Génération et Évaluation de Visualisations avec des techniques d'Apprentissage Automatique

Loann Giovannangeli

université de BORDEAUX

LaBRI

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR
DE L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE MATHÉMATIQUES ET INFORMATIQUE

Par **Loann GIOVANNANGELI**

Génération et Évaluation de Visualisations avec des techniques d'Apprentissage Automatique

Sous la direction de : **Romain BOURQUI**

Soutenue le 20 novembre 2023

Membres du jury :

M. Daniel ARCHAMBAULT	Professeur des Universités	Université de Newcastle	Rapporteur
M. Alexandru C. TELEA	Professeur des Universités	Université d'Utrecht	Rapporteur
M. Guy MELANÇON	Professeur des Universités	Université de Bordeaux	Examinateur
M. Arnaud SALLABERRY	Maître de Conférences	Université de Montpellier	Examinateur
M. Akka ZEMMARI	Professeur des Universités	Université de Bordeaux	Président
M. Romain BOURQUI	Maître de Conférences	Université de Bordeaux	Directeur
M. David AUBER	Professeur des Universités	Université de Bordeaux	Invité
M. Romain GIOT	Maître de Conférences	Université de Bordeaux	Invité

A mon grand-père, Jacques.

"*Le travail, c'est magnifique... je le regarderais faire pendant des heures.*"

Remerciements

Premièrement, je veux remercier le Ministère français de l'Enseignement supérieur et de la Recherche de m'avoir attribué une bourse pour financer mes travaux de thèse, ainsi que l'Université de Bordeaux et le LaBRI pour m'avoir permis de les réaliser dans de bonnes conditions. Je remercie également tous les membres de mon jury de thèse, **Daniel Archambault**, **Alexandru C. Telea**, **Guy Melançon**, **Arnaud Sallaberry** et **Akka Zemmari** d'avoir accepté de relire et d'évaluer mes travaux de thèse. Vos remarques et nos discussions ont été une grande source d'amélioration.

Je remercie chaleureusement **Romain Bourqui**, Maître de conférences à l'Université de Bordeaux, de m'avoir donné la chance de réaliser mes travaux auprès de lui et de m'avoir transmis sa culture scientifique. Je tiens à lui exprimer toute ma gratitude et mon amitié pour tous les échanges que nous avons eus et le temps qu'il m'a consacré. Cela aura été un honneur et un plaisir de relever les défis que peut comporter une thèse avec lui.

Je souhaite également remercier **David Auber**, Professeur à l'Université de Bordeaux, de m'avoir accompagné dans mes études. Tout d'abord en Master, puis en Doctorat, il m'a toujours encouragé à aller plus loin. Je n'aurais jamais osé me lancer dans un doctorat sans une certaine obstination de sa part, et je lui en suis reconnaissant.

Je tiens également à remercier **Romain Giot**, Maître de conférences à l'Université de Bordeaux, et **Frédéric Lalanne**, Ingénieur de recherche à l'Université de Bordeaux, pour leur accompagnement et leur soutien quotidien. De la conception à la rédaction, en passant par l'implémentation, les travaux réalisés dans cette thèse n'auraient pu être réalisés sans leur appui.

Je remercie d'autre part tous les membres de l'équipe BKB, du département SeD et de l'axe IA pour leur bonne humeur et leurs qualités humaines indéniables. Je tiens à signifier une reconnaissance particulière aux doctorants de l'équipe BKB, Luc, Yanis, Elsa, Galadriel, Myriam et Adrien pour la bonne ambiance et l'émulation qui m'aura motivé à chaque instant.

Enfin, je tiens à remercier tous les membres de ma famille pour le soutien qu'ils m'ont apporté. Je remercie tout particulièrement ma grand-mère pour son investissement et les sacrifices qu'elle a réalisés pour me permettre de réaliser ce doctorat dans de telles conditions. Je remercie également ma mère et mon oncle pour leur soutien et leurs conseils, et je pense également à mon défunt grand-père qui m'a donné sa passion pour la culture et la connaissance, et à qui je dédie cette thèse. Pour finir, je souhaite remercier C., D.N., H. et S. pour leur fidèle support et pour m'avoir réconforté dans les moments difficiles.

Génération et Évaluation de Visualisations avec des techniques d'Apprentissage Automatique

Résumé : L'essor de l'Internet des Objets, des modèles de stockage et de traitement des données a conduit à une explosion de la quantité et de la complexité des données que nous collectons aujourd'hui. Pour mieux les comprendre et les manipuler, les experts ont recours à des visualisations de l'information. Cependant, la complexité nouvelle des données rend inefficace des techniques de visualisation utilisées jusqu'alors. Il est donc nécessaire de concevoir de nouvelles techniques de visualisation et de revoir les méthodes d'évaluation qui permettent de mesurer leur efficacité.

Cette thèse présente des contributions sur deux aspects principaux du domaine de la Visualisation d'Information : la génération et l'évaluation automatique de visualisations. Pour ces deux axes, nos travaux tirent parti des techniques d'apprentissage automatique, notamment profond, qui ont démontré leurs capacités à traiter efficacement des grands volumes de données. Les cas d'applications des contributions présentées dans ce manuscrit utilisent des modèles d'apprentissage automatique pour le Dessin de Graphe par représentation Nœud-Lien, la Suppression de Chevauchements dans des nuages de points, et l'Évaluation automatique de Visualisations.
Mots-clés : Visualisation, Évaluation, Apprentissage Automatique, Apprentissage Profond

---

Generation and Evaluation of Visualizations with Machine Learning Techniques

Abstract: The rise of the Internet of Things, data storage and management models has led to an explosion in the quantity and complexity of the data we collect today. To understand these data and better manipulate them, experts are relying on information visualization. However, the actual complexity of these new data makes former visualizations ineffective. It becomes necessary to design new visualization techniques and review the evaluation methods used to measure their effectiveness.

This thesis presents contributions to the field of Information Visualization on two main aspects: automatic generation and evaluation of visualizations. For these two axes, our work leverages machine and deep learning techniques that have demonstrated their ability to efficiently process large volumes of data. The applications of the contributions presented in this manuscript make use of Machine and Deep Learning models for Graph Drawing with Node-Link representations, Overlap Removal in point clouds (scatter plots), and Automated Evaluation of Visualizations.
Keywords: Visualization, Evaluation, Machine Learning, Deep Learning

---

Unité de recherche
LaBRI (UMR CNRS 5800), 351 Cours de la Libération, 33405 Bordeaux, France.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

	Anglais	*Français*
VIS	(Information) Visualization	Visualisation (d'Information)
AM	Adjacency Matrix	Matrice d'Adjacence
NL	Node-Link	Nœud-Lien
MDS	Multi-Dimensional Scaling	Positionnement Multi-dimensionnel
OR	Overlap Removal	Suppression de Chevauchements
GS	Geometric Space	Espace Géométrique
VS	Visual Space	Espace Visuel
AI et **IA**	Artificial Intelligence	Intelligence Artificielle
ML	Machine Learning	Apprentissage Automatique
DL	Deep Learning	Apprentissage Profond
XAI	eXplainable Artificial Intelligence	Intelligence Artificielle Explicable
SGD	Stochastic Gradient Descent	Descente de Gradient Stochastique
GT	Ground Truth	Vérité Terrain
DNN	Deep Neural Network	Réseau de Neurones Profond
CNN	Convolutional Neural Network	Réseau de Neurones Convolutif
GNN	Graph Neural Network	Réseau de Neurones de Graphe
GCN	Graph Convolutional Network	Réseau Convolutif de Graphe
ER	Error Rate	Taux d'Erreur
RT	Response Time	Temps de Réponse

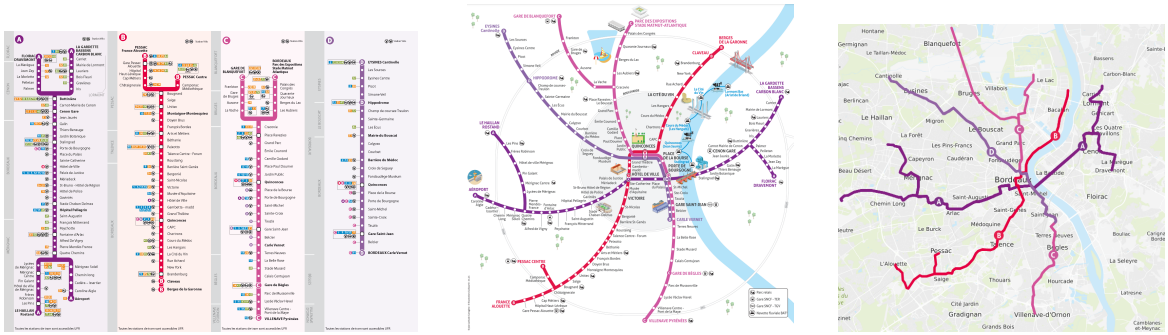
Chapitre I

Introduction

I.1 Visualisation d'Information (VIS)

L'essor conjoint des systèmes de stockage de données, traitements de données et modèles de calculs a mené à une explosion de la complexité des systèmes d'information. Ces avancées ont été le support de la démocratisation de l'*Internet des Objets*, lui-même à la base de nouvelles visions de l'organisation de notre société (*e.g.*, *Smart cities*, voitures autonomes communicantes). La philosophie des systèmes d'information *Big data* et ses *lacs de données* implique que toute information doit être stockée car celle-ci pourrait un jour être valorisable. Les volumes et la complexité de ces nouvelles données soulèvent alors de nouveaux problèmes, notamment leur analyse et leur exploitation efficace tandis que les outils classiques (*e.g.*, graphiques, tableurs) ne permettent plus aux experts de les manipuler facilement.

La *Visualisation d'Information* est une des techniques établies pour explorer efficacement ces grands volumes de données trop complexes pour être manipulés avec des méthodes traditionnelles (*e.g.*, requêtage). Elle consiste à proposer des représentations abstraites des données facilement explorables pour permettre aux utilisateurs/experts d'extraire de la connaissance afin de résoudre des tâches spécifiques (*e.g.*, trouver des éléments ou des motifs) ou de découvrir de nouvelles structures. Pour être efficace, la conception de la visualisation repose sur trois axes principaux. Premièrement, le choix de la technique de visualisation. C'est elle qui détermine quelles sont



(a) Séquence et embranchements (b) Carte abstraite (c) Carte réelle

FIGURE I.1 : Exemples de trois représentations des quatre lignes de tramway de la ville de Bordeaux. Chaque figure est le résultat d'une approche différente à la visualisation de ces lignes. En fonction de l'approche utilisée, il est possible d'encoder d'autres informations plus ou moins facilement (*e.g.*, points d'intérêts, correspondances). Les trois approches ne mettent pas en valeur les mêmes propriétés du réseau et ne sont pas efficaces pour les mêmes raisons.

les données à représenter ainsi que les composants constituant la visualisation. Une fois la technique de visualisation choisie, le deuxième axe est le positionnement des éléments à afficher dans le plan. Cette étape peut s'effectuer manuellement, ou être le résultat d'un algorithme de positionnement. Enfin, une fois positionnés, il faut choisir une méthode de représentation graphique de ces éléments : choisir des attributs visuels (*e.g.*, forme, couleur) et leurs valeurs (*e.g.*, carré, rond). La Figure I.1 illustre différents exemples de représentation de la même donnée en entrée : les lignes de tramway de la ville de Bordeaux. Dans la Figure I.1a, l'approche de représentation choisie est la juxtaposition des lignes, elles-mêmes représentées comme des séquences présentant les arrêts et leurs correspondances. Les éléments sont positionnés dans le sens de la ligne, de manière orthogonale pour suggérer le concept de *chaîne* qui modélise bien le trajet d'un tramway. Enfin, les couleurs, la forme, et la police déterminent le type des arrêts. Dans les Figures I.1b et I.1c, les choix pour représenter cette même information sont différents et se traduisent par des visualisations hétérogènes n'ayant pas tout à fait le même objectif. La Figure I.1a décrit le plus simplement possible l'information des lignes de tramway, et permet une recherche rapide d'une information sur une de ces lignes. C'est également la visualisation qui nécessite le moins de place et est particulièrement adaptée à l'affichage à l'intérieur d'une rame de tramway. Les Figures I.1b et I.1c donnent une vision plus générale du réseau de la ville de Bordeaux, rendant la recherche d'un arrêt en particulier plus difficile, mais facilitant l'identification de correspondances et de points d'intérêts. Dans tous les cas, les choix réalisés lors de la conception d'une technique de visualisation ont pour but de faciliter la transmission d'informations. Pour cela, les recherches en Visualisation d'Information se basent fortement sur les travaux de Perception (voir Section II.4.1) afin de tirer le meilleur parti de l'acuité visuelle des utilisateurs [1].

Plus formellement, dos Santos et Bordlie [2] ont proposé un schéma de traitement de l'information dans le processus de visualisation, appelé *Visualization pipeline*. La Figure I.2 illustre une partie de ce schéma en rapport avec les travaux présentés dans ce document. Dans

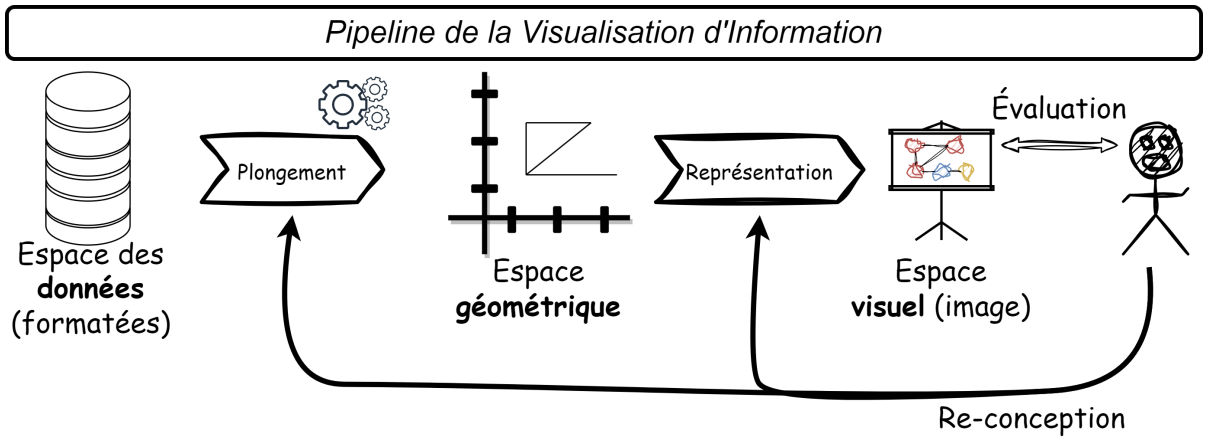


FIGURE I.2 : Schéma représentant la chaîne de traitement de l'information pour produire une visualisation. Les premières étapes de collecte et nettoyage de données sont éludées, de telle sorte que le traitement commence à partir de données formatées. Ces données sont plongées dans un espace *géométrique* pour satisfaire certains critères, puis représentées dans un espace *visuel* pour être interprétées par un utilisateur final qui peut ensuite rétro-agir sur le processus de conception de la visualisation. L'exemple utilisé dans ce schéma est le dessin de graphe nœud-lien (voir Section II.1.1). Cette figure est inspirée du *Visualization pipeline*[1] de dos Santos et Brodlie [2].

[1] `www.infovis-wiki.net/wiki/Visualization_Pipeline`

ce schéma, nous décomposons le contenu de l'étape de *Mapping* de dos Santos et Bordlie [2]. Pour faire le lien avec les axes exposés précédemment : (i) la structure des données formatées détermine l'abstraction utilisée et les éléments à afficher, (ii) l'algorithme de plongement définit la stratégie de positionnement des éléments dans le plan ; et (iii) une étape de représentation permet d'associer des propriétés visuelles aux éléments et au contexte (*e.g.*, arrière plan) pour produire l'image de la visualisation. Lors du processus de conception de la technique de visualisation, il est courant d'évaluer la qualité de la visualisation produite par le système et de rétro-agir dessus dans le but de l'améliorer. Cette évaluation peut être subjective : basée sur l'expérience de l'expert, le plus souvent durant la phase de conception initiale ; ou formelle : mesurée pendant une évaluation, pour comparer son efficacité aux techniques existantes.

Dans cette thèse, nous nous concentrons principalement sur le plongement et la représentation des données, ainsi que l'évaluation des visualisations qui en résultent. Les cas d'application de nos travaux sont principalement le dessin de graphe, et la visualisation de nuages de points. Dans ces deux cas, l'objectif est de plonger un ensemble de données en hautes dimensions vers un espace 2D qui préserve certaines propriétés de cet ensemble.

I.2 Intelligence Artificielle (IA)

Si n'importe quel algorithme manipulant des données peut être considéré *intelligent*, l'Intelligence Artificielle réfère de nos jours aux méthodes basées sur l'apprentissage : l'Apprentissage Automatique (ML pour « Machine Learning ») et Apprentissage Profond (DL pour « Deep Learning »). L'intuition du fonctionnement de ces algorithmes d'apprentissage est qu'à partir des volumes de données dont nous disposons, il est possible de paramétrer un modèle pour qu'il sache reconnaître les motifs constituant ces données. Dans les approches ML standard, ce paramétrage est en partie manuel : l'expert doit extraire lui-même les caractéristiques importantes des données, le modèle se contentant d'apprendre à les exploiter. Le domaine DL est une sous-partie du domaine ML dans laquelle les modèles sont notamment capables d'apprendre à extraire ces caractéristiques importantes par eux-mêmes. La régression linéaire et les forêts d'arbres décisionnels [3] sont des exemples classiques d'algorithmes de ML, tandis que les modèles de DL sont exclusivement des réseaux de neurones profonds (DNN pour « Deep Neural Networks »).

Aujourd'hui, les algorithmes ML sont généralement préférés lorsque le volume de données disponible pour l'apprentissage est faible, tandis que l'entraînement de DNN nécessite de grands volumes. Compte tenu de ces contraintes, couplées à la philosophie du *Big data*, il n'est donc pas surprenant que les DNN soient à l'origine des grandes avancées en matière d'IA au cours de ces dernières années (*e.g.*, traitement d'image [4, 5], du langage [6]).

Si l'essor de ces techniques provient en partie des grands volumes de données que nous sommes aujourd'hui capables de collecter pour les entraîner, leurs avancées sont également supportées par les améliorations du matériel sur lequel ces modèles sont entraînés. En effet, la plupart s'implémentent sous la forme de calculs matriciels sur des *lots* (*batch* en anglais) de données. Ils peuvent ainsi tirer parti des accélérations de calculs proposées par les Processeurs Graphiques (GPU pour « Graphics Processing Unit ») [7]. Nous détaillons le fonctionnement de ces modèles d'apprentissage dans la Section II.1.

L'utilisation des modèles d'apprentissage s'est démocratisée et ceux-si sont désormais utilisés dans des domaines variés pour réaliser tout type de tâche. Le domaine de la Visualisation d'Information ne fait pas exception à cet engouement. Ainsi, deux thématiques principales ont

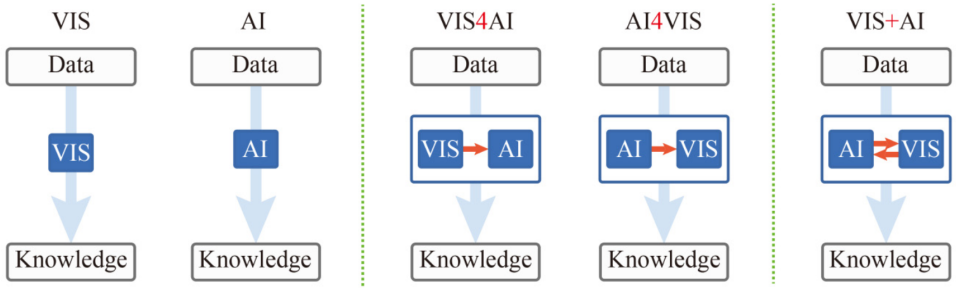


FIGURE I.3 : Illustration des interactions entre algorithmes de VIS et IA pour extraire de la connaissance des données. Cette illustration est tirée de l'étude VIS+AI [8].

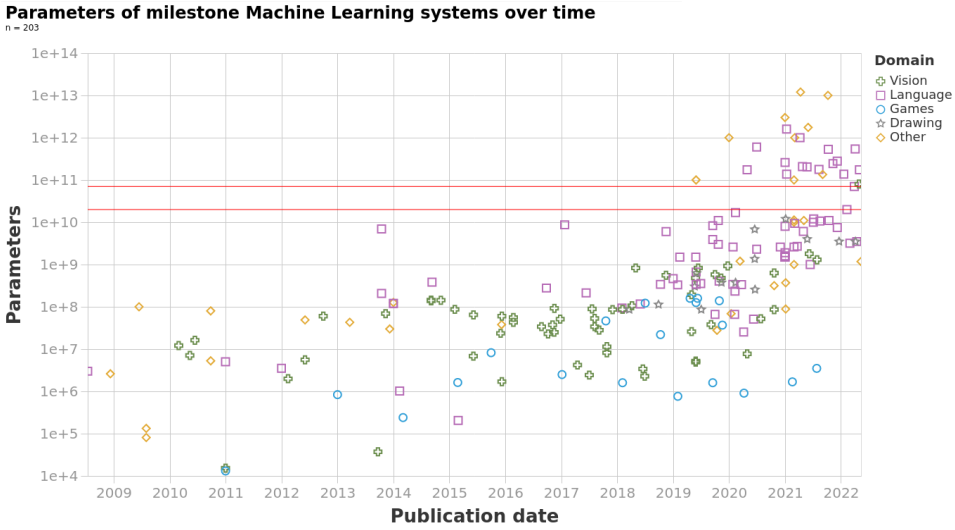


FIGURE I.4 : Évolution du nombre de paramètres dans les réseaux de neurones profonds de la littérature. Cette illustration est tirée de l'article de Villalobos *et al.* [9].

émergé : VIS4IA et IA4VIS (voir Figure I.3).

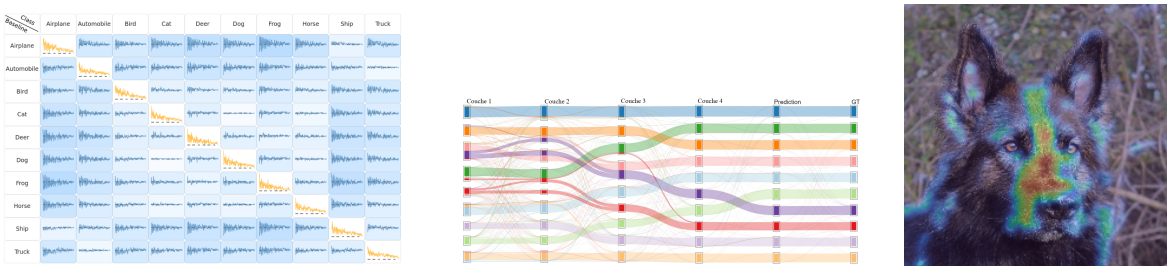
La création de la thématique VIS4IA a été motivée par le constat que les techniques d'IA, notamment les DNN, sont des modèles *boîtes noires* dont il est difficile de comprendre le fonctionnement et la prise de décision. En effet, bien que les opérations qui les constituent soient connues et maîtrisables, les DNN agencent parfois des milliards voire des milliers de milliards de paramètres d'apprentissage [9] (voir Figure I.4). Il devient alors complexe de relier les estimations du modèle d'apprentissage à sa donnée en entrée. La thématique VIS4IA a donc pour objectif de proposer des visualisations ou des systèmes de visualisation pour faciliter la conception des DNN et expliquer leurs comportements une fois appris (*e.g.*, [10–14]). Ici, le modèle d'IA est alors l'objet d'étude.

À l'inverse, les applications relatives à IA4VIS sont utilisatrices de modèles d'apprentissage. Le temps d'exécution très court de l'inférence des modèles d'IA entraînés devrait permettre le passage à l'échelle de certaines techniques de visualisation, et améliore donc leur interactivité. L'IA peut être utilisée pour simplifier la donnée à visualiser, projeter la donnée dans le plan, voire évaluer les visualisations produites. De plus, l'entraînement de ces modèles peut être volontairement biaisé pour leur faire intégrer des comportements humains. Leur performance et flexibilité ont ouvert de nouvelles possibilités qui sont désormais étudiées par les experts en visualisation [8, 15, 16].

I.3 VIS4IA : VIS pour l'IA

Sur des tâches telles que la classification d'image [4] ou le traitement du langage [6], les méthodes d'IA qui obtiennent aujourd'hui les meilleures performances sont principalement les approches basées sur des réseaux de neurones profonds. Malgré leurs capacités, ces réseaux ont des architectures complexes, composées de plusieurs millions voire milliards de paramètres, ce qui les rend difficiles à appréhender pour un humain. Leur complexité devient d'autant plus problématique que ces réseaux ne sont pas infaillibles et peuvent aussi prédire de mauvaises estimations pouvant s'avérer particulièrement critiques dans certains cas d'application (*e.g.*, une voiture autonome ne détecte pas un obstacle devant elle [17]). Pour pouvoir déployer ce type de modèle à large échelle, il devient alors impératif de comprendre les causes qui ont mené ces modèles à leurs prédictions, qu'elles soient correctes ou non. L'objectif est de construire une *IA de confiance* [18] pour favoriser son acceptabilité dans certains domaines. Par ailleurs, la compréhension de ces modèles permet d'améliorer leur conception et ainsi augmenter leurs performances. La méthode d'explication elle-même peut parfois améliorer les performances de l'IA [10]. Enfin, le droit à l'explication des décisions issues de processus automatiques, dont les modèles d'apprentissage font partie, fait l'objet d'une règle imposée par la loi européenne RGPD[1] depuis 2018.

Pour répondre à ce défi sur une technologie en plein essor, une partie de la communauté des chercheurs utilisant les réseaux de neurones profonds s'est impliquée dans une nouvelle thématique : l'explicabilité (XAI pour « eXplainable Artificial Intelligence »). Plusieurs axes ont été explorés pour analyser le comportement du réseau [11–14, 21], du point de vue de ses composantes (couches, neurones) ; ou du point de vue des données utilisées pour le nourrir. L'objectif final des méthodes XAI étant de permettre aux humains de comprendre le fonctionnement de ces réseaux *boîtes noires*, la plupart de ces analyses proposent des visualisations aux utilisateurs. Le type de visualisation dépend de l'objet étudié et de la méthode XAI employée. Par exemple, le rôle des paramètres d'une couche d'un réseau de neurones convolutif peut être illustré à travers leur pouvoir discriminatoire, dans une matrice [10]. Comme présenté dans la Figure I.5a, chaque case de cette matrice montre la capacité d'un ensemble de paramètres à distinguer deux classes d'un jeu de données à l'aide d'une sparkline. Dans d'autres approches, l'ensemble des données d'une même classe peut être visualisé grâce à un diagramme de Sankey (ou diagramme de flux) illustrant comment les données sont regroupées ou discriminées tout au long de leur passage dans le réseau [19] (voir Figure I.5b). Ce type d'approche se base sur les sorties des neurones



(a) Matrice de Sparklines [10] (b) Diagramme de Sankey [19] (c) Carte de saillance [20]

FIGURE I.5 : Exemples de visualisations utilisées dans des méthodes d'explicabilité. Chacune de ces techniques sert à visualiser une partie du processus de prise de décision d'un modèle de réseau de neurones profond afin d'en améliorer notre compréhension.

[1] www.privacy-regulation.eu/fr/r71.htm

de chaque couche et permet notamment de visualiser efficacement quelle profondeur de réseau est requise pour distinguer certaines classes. Enfin, les approches les plus étudiées aujourd'hui permettent d'expliquer les caractéristiques importantes utilisées par un réseau pour produire son estimation sur une donnée [22–24]. Cette importance est généralement visualisée comme une carte de chaleur (ou saillance) superposée à la donnée d'origine (voir Figure I.5c). Par exemple, H^2O [20] est une méthode d'explicabilité par occultation hiérarchique qui permet d'obtenir des cartes de saillance précises et fortement localisées mettant en évidence les caractéristiques importantes à la prise de décision du réseau.

I.4 IA4VIS : IA pour la VIS

IA4VIS est une thématique dans laquelle les algorithmes d'Intelligence Artificielle sont utilisés pour visualiser la donnée de manière efficace. Concrètement, ces algorithmes peuvent être utilisés pour *générer* [25, 26], *traiter* [27] ou *analyser* [28–30] des visualisations. Sous forme d'algorithme [25, 26, 29] ou dans un système complet [31–33], l'objectif de cette approche est de tirer parti des capacités des modèles d'apprentissage pour (i) extraire automatiquement les caractéristiques principales d'une donnée, de manière à réduire le nombre d'éléments à afficher ; et/ou (ii) produire une visualisation efficace pour explorer un ensemble de données.

Cette thématique a émergé dans les années 2015 et se démocratise de plus en plus dans la littérature de Visualisation, comme illustré dans la Figure I.6. Aujourd'hui, les trois études recensant les principaux travaux dans cette thématique sont AI4VIS [15], VIS+AI [8] et ML4VIS [16]. AI4VIS [15] et VIS+AI [8] synthétisent principalement les tâches et objectifs de IA4VIS, et expliquent le *quoi* et *pourquoi* de la thématique. ML4VIS [16] synthétise la manière dont les études de la littérature ont formalisé les problématiques de VIS pour être traitées par des modèles d'apprentissage. L'étude s'intéresse également aux méthodes de mise en œuvre de ces modèles d'apprentissage (*e.g.*, type de modèle, tâche étudiée). Elle décrit ainsi le *comment* de la thématique.

Dans la suite de cette section, nous présentons les tâches de IA4VIS traitées dans nos travaux qui seront détaillés dans les chapitres de ce document. En outre, nous concentrons nos études sur les tâches de *génération* et *analyse* (*i.e.*, évaluation) via IA4VIS.

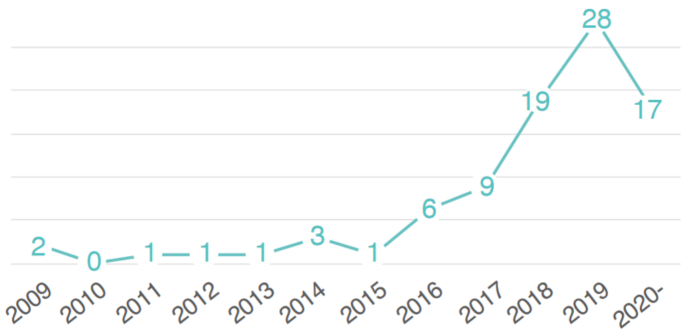


FIGURE I.6 : Nombre d'articles publiés traitant de IA4VIS dans des revues de Visualisation (VIS), Interface Homme-Machine (HCI) et Exploration de données (DMM). Cette figure est tirée de l'étude ML4VIS [16].

I.4.1 IA4VIS Génération

L'idée d'utiliser des algorithmes d'IA pour générer des visualisations vient du constat qu'un certain nombre de visualisations sont produites via des plongements de données. Le fondement de ces algorithmes de visualisation est alors la méthode de plongement utilisée. Il n'est donc pas surprenant que les experts se soient intéressés aux méthodes d'IA dont les performances ne cessent de s'améliorer depuis plusieurs années, et qui sont par nature capables de traiter des problèmes de plongement (*i.e.*, régression). Par exemple, l'étude de Espadoto *et al.* [34] a montré qu'il était possible d'apprendre à des réseaux de neurones profonds à reproduire les algorithmes de plongement en nuages de points de la littérature (*e.g.*, t-SNE [35], UMAP [36]). De nombreux algorithmes de dessin de graphe se basent également sur la descente de gradient stochastique [37–41] ou des réseaux de neurones plus ou moins profonds [25, 26, 42–45].

Si l'utilisation de descente de gradient stochastique est commune dans la visualisation d'information, la nouveauté consiste à utiliser des réseaux de neurones profonds. L'intérêt de ces réseaux est multiple dans le contexte de la génération de visualisations. (i) Ils sont capables d'apprendre par eux-même l'information importante à extraire de la donnée d'origine, ce qui permet à l'algorithme de visualisation de s'adapter à sa donnée en entrée, et nécessite donc moins de paramétrage manuel. (ii) Il est possible de les biaiser de manière à intégrer des problématiques métiers, voire du raisonnement humain, afin de produire des visualisations ayant un objectif précis. Enfin, (iii) une fois entraînés, leur temps d'exécution est relativement court. Beaucoup d'algorithmes de visualisation ou de plongement ayant une complexité quadratique sur le nombre d'éléments, l'utilisation de réseaux de neurones profonds permet de bénéficier d'algorithmes produisant des visualisations de qualité en un temps d'exécution permettant l'interaction avec un utilisateur.

Les applications mettant en œuvre l'approche IA4VIS prennent la forme d'algorithmes, voire de systèmes entiers. Par exemple, VividGraph [32] ainsi que son extension GraphDecoder [33] proposent des solutions d'extraction de dessin de graphe pour une re-conception ultérieure. Si l'objectif original de ces méthodes est d'extraire la donnée originale à partir d'une image de leur représentation, ces systèmes permettent également la visualisation interactive de la donnée extraite afin d'en faciliter l'exploration. GraphDecoder propose d'ailleurs différentes techniques d'affichage pour cela. DashBot [31] est un autre exemple de système complet IA4VIS. Ce dernier est dédié à la création de tableau de bord (*Dashboard* en anglais) à partir de données tabulaires (CSV). L'IA du système permet d'automatiser l'exploration des colonnes du CSV en fonction de leur type et variabilités. Elle est ainsi capable de déterminer l'importance et le rôle des colonnes, puis de proposer un tableau de bord décrivant la table de données. Elle recommande également à l'utilisateur des *insights* pour re-organiser automatiquement le tableau de bord en fonction d'autres critères.

Nous pouvons également regrouper les algorithmes de *traitement* de visualisation avec ceux de *génération* car leur objectif est dans tous les cas de produire une nouvelle visualisation, qu'elle soit générée ou ajustée. La différence réside donc dans le type de données qu'ils traitent en entrée. Ainsi, les algorithmes d'*amélioration* de visualisation [46–49] sont considérés comme des méthodes de *génération* de visualisation.

Dans une moindre mesure, les algorithmes de peinture [50] ou générateurs d'images [51, 52] peuvent également être considérés comme des applications visant à générer des visualisations via des techniques d'IA. Si ces techniques génèrent bien des images (*e.g.*, DALL.E[1]), celles-ci ne sont pas proprement dédiées à la visualisation d'information (*i.e.*, il n'y a pas de *donnée* à visualiser).

[1] `www.openai.com/dall-e-2`

I.4.2 IA4VIS Évaluation

La Visualisation d'Information s'étant établie comme une manière efficace d'explorer des données complexes, son essor a mené à une explosion du nombre de techniques de représentation des données [53]. Il est alors commun de se questionner sur la technique à utiliser pour visualiser une donnée particulière [54, 55], ou pour comparer une nouvelle méthode à celles déjà établies dans la littérature. Cependant, leur grande variété a mené à une question proche du domaine de la Perception d'Information : comment évaluer l'efficacité d'une technique de visualisation pour résoudre une tâche ?

La manière de répondre à cette question la plus admise dans la littérature est l'*évaluation utilisateur*. À partir d'un ensemble de paramètres fixes (*e.g.*, données, environnement), l'idée est de mesurer les performances de groupes d'utilisateurs pour résoudre une tâche. En comparant leur performances, il est alors possible de déduire l'impact d'un paramètre de la visualisation ou de sa donnée en entrée sur la difficulté de la tâche. En réalité, puisque quantifier la qualité d'une visualisation est compliqué, la plupart des évaluations comparent plusieurs techniques de visualisation pour mesurer leurs performances relatives (*i.e.*, « VIS_A *est meilleur que* VIS_B *quand [condition]* »). Le principal avantage de cette méthode d'évaluation est que l'efficacité de la technique de visualisation est directement mesurée sur un échantillon de la population d'utilisateurs finaux potentiels (*i.e.*, des humains). Cependant, cette approche comprend plusieurs limites et défauts contraignants. Par exemple, (i) ces expériences sont coûteuses (*e.g.*, en temps) à mettre en place et à faire passer à des participants. (ii) Une attention très particulière doit être portée aux choix de conception (*e.g.*, sélection des cas de tests, durée de l'évaluation, population de participants). (iii) Les conditions d'expérimentation (*i.e.*, l'environnement) doivent être équitablement propices au passage de l'évaluation pour tous les participants. (iv) Il peut être difficile de recruter un nombre significatif de participants. (v) D'une manière générale, ces expérimentations sont difficilement reproductibles car trop de paramètres d'environnement et de choix de conception peuvent affecter les résultats. Bien qu'il soit possible de contrôler et minimiser l'impact de ces inconvénients sur l'expérience, il reste systématiquement des choix de conception subjectifs contestables, ou qui deviendront contestables à mesure que les standards des protocoles d'évaluation évolueront.

Une alternative qui pallie la plupart de ces inconvénients est l'*évaluation automatique*. La méthode conventionnelle pour la mettre en œuvre s'effectue au travers d'une évaluation quantitative basée sur des métriques de qualité. Elle permet notamment de comparer différents algorithmes de calcul, ou de rendu, sur les mêmes types de représentation, comme illustré dans la Figure I.7. Cependant, cette approche aussi a ses limites. Si certaines études ont validé la corrélation entre l'optimisation d'une métrique avec la préférence humaine [56–58], ce n'est pas le cas pour toutes les métriques. Il est alors difficile de savoir si un gain sur une métrique est réellement représentatif d'une amélioration de la qualité de la représentation pour un humain. De plus, il est difficile pour ces métriques de refléter la qualité d'une visualisation pour résoudre une tâche de haut niveau qui peut nécessiter l'interprétation d'éléments graphiques à différents niveaux d'abstraction. Enfin, il n'existe pas (ou peu) de métrique capable de s'adapter à des techniques de visualisation foncièrement différentes permettant de visualiser une même donnée (*e.g.*, Figure I.8).

C'est pour proposer une solution aux problèmes posés par l'évaluation utilisateur et l'évaluation automatique par métrique qu'émerge l'idée d'utiliser des modèles d'apprentissage profond pour évaluer des techniques de visualisation [15, 16, 29, 61]. Cette idée se base sur la capacité de ces réseaux à apprendre par eux-même à extraire l'information des visualisations, et la traiter à différents niveaux d'abstraction. D'une certaine manière, leur traitement de l'information est

(a) Dessin par Kamada-Kawai [59]
$Stress = 45$

(b) Dessin par PivotMDS [60]
$Stress = 39$

FIGURE I.7 : Deux visualisations du même graphe par représentation nœud-lien. L'utilisation de la même technique d'encodage de l'information (*i.e.*, formes géométriques reliées par des traits) permet l'évaluation automatique de l'impact des paramètres de la visualisation (*e.g.*, positionnement, couleur, transparence des nœuds) sur sa qualité avec des métriques. Par exemple, une évaluation du *Stress* permet de quantifier automatiquement la qualité du positionnement des nœuds.

(a) Matrice d'adjacence (AM)

(b) Représentation nœud-lien (NL)

FIGURE I.8 : Deux techniques de représentation de graphe, par (a) matrice d'adjacence et (b) nœud-lien. Ces deux techniques servent à visualiser le même objet (graphe) avec ses composants (nœuds et arêtes) mais n'utilisent pas du tout les mêmes encodages. Il est quasi-impossible de les évaluer automatiquement car peu de métriques sont applicables aux deux.

transposable au système de perception humain. Il est alors possible de leur apprendre toute tâche exprimable programmatiquement. Il est également possible de les entraîner directement à partir de l'image produite par une visualisation, afin de mesurer l'impact de certains choix de rendu dans la visualisation. En fonction du temps que met le modèle à apprendre et des performances qu'il obtient, il est alors possible d'évaluer la difficulté de la tâche à résoudre, et par extension la qualité de la visualisation. Cette méthode d'évaluation automatique est également reproductible. En effet, il est possible de partager les données, l'architecture du réseau de neurones utilisée, ainsi que les valeurs de ses paramètres appris pour permettre à d'autres experts de reproduire ou étendre une évaluation. Enfin, un dernier avantage majeur est qu'elle est capable, et à même besoin, de traiter un grand volume de données. Dans la plupart des techniques de visualisation, les variations possibles des paramètres de l'algorithme et des propriétés des données en entrée (*i.e.*, l'*espace de paramètres*) conduisent à l'explosion combinatoire des images possibles en sortie (des centaines de milliers, voire des millions). Or, dans une évaluation utilisateur, la durée de concentration des participants, ainsi que le temps qu'ils peuvent accorder à l'expérience, sont limités. Il est donc commun de sous-échantillonner l'espace de paramètres pour ne tester que certains cas jugés intéressants à évaluer. Ces choix sont généralement réalisés sur une stratégie d'échantillonnage régulière, aléatoire ou arbitraire. Cependant, là où le nombre de

cas de tests doit être limité pour une évaluation utilisateur, les réseaux de neurones profonds, quant à eux, bénéficient largement de jeux d'apprentissage volumineux. Aussi, il est possible, et même souhaitable, d'explorer largement l'espace de paramètres pour entraîner le réseau. Cela permet à la fois de créer un jeu d'apprentissage suffisamment conséquent pour entraîner de tels modèles, et d'évaluer les variations de la difficulté de la tâche à résoudre sur un ensemble plus large de l'espace de paramètres. Il est également connu que des réseaux de neurones profonds arrivent à résoudre la plupart des tâches, tant que leur jeu de données est assez volumineux et leur paramétrage affiné. Dans le contexte de l'évaluation automatique, ce n'est pas le but recherché. L'objectif est plutôt de prendre des architectures communes issues de la littérature, en fonction de la donnée à traiter, de la tâche à résoudre et des opérations qui les composent. En limitant leur paramétrage, il devient alors plus facile de relier des variations dans leurs performances avec des propriétés de la donnée en entrée. En outre, il n'est pas nécessaire, et encore moins souhaitable, que le réseau de neurones utilisé se sur-spécialise. Un modèle ayant de trop bonnes performances ne permettrait pas d'étudier les conditions limites qui le font échouer car ces cas seraient trop peu nombreux.

Cette approche se base sur une supposition forte : les performances d'un réseau de neurones profond peuvent être corrélées à celles d'humains sur des tâches de perception. La véracité de cette supposition dépend fortement de l'architecture utilisée et de la tâche à résoudre. Par exemple, un réseau de neurones convolutif traite la donnée en entrée (*i.e.*, tableau de pixels) en extrayant de l'information vers des niveaux d'abstraction de plus en plus élevés. Ce fonctionnement est très proche du traitement de l'information visuelle faite par la perception humaine dans le cas où nous recherchons un élément dans une image, avec une stratégie *bottom-up* (*e.g.*, « Où est Charlie ? », voir Figure I.9). Le processus *bottom-up* consiste à se concentrer sur les petit éléments considérés *atomiques*, puis d'agréger des informations acquises à ce niveau pour déduire et interpréter des concepts de plus haut niveau. Dans l'exemple de la Figure I.9, cela revient à diriger son attention séquentiellement sur chaque personnage de l'image, et le comparer au concept plus haut niveau de Charlie. À l'inverse, le processus *top-down* correspond à la stratégie qui consiste à balayer de larges régions de l'image, puis restreindre la concentration finement à des éléments

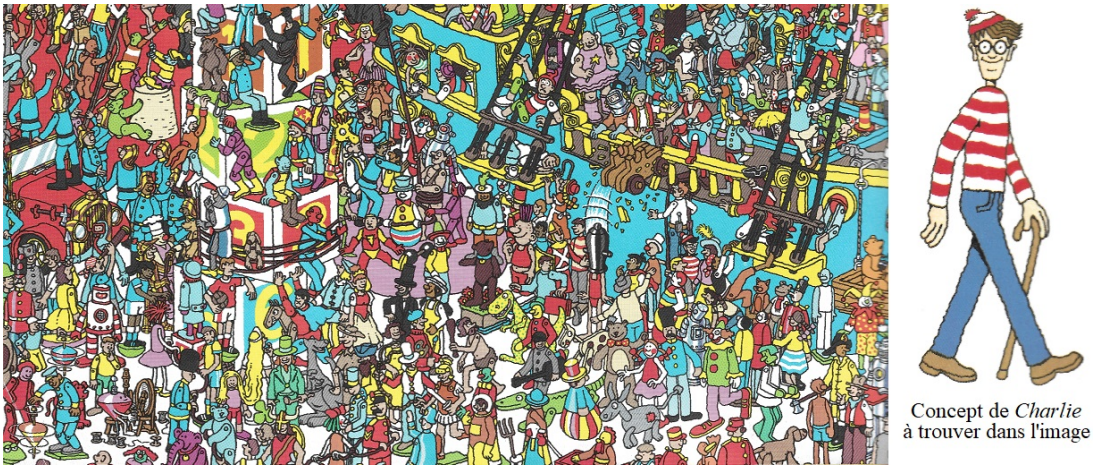


FIGURE I.9 : Image d'un ouvrage de « Où est Charlie ? » [62]. L'objectif est de trouver un personnage unique avec des vêtements et un visage connu, toujours le même (*i.e.*, Charlie). Il est commun que les individus confrontés à cet exercice ressentent les deux stratégies de la perception : « *top-down* » (étude rapide d'éléments ressemblant au concept connu de Charlie) puis « *bottom-up* » (étude séquentielle des personnages pour vérifier s'ils correspondent au concept, Charlie).

de plus en plus bas niveaux attirant l'attention. Dans l'exemple de la Figure I.9, cela consiste à diriger son attention vers les régions où des textures rappellent celles de Charlie, puis de vérifier si ces textures proviennent bien de ce personnage. D'une manière plus générale, l'hypothèse de la corrélation entre le système de traitement de l'information des réseaux de neurones et des humains reste à approfondir. Nous présentons une étude sur ce sujet dans les travaux du Chapitre V.

I.5 Organisation du Document

Ce manuscrit est organisé en six chapitres principaux présentant les travaux réalisés au cours de ma thèse et qui ont fait l'objet de plusieurs publications.

Chapitre 1 : Introduction

Ce chapitre expose le contexte, les motivations et l'intérêt des travaux abordés dans cette thèse. Il présente également l'organisation du contenu du document.

Chapitre 2 : État de l'Art

Ce chapitre introduit les principales définitions utiles à la compréhension du document, et décrit les travaux de la littérature en lien avec ceux détaillés dans ce manuscrit. Concrètement, il présente (i) la littérature du Dessin de Graphe ainsi que les techniques d'apprentissage automatique sur les graphes. Puis, (ii) les principaux travaux sur la Suppression de Chevauchements dans les visualisations, avec une distinction entre les espaces *données*, *géométriques*, et *visuels*. Enfin, (iii) il décrit l'état de l'art de la Perception sur la tâche de Recherche Visuelle d'un intrus, et l'évaluation automatique de visualisations.

Chapitre 3 : Dessin de Graphe avec des Réseaux de Neurones Profonds

Ce premier chapitre de contribution présente nos travaux sur la conception d'algorithme de dessin de graphes nœud-lien avec des réseaux de neurones profonds. Un algorithme de dessin nœud-lien étant analogue à un plongement d'un espace hautement dimensionnel à un espace 2D, nous proposons d'utiliser une méthode d'apprentissage basée sur les *Réseaux de Graphes Convolutifs* pour projeter la structure discrète dans un plan ensuite interprété comme le dessin. L'avantage de notre méthode est d'exploiter les capacités des réseaux de neurones profonds pour apprendre efficacement à résoudre la tâche *de manière non-supervisée*. Avec cette approche originale qui utilise des réseaux de neurones profonds, nos travaux se concentrent également sur l'évaluation de l'apprentissage de nos modèles afin de connaître leur limites. Cette étude est nécessaire dans tout domaine utilisant les réseaux de neurones profonds afin d'affiner la conception de ces modèles pour en tirer le meilleur parti.

Ce travail a fait l'objet de deux publications, en conférence et en journal :

[25] **L. Giovannangeli**, F. Lalanne, D. Auber, R. Giot, and R. Bourqui. « Deep Neural Network for DrawiNg Networks, $(DNN)^2$ ». In *Graph Drawing and Network Visualization*, 2021

[63] **L. Giovannangeli**, F. Lalanne, D. Auber, R. Giot, and R. Bourqui. « Toward Efficient Deep Learning for Graph Drawing (DL4GD) ». *IEEE Transactions on Visualization and Computer Graphics*, 2022

Le code source de *(DNN)*^{2} est disponible en ligne [64].

Chapitre 4 : Suppression de Chevauchements par Descente de Gradient Stochastique

Après avoir proposé une nouvelle méthode de génération de visualisations dans le Chapitre III, ce nouveau chapitre présente une méthode d'ajustement. En effet, avec l'augmentation de la complexité des données à représenter, la densité des plongements produits est telle que les éléments graphiques qui la constituent peuvent se chevaucher. Les travaux présentés dans ce chapitre abordent la Suppression de Chevauchements dans des visualisations 2D. Nos contributions sont séparées en deux parties pour deux algorithmes : FORBID travaillant dans l'espace géométrique, et GIST dans l'espace visuel. L'objectif de ces deux algorithmes est d'ajuster des plongements en entrée de telle sorte que toutes les données soient visibles dans le plongement en sortie. Ces algorithmes combinent une recherche dichotomique de facteur d'échelle avec une modélisation de mouvement des données inspirée du *Stress* et optimisé par descente de gradient stochastique. La combinaison de ces deux composants permet aux algorithmes de proposer un compromis unique entre (i) la préservation du plongement initial, et (ii) la compacité du plongement ajusté.

Ces travaux ont fait l'objet de deux publications en conférence :

[27] **L. Giovannangeli**, F. Lalanne, R. Giot, and R. Bourqui. « FORBID : Fast Overlap Removal By stochastic gradIent Descent for Graph Drawing ». In *International Symposium on Graph Drawing and Network Visualization*, 2022, **Best Paper Award**

[65] **L. Giovannangeli**, F. Lalanne, D. Auber, R. Giot, and R. Bourqui. « Guaranteed Visibility in Scatterplots with Tolerance ». In *VIS 2023, Special Issue in IEEE Transactions on Visualization and Computer Graphics*, 2023

Le code source de FORBID est ouvert [66], et une démonstration interactive de GIST est disponible en ligne [67].

Chapitre 5 : Évaluation Automatique de Visualisations par Apprentissage Profond

Enfin, après avoir proposé des méthodes de génération dans les précédents chapitres, ce cinquième chapitre se concentre sur l'évaluation de visualisations. Il présente (i) une méthodologie d'évaluation automatique de visualisations en utilisant des réseaux de neurones profonds. L'idée de cette méthodologie est d'entraîner un modèle à résoudre la même tâche que celle que devra résoudre l'utilisateur final. En mesurant ses performances, il est alors possible de quantifier l'efficacité d'une visualisation pour résoudre une tâche, et donc de comparer différentes visualisations. Cette évaluation automatique nous permet d'affiner (ii) une évaluation utilisateur étudiant l'impact de l'hétérogénéité d'une visualisation sur la recherche d'intrus. Concrètement, l'expérience étudie le nombre maximum de *couleurs* et *formes* utilisable dans une visualisation avant que celle-ci ne devienne trop complexe. Elle évalue également l'impact d'autres paramètres que nous ne détaillerons pas ici. Enfin, (iii) le chapitre présente une analyse de corrélations entre les humains et les réseaux de neurones profonds sur la tâche de recherche d'intrus considérée dans les deux évaluations précédentes : (i) automatique et (ii) utilisateur.

Ce chapitre synthétise les contributions présentées dans les trois publications suivantes :

[68] **L. Giovannangeli**, R. Bourqui, R. Giot, and D. Auber, "Toward automatic comparison of visualization techniques : Application to graph visualization," *Visual Informatics*, 2020

[30] **L. Giovannangeli**, R. Bourqui, R. Giot, and D. Auber, "Color and Shape efficiency for outlier detection from automated to user evaluation," *Visual Informatics*, 2022

[69] **L. Giovannangeli**, R. Giot, D. Auber, J. Benois-Pineau, and R. Bourqui, “Analysis of Deep Neural Networks Correlations with Human Subjects on a Perception Task,” in *25th International Conference Information Visualisation (IV)*, 2021

Chapitre 6 : Synthèse, Conclusion et Perspectives

Ce dernier chapitre récapitule les travaux et synthétise les contributions réalisées au cours de cette thèse. Il présente également leurs perspectives et possibles améliorations.

Chapitre II

État de l'Art

Sommaire

Ce chapitre présente les travaux de l'état de l'art en lien avec nos propres contributions détaillées dans les chapitres suivants. Dans la Section II.1, nous commençons par définir les termes et notions fondamentales pour les travaux abordés dans ce document. Les sections suivantes sont dédiées aux recherches dans la littérature en lien avec chacun des chapitres de ce document. Plus spécifiquement, la Section II.2 présente les travaux autour du Dessin de Graphe sur lesquels se basent nos contributions détaillées dans le Chapitre III. De la même manière, la Section II.3 présente les travaux de la littérature autour de la tâche de Suppression de Chevauchements en lien avec le Chapitre IV. Enfin, la Section II.4 décrit les premiers travaux sur l'Évaluation Automatique de Visualisations dans le contexte de la tâche de Recherche Visuelle employé dans le Chapitre V, en relation avec le domaine de la Perception.

II.1 Définitions et Prérequis

II.1.1 Graphes

Ce manuscrit expose des travaux entrepris au cours de ma thèse, traitant et utilisant des *graphes* (Chapitres III et IV). Dans la présente section, nous définissons les notions qui seront utilisées dans les descriptions de nos travaux. Des définitions complètes et détaillées de graphes sont

disponibles dans le livre de West [70].

Un *graphe*, noté $G = (V, E)$, est une structure (ou modèle) mathématique formelle d'objets définissant des liens E entre des entités V appelées *nœuds*. Un graphe peut être *orienté* ou non. Dans un graphe non orienté, les liens sont nommés *arêtes*. Dans un graphe orienté, les liens sont nommés *arcs* et sont unilatéraux (*i.e.*, $\{v_i, v_j\} \neq \{v_j, v_i\}$ où $v_i, v_j \in V$ et $i \neq j$). Un graphe peut être *pondéré* (ou valué) sur ses arêtes ou ses nœuds. Dans ce cas, il est défini comme $G = (V, E, f)$ où f est une fonction associant une pondération à chaque nœud et/ou arête. Des nœuds sont *voisins* ou *adjacents* s'ils sont directement reliés par une arête. Ainsi est nommée *matrice d'adjacence* la matrice de taille $|V| \times |V|$ où chaque cellule contient l'information d'adjacence (valuée ou binaire, pour un graphe pondéré ou non) entre le nœud source en ligne et celui destination en colonne. Le nombre de voisins d'un nœud est appelé son *degré*. Dans un graphe orienté, chaque nœud possède un degré *entrant* et *sortant*, mesurant son nombre d'arcs entrants et sortants. La matrice des degrés d'un graphe, de taille $|V| \times |V|$, contient sur sa diagonale le degré de chaque nœud, et des 0 partout ailleurs. Dans ce document, nous ne traiterons pas de graphe orienté. La suite des définitions omet donc volontairement la distinction entre graphes orientés et non-orientés.

La *topologie* du graphe désigne communément l'ensemble des structures formées par les entités et liens qui le constituent. Il est important de noter que cette définition diverge de celle employée par Gross et Tucket [71], et de la littérature qui découle de leurs travaux. Un *chemin* est une séquence d'arêtes reliant des nœuds dans un graphe. Un *cycle* est un chemin dont le nœud de départ et d'arrivée sont identiques. Par extension, le *plus court chemin* entre deux nœuds v_i et v_j est la séquence d'arêtes la plus courte reliant v_i et v_j. La *distance topologique* entre deux nœuds est le nombre d'arêtes dans le plus court chemin qui les sépare. Le *diamètre* d'un graphe est le plus long de ses plus courts chemins (*i.e.*, la distance maximale parmi toutes les distances topologiques du graphe). Enfin, la *matrice de distances* de taille $|V| \times |V|$ est la structure dans laquelle chaque cellule contient la distance entre le nœud en ligne et celui en colonne.

Un graphe est dit *connexe* s'il existe un chemin entre toute paire de nœuds. Deux graphes sont *isomorphes* s'ils ont la même topologie, c'est-à-dire qu'il existe une bijection $f : V(G_1) \rightarrow V(G_2)$ entre les nœuds de deux graphes G_1 et G_2, qui préserve les arêtes. Enfin, certains graphes possèdent des propriétés communes formant ce que nous appelons des *familles*. Dans ce document, nous utilisons notamment les familles :

- *Complet* : chacun des nœuds du graphe est relié à tous les autres. On note K_n un graphe complet à n nœuds.
- *Biparti* : les nœuds du graphe peuvent être séparés en deux sous-ensembles tels que deux nœuds adjacents soient toujours de sous-ensembles différents.
- *Biparti complet* : les nœuds du graphes peuvent être séparés en deux sous-ensembles, et chaque nœud d'un sous-ensemble est relié à tous les nœuds de l'autre sous-ensemble. On note $K_{m,n}$ un graphe biparti complet où chaque sous-ensemble est respectivement constitué de m et n nœuds.
- *Planaire* : peut être dessiné sur une surface de genre 0 (*e.g.*, plan ou sphère) sans croisement d'arêtes. Formellement, tout graphe qui ne compte ni K_5 ni $K_{3,3}$ parmi ses mineurs est planaire. Un *mineur* d'un graphe G est un graphe obtenu en contractant les arêtes de G.
- *Arbre* : ne contient aucun cycle (*i.e.*, acyclique) et est connexe.

Nous nommons *dessin* d'un graphe le plongement de ses composants dans un espace en basses dimensions (*e.g.*, 2D ou 3D) dans le but d'être visualisé. Dans notre terminologie, les

concepts *dessin*, *plongement*, *représentation* et *visualisation* d'un graphe désignent le même processus, sauf indication contraire. Il existe différentes techniques de visualisation de graphes (*e.g.*, liste d'adjacence, matrice d'adjacence, nœud-lien), et celles-ci seront présentées dans la Section II.2.1. Dans la plupart des cas, dessiner un graphe revient à calculer une position de ses nœuds (*i.e.*, un nuage de points). Les arêtes sont ensuite représentées dans le dessin avec un style dépendant de la technique utilisée. Néanmoins, la représentation des arêtes peut également être optimisée, comme c'est notamment le cas dans la thématique de Regroupement d'Arêtes [72–74] (*Edge Bundling* en anglais).

La *Triangulation de Delaunay* [75] d'un nuage de points P, notée $DT(P)$, est le graphe connexe dont les nœuds sont les points de P, et les arêtes sont définies de telle sorte qu'aucun nœud ne soit à l'intérieur d'un cercle circonscrit d'une des faces formées par les arêtes de $DT(P)$.

II.1.2 Apprentissage Automatique

Cette section présente le fonctionnement et définit les termes utilisés dans le contexte de l'Apprentissage Automatique et des Réseaux de Neurones Profonds (DNN pour « Deep Neural Networks »). Nous présenterons d'abord les concepts de plus haut niveau, puis préciserons progressivement leur fonctionnement. Néanmoins, le niveau d'abstraction final de ces explications restera relativement élevé. Des descriptions complètes et approfondies des modèles d'Apprentissage Automatique sont disponibles dans le livre d'Aggarwal [7].

II.1.2.1 Principe de l'Apprentissage : Tâche, Jeu de Données et Entraînement

Dans le contexte d'Apprentissage Automatique, le cycle de vie d'un modèle est séparé en deux phases : l'apprentissage, et la mise en production. Lors de la phase d'apprentissage, le modèle est entraîné sur banc d'essai à résoudre une *tâche* à partir de données récoltées, nettoyées et préparées dans ce but. Par exemple, il peut être entraîné à *classer* des images [4], ou *estimer* des quantités [29, 68]. La phase d'apprentissage fait intervenir différents composants qui ne seront plus nécessaires une fois le modèle considéré *appris* et mis en production. Par la suite, nous définissons les termes et concepts de la phase d'apprentissage relatifs aux données et aux tâches à résoudre, et dont une vue d'ensemble est présentée dans la Figure II.1.

Nourrir le modèle. Les données d'apprentissage peuvent être de types divers et variés (*e.g.*, image, vidéo, graphe). Pour être traitées par un modèle d'apprentissage, ces données doivent être

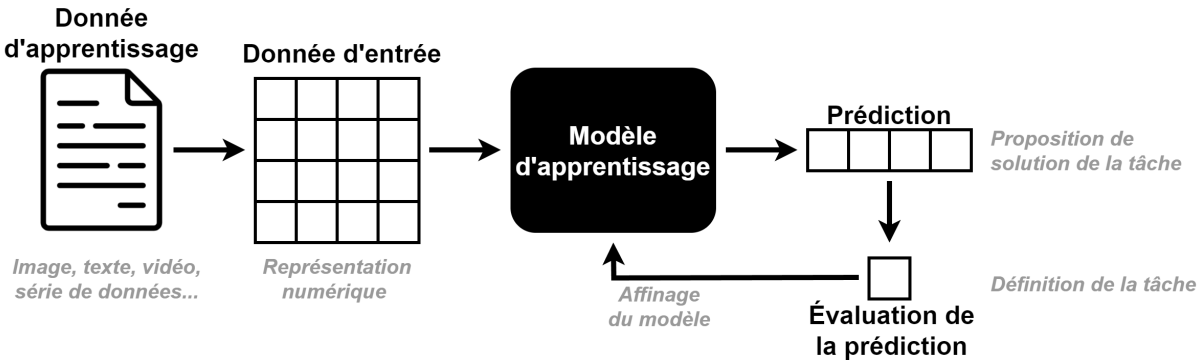


FIGURE II.1 : Schéma de la phase d'apprentissage d'un modèle sur une donnée. La donnée d'apprentissage est convertie en un format numérique que le modèle est capable de traiter. Celui-ci produit alors une prédiction interprétée comme une solution de la tâche. Enfin, la prédiction est évaluée et le modèle affiné en fonction de l'exactitude de sa prédiction.

mises en forme dans une structure numérique commune. Par exemple, si les données en entrée sont des images, leur représentation numérique est communément définie comme un tenseur de taille $W \times H \times 3$ où W et H sont respectivement la largeur et la hauteur des images (en nombre de pixels), et 3 est le nombre de valeurs décrivant chaque pixel (valeurs des canaux Rouge, Vert, Bleu), aussi appelé leurs *caractéristiques*. D'autres encodages de ces données existent, l'essentiel étant de définir (i) des *éléments* constituant la donnée (*e.g.*, les pixels d'une image, les nœuds d'un graphe), et (ii) les caractéristiques représentant ces éléments. Une donnée peut également être nommée un *échantillon*. Pour apprendre à résoudre une tâche efficacement, un modèle est généralement entraîné sur plusieurs exemples de données dont l'ensemble est appelé *jeu de données*. Une optimisation du modèle pour résoudre la tâche sur toutes les données du jeu d'entraînement est appelée *une époque* d'apprentissage. Le nombre d'époques pendant lequel un modèle est entraîné dépend du contexte de la tâche et de ses données.

Résoudre la tâche. La tâche peut être vue comme le type de question auquel le modèle d'apprentissage doit répondre et détermine son format de sortie (*i.e.*, *prédiction*). Les exemples les plus classiques de tâches sont la *régression* et la *classification*. Dans les problèmes de régression, la sortie du réseau est constituée d'une ou plusieurs valeurs numériques interprétées comme des estimations d'une valeur ou quantité. Cela signifie qu'une distance entre l'estimation prédite et une *vérité terrain* observée peut être calculée. Ce type de tâche est par exemple utilisé pour quantifier des éléments (*e.g.*, nombre de nœuds dans un graphe) ou prédire l'évolution d'une courbe (*e.g.*, évolution du cours de l'immobilier). Dans les problèmes de classification, l'objectif est d'associer la donnée en entrée à une ou plusieurs catégories parmi un ensemble prédéfini. La sortie du modèle d'apprentissage est donc un vecteur de probabilités de l'appartenance de la donnée en entrée à chacune des classes. Il est alors considéré que la classe prédite par le modèle est celle dont la probabilité est la plus élevée. Dans un cas de classification multi-label où il y a k classes à prédire, les classes prédites sont les k dont la probabilité est la plus élevée. L'exemple typique de cette tâche est la classification d'image, où le modèle doit prédire le concept prédéfini le plus proche de ce que représente l'image (*e.g.*, chien, chat, loup).

Dans la Figure II.1, la définition de la tâche est positionnée à l'évaluation de la prédiction car c'est elle qui permet de modéliser la notion *d'exactitude* de la prédiction du modèle. C'est ici qu'est encodée la problématique métier que le modèle d'apprentissage doit satisfaire.

Évaluer une prédiction. Que la tâche à résoudre soit une régression ou une classification, l'évaluation de la prédiction du modèle peut être *supervisée* ou non. Dans un apprentissage *supervisé*, la solution correcte de la tâche sur chaque donnée en entrée est déjà connue. Cette solution est alors appelée *label* ou *vérité terrain* (GT pour « Ground Truth »). Dans ce cas, la qualité de la prédiction est calculée avec une fonction de distance entre la prédiction et la GT. En apprentissage non supervisé, les données d'entrée ne sont pas associées à des GT. L'évaluation de la prédiction se fait alors en fonction d'autres critères, principalement basés sur la donnée en entrée elle-même. Par exemple, dans le travail présenté dans le Chapitre III, nos modèles sont entraînés à prédire un positionnement efficace des nœuds d'un graphe sur un plan 2D. Puisqu'il n'existe pas un positionnement idéal unique, cet entraînement ne peut pas être supervisé par une vérité terrain. Néanmoins, il est possible de quantifier la qualité du positionnement prédit à partir de la topologie du graphe en entrée.

Évaluer l'apprentissage d'un modèle. L'évaluation d'un modèle pendant l'apprentissage est appelée *validation* et se base sur deux critères. Le premier critère est l'efficacité du modèle à résoudre la tâche, mesuré à partir de l'évaluation de ses prédictions sur son jeu d'entraînement, comme vu précédemment. Cependant, ce seul critère ne suffit pas à évaluer les capacités d'un modèle. Celui-ci étant entraîné sur banc d'essai avec des données préparées voire générées dans ce but, il est nécessaire d'évaluer ses performances sur des données qu'il n'a jamais vues pendant

son entraînement afin de s'assurer qu'elles restent stables une fois mis en production dans un environnement moins contrôlé. Le second critère relève donc de l'évaluation des capacités du modèle à *généraliser* son apprentissage, et dont la mesure est appelée la *généralisabilité* du modèle. Le *sur-apprentissage* est alors défini comme un état du modèle où il obtient de bonnes performances sur banc d'essai mais une mauvaise *généralisabilité*.

Différentes stratégies de validation existent pour vérifier la *généralisabilité* d'un modèle [76]. Dans les chapitres suivants, nous mettrons en œuvre les stratégies *Hold-Out* et *Leave One Out*. *Hold-Out* est la stratégie la plus commune dans l'apprentissage profond. Elle consiste à séparer le jeu de données initial en plusieurs parties distinctes et ayant chacune un rôle. La première partie est appelée *jeu d'apprentissage* ou d'*entraînement*. Ces données sont directement utilisées pour apprendre au modèle à résoudre la tâche. La seconde partie est nommée *jeu de validation* et est utilisée au cours de l'apprentissage pour s'assurer que le modèle garde toujours une certaine *généralisabilité* au cours de l'entraînement, et prévenir le sur-apprentissage. Enfin, la troisième partie est appelée *jeu de test* et est optionnelle. Elle est dédiée à la comparaison de performances de différents modèles sur une même tâche. Ces modèles étant potentiellement issus d'entraînements ou d'environnements différents, il est préférable de les évaluer sur un jeu de test isolé (*i.e.*, composé de données qu'aucun d'entre eux n'a jamais vu pendant son entraînement). Cette stratégie nécessite d'avoir un volume de données conséquent (*i.e.*, plusieurs milliers d'échantillons) car la distribution des données dans chaque sous-ensemble doit suivre celle du jeu de données complet. La séparation est généralement réalisée aléatoirement selon les proportions : 70 à 80% pour le jeu d'entraînement, 10 à 20% pour le jeu de validation, et jusqu'à 10% pour le jeu de test. Ces proportions sont données à titre indicatif et doivent être adaptées au problème considéré et à ses données. À l'inverse, la stratégie de validation *Leave One Out* est dédiée aux cas où le volume de données à disposition est faible. Séparer les données en deux ou trois ensembles est alors impossible car les distributions des différentes sous-parties risqueraient fortement de ne pas être identiques. L'idée de *Leave One Out* est d'entraîner plusieurs fois le modèle avec toutes les données sauf une, dite *exclue*. Cette donnée exclue est différente à chaque entraînement et la généralisabilité du modèle est évaluée dessus. Cette stratégie est relativement coûteuse car elle nécessite d'entraîner le modèle autant de fois qu'il y a d'échantillons dans le jeu de données. Néanmoins, elle permet de s'assurer d'avoir suffisamment de données pour que le modèle réussisse à apprendre tout en maintenant sa capacité à généraliser son apprentissage. Pour atténuer le coût de cette validation, il est possible d'exclure un lot de plusieurs données à chaque apprentissage. Cette stratégie est nommée *Leave K Out* où K est le nombre de données réservées à la validation à chaque étape.

II.1.2.2 Optimisation

Un modèle d'apprentissage est composé de *paramètres*. Ce terme générique désigne ses composants internes qui sont amenés à varier au cours de l'apprentissage. Ces paramètres déterminent comment la donnée en entrée est modifiée pour obtenir une prédiction. Un modèle est dit *appris* une fois que les valeurs de ses paramètres sont suffisamment affinées pour que ses prédictions atteignent un niveau de performance souhaité. Ceux-ci sont à distinguer des *hyper-paramètres* qui ont vocation à contrôler l'apprentissage du modèle, et qui ne seront plus utilisés une fois le modèle mis en production.

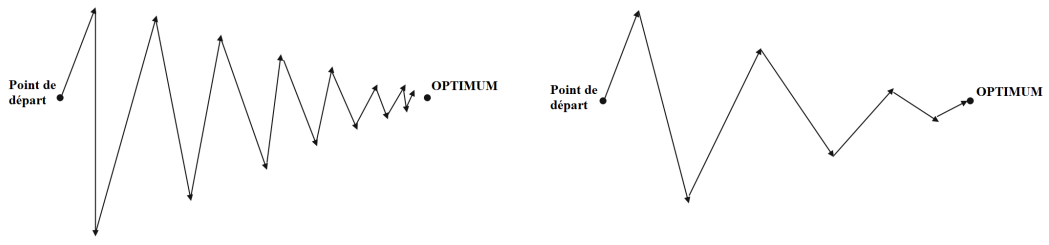
L'*apprentissage* d'un modèle fait référence à l'optimisation de ses paramètres : la recherche de leur valeur optimale pour résoudre la tâche efficacement, quelle que soit la donnée en entrée. L'efficacité de la résolution de la tâche doit être exprimée avec une fonction capable de quantifier l'exactitude de la prédiction du modèle. Cette fonction, dite *de coût*, permet de quantifier à quel

point le modèle s'éloigne de « la courbe qui doit relier toutes les données » dans un problème de régression, ou « la courbe permettant de séparer toutes les données » pour de la classification. Contrairement aux métriques d'évaluation métier, cette fonction de coût doit être dérivable, au moins par parties, afin de permettre la mise à jour des paramètres du modèle en fonction de l'efficacité de sa prédiction pour résoudre la tâche sur chaque échantillon du jeu de données. Il existe différents algorithmes d'optimisation (dits *optimiseurs*) pour entraîner des modèles d'apprentissage. Les exemples les plus connus sont Adam [77], AdaGrad [78] et RMSProp [79]. Ces méthodes sont des optimiseurs basés sur la *descente de gradient stochastique* avec un *pas d'apprentissage dégressif par paramètre*, et avec *momentum*.

Descente de Gradient Stochastique. Dite SGD pour « Stochastic Gradient Descent », c'est une méthode itérative d'optimisation de fonction étendant la descente de gradient. Étant donné une donnée en entrée x, un modèle d'apprentissage M et une fonction de coût L évaluant la prédiction $M(x)$, l'efficacité du modèle peut être décrite par $loss = L(M(x))$. Pendant l'optimisation, l'idée est de mettre à jour M pour optimiser la fonction de coût L, et donc le score $loss$ obtenu. Pour cela, l'approche par SGD modifie les paramètres du modèle dans la direction opposée du gradient de $L(M(x))$, pour que la prochaine évaluation du modèle se rapproche systématiquement de l'optimum. Le calcul de la dérivée de la fonction de coût pour chaque donnée du jeu d'entraînement étant trop coûteux, la SGD propose de regrouper les données en *lots* (*batch* en anglais) et de mettre à jour les paramètres une seule fois pour chaque lot. Le gradient est ainsi calculé sur $L(M(X))$ où X est un ensemble de données d'apprentissage, et où $|X|$ est un hyper-paramètre appelé taille du lot (*batch size* en anglais).

Pas d'apprentissage dégressif par paramètre. Pour garantir la convergence de l'optimisation par SGD, la mise à jour des paramètres n'est pas uniquement fonction du gradient de la fonction de coût, mais également d'un *pas d'apprentissage* dégressif. Cet hyper-paramètre est un facteur multiplicateur du gradient lors de la mise à jour des paramètres. S'il est constant mais trop faible, l'optimisation prend trop de temps à se rapprocher d'une solution. S'il est trop grand, l'optimisation se rapproche rapidement de la solution mais oscille ensuite autour d'elle sans pouvoir s'en approcher davantage. Pour garantir la convergence, les optimiseurs utilisent un pas d'apprentissage dégressif. Ainsi, il commence à une valeur suffisamment élevée pour se rapprocher rapidement de la solution, puis diminue soit régulièrement, soit lorsque le modèle n'arrive plus à améliorer ses performances.

De plus, les optimiseurs mentionnés plus haut ont un pas d'apprentissage *par paramètre*. L'idée est que chaque paramètre du modèle ne va pas contribuer équitablement à la prise de décision (*i.e.*, prédiction). Cette approche monitore donc, pour chaque paramètre, si la dérivée partielle de la fonction de coût par rapport à lui change de signe régulièrement. L'objectif du



(a) Sans momentum (b) Avec momentum

FIGURE II.2 : Illustration de l'effet du momentum sur un problème d'optimisation. L'inertie préservée par le momentum atténue l'oscillation de l'optimisation autour de l'optimum. Cette illustration est inspirée de Aggarwal [7].

pas d'apprentissage par paramètre est alors de *lisser* l'optimisation en atténuant certains effets d'oscillation.

Momentum. Ces optimiseurs se basent également sur le concept de *momentum*. Cette approche part du principe que pour n'importe quelle optimisation par SGD, il est admis qu'elle va *osciller* à un moment donné. Ainsi, le *momentum* a pour objectif d'atténuer cet effet pour faciliter la convergence, comme illustré dans la Figure II.2. Pour cela, il garde en mémoire les gradients des itérations précédentes. La mise à jour des paramètres au cours d'une itération dépend alors du gradient de l'itération courante ainsi que des précédentes, de manière à préserver une forme *d'inertie* dans l'optimisation. Cette inertie peut parfois créer de légers dépassements de la solution, mais son utilisation reste bénéfique dans la majorité des cas.

II.1.2.3 Architecture et Réseaux de Neurones Profonds

Jusqu'ici, nous avons défini un modèle comme un ensemble de paramètres dont les valeurs sont modifiées durant la phase d'apprentissage, pour satisfaire une fonction de coût définissant la tâche à apprendre. Cette définition convient à de nombreux algorithmes tels que les Forêts d'Arbres Décisionnels [3], K-Means [80] et les Machines à Vecteurs de Support [81].

Architecture. Tel que nous le définissons, les trois algorithmes précédents peuvent être considérés comme des *architectures* de modèle d'apprentissage. Une architecture est un patron de conception définissant le type et l'agencement des paramètres du modèle. Par exemple, l'architecture en Forêts d'Arbres Décisionnels définit l'organisation des paramètres en arbres de décisions, eux-mêmes constitués de nœuds décrivant des tests sur les variables de la donnée d'apprentissage. Le nombre d'arbres dans la forêt et le nombre de nœuds dans les arbres sont des hyper-paramètres à définir lors de l'instanciation de l'architecture en un modèle.

Réseau de Neurones Profond – Les DNN sont des modèles d'apprentissage revenus en force dans le début des années 2010 [4, 82] et sont désormais à la base de la plupart des algorithmes d'intelligence artificielle (*e.g.*, Traitement du Langage [6], Traitement d'Images [4]). Ces réseaux constituent la thématique d'Apprentissage Profond, elle-même inclue dans l'Apprentissage Automatique.

Les modèles d'apprentissage profond sont constitués d'ensembles de paramètres regroupés en *couches* et/ou en *réseaux*. Les *couches* et les *réseaux* sont des concepts abstraits qui factorisent des sous-ensembles de paramètres pour faciliter leur description et leur manipulation. Ces concepts étant abstraits, il est possible de définir des couches à partir de réseaux, des réseaux de réseaux, *etc*. Pour simplifier cette terminologie, il est communément admis d'organiser un modèle d'apprentissage hiérarchiquement comme suit : un réseau est composé de couches, elles-mêmes constituées de paramètres. Les couches peuvent être : (i) d'entrée, (ii) cachée, et (iii) de sortie. Le format (*i.e.*, la forme du tenseur) des données que le réseau peut traiter est défini par sa couche d'entrée. Une donnée traitée par le réseau passe ensuite dans chaque couche cachée où elle va être transformée. Ces couches cachées implémentent un type de traitement de la donnée et définissent l'organisation des paramètres du modèle. Il existe de nombreux types de couches définissant des opération différentes. Parmi les plus connues, nous pouvons mentionner les couches de convolution [83] utilisées pour le traitement du signal, les couches d'auto-attention [6] notamment employées dans le traitement du langage naturel, et les couches denses (*i.e.*, totalement connectées) utilisées pour la régression en général. D'autres types de couches, utilisées dans nos contributions, seront détaillées dans les prochains chapitres. Enfin, la couche de sortie définit le format de sortie du réseau. Dans l'exemple donné sur la gauche de la Figure II.3, la couche de sortie du réseau ne contient qu'un neurone : le réseau ne prédit donc qu'une seule valeur. Cet exemple peut être représentatif d'un problème de régression vers une

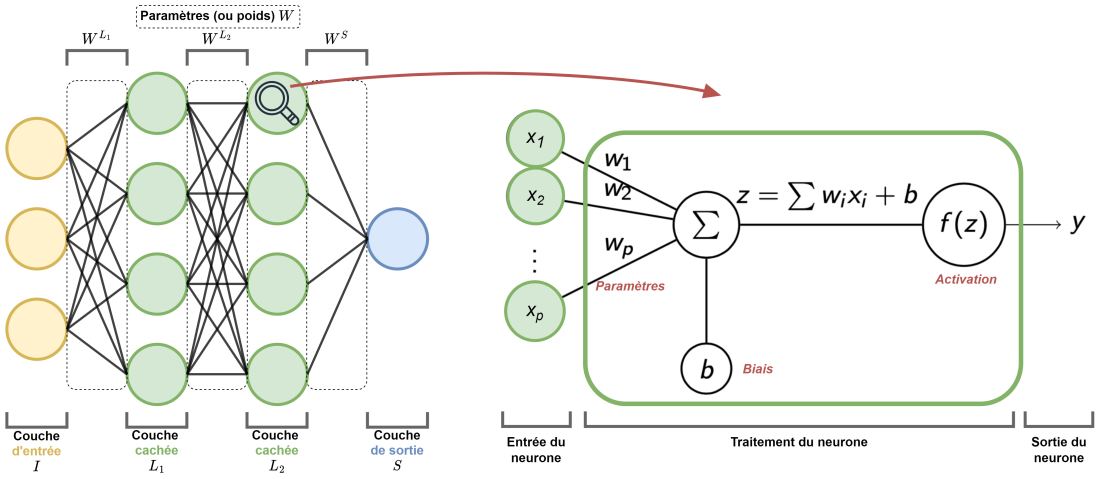


FIGURE II.3 : Schéma explicatif d'une architecture de réseau de neurones (*i.e.*, chaque cercle coloré est un neurone). Sur la gauche, le réseau est composé d'une couche d'entrée, de deux couches cachées et d'une couche de sortie. Les neurones sont reliées par des *paramètres* (ou poids) dont les valeurs sont apprises pendant l'entraînement. Sur la droite, la description d'un neurone formel montre le calcul de sa sortie en fonction des paramètres en entrée, du biais et de la fonction d'activation.

valeur numérique, ou d'une classification binaire. La *profondeur* d'un DNN est définie par son nombre de couches cachées : plus il y a de couches, plus le réseau est profond. Le nombre, le type et l'agencement des couches forment l'*architecture* du DNN. *The Neural Network Zoo* [84] recense une large majorité des architectures de DNN utilisées dans la littérature.

Si l'architecture du réseau de neurones profond définit la structure du modèle, ce sont bien ses paramètres, aussi appelés *poids*, qui permettent son apprentissage. Chaque couche reçoit un tenseur de données en entrée et le transforme en utilisant ses poids. De cette manière, la sortie du réseau est le résultat d'une série d'applications de poids sur la donnée en entrée. Durant l'apprentissage, il est alors possible de calculer l'impact des poids sur la prédiction du modèle en fonction des caractéristiques de la donnée en entrée. Cela permet de mettre à jour ces poids pour que les prochaines prédictions améliorent la résolution de la tâche (voir Section II.1.2.2). Ce processus de modification des poids partant de prédiction du modèle et remontant à travers ses paramètres jusqu'à la donnée en entrée est appelé *rétropropagation* (*backpropagation* en anglais).

Enfin, il nous reste à expliquer le fonctionnement d'un neurone dans un DNN. L'idée de ce type de réseau est de générer une prédiction à partir de transformations de la donnée en entrée via ses paramètres appris. Concrètement, un neurone définit une transformation des caractéristiques de la donnée en entrée vers un état intermédiaire : c'est *l'extraction de caractéristiques* (*feature extraction* en anglais). Tel qu'illustré dans la partie droite de la Figure II.3, un neurone formel peut ainsi être défini comme une fonction $z = wx + b$ où x est la donnée en entrée, w un paramètre appris et b est un biais appris, décrit dans la suite. L'objectif de l'entraînement est alors de trouver la bonne valeur w qui permet d'obtenir une solution satisfaisante à la tâche, quelle que soit la donnée en entrée. Pour s'adapter à l'hétérogénéité potentielle des données, chaque neurone définit en réalité un ensemble de paramètres qui permet au modèle de trouver une stratégie plus fine pour résoudre la tâche. Ainsi, la solution à la tâche peut être définie comme un ensemble de transformations de la donnée en entrée $z = \sum_{i<|W|} w_i x_i + b$ où W est l'ensemble des paramètres d'un neurone. La variable b, nommée *biais*, est optionnelle et également apprise. Contrairement

aux paramètres W qui peuvent être multiples, il y a au plus un biais par neurone. En outre, un *poids* est associé à chaque entrée d'un neurone, et un *biais* peut être associé à sa sortie. Enfin, il est commun d'utiliser une *fonction d'activation* en sortie de neurone pour lisser et simplifier l'apprentissage du réseau. Un neurone formel est donc défini par $y = f(\sum_{i<|W|} w_i x_i + b)$ où f est une fonction d'activation et y la sortie du neurone.

Si le fonctionnement des DNN est connu et intuitif, leur profondeur et la multiplication des poids qui les composent rendent l'interprétation de leurs résultats difficiles, voire impossibles. Aujourd'hui, ils sont considérés comme des *boîtes noires*, ce qui a conduit à de nouvelles recherches sur l'*intelligence artificielle explicable* (XAI pour « eXplainable AI »). L'objectif de cette thématique est de proposer des solutions pour comprendre et interpréter la prise de décision de ces modèles [13, 14], afin de tendre vers la conception d'*IA de confiance* [18].

II.2 Dessin de Graphe

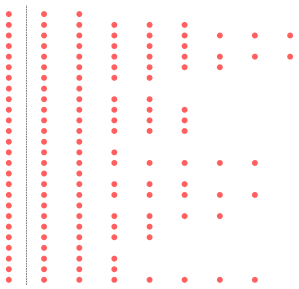
Les graphes sont des objets largement utilisés dans de nombreux domaines [85–89] car l'organisation des données en schémas relationnels est de plus en plus courante, et des algorithmes efficaces permettent de résoudre une multitude de tâches (*e.g.*, recherche d'élément, de structure ou de propriétés). Cependant, la complexité croissante des données collectées et des graphes qui les modélisent complique le travail des experts. L'objectif du Dessin de Graphe est alors de proposer des visualisations efficaces pour améliorer l'appréhension de ces données et faciliter leur manipulation.

Cette section présente les travaux de la littérature de la communauté de Dessin de Graphe en lien avec les travaux décrits dans le Chapitre III. Tout d'abord, nous exposons les principes de la tâche de dessin de graphe, puis les approches et algorithmes principaux de la littérature. Dans un second temps, nous présentons des structures de données qui ont été développées pour permettre et améliorer l'apprentissage sur les graphes. Enfin, nous présentons les travaux de la littérature sur le dessin de graphe via des réseaux de neurones profonds.

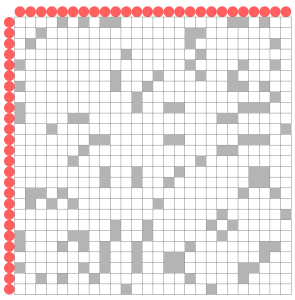
II.2.1 Les Approches au Dessin de Graphe

Comme indiqué dans la Section II.1.1, les graphes sont des structures de données définissant des *liens* (*i.e.*, arêtes) entre des entités (*i.e.*, nœuds). Si le graphe en tant que tel est utile pour traiter efficacement la donnée qu'il encode (*e.g.*, base de données graphes [91]), les experts ont parfois besoin de le visualiser pour comprendre l'agencement du réseau formé par les entités [92, 93] afin de mieux le manipuler.

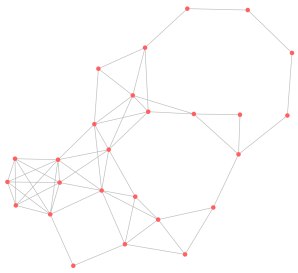
Il existe différentes manières de visualiser un graphe. Sans utiliser d'algorithme, la première approche est l'affichage de la donnée brute. Par exemple, la topologie du graphe peut être visualisée sous forme de liste d'adjacence : chaque ligne représente un nœud, suivi de tous ceux avec qui il est connecté (voir Figure II.4a) [94, 95]. Cette méthode est cependant peu efficace sans utiliser d'encodage particulier permettant de distinguer les nœuds, et demeure relativement peu employée. Une autre visualisation de la topologie est l'affichage direct de la matrice d'adjacence (AM pour « Adjacency Matrix », voir Figure II.4b) [96]. Ici, l'information est plus facile à lire car il est possible d'identifier chaque nœud et ses arêtes incidentes. Pour améliorer la visualisation par matrice d'adjacence, il est également possible d'ordonner les nœuds. Par exemple, dans la Figure II.4d, les nœuds sont triés par leur communauté d'appartenance, calculée avec l'algorithme Louvain [90]. Encore aujourd'hui, la visualisation de graphe par matrice d'adjacence est parmi les plus utilisées, avec celle qui nous intéressera dans la suite de cette section : la représentation



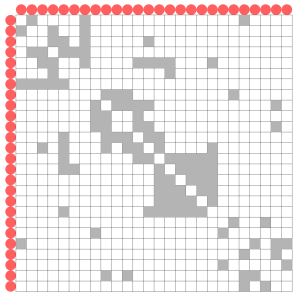
(a) Liste d'adjacence



(b) Matrice d'adjacence



(c) Dessin nœud-lien [59]



(d) Matrice d'adjacence ordonnée [90]

FIGURE II.4 : Exemples de dessins d'un même graphe avec différentes techniques.

nœud-lien (NL, voir Figure II.4c). Dans ce type de visualisation, les nœuds sont dessinés avec des formes géométriques (ici, des cercles), reliés par des traits représentant les arêtes. Produire une visualisation NL efficace revient alors à positionner correctement les nœuds et les arêtes sur le plan 2D. Cette visualisation est souvent opposée à AM car les deux sont couramment utilisées pour représenter des graphes, et la question de leur efficacité relative n'est pas encore tranchée [54, 55, 68]. Le consensus admis aujourd'hui est que les visualisations NL sont systématiquement efficaces sur des *petits* graphes (*i.e.*, faible nombre de nœuds) et particulièrement pour résoudre des tâches de recherche de chemins. Les représentations AM sont moins affectées que NL lorsque le nombre de nœuds devient significativement grand et sont donc mieux adaptées à ces cas. Elles sont aussi plus efficaces pour les tâches de visualisation de bas niveau (*e.g.*, comptage) ou d'identification de voisinage direct.

Dans ce document, nous nous intéressons aux représentations NL [97]. Les algorithmes de dessin dont nous parlons par la suite ont pour objectif de trouver un plongement 2D efficace des nœuds du graphe, et produire des dessins *plaisants*. Pour évaluer leur degré d'efficacité, des métriques de qualité esthétiques ont été définies [56]. Ces métriques sont les pierres angulaires du domaine du Dessin de Graphe pour deux raisons. Premièrement, elles permettent l'évaluation quantitative, automatique et reproductible des différents algorithmes de dessin de graphe. Il devient donc nécessaire de systématiquement évaluer et comparer chaque nouvelle méthode à ses prédécesseures. Le deuxième avantage significatif de ces métriques est que leurs corrélations avec la perception humaine ont déjà été étudiées [53, 57, 58]. Il est donc possible de savoir dans quelle mesure un dessin est réellement efficace pour la perception humaine à partir des scores des métriques esthétiques. À titre d'exemple, voici quelques métriques de dessin de graphe de l'état de l'art utilisées dans ces études :

- *Nombre de croisements d'arêtes* : compte le nombre de fois que des arêtes se croisent dans le dessin. L'idéal est de minimiser le nombre de croisements.

- *Résolution angulaire des nœuds* : angle minimal formé par deux arêtes incidentes à un

nœud dans le dessin. L'idéal est de maximiser cette métrique, de telle sorte que les arêtes incidentes à un nœud soient uniformément réparties autour du nœud.

- *Orthogonalité* : quantification de l'orthogonalité du positionnement des nœuds. Cela mesure la régularité des espacements entre les nœuds, favorisant le positionnement sur une grille régulière.

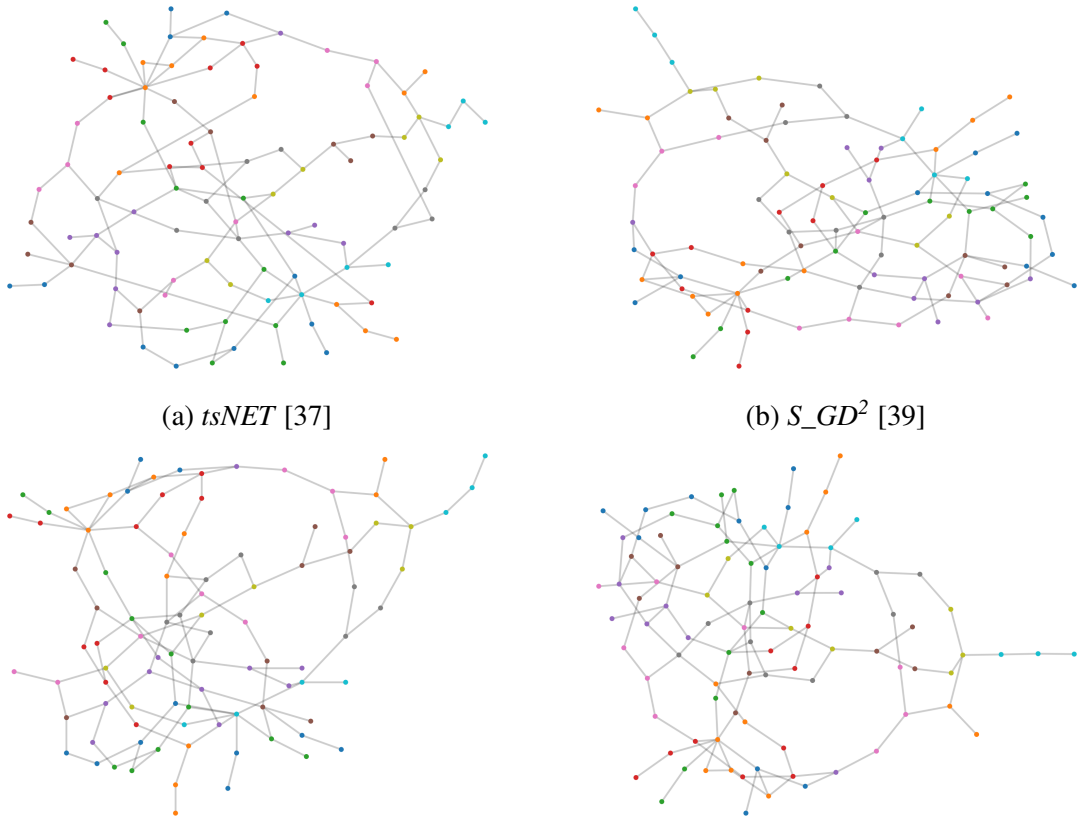
- *Symétrie* : quantification de la symétrie des structures locales et globales formées par le positionnement des nœuds.

L'étude de Di Bartolomea *et al.* [98] recense la plupart de ces métriques ainsi que d'autres informations (*e.g.*, jeux de données, types de tâches) sur les protocoles d'évaluation des méthodes de dessin de graphe. Si l'aspect esthétique de la visualisation NL est important, la préoccupation majeure des algorithmes de dessin est de produire un plongement dans lequel le positionnement des nœuds en 2D permet à l'utilisateur d'appréhender la topologie du graphe dessiné.

Historiquement, nous distinguons trois grandes familles de Dessin de Graphe pour des représentations NL dans la littérature. La première est celle des algorithmes *dédiés* à des familles de graphe. Par exemple, de nombreux algorithmes permettent de dessiner les graphes *planaires* sans croisement [99, 100]. Les arbres peuvent, eux aussi, être dessinés avec des algorithmes spécifiques tirant parti de leur topologie, et mettant en évidence leur structure arborescente [101–103]. TopoLayout [104] permet de généraliser ce type d'approche à des graphes quelconques. En décomposant hiérarchiquement le graphe en entrée, la méthode est capable de détecter des topologies particulières dans les sous-graphes décomposés, et d'utiliser des algorithmes de dessins dédiés à ces topologies.

La seconde famille de Dessin de Graphe est constituée des algorithmes par *modèle de force*. Popularisée par Kamada et Kawai [59], cette approche consiste à modéliser le graphe comme un système où tous les nœuds sont reliés par des *ressorts* dont la raideur est définie par la distance topologique (*i.e.*, nombre d'arêtes dans le plus court chemin) entre les nœuds. L'objectif de l'algorithme est ensuite de simuler une convergence de ce système minimisant sa tension globale. L'avantage de cette approche est de pouvoir s'adapter à n'importe quel graphe. En effet, peut importe sa topologie, il est simplement nécessaire de connaître les plus courts chemins entre les paires de nœuds du graphe pour ensuite être en mesure de calculer les forces d'attraction et de répulsion entre eux. Le principal inconvénient des modèles de force est leur coût calculatoire quadratique, voire cubique sur le nombre de nœuds. En effet, puisque le système considère des forces entre toutes les paires de nœuds, les algorithmes classiques par modèle de force [59, 105–107] doivent traiter N^2 forces, où N est le nombre de nœuds dans le graphe. La complexité de ces modèles a motivé la recherche d'approches plus rapides mettant en œuvre des systèmes moins coûteux. Par exemple, Meidiana *et al.* [108, 109] ont proposé une méthode de dessin de graphe qui échantillonne les forces calculées afin d'obtenir une complexité linéaire, voire sous-linéaire. Hachul et Jünger [110] ont également proposé un algorithme multi-échelle pour dessiner des grands graphes par modèle de force en temps linéaire.

Enfin, la troisième famille est l'adaptation d'algorithmes de positionnement multidimensionnel (MDS pour « Multidimensional Scaling ») au Dessin de Graphe [60, 111–113]. L'idée de ces algorithmes est de modéliser chaque nœud comme un vecteur de caractéristiques en haute dimension. Les caractéristiques de chaque nœud sont ensuite réduites dans l'espace cible, par exemple en 2D. Ces méthodes de dessin de graphe par MDS se basent sur la matrice d'adjacence, de distance ou la décomposition en vecteurs propres de ces matrices pour définir les caractéristiques initiales des nœuds. Elles ont récemment été étendues avec les nouvelles méthodes d'apprentissage. En effet, les méthodes d'apprentissage se montrant très efficaces dans les problèmes



(a) *tsNET* [37] (b) S_GD^2 [39]

(c) FM^3 [110] (d) Kamada et Kawai [59]

FIGURE II.5 : Exemples de dessins d'un même graphe avec différents algorithmes.

de régression [29, 34, 68], il est possible de les utiliser pour produire des dessins de graphes à partir de vecteurs de caractéristiques représentant les nœuds. Par exemple, Ahmed *et al.* [40, 41] ont proposé d'utiliser des métriques esthétiques de dessin de graphe comme fonction de coût pour produire des plongements de nœuds par SGD. Kwon *et al.* [38] ont proposé une méthode d'apprentissage automatique dédiée aux grands graphes et capable de produire des dessins d'un graphe dans plusieurs styles tout en estimant les scores de métriques esthétiques associées aux dessins. Cependant, les deux principaux algorithmes de dessin de graphe par apprentissage automatique sur lesquels nous souhaitons insister ici sont *tsNET* [37] et S_GD^2 [39]. *tsNET* [37] adapte l'algorithme t-SNE [35] au contexte du dessin de graphe. Son objectif est de *préserver le voisinage* des nœuds dans le plongement 2D en optimisant la fonction objectif utilisée par t-SNE : la divergence de Kullback-Leibler [114]. S_GD^2 [39] est un autre algorithme dont l'objectif est de *préserver les distances* entre les nœuds dans le plongement 2D. Il reprend la modélisation du Stress par contrainte de Dwyer [115] et l'optimise par SGD. L'algorithme modélise ainsi les déplacements de nœuds comme des contraintes à satisfaire. Les paires de nœuds sont traitées individuellement et déplacées en fonction du gradient du Stress pour chaque paire. Par opposition à *tsNET* qui est une méthode de préservation de voisinage, S_GD^2 se concentre sur la préservation des distances. Ces deux algorithmes étant à la base de deux contributions présentées dans ce document, leur fonctionnement sera détaillé dans les Chapitres III et IV. Des exemples de dessins de graphes de certains algorithmes de la littérature sont présentés dans la Figure II.5

Plus récemment, la famille des algorithmes de Dessin de Graphe par MDS a encore été étendue avec l'arrivée des techniques d'apprentissage profond dont les modèles peuvent être entraînés à réaliser le plongement du graphe de l'espace topologique vers le dessin 2D.

II.2.2 Outils pour l'Apprentissage Profond sur les Graphes

L'engouement pour les modèles d'apprentissage automatique a motivé le développement d'outils capables d'améliorer leur entraînement. Cet intérêt croissant concerne également l'apprentissage profond pour le traitement de graphes dont nous détaillons certaines contributions de la littérature utiles à la lecture de ce document.

Comme nous l'avons vu précédemment, les méthodes de Dessin de Graphe par MDS plongent des vecteurs de caractéristiques représentant les nœuds d'un espace hautement dimensionnel vers un espace en 2D. Si la plupart des méthodes classiques utilisent la matrice de distance ou d'adjacence où chaque ligne représente les caractéristiques d'un nœud, des algorithmes ont été conçus pour produire des caractéristiques de nœuds personnalisables (*e.g.*, nombre de dimensions paramétrable) et représentatives des particularités de ces nœuds dans le graphe. DeepWalk [116] est un exemple d'algorithme générant ce type de vecteur de caractéristiques (appelé *espace latent*). Pour chaque nœud du graphe, DeepWalk réalise une courte marche aléatoire de taille fixe dans ses voisins. Les caractéristiques du nœud courant sont ensuite mises à jour en utilisant un algorithme de modélisation du langage [117]. Cette mise à jour s'effectue en tenant compte des voisins visités durant la marche aléatoire et optimise une fonction de coût conçue pour la tâche. node2vec [118] est une méthode proche de DeepWalk. La principale différence est que node2vec permet une meilleure paramétrisation des marches aléatoires, et peut ainsi adapter son calcul des caractéristiques à la topologie du graphe. Contrairement à DeepWalk [116] et node2vec [118] dont les caractéristiques calculées sont essentiellement représentatives du voisinage direct des nœuds, Line [119] est un autre algorithme dont l'objectif est de capturer deux types de proximité pour les encoder dans les caractéristiques des nœuds. Les auteurs soutiennent que, notamment dans les graphes de réseaux sociaux, une grande partie des relations entre les entités ou communautés n'est pas encodée par des arêtes du graphe [120, 121]. Ils dissocient alors la proximité de *premier ordre* (*i.e.*, les nœuds reliés par des arêtes) de celle de *second ordre*. Les nœuds ayant une forte proximité de second ordre ne sont pas directement reliés, mais leur emplacement relatif dans le graphe par rapport à tous les autres nœuds suggère qu'il existe une relation entre eux. Ce type de proximité est notamment présent dans des graphes de réseaux sociaux où des nœuds d'une même communauté peuvent ne pas être adjacents, mais leur proximité reste plus forte qu'avec les nœuds d'autres communautés. Line optimise donc le calcul des caractéristiques descriptives des nœuds pour chaque ordre de proximité avant de les combiner, produisant ainsi des caractéristiques représentatives des structures locales et globales du graphe.

Le second type d'outils nécessaires à l'apprentissage de modèles de réseaux de neurones profonds sur des graphes que nous présentons ici sont les couches neuronales définissant les opérations de traitement de l'information du graphe, qui considèrent à la fois ses nœuds et ses arêtes. Les premières structures permettant l'apprentissage profond sur les graphes ont émergé en 2008 avec *Graph Neural Netwok* de Scarselli *et al.* [122]. Récemment, ces structures ont regagné en popularité avec les Réseaux Convolutifs de Graphe (GCN pour « Graph Convolutional Networks ») proposés par Kipf et Welling [123]. Malgré l'appellation *convolution* et le parallèle qui peut effectivement être fait avec, la couche proposée par les auteurs est une transformation standard de l'entrée (*i.e.*, les caractéristiques des nœuds) avec les paramètres de la couche. L'originalité de cette couche est dans l'ajout d'une pondération de la transformation des données par une structure encodant la topologie du graphe. En outre, la transformation des caractéristiques descriptives d'un nœud dépend de celles de tous les autres nœuds du graphe, avec une pondération différente en fonction des forces de leurs liens. De ce point de vue, cette opération est aussi proche d'une couche d'*attention* que de *convolution*. Le problème principal

de ces convolutions de graphes est leur difficulté à différencier des nœuds ayant un voisinage similaire [25, 124, 125]. En effet, si deux nœuds ont des caractéristiques similaires et le même voisinage, il a été observé qu'un réseau exclusivement constitué de couches de convolutions de graphes devient rapidement incapable de différencier ces nœuds et finit par les projeter au même endroit. Depuis, ces couches de convolutions de graphes ont été déclinées plusieurs fois pour remédier à ce défaut (*e.g.*, FastGCN [126], GraphSAGE [127] et P-GNN [124]). D'autres structures ont également été proposées pour permettre l'apprentissage d'extraction de caractéristiques de graphes, telles que les réseaux d'attention de graphe (GAT pour « Graph Attention Network ») [128] s'inspirant des réseaux auto-attentifs de la littérature de traitement du langage [6] ; ou les réseaux d'isomorphisme de graphe (GIN pour « Graph Isomorphism Network ») [125] et ses extensions [129–131]. Ces trois types de réseaux de neurones implémentent le concept de Transfert de Message [132]. Dans ce paradigme, les caractéristiques d'un nœud sont mises à jour itérativement via une agrégation de ses propres caractéristiques avec celles de ses voisins. Les principales différences entre ces méthodes sont les types d'agrégations utilisés, et l'organisation des paramètres internes à chaque couche.

La plupart des cas d'application de ces outils sont la classification de graphes, nœuds ou arêtes [133–135]. Cependant, certaines études se sont intéressées à la problématique du Dessin de Graphe avec des réseaux de neurones profonds.

II.2.3 Apprentissage Profond pour le Dessin de Graphe

Depuis les travaux de Haleem *et al.* [29] sur l'évaluation automatique de la qualité de dessins nœuds-liens avec des réseaux de neurones profonds, l'idée d'utiliser des réseaux de neurones pour le dessin de graphe a émergé dans la communauté.

DeepDrawing [42] propose d'utiliser des réseaux LSTM [136] pour apprendre à dessiner des graphes de manière supervisée. Un LSTM (pour « Long-Short Term Memory ») est un réseau de neurones récurrent initialement dédié au traitement du langage. Pour le faire fonctionner sur des graphes, DeepDrawing constitue des *phrases* avec les nœuds d'un graphe. L'ordre des nœuds dans la phrase est déterminé par un parcours en largeur, et les vecteurs de caractéristiques représentant les nœuds encodent leur connexions avec les nœuds précédents dans la phrase. L'entraînement de DeepDrawing étant supervisé, il nécessite de pré-calculer des dessins NL, dits *vérités terrain*, des graphes du jeu d'entraînement. Ces vérités terrain sont générées avec un algorithme de référence. Chaque plongement prédit par le modèle est ensuite évalué en fonction de la distance entre les positions des nœuds dans le dessin prédit et le dessin vérité terrain. Le modèle apprend donc à reproduire les dessins créés par l'algorithme de référence. Cette approche présente cependant plusieurs défauts. (i) Le fait de devoir calculer les vérités terrain pour l'entraînement est coûteux. Un des avantages principaux des réseaux de neurones profonds étant leur rapidité de calcul, entraîner des réseaux à reproduire des techniques de plongement coûteuses mais efficaces est commun [34]. Par nature, les algorithmes de dessin que DeepDrawing cherche à imiter sont donc plutôt lents, et le coût du calcul des vérités terrain peut devenir prohibitif. Nous rappelons également que le paramétrage des algorithmes peut nécessiter de les exécuter plusieurs fois afin d'obtenir des résultats satisfaisants, et le paramétrage optimal peut varier en fonction de la donnée en entrée. Avec une méthode d'apprentissage supervisée, il faut donc paramétrer l'algorithme calculant les vérités terrain en plus du modèle d'apprentissage. (ii) Le second défaut de cette approche est que le modèle apprend également à reproduire les défauts de la méthode de référence qui a généré les vérités terrain. (iii) Enfin, nous pensons qu'un apprentissage supervisé bride le réseau dans son extraction des caractéristiques liées à la topologie du graphe. Ce type d'approche est très sensible au sur-apprentissage où

les paramètres du modèle se spécialisent sur les données du jeu d'entraînement, ce qui rend incertaines les capacités du modèle à généraliser son apprentissage. D'une manière générale, l'apprentissage supervisé ne semble pas adapté à la tâche de dessin de graphe puisqu'il n'existe pas de plongement *idéal* pour un graphe. En fonction de la tâche à réaliser sur la visualisation, des dessins différents d'un même graphe peuvent être efficaces.

Depuis, les méthodes dédiées aux graphes ont été développées, utilisant des opérations sur les graphes (voir Section II.2.2) et via des apprentissages non-supervisés. GraphTSNE [44] est un réseau de neurones peu profond utilisant des convolutions de graphe [123] pour produire des plongements basés sur l'extraction de caractéristiques des nœuds et de la topologie du graphe. Inspiré par tsNET [37], le réseau optimise une fonction de coût basée sur celle de l'algorithme t-SNE [35] adaptée aux graphes. L'originalité de GraphTSNE est de réaliser un apprentissage pour chaque graphe à dessiner, ce qui rend toutefois son temps d'exécution relativement élevé. Tiezzi *et al.* [43] comparent des réseaux de neurones peu profonds constitués exclusivement de GCN, GIN ou GAT (voir Section II.2.2) pour dessiner des graphes. Ils étudient également trois types de fonction de coût : (i) supervisée, (ii) non supervisée, et (iii) supervisée par l'estimation d'un *réseau esthète*. L'idée du réseau esthète est inspirée des travaux de Haleem *et al.* [29] qui utilisent un réseau de neurones non dédié au graphe pour apprendre de manière supervisée à prédire la qualité d'un dessin. Leurs résultats montrent que les couches GAT sont plus efficaces pour la tâche de dessin de graphe, et soulignent les difficultés des couches GCN à dissocier des nœuds avec un voisinage similaire. Ils confirment également les inconvénients de l'apprentissage supervisé de ce type de réseau, et se montrent encourageants sur l'utilisation d'un *réseau esthète* pour estimer la qualité des dessins produits par le réseau pendant l'apprentissage. L'utilisation de ce réseau esthète permettrait d'améliorer les capacités à généraliser du réseau entraîné à dessiner.

Aujourd'hui, les méthodes de dessin de graphe par réseau de neurones profond les plus représentatives du concept sont *(DNN)*^{2} [25, 63] et DeepGD [26]. Dans ces deux méthodes, l'architecture du réseau de neurones profond utilisée est calquée sur des architectures de la littérature du traitement d'image [137, 138], dans lesquelles les convolutions standards sont remplacées par des convolutions de graphe. Ces réseaux sont donc profonds et entraînés de manière non-supervisée. Les différences principales de ces deux travaux sont (i) la définition de la couche de GCN, (ii) les fonctions de coût optimisées, et (iii) l'objectif de leur évaluation. Dans *(DNN)*^{2} (présenté en détail dans le Chapitre III), la couche GCN est strictement définie comme l'originale utilisée par Kipf et Welling [123] basée sur les convolutions spectrales [139]. Dans nos études [25, 63], les implémentations de *(DNN)*^{2} n'ont pas vocation à en faire un algorithme prêt à l'emploi. L'objectif de l'évaluation de notre modèle est d'étudier le comportement de ces approches de dessin de graphe par GCN en fonction des variations de plusieurs hyper-paramètres. Dans DeepGD [26], les couches de GCN travaillent sur la matrice d'adjacence du graphe (par opposition au spectre du graphe pour *(DNN)*^{2}) et comportent un noyau d'apprentissage supplémentaire. Ce dernier, interne à chaque couche GCN, est appliqué sur la structure topologique du graphe et permet à chacune de ces couches d'apprendre une lecture différente de la topologie du graphe, adaptée à sa position dans l'architecture. Enfin, là où *(DNN)*^{2} optimise des fonctions de coût dérivées de t-SNE [35] (inspirées par tsNET [37] et GraphTSNE [44]), DeepGD optimise toutes sortes de critères esthétiques de dessin de graphe. Si ces fonctions de coût engendrent des différences fondamentales dans les résultats des deux méthodes, elles sont uniquement dues à des choix de conception différents, réalisés pour l'évaluation dans leurs articles respectifs. Les méthodes *(DNN)*^{2} et DeepGD étant fonctionnellement très proches, elles peuvent être entraînées pour optimiser les mêmes fonctions de coût.

Depuis, d'autres approches ont proposé d'employer des GCN pour dessiner des graphes, en utilisant des architectures différentes mais toujours dérivées de l'état de l'art de l'apprentissage

profond. GRAPHULY [45] propose d'organiser les couches GCN en une architecture U-Net [140, 141], avec une phase de rétrécissement suivie d'une phase d'affinage de la donnée en entrée. L'architecture type U-net permet au modèle de reproduire un comportement d'algorithme de dessin multi-échelles. Couplé à une fonction de coût multi-échelles, le modèle est capable de raffiner le plongement des nœuds efficacement. Enfin, SmartGD [142] propose d'organiser les couches GCN dans une architecture GAN [143] (pour « Generative Adversarial Network »). L'organisation en GAN singifie que l'architecture est composée en deux parties : un réseau *générateur* et un réseau *discriminateur*. L'objectif du *discriminateur* est de savoir dissocier une vraie donnée d'une fausse. Le *générateur* est entraîné à générer des données réussissant à tromper le *discriminateur*. Dans SmartGD, ce principe est détourné pour permettre un entraînement potentiellement infini du modèle. Le discriminateur est pré-entraîné pour être capable, à partir de plusieurs dessins, d'élire le meilleur. Initialement, il est entraîné sur des exemples *gagnants* de dessins issus de méthodes de la littérature. L'objectif du *générateur* est alors d'apprendre à produire des plongements meilleurs que ces exemples *gagnants* afin de tromper le discriminateur. Une fois qu'il y parvient, les exemples *gagnants* sont remplacés par les meilleurs dessins du générateur. Ce dernier doit donc continuellement apprendre à produire des dessins meilleurs que ses précédents pour tromper le discriminateur, ce qui permet hypothétiquement au modèle de converger vers un dessin considéré comme optimal.

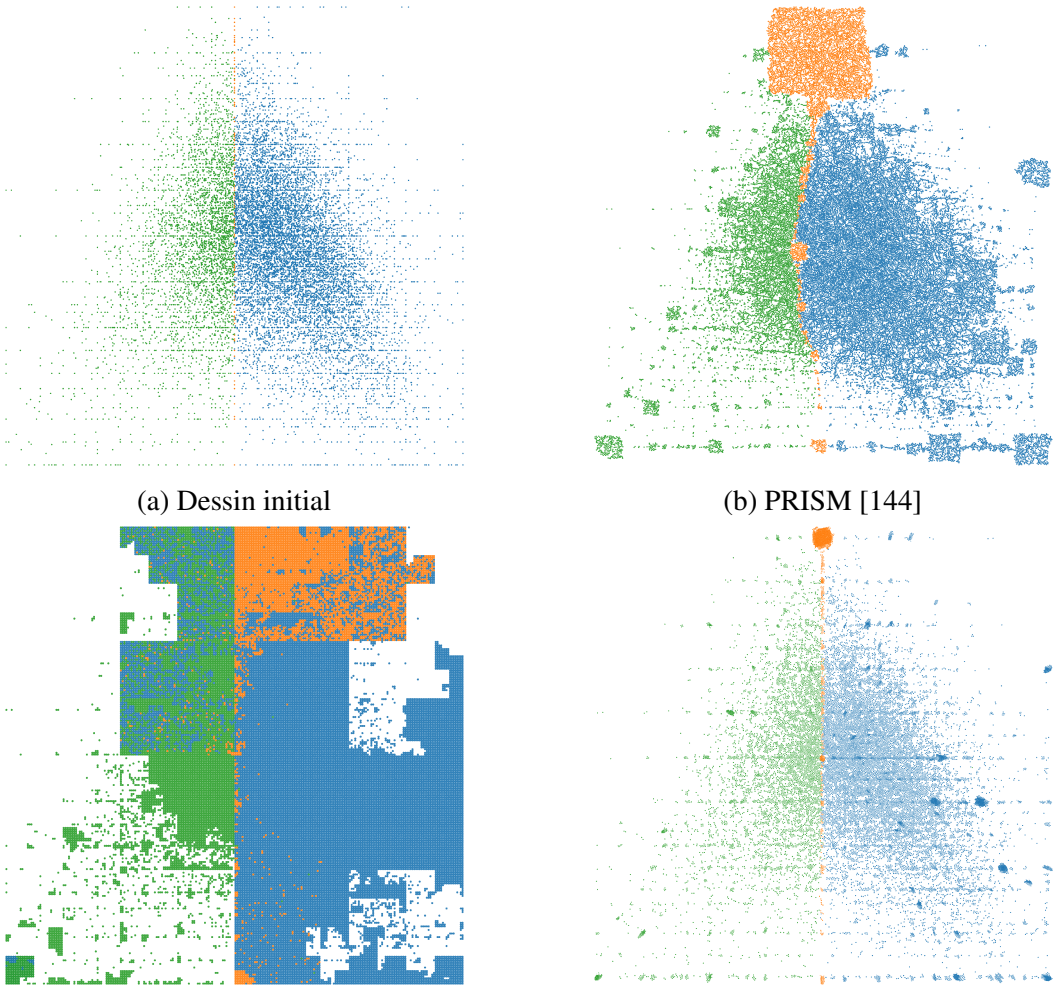
II.3 Suppression de Chevauchements

La plupart des algorithmes de plongement ayant vocation à produire une représentation 2D (*e.g.*, [35, 36, 111]) traitent les données dans l'espace comme des points n'ayant pas de forme ou taille. Cependant, il est courant de les représenter à l'écran avec des formes géométriques pour encoder de l'information supplémentaire (*e.g.*, étiquette textuelle, couleur ou forme encodant une catégorie) ou simplement pour obtenir une visualisation plus plaisante. Toutefois, ces formes et tailles sont souvent ajoutées de manière expérimentale par l'utilisateur après le plongement des données en 2D. L'ajout de ces attributs visuels modifie les distances relatives entre les données et peut créer des *chevauchements* qui entravent le processus d'exploration visuelle. Cette section présente les principaux travaux de la littérature de la Suppression de Chevauchements (OR pour « Overlap Removal ») en lien avec les contributions présentées dans le Chapitre IV. Cette présentation est séparée en trois catégories d'algorithmes ayant des objectifs et moyens différents. Ces catégories correspondent à la fois au type de données en entrée sur lequel travaillent les algorithmes, et leur objectif. Les trois catégories que nous distinguons sont : (i) l'Espace des Données, (ii) l'Espace Géométrique, et (iii) l'Espace Visuel. La suite de cette section définit chacun de ces espaces et décrit des algorithmes qui y sont associés. Des exemples visuels d'application d'algorithmes de suppression de chevauchements sont présentés dans la Figure II.6.

II.3.1 Espace des Données

Tel que nous le définissons, les méthodes d'OR dans l'*espace des données* ont connaissance des données hautement dimensionnelles (brutes), avant plongement en 2D. Ces méthodes tirent alors parti de ces informations pour produire des plongements fiables représentant les données d'origine.

Dans la majorité des cas, les approches dans l'espace des données sont basées sur l'agrégation ou le sous-échantillonnage des données initiales [147–150]. La principale différence entre ces méthodes est leur stratégie d'échantillonnage, visant toujours à faire en sorte que la densité



(a) Dessin initial (b) PRISM [144]

(c) HaGrid [145] (d) ScatterplotUnfold [146]

FIGURE II.6 : Exemples de plongements solutions produits par différents algorithmes de Suppression de Chevauchements. Les plongements illustrent : (b) une méthode dans l'espace géométrique ; (c) une méthode de gridification dans l'espace visuel ; et (d) une méthode de préservation de densités dans l'espace visuel.

des données plongées en 2D soit représentative de la distribution des données dans l'espace d'origine en hautes dimensions. D'autres méthodes [151, 152] produisent un plongement en tenant compte des tailles des nœuds. Par exemple, Harel et Koren [152] proposent une adaptation de l'algorithme de dessin de graphe par modèle de forces de Kamada et Kawai [59] qui tient compte de la taille des nœuds.

Étant donné que nous ne nous intéresserons pas à cette catégorie d'algorithmes par la suite, nous ne détaillons pas plus leur fonctionnement. Pour cela, nous recommandons l'étude de Yuan *et al.* [153] qui recense et compare les algorithmes d'échantillonnage dédiés à la visualisation de nuages de points.

II.3.2 Espace Géométrique (Dispersion de Nœuds)

Les algorithmes de suppression de chevauchements dans l'Espace Géométrique (GS pour « Geometric Space ») prennent en entrée un plongement en 2D et produisent un nouveau plongement 2D, dit *ajusté* ou *solution*. Nous parlons ainsi de GS car ils travaillent uniquement sur les ca-

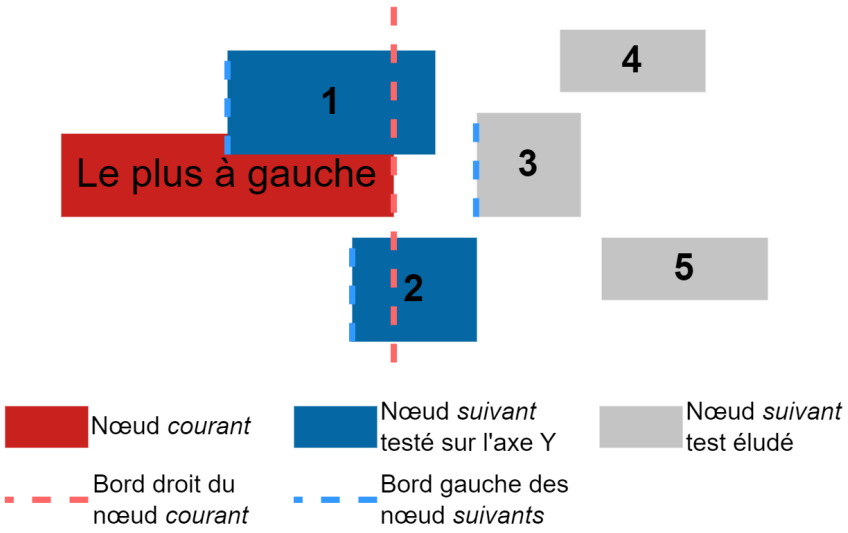


FIGURE II.7 : Illustration des tests effectués dans l'algorithme *scan-line* [154] pour détecter les chevauchements. Les nœuds sont tout d'abord triés sur l'axe X, puis comparés au nœud le plus à gauche (en rouge). Si un chevauchement est détecté sur l'axe X, les coordonnées sur l'axe Y sont vérifiées (en bleu, 1 et 2). Si la coordonnée sur du bord gauche d'un nœud est plus grande que celle du bord droit du nœud rouge (nœud 3), les nœuds ne peuvent pas se chevaucher, et les tests avec les nœuds à droite du 3 sont éludés.

ractéristiques *taille* et *position* des nœuds. L'objectif de ces algorithmes est de supprimer les chevauchements tout en minimisant les déformations du plongement initial.

Pour être capable de supprimer les chevauchements, il faut tout d'abord les détecter. Pour cela, une alternative plus efficace que la comparaison exhaustive deux à deux est proposée avec *scan-line* [154], illustrée dans la Figure II.7. Dans cet algorithme, les nœuds sont d'abord triés sur l'axe X, puis traités un par un dans cet ordre. Chaque nœud courant est comparé aux nœuds *suivants* (*i.e.*, à sa droite). Si le bord droit du nœud courant a une coordonnée plus grande que le bord gauche du nœud suivant, ils se chevauchent sur l'axe X et il faut alors vérifier s'il y a également chevauchement des coordonnées sur Y. Sinon, on passe au prochain nœud suivant. Si le bord droit du nœud courant a une valeur plus faible que le bord gauche du nœud suivant, cela signifie que les deux nœuds sont séparés sur l'axe X. Qu'importe leur position sur l'axe Y, ils ne peuvent donc pas se chevaucher. De plus, les nœuds étant triés, cela signifie que le nœud courant ne peut pas être en chevauchement avec les nœuds à droite du nœud suivant. L'algorithme permet donc d'élaguer une grande partie des tests à effectuer. Sa complexité pour détecter si un plongement $\mathbb{R}^{N\times 2}$ à N points contient des chevauchements est $\mathcal{O}(N \log N)$. Néanmoins, sa complexité pour trouver l'ensemble des chevauchements reste nécessairement $\mathcal{O}(N^2)$ puisque c'est le nombre maximum de chevauchements que peut contenir un plongement. En pratique, bien que cette complexité reste quadratique, l'algorithme a un temps d'exécution beaucoup plus efficace qu'une recherche exhaustive naïve.

L'étude de Chen *et al.* [49,155] présente les principaux algorithmes de suppression de chevauchements dans l'espace géométrique. Tous ces algorithmes suppriment les chevauchements par dispersion (*i.e.*, mouvement) de nœuds, et peuvent être regroupés en trois catégories présentées ci-après. Le seul algorithme que nous n'associons à aucune de ces catégories et qui sert de référence est *Scaling* [49, 155]. Il consiste à multiplier les coordonnées de tous les nœuds par le facteur minimum qui permet la suppression de tous les chevauchements. Multiplier les coordonnées de tous les nœuds par un même facteur revient à produire des mouvements uniformes

dans tout le plongement, ce qui préserve parfaitement les distances relatives mais augmente significativement l'espace utilisé par le plongement solution.

Séquentiels. Les algorithmes que nous appelons séquentiels se basent sur *scan-line* [154] pour détecter les chevauchements et les traitent en série avec des mouvements, par exemple orthogonaux. PFS [156], PFS' [157] et FTA [158] sont constitués de deux passes pour gérer les mouvements de nœuds verticaux et horizontaux séparément. À l'inverse, RWordle [159] déplace les nœuds chevauchés sur les deux axes en même temps. Concrètement, pour chaque paire de nœuds chevauchés, l'un est défini comme référence, et l'autre est positionné sur une *spirale* partant du nœud de référence, dont la courbure est définie par la taille des nœuds. Tous ces algorithmes ont une complexité quadratique sur le nombre de nœuds du plongement.

Par contrainte. Certains algorithmes modélisent l'ensemble des chevauchements du plongement initial comme un ensemble de contraintes à satisfaire. C'est par exemple le cas de VPSC [154, 160] qui modélise à la fois des contraintes entre les paires de nœuds chevauchées (à déplacer) et les paires non chevauchées (à préserver). Cet algorithme a malgré tout tendance à largement déformer le plongement initial. Sa complexité est $\mathcal{O}(CN\log C)$ où C est le nombre de contraintes en $\mathcal{O}(N)$; menant à une complexité finale de $\mathcal{O}(N^2\log N)$. Diamond [161] est un autre algorithme basé sur la programmation par contraintes en $\mathcal{O}(N^2)$ qui optimise la préservation de l'ordre orthogonal des nœuds dans le plongement. L'algorithme suppose que les nœuds sont représentés par des carrés et montre qu'il est possible de faciliter la relaxation des contraintes en effectuant une rotation de 45° de ces carrés, les représentant ainsi comme des *losanges*.

Par optimisation du Stress. Enfin, la dernière catégorie que nous présentons ici est celle des algorithmes d'optimisation du Stress dont fait partie notre algorithme FORBID [27] décrit dans le Chapitre IV. De manière générale, le Stress d'un plongement est exprimé ainsi :

$$\sigma(X) = \sum_{i,j} \delta_{ij}^{-2}(d(X_i,X_j) - \delta_{ij})^2 \tag{II.1}$$

où X est un plongement dans $\mathbb{R}^{N\times 2}$, $d(X_i,X_j)$ est la distance euclidienne entre les points X_i et X_j dans le plongement, et δ est l'ensemble des distances idéales entre les paires de nœuds.

L'idée des approches par optimisation du Stress est de modéliser l'ensemble des distances à préserver (*i.e.*, entre des nœuds qui ne se chevauchent pas) et celles à modifier (*i.e.*, entre des nœuds qui se chevauchent) comme un problème d'optimisation de Stress. La distance idéale entre les nœuds qui ne se chevauchent pas est alors fixée à leur distance initiale, de sorte que l'optimisation préserve cette distance. À l'inverse, la distance entre les paires de nœuds qui se chevauchent est fixée à une certaine valeur, dépendant des choix de conception de l'algorithme, telle que les nœuds ne se chevauchent plus. Cette approche revient à créer un espace K-dimensionnel dans lequel toutes les distances sont préservées, sauf celles entre les nœuds qui se chevauchent. L'algorithme d'optimisation a ensuite comme rôle de re-plonger cet espace en 2D pour produire un dessin qui supprime les chevauchements tout en préservant le plongement initial. Par conception, l'optimisation du Stress a une complexité quadratique sur le nombre de nœuds.

PRISM [144] est un exemple d'algorithme de suppression de chevauchements par optimisation de Stress. Il crée un *graphe de proximité* du plongement initial via une Triangulation de Delaunay [75]. Les arêtes de ce graphe de proximité modélisent les distances les plus courtes dans le plongement initial, c'est-à-dire l'endroit où la probabilité qu'il y ait des chevauchements est la plus forte. PRISM optimise ensuite le Stress le long de ces arêtes pour supprimer les chevauchements. Néanmoins, comme le graphe de proximité ne capture pas nécessairement tous les chevauchements du plongement initial, l'algorithme augmente les arêtes du graphe de proximité en recherchant les chevauchements dans le plongement, puis re-optimise le Stress.

Les détails de la modélisation du Stress de PRISM seront décrits dans le Chapitre IV pour la dissocier de celle de FORBID. La complexité de PRISM est $\mathcal{O}(t(mkN + N\log N))$ où m et k sont des hyper-paramètres d'optimisation, et t dépend du nombre de chevauchements (et peut être supérieur à N dans le pire des cas). Enfin, GTREE [162] est un autre algorithme de suppression de chevauchements par optimisation de Stress qui s'inspire fortement de PRISM. Sa contribution principale est l'ajout du calcul d'un arbre couvrant minimal du graphe de proximité. Cet arbre couvrant permet de réduire le nombre de déplacements de nœuds à calculer et améliore sensiblement le temps d'exécution, bien que la complexité de l'algorithme reste quadratique.

II.3.3 Espace Visuel

La dernière famille d'algorithmes que nous présentons ici travaille dans l'Espace Visuel (VS pour « Visual Space »). Ainsi, ils ont connaissance, voire optimisent des paramètres relatifs à l'espace de rendu final du plongement (*i.e.*, l'image résultante de la visualisation). En effet, l'objectif des algorithmes travaillant dans GS étant de produire des plongements sans chevauchements, ils ne tiennent pas compte de l'environnement de rendu du plongement. Cela pose de sérieux problèmes pour les algorithmes tels que *Scaling* [49, 155] qui augmentent drastiquement les dimensions du plongement. Considérant qu'un plongement est rendu dans un espace *cible* de taille fixe (*e.g.*, une résolution d'image), augmenter la taille du dessin (*i.e.*, sa boîte englobante) revient à réduire la taille des nœuds lors de la projection dans l'espace *cible*. Les plongements solutions produits par des méthodes telles que *Scaling* sont tellement grands que les nœuds deviennent extrêmement petits, voire invisibles dans l'image (*i.e.*, largeur inférieure à 1 pixel), une fois adaptés à la résolution cible. À l'inverse, les méthodes travaillant dans VS ont pour objectif de produire des visualisations dans lesquelles les données sont *visibles*. Cela implique que les chevauchements doivent être *résolus* (et pas nécessairement supprimés) de telle sorte que chaque nœud puisse avoir une aire d'affichage qui lui est exclusivement dédiée. Enfin, cela implique également que la taille des nœuds dans l'espace de rendu (*i.e.*, l'image) soit toujours visible, si ce n'est maximale.

Une première approche pour garantir la suppression de chevauchements tenant compte d'une forme de résolution d'affichage est l'ajustement du plongement en positions entières, aussi appelé *gridification*. Le fait de positionner les nœuds dans les cellules d'une grille garantit qu'ils ne peuvent plus se chevaucher, et la dimension de la grille peut être interprétée comme une résolution minimale où les cellules feraient un pixel de large. DGrid [163] et HaGrid [145] sont deux algorithmes de gridification pour la suppression de chevauchements. Le fonctionnement en grille et son découpage récursif leur permettent d'atteindre de très bonnes complexités. DGrid [163] dessine une grille sur le plongement initial, puis détermine quelles sont les zones denses du plongement. À l'extérieur de ces zones, l'algorithme rajoute des points factices pour représenter les espaces vides du plongement. Enfin, un algorithme de découpage récursif de la grille se charge d'assigner chaque nœud à sa cellule, avec certaines contraintes permettant de minimiser la déformation du plongement initial. Dans HaGrid [145], les positions des nœuds sont attribuées à l'aide d'une courbe de remplissage de l'espace (*space filling curve*) dont la profondeur de récursion est définie pour produire des plongements plus ou moins compacts. Ces deux algorithmes ont une complexité en $\mathcal{O}(Nlog(N))$. Ils sont donc les plus rapides des algorithmes d'ajustement décrits jusqu'à présent, mais peuvent créer de sévères déformations du plongement initial en essayant de le compacter.

Les méthodes de *gridification compacte* peuvent également être considérées comme des solutions au problème de chevauchement. En effet, elles fonctionnent elles aussi en assignant à chaque nœud une position exclusive, garantissant sa visibilité. Si DGrid et HaGrid *peuvent* être

paramétrés pour produire ce type de visualisation, certains algorithmes sont explicitement conçus dans ce but. Par extension, nous les intégrons donc aux travaux de la littérature sur l'ajustement de plongements dans VS. Par exemple, SSM [164] projette les nœuds dans une grille de taille minimale (pouvant contenir tous les nœuds) et effectue des permutations aléatoires jusqu'à satisfaire une métrique de similarité. VRGrid [165] est un autre exemple d'ajustement compact qui calcule des tesselations de Voronoï pour scinder l'espace visuel et attribuer une position à chaque nœud.

Enfin, il existe peu de méthodes d'ajustement dans VS qui ne correspondent pas à une des sous-catégories précédentes. L'une d'entre elles est ScatterplotUnfold [146]. De par sa connaissance, voire l'optimisation de certains paramètres de rendu du plongement solution, c'est la méthode la plus proche de notre contribution GIST [65] (voir Chapitre IV). L'objectif de ScatterplotUnfold est de s'assurer que la distribution des densités de nœuds dans le plongement solution soit visuellement représentative de cette distribution dans le plongement initial. La densité dans une visualisation de nuage de points étant relative à la distribution des points dans l'espace, l'algorithme inclut une sécurité qui s'assure que les nœuds dans les zones les moins denses restent visibles. Pour atteindre son objectif, il déplace les nœuds en utilisant un algorithme dérivé de l'empilement de cercles [166] (*circle packing* en anglais) pour garantir la visibilité des nœuds dans l'espace visuel. L'algorithme peut modifier la taille des nœuds à l'écran pour modifier la densité visuelle et ainsi garantir que la distribution des densités dans le plongement solution soit bien représentative de celle du plongement initial. Malgré la sécurité implémentée dans l'algorithme, certains nœuds peuvent tout de même devenir trop petits pour être visibles.

Si nous pensons qu'il est nécessaire que les algorithmes d'ajustement se préoccupent de l'espace de rendu afin de s'assurer que les données soient visibles, il serait encore plus efficace de s'intéresser à la *perceptibilité* des données. Ainsi, nous identifions une quatrième catégorie d'algorithmes qui travaille dans un *Espace de Perception*. Bien que nous ne les utiliserons pas dans le reste de ce document, il est important de noter leur existence. Par exemple, Micallef *et al.* [167] utilisent les données brutes (en hautes dimensions) pour optimiser une fonction de coût basée sur des critères de perception des utilisateurs finaux. Ils peuvent pour cela modifier des paramètres du rendu final tels que l'opacité des nœuds. ClusterNet [168] propose d'utiliser un réseau de neurones profond pour calculer les groupements de nœuds dans des nuages de points. L'originalité de cette approche est de biaiser l'entraînement du modèle à partir de données récoltées lors d'une évaluation utilisateur. Ainsi, le modèle entraîné encode à la fois l'efficacité d'un algorithme de groupement, et un comportement adapté aux préférences des utilisateurs.

II.4 Efficacité de Visualisations

Dans les sections précédentes, nous avons présenté des travaux sur la génération de visualisations (*i.e.*, dessin de graphes, voir Section II.2) et le traitement de visualisations (*i.e.*, suppression de chevauchements, voir Section II.3). Cependant, la conception même des travaux sur ces deux thématiques repose sur un prérequis fondamental : la capacité de formellement évaluer la qualité des visualisations. Ce prérequis est nécessaire afin de définir des métriques et des protocoles d'évaluation fiables, permettant de mesurer les réelles améliorations apportées par les nouvelles techniques de visualisation.

L'évaluation utilisateur est la méthode standard pour quantifier l'efficacité de visualisations pour la perception humaine. Le principal avantage de cette méthode est que l'efficacité de la visualisation est directement évaluée sur un échantillon de la population d'utilisateurs finaux po-

tentiels. Cependant, elle présente également plusieurs défauts (mentionnés dans la Section I.4.2) qui ont motivé la recherche de nouveaux protocoles d'évaluation automatisés. Les objectifs principaux de ceux-ci sont de (i) réduire le coût de conception et de mise en œuvre des évaluations, (ii) proposer des supports quantitatifs à leur conception, et (iii) améliorer leur reproductibilité.

Cette section est dédiée à la présentation des travaux de la littérature concernant la Perception humaine sur la tâche de Recherche Visuelle pour la détection d'intrus, en lien avec le Chapitre V. De plus, nous présentons les études existantes en matière d'évaluation automatique de visualisations qui sont à la base de celle développée dans nos travaux.

II.4.1 Perception, Recherche Visuelle et Détection d'Intrus

La théorie fondatrice du domaine de Recherche Visuelle est la *Théorie de l'intégration des caractéristiques* (FIT pour « Feature-Integration Theory ») de Treisman et Gelade [169]. Elle définit l'attention comme un système à deux étapes, d'abord *pré-attentive* puis *attentive*. Certains attributs visuels peuvent être considérés pré-attentifs si leurs caractéristiques peuvent être traitées parallèlement par notre perception [170]. Elle distingue également la recherche de caractéristiques *simples* (*e.g.*, recherche d'une couleur), de la *conjonctive* (*e.g.*, recherche d'un objet défini par une couleur et une forme). Aujourd'hui, la théorie la plus consensuelle est *Guided Search* de Wolfe [171, 172] qui est régulièrement mise à jour (la plus récente étant la version 6.0 [173]). Selon cette théorie, le système d'attention peut suivre deux stratégies pour résoudre une tâche : *bottom-up* ou *top-down*. La stratégie *bottom-up* est centrée sur les stimuli : l'attention se dirige sur un stimuli, puis le cerveau essaie de l'interpréter et le faire correspondre à un concept. À l'inverse, la stratégie *top-down* est centrée sur l'utilisateur. Dans la plupart des cas, cette deuxième stratégie est employée quand la cible est connue. Le cerveau part alors d'une vision du concept et dirige son attention vers des stimuli susceptibles de lui correspondre. Néanmoins, la théorie *FIT* a été essentielle au domaine de la Recherche Visuelle en Perception, comme montré dans l'édition spéciale dédiée à la mémoire d'Anne Treisman [174, 175].

Duncan et Humphreys [176] ont basé leur théorie sur la similarité entre les stimuli et les modèles visuels. Ils séparent les stimuli en deux catégories : les cibles (T pour « target ») et les distracteurs (N pour « non-target »). Dans leur étude, les auteurs démontrent que lorsque la similarité entre T et N augmente, la tâche devient plus difficile à résoudre. La difficulté croît encore si la similarité N-N augmente, car cela signifie que tous les distracteurs se ressemblent entre eux, et ressemblent à la cible. À l'inverse, lorsque la similarité T-N reste faible, la difficulté de la tâche n'est pas corrélée à l'augmentation de la similarité N-N (*i.e.*, la ressemblance entre les distracteurs n'a pas d'impact négatif s'ils sont facilement distinguables de la cible). De plus, le nombre de distracteurs possibles dans la représentation, appelé *hétérogénéité des distracteurs*, complexifie sévèrement la tâche. Enfin, ils montrent que si un intrus peut être identifié via une dimension (*i.e.*, attribut visuel) spécifique, appelée dimension *pertinente*, l'hétérogénéité dans les autres dimensions ne devrait avoir qu'un impact mineur sur la difficulté de la tâche, ce qui corrobore certains travaux de Treisman [177]. Pashler [178] a étendu cette étude à l'hétérogénéité dans les dimensions non pertinentes et a montré que celle-ci n'avait pas d'impact sur la difficulté de la tâche, *même lorsque l'intrus est inconnu*. Cela signifie que même si un individu ne connaît pas la forme et la couleur de ce qu'il recherche, si ce qu'il cherche est identifiable par sa couleur unique, l'hétérogénéité des formes n'affecte pas la difficulté de sa tâche. En résumé, les efforts que nécessitent la recherche visuelle dépendent des similarités T-N et N-N, de l'hétérogénéité de la représentation, ainsi que des dimensions dans lesquelles l'hétérogénéité est représentée.

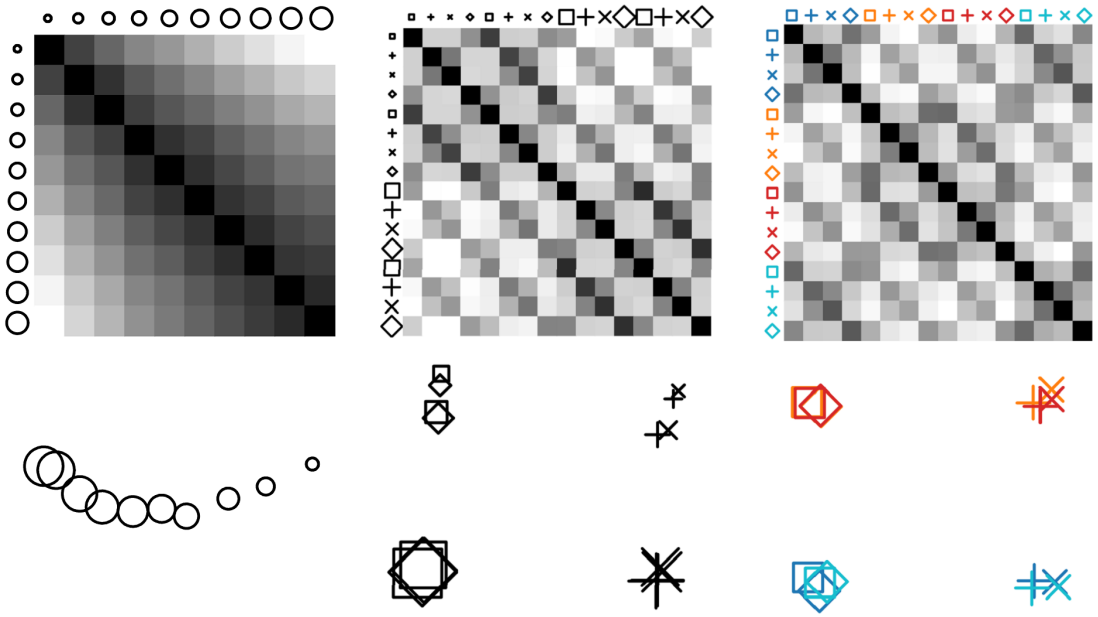
Quinlan et Humphreys [179] ont montré que la difficulté de la recherche d'une cible définie par sa *forme* augmente linéairement avec le nombre total de stimuli, tandis qu'elle n'augmente

pas si la cible est définie par sa couleur. Cela corrobore les études définissant la *couleur* comme un attribut visuel pré-attentif [169, 170]. Ils démontrent également que la similarité T-N a plus d'impact en recherche conjonctive où les cibles sont définies par des conjonctions d'attributs visuels. D'une manière générale, ils en concluent que plus une cible partage des attributs visuels avec les distracteurs, plus la difficulté de la tâche est grande et sensible à l'hétérogénéité.

Si le domaine de la Perception a mené des études théoriques du système perceptif humain, ses recommendations ne sont pas toujours adaptées aux cas d'application en Visualisation d'Information. Par exemple, Treisman et Gelade [169] revendiquent que l'« *on ne peut pas détecter une cible dans un ensemble de distracteurs si on ne connaît pas au moins l'attribut visuel qui le différencie* ». Toutefois leur évaluation était basée sur des cas expérimentaux présentés à des utilisateurs pendant seulement 3 secondes. Un tel cas d'application n'est pas représentatif des tâches réalisées sur les visualisations, notamment les plus complexes qui nécessitent parfois un temps de déchiffrage important malgré leur efficacité (*e.g.*, trouver une ville sur la carte d'un pays inconnu). Healey et Enns [180] ont dressé un tableau de la littérature de la perception visuelle dédiée aux applications de la communauté de Visualisation. Depuis lors, cette communauté a poursuivi ses études sur l'efficacité du système perceptif humain dans son contexte applicatif, en complément des recherches de la communauté de Perception.

Haroz et Whitney [181] ont étudié comment les *groupements de couleurs* et les *mouvements* influencent l'efficacité des visualisations pour la détection d'une cible. Ils ont conduit plusieurs expériences avec 5 participants devant décider si une cible était présente dans une visualisation, sur 960 exemples chacun. Cela leur a permis d'étudier l'impact de certains paramètres tels que le nombre de couleurs, ou le positionnement de groupes de couleur. Ils en concluent que grouper les couleurs facilite significativement la tâche lorsque l'intrus est inconnu. De plus, lorsque les couleurs sont groupées, il est plus facile d'extraire de l'information générale sur la visualisation, tel que le nombre total de couleurs. Gramazio *et al.* [182] ont étendu ce travail en étudiant l'impact du nombre de stimuli total et leur disposition dans le plan sur la détection de cible. Dans cette étude, les auteurs font varier l'hétérogénéité de la représentation avec le nombre de stimuli, leur disposition, leur taille et le nombre total de couleurs. Ils concluent que grouper les attributs visuels par similarité de leur valeur (*e.g.*, grouper les couleurs par similarité dans un espace tel que CIELAB) améliore significativement la lisibilité de la visualisation. Lorsque les stimuli sont groupés par similarité, la tâche de recherche visuelle n'est pratiquement pas affectée par l'hétérogénéité de la représentation. Enfin, ils recommandent d'utiliser des stimuli dont la taille est $> 0.508°$ d'angle visuel ($1° \approx 1.064$ cm, les participants étant approximativement à 60 cm de l'écran) pour ne pas rendre la tâche difficile. Leur étude montre également que passé une certain seuil, il n'y a plus d'amélioration significative à augmenter la taille des stimuli, les performances étant sensiblement équivalentes avec des stimuli de taille $0.762°$ à $1.271°$.

L'étude de Demiralp *et al.* [183] introduit la notion de *noyau perceptuel*, une matrice de distances qui représente la perceptibilité des différents stimuli qui peuvent être issus de combinaisons d'attributs visuels (voir Figure II.8). Leur expérience étudie les combinaisons de couleurs, formes et tailles de stimuli. Elle permet de mettre en place une méthode d'évaluation de l'efficacité perceptuelle des différentes combinaisons réalisables avec ces attributs. Leur travail montre que la couleur et la forme ont des noyaux perceptuels très différents. Dans le noyau des formes, nous pouvons observer que certaines formes sont très proches, tandis que d'autres sont très distantes. À l'inverse, le noyau perceptuel des couleurs montre une utilisation plus uniforme de l'espace. Néanmoins, tous les stimuli issus de la combinaison couleur-forme sont regroupés en un faible nombre de groupes. Leur expérience comporte 4 couleurs et 4 formes (*i.e.*, 16 stimuli différents), mais seulement 4 groupes de stimuli sont distinguables dans le noyau perceptuel



(a) Taille (b) Taille-Forme (c) Couleur-Forme

FIGURE II.8 : Exemples de *noyaux perceptuels* et de leur projection provenant de l'étude de Demiralp *et al.* [183]. Les matrices sur la ligne du haut représentent les noyaux. Les valeurs dans la matrice sont estimées à partir des *distances perceptuelles* mesurées entre les stimuli en ligne et en colonne pendant une évaluation utilisateur. Les visualisations de la ligne du bas représentent les projections 2D des stimuli via positionnement multidimensionnel (MDS) à partir de leurs caractéristiques dans le noyau.

de *couleur-forme*. Cette étude ne propose pas de conclusion sur l'évaluation de l'efficacité relative des stimuli considérés pendant leur expérience, mais elle souligne l'importance du choix des attributs visuels à utiliser dans une visualisation pour que l'encodage de l'information soit efficace. Dans le Chapitre V, les choix d'attributs visuels et de leurs valeurs dans notre expérience sont en partie motivés par les résultats de cette étude.

II.4.2 Évaluation de Visualisations avec des Réseaux de Neurones Profonds

Évaluer des visualisations et des techniques de visualisation est un processus complexe qui nécessite un cadre spécifique et des tâches bien définies. Jusqu'à récemment, la seule méthode consensuellement acceptée pour réaliser ce type d'évaluation était les *évaluations utilisateurs* [184]. Aujourd'hui, l'avancée des techniques de vision par l'ordinateur (*Computer Vision* en anglais) permet d'envisager leur utilisation pour évaluer automatiquement des visualisations. Étant donné la récence de cette thématique, peu de travaux traitent exclusivement de l'évaluation automatique avec des techniques de vision par l'ordinateur. Cette section en présente les travaux principaux sur lesquels nous basons nos contributions détaillées dans le Chapitre V. Pour une description plus détaillée, nous recommandons les études AI4VIS [15], ML4VIS [16] et VIS+AI [8] qui présentent les différents liens entre la visualisation d'information et l'intelligence artificielle. Dans leur terminologie, les auteurs nomment l'évaluation automatique de visualisations : *quality assessment*.

L'idée même d'utiliser des réseaux de neurones profonds pour l'évaluation de visualisations a émergé de plusieurs études. Dans leur travail, Behrisch *et al.* [185] étudient différentes métriques de qualité pour évaluer des visualisations, et mentionnent l'utilisation des réseaux

de neurones pour estimer la qualité de visualisations comme une approche prometteuse. Bethge *et al.* [186] utilisent par exemple un réseau de neurones pour prédire les performances d'algorithmes de cartes proportionnelles (*treemap* en anglais). Le travail de Haehn *et al.* [28] a reproduit l'évaluation de Cleveland et McGill [187] avec des réseaux de neurones convolutifs (CNN pour « Convolutional Neural Network ») pour comparer leur comportement à ceux de participants humains sur différentes tâches élémentaires de perception graphique. Ils concluent que les humains et les CNN se comportement différemment sur ces tâches élémentaires sauf dans certains cas spécifiques, mais restent optimistes quant à leur utilisation pour l'évaluation de visualisations.

Haleem *et al.* [29] ont entraîné un CNN à prédire différentes métriques de qualité de dessins de graphes nœud-lien. L'originalité de cette étude est de nourrir le réseau avec des images issues des dessins de graphes, et non pas à partir de vecteurs numériques représentant la donnée brute. Leur approche laisse au réseau la responsabilité d'apprendre à extraire les caractéristiques importantes de l'image pour estimer sa qualité. Ainsi, le CNN est affecté par les choix de rendu (*e.g.*, couleur, taille, forme des nœuds) et doit apprendre à *interpréter* l'image pour résoudre la tâche, comme le ferait un participant humain. Leur modèle atteint 85% d'exactitude. Les scores de métriques que le CNN a appris à estimer étant liés aux préférences de participants humains [53, 56–58] (voir Section II.2.1), cela signifie que le réseau est capable d'estimer avec précision la qualité de la visualisation.

Dans un de nos travaux [68], nous proposons une méthodologie de comparaison automatique de visualisations. L'idée est de fixer un jeu de données, une tâche, une architecture de DNN ainsi que tous ses hyper-paramètres, et le protocole d'évaluation de l'efficacité du réseau pour résoudre la tâche. Ensuite, nous entraînons un DNN à résoudre la tâche sur les images produites par des techniques de visualisations séparément. La seule différence (contrôlée) entre les deux entraînements est la technique de visualisation utilisée, et donc les images passées en entrée du réseau. En comparant les performances des deux instances de notre modèle, nous pouvons alors voir lequel a le mieux appris à résoudre la tâche. Puisque la technique de visualisation est la seule variable isolée entre les deux entraînements, nous pouvons conclure que le réseau qui obtient les meilleures performances atteint son niveau d'efficacité car la technique de visualisation qui a produit les images qui l'ont nourri est plus efficace pour transmettre l'information nécessaire à la résolution de la tâche. En appliquant ce protocole à la comparaison de représentations de graphes par nœud-lien et par matrice d'adjacence, nous avons retrouvé les résultats d'expériences de la littérature réalisés sur des participants humains [54, 55].

Cai *et al.* [188] proposent de nouvelles métriques dédiées à l'évaluation fidèle de la difficulté de la lecture d'un chemin dans un graphe. Le premier objectif de ces métriques est de quantifier précisément la qualité d'une visualisation de graphe par nœud-lien pour permettre à un utilisateur de résoudre la tâche de *recherche d'un plus court chemin* dans un dessin de graphe. Les auteurs évaluent tout d'abord l'efficacité de leurs métriques en étudiant les coefficients de corrélation entre celles-ci et les performances d'utilisateurs mesurées pendant une évaluation. Cette étude de corrélations montre que leurs métriques sont plus efficaces que certains critères établis dans la littérature pour estimer la difficulté de la tâche, bien que tous les coefficients de corrélation de Pearson [189] soient < 0.4. Ce travail propose également une évaluation automatique de la difficulté de la tâche de recherche d'un plus court chemin via de l'apprentissage automatique (M), de l'apprentissage profond (SP), et la combinaison des deux (MSP). L'objectif de ces approches est d'estimer les performances mesurées des participants humains à partir de certains scores de métriques de qualité des visualisations nœud-lien, et/ou des images résultant de ces visualisations. Les performances du meilleur modèle de cette étude (MSP) s'approchent à $MSE = 0.3555$ d'erreur quadratique moyenne (MSE pour « Mean Squared Error ») du temps de

réponse des participants. De plus, le modèle estime avec une exactitude moyenne de 71.69% la justesse de la réponse des participants. Cependant, le modèle n'obtient que 33.37% d'exactitude moyenne sur l'estimation de la difficulté de la tâche perçue par les utilisateurs, mesurée via une échelle de Likert [190] entre 1 et 9. Ces résultats montrent qu'il est possible de finement estimer les performances d'utilisateur avec des modèles d'apprentissage, mais qu'il est difficile de capturer leur *impression* de la difficulté.

Enfin, comme mentionné dans les études AI4VIS [15], ML4VIS [16] et VIS+AI [8], le nombre d'études traitant à la fois de Visualisation et d'apprentissage profond a fortement augmenté depuis quelques années. Désormais, de plus en plus de travaux proposent des aides à la conception, voire des recommendations à disposition des experts pour créer des visualisations efficaces [31, 191, 192].

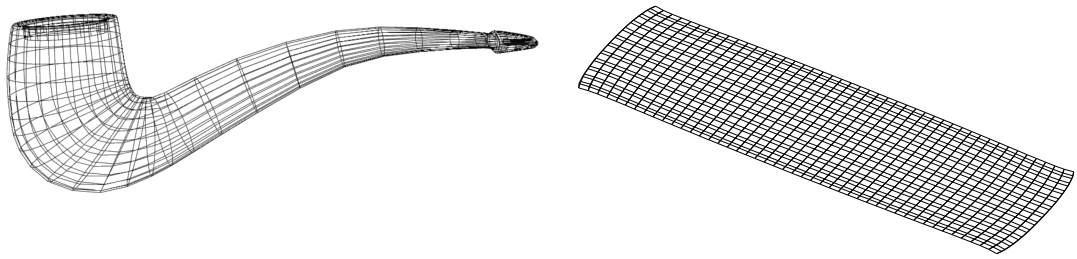
Chapitre III

Dessin de Graphe avec des Réseaux de Neurones Profonds

Sommaire

III.1 Problématique

Les graphes sont des structures de données discrètes définissant des relations entre des entités. Leurs applications sont nombreuses étant donnée la large variété de domaines métiers qui les utilisent (*e.g.*, réseaux sociaux [85, 86], biologie [87, 88], trafic de différentes sortes [89]). Cette structure est étudiée depuis des décennies et des algorithmes existent désormais pour la traiter efficacement afin de résoudre une variété de tâches (*e.g.*, chercher un élément, détecter une structure topologique particulière). Plus que jamais, avec la démocratisation de l'Internet des Objets où chaque composant est intelligent et communique avec d'autres, les données que nous manipulons deviennent massives et des algorithmes efficaces et scalables (*i.e.*, capables de passer à l'échelle) sont nécessaires pour les traiter.



Ceci n'est pas une pipe.

FIGURE III.1 : Deux dessins d'un même graphe. À gauche, le positionnement des nœuds suggère le concept abstrait d'une pipe. Dans le dessin de droite, la structure en maillage du graphe est mise en évidence. Ces représentations montrent qu'un même graphe peut être dessiné de manières différentes, pour mettre en valeur divers critères esthétiques. Cette figure est inspirée de la peinture surréaliste « *La Trahison des images* » (1929) de R. Magritte dans laquelle l'artiste voulait montrer que la représentation d'un objet (ici, un dessin de graphe) est différente de l'objet lui-même (ici, un graphe).

Ce travail se concentre sur la tâche de Dessin de Graphe par visualisation Nœud-Lien. Cette technique consiste à représenter chaque entité (*i.e.*, nœud) par une forme géométrique (*e.g.*, cercle, carré) et les relations (*i.e.*, arêtes) par des traits connectant les formes. L'objectif est alors de trouver un *positionnement* optimal (*i.e.*, une position pour chaque nœud et/ou arête) qui met en valeur la structure du graphe. Un même graphe peut avoir différents positionnements optimaux mettant en valeur différents critères esthétiques, comme illustré dans la Figure III.1.

Comme présenté en Section I.2, les techniques d'Apprentissage Profond (DL pour « Deep Learning ») ont récemment connu un essor grâce à leurs excellentes performances dans plusieurs domaines d'application (*e.g.*, Traitement d'Images [4], Traitement du Langage [6]). Le principal avantage de ces modèles d'apprentissage est leur capacité à apprendre par eux-mêmes à extraire les caractéristiques importantes pour résoudre une tâche. Un second avantage de ces méthodes est leur court temps d'exécution. Au prix d'un entraînement certes coûteux, la prédiction d'un modèle de DL est le résultat d'une transformation polynomiale de leur entrée, la complexité du polynôme étant définie par les paramètres (*i.e.*, poids) du modèle. Pour la plupart des architectures de modèles DL, ces opérations sont d'autant plus rapides qu'elles sont exécutées en parallèle sur des processeurs graphiques (GPU pour « Graphic Processing Unit ») [7]. La démocratisation de ces techniques s'est aussi évidemment étendue aux tâches sur les graphes. De nouvelles architectures ont été conçues pour traiter cette structure de données : les Réseaux Neuronaux de Graphes (GNN pour « Graph Neural Networks ») [122, 123]. Jusqu'alors, la mise en œuvre de ces architectures était principalement dédiée aux tâches de classification de graphes, nœuds ou arêtes [133–135].

Ce chapitre est consacré à la présentation de nos travaux *(DNN)*^{2} [25] (pour « Deep Neural Network for drawiNg Networks ») ainsi que sa version étendue DL4GD [63], publiés respectivement dans la conférence *Graph Drawing and Network Visualization 2021* et la revue *IEEE Transactions on Visualisation and Computer Graphics*. Le code source de *(DNN)*^{2} est en accès libre en ligne [64]. La principale contribution de ces travaux est une nouvelle méthode de dessin de graphes tirant partie de techniques d'apprentissage profond. Cette nouvelle méthode, *(DNN)*^{2}, s'inspire de la manière dont les Réseaux de Neurones Convolutifs (CNN pour « Convolutional Neural Network ») traitent les images. L'idée est de tirer parti de l'efficacité d'architectures éprouvées dans d'autres domaines en les adaptant à notre contexte (*i.e.*, graphes) et notre tâche (*i.e.*, dessin de graphes). Les CNN ayant été définis pour traiter des images, des ajustements

sont nécessaires pour les entraîner. Ces changements, détaillés dans la Section III.3, incluent par exemple le remplacement des convolutions classiques par des convolutions de graphe. L'utilisation d'architecture de CNN donne au modèle une capacité à apprendre par lui-même à extraire les caractéristiques essentielles du graphe en entrée pour résoudre sa tâche. Une fois extraites, ces caractéristiques sont passées au travers de couches denses (totalement connectées) pour effectuer une régression vers le nombre de dimensions désirées. Par conception, *(DNN)*^{2} est donc capable de résoudre la tâche générique d' Incorporation de Nœud (*Node Embedding* en anglais) en n'importe quelle dimension, déterminée au moment de l'instanciation de l'architecture, avant l'entraînement. Dans ce travail, nous nous concentrons sur la projection des nœuds en 2 dimensions (2D) et interprétons le résultat comme le dessin du graphe sur un plan euclidien orthonormé.

La nouveauté de *(DNN)*^{2} est son apprentissage *non-supervisé*. Cette approche permet d'apporter une solution aux limitations des approches supervisées (voir Section III.2). L'efficacité de la méthode est étudiée au cours de plusieurs expériences évaluant les performances de nos modèles en fonction de différents critères et paramètres. L'objectif de ces évaluations n'est pas de démontrer la supériorité des performances d'une méthode par rapport à une autre. Ainsi, l'implémentation de *(DNN)*^{2} faite ici n'utilise pas nécessairement des hyper-paramètres menant à des résultats optimaux, mais isole plutôt certains d'entre eux afin d'étudier l'impact de leur variation sur les performances du modèle. La nécessité de ce type d'évaluation résulte de la complexité du paramétrage de l'apprentissage des réseaux de neurones profonds. *(DNN)*^{2} étant une nouvelle approche dans le domaine, l'étude générale de l'impact des principaux hyper-paramètres est nécessaire pour cerner leurs effets concrets sur les performances du modèle avant d'effectuer des mises au point pour optimiser ces performances. Plus précisément, nous étudions les effets de l'hétérogénéité du jeu d'entraînement sur les performances du modèle et ses capacités à généraliser sa stratégie aux données qu'il n'a jamais vues. Ce paramètre est d'autant plus critique qu'il n'existe pas de base de données de graphes suffisamment générique dédiée à l'apprentissage de réseaux de neurones pour le dessin de graphes quelconques. Chaque concepteur doit alors nécessairement utiliser les données à sa disposition, voire générer son propre jeu de données, ce qui pose d'évidentes questions sur la généralisabilité des modèles. Nous étudions également l'intérêt du transfert d'apprentissage dans notre approche, c'est-à-dire l'entraînement d'un réseau sur des graphes génériques (aléatoires) avant de le spécialiser avec un entraînement sur une famille spécifique de graphes. Ce transfert d'apprentissage est une des alternatives à l'initialisation aléatoire des paramètres du modèle. Enfin, nous comparons *(DNN)*^{2} avec d'autres méthodes de l'état de l'art pour voir à quel niveau de performance se situe notre modèle dans la littérature.

Dans ce travail, les graphes sont considérés connexes, non-orientés et sans pondération. De manière classique, les graphes non-connexes peuvent être gérés en traitant individuellement chaque composante connexe. Bien que nous ne considérions pas les graphes orientés et pondérés, la conception de *(DNN)*^{2} permet techniquement de les gérer (voir Section III.3.4).

Le reste de ce chapitre est organisé comme suit. La Section III.2 rappelle les travaux principaux de la littérature autour du dessin de graphes et des applications des techniques d'apprentissage profond sur les graphes. La Section III.3 décrit la méthode *(DNN)*^{2}, tandis que la Section III.4 en présente les implémentations et évaluations. Enfin, la Section III.5 discute certains choix de conception ainsi que les limitations de la méthode, et la Section III.6 fait la synthèse du travail réalisé et propose des pistes d'amélioration pour des travaux futurs.

Notation

Soit $G = (V,E)$ un graphe où $V = \{v_0, v_1, ..., v_{N-1}\}$, est son ensemble de $N = |V|$ nœuds et $E \subseteq V \times V$ son ensemble d'arêtes. Un dessin (ou positionnement, plongement) du graphe est défini par un vecteur $X \in \mathbb{R}^{N \times D}$ où X_i est la projection du noeud v_i en D dimensions (ici, $D = 2$). La distance euclidienne (resp. plus court chemin, *i.e.*, la distance *théorique*) dans l'espace projeté (resp. dans l'espace théorique du graphe) entre deux noeuds v_i et v_j est notée $||X_i - X_j||$ (resp. δ_{ij}).

III.2 Rappel de l'État de l'Art

Dans cette section, nous rappelons brièvement les travaux de la littérature sur le dessin de graphes par représentation nœud-lien, l'application d'apprentissage profond sur des graphes et plus particulièrement l'apprentissage pour le dessin de graphe. Pour plus de détails sur ces techniques, le lecteur peut se reporter à la Section II.2.

Historiquement, les algorithmes de Dessin de Graphe de la littérature peuvent être répartis dans trois grandes familles. La première est celle des algorithmes *dédiés* aux propriétés des graphes (*e.g.*, planaires [100], arbres [103]). Ces algorithmes présument que le graphe en entrée possède certaines propriétés et tirent parti de celles-ci pour produire un dessin. La deuxième famille d'algorithme regroupe les modèles de force [59, 106]. Les arêtes sont modélisées comme des ressorts entre des nœuds ayant une masse. Ces ressorts sont initialement trop ou pas assez tendus en fonction du positionnement initial des nœuds et des forces d'attraction/répulsion induites par leur masse. L'objectif de ces modèles est alors d'optimiser le positionnement des nœuds de manière à relaxer la tension des ressorts, communément appelée *Stress*. Enfin, la dernière famille comprend les algorithmes de réduction de dimensions [60, 111]. Originellement conçues pour plonger des données tabulaires en haute dimension dans un espace plus restreint, ces techniques ont été adaptées au contexte des graphes. De manière assez naturelle, un graphe peut être représenté de différentes manières sous forme de table (*e.g.*, matrice d'adjacence, de distances, décomposition spectrale) de telle sorte que chaque ligne de la matrice (*i.e.*, les caractéristiques de chaque nœud) peut être plongée en deux dimensions. Plus récemment, une nouvelle famille d'algorithmes a émergé : le dessin de graphes par Apprentissage Automatique (ML pour « Machine Learning ») [37–41]. Plus génériques que les modèles d'apprentissage profonds, ces algorithmes mettent en œuvre des techniques d'apprentissage statistique basées sur des caractéristiques du graphe définies par l'expert, dans le but d'optimiser une fonction de coût. Ces fonctions de coût peuvent ou non être définies pour optimiser les critères esthétiques standard en dessin de graphe [40, 41]. Pour plus de détails sur ces critères, le lecteur peut se reporter à la Section II.2.1.

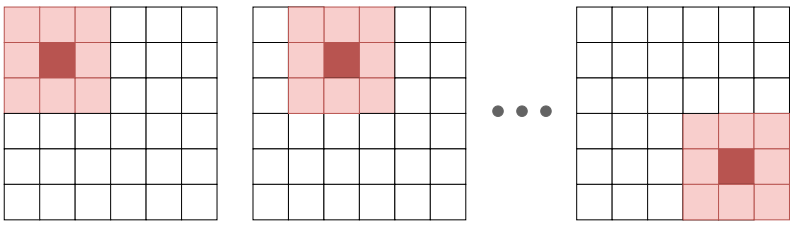
L'apprentissage profond a déjà été appliqué aux graphes, notamment pour résoudre des tâches de classification de graphe, nœud, ou arête [133–135]. Dans ce but, certaines techniques sont adaptées ou conçues pour améliorer les capacités d'apprentissage des modèles sur des données structurées en graphe. Par exemple, différentes techniques [116, 118, 119] sont capables de créer un plongement numérique initial pour chaque nœud. Ces plongements permettent techniquement à un modèle de réseau de neurones de traiter la donnée en transformant l'information discrète en vecteurs numériques, et sont optimisés pour favoriser l'extraction de caractéristiques des modèles. Différentes couches de Réseau Convolutif pour Graphes (GCN pour « Graph Convolutional Network ») [123–128] ont aussi été conçues ou adaptées pour efficacement traiter les données structurées en graphe dans les réseaux de neurones profonds.

Enfin, plusieurs techniques de Dessin de Graphe par réseau de neurones existent désormais.

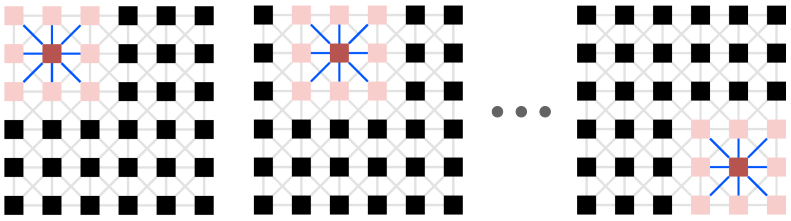
GraphTSNE [44] propose d'utiliser des réseaux de neurones peu profonds pour dessiner des graphes, avec pour originalité d'entraîner une instance du modèle spécifique à chaque graphe à dessiner. GND [43] utilise la tâche de dessin de graphe pour comparer l'efficacité de différentes couches de GCN. DeepDrawing [42] est la première technique de dessin de graphes par réseau de neurones profond. Sa principale limitation est d'être supervisée, ce qui nécessite de pré-calculer des dessins qui servent ensuite de vérité terrain. Superviser l'entraînement est en réalité peu satisfaisant car la génération des dessins vérités terrain est coûteuse et peut biaiser l'apprentissage. En effet, le modèle n'apprend pas seulement à reproduire les qualités des dessins de référence, mais aussi leurs défauts. Enfin, DeepGD [26] et son extension SmartGD [142] sont les méthodes les plus proches de *(DNN)²* [25]. SmartGD est un réseau antagoniste génératif tandis que DeepGD et *(DNN)²* sont des réseaux convolutifs. La différence entre DeepGD et *(DNN)²* réside dans la définition des noyaux d'apprentissage des convolutions. En particulier, DeepGD inclue une couche d'apprentissage supplémentaire sur les arêtes du graphe dans chaque convolution.

III.3 Méthode *(DNN)²*

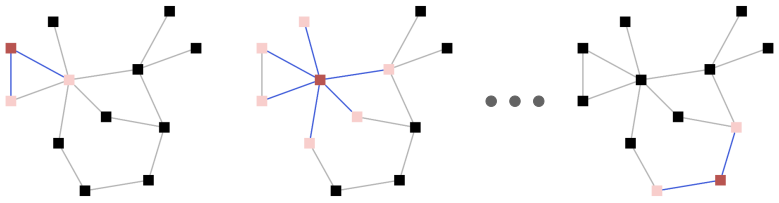
Cette section présente et détaille la conception de la méthode *(DNN)²* (pour « Deep Neural Network for drawiNg Networks »).



(a) Convolution standard sur une image



(b) Vision orientée graphe d'une convolution standard sur une image



(c) Convolution de graphe

FIGURE III.2 : Illustration d'une (a) convolution standard sur une image ; (b) convolution standard sur une image avec une vision orientée graphe (les pixels sont des nœuds, les nœuds sont connectés si les pixels qui leur correspondent sont adjacents dans la grille) ; et (c) une convolution standard sur un graphe.

III.3.1 Architecture

Comme précédemment évoqué, les CNN ont montré de grandes capacités pour résoudre efficacement différentes tâches dans des domaines divers tels que le Traitement d'Image [4]. Par conception, ces réseaux convoluent les pixels de l'image avec leurs voisins (voir Figure III.2a). Vu ainsi, leur adaptation au contexte des graphes peut être triviale : les pixels peuvent être considérés comme des *nœuds* connectés par une *arête* s'ils sont adjacents dans l'image, comme illustré dans la Figure III.2b. En donnant une structure de données encodant l'adjacence au réseau, il est possible de lui spécifier finement l'information d'adjacence entre les nœuds. Il devient alors possible de calculer une convolution de graphe, comme présenté dans la Figure III.2c. L'idée de *(DNN)*^{2} est de tirer parti des architectures de CNN largement éprouvées pour traiter des graphes et produire leur dessin. Adapter ces architectures existantes est un moyen de pouvoir s'appuyer sur les connaissances et le recul de la communauté pour concevoir notre méthodologie en limitant les biais d'implémentation. L'adaptation de ces CNN est effectuée en remplaçant les couches de convolution par des couches de convolution de graphe [123, 139].

Dans nos expérimentations, nous proposons d'utiliser *ResNet50* [138] comme squelette d'architecture de CNN. Dans cette architecture, les convolutions sont organisées en groupes appelés *blocks résiduels*, eux-mêmes rassemblés en *piles* de *blocks*. La séquence de *piles* forme la partie d'*extraction de caractéristiques* de l'architecture. Un block résiduel (voir Figure III.3) est un concept constitué de deux branches : une *principale* dans laquelle les données en entrée sont convoluées ; et une branche *raccourcie* où elles ne le sont (pratiquement) pas. A la fin d'un block, les deux branches sont agrégées, classiquement avec une addition. Ces deux branches

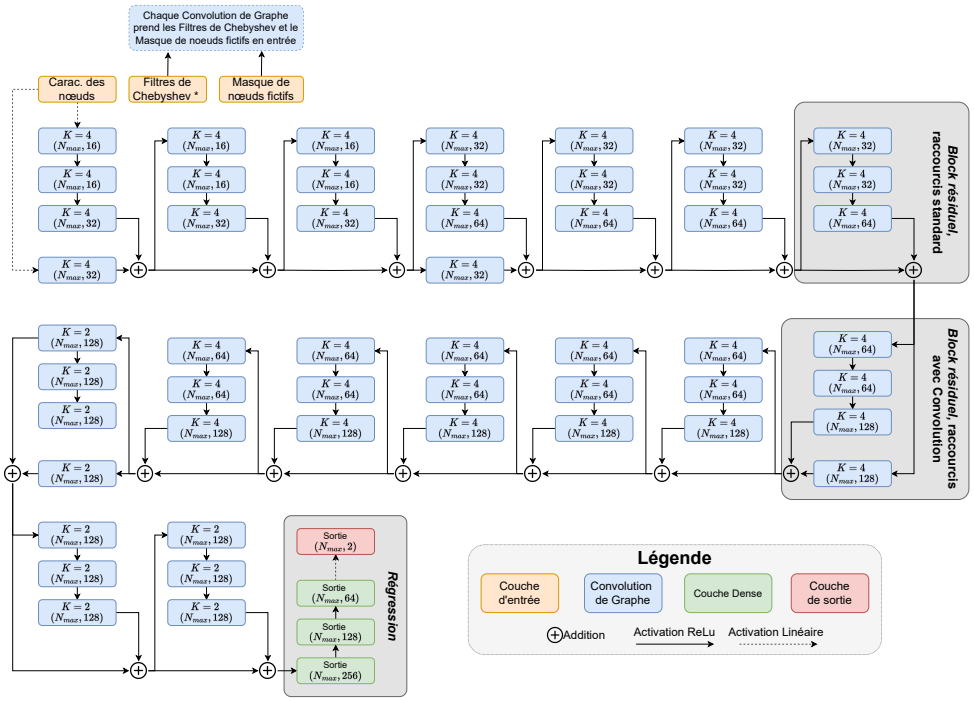


FIGURE III.3 : Schéma de l'architecture de *(DNN)*^{2} basée sur ResNet50 [138]. Seulement deux des seize blocks résiduels sont détourés. Les filtres de Chebyshev sont fournis jusqu'à l'ordre $K = 4$ à toutes les convolutions sauf les dix dernières (*i.e.*, les trois derniers blocks résiduels) où ils ne sont seulement fournis que jusqu'à l'ordre $K = 2$. Le nombre maximum de nœuds pendant l'entraînement est fixé à $N_{max} = 128$. A la fin de chaque block résiduel, les caractéristiques des nœuds sont normalisées et le masque des nœuds fictifs est appliqué.

parallèles permettent au modèle de raisonner sur différents niveaux d'abstraction de la donnée en même temps. La branche *raccourci* permet également au gradient de remonter plus facilement, ce qui amenuise les problèmes de disparition du gradient [193, 194].

Une fois les caractéristiques des nœuds extraites par cette partie de l'architecture, elles passent au travers d'une série de couches denses qui vont régresser ces caractéristiques vers l'espace cible (ici 2D). Une couche *dense* est dite totalement connectée (*fully-connected* en anglais). En outre, chaque neurone d'une couche dense L^i est connecté à *tous* les neurones de la couche L^{i-1}, résultant en une matrice de poids à apprendre de taille $|L^{i-1}| \times |L^i|$ où $|L^i|$ est le nombre de neurones dans la couche L^i. Dans notre modèle de régression, chaque couche dense applique sa matrice de poids aux caractéristiques de tous les nœuds. Ainsi, chaque couche dense peut être considérée comme une traduction de l'espace des caractéristiques des nœuds en entrée vers un espace cible (en plus basse dimension). La matrice des poids de la couche est alors une sorte de dictionnaire appliquée indépendamment à tous les nœuds pour réduire la dimension de ses caractéristiques. Ce choix de conception est principalement guidé par la nécessité de préserver l'identité de chaque nœud tout au long de l'architecture. Chaque nœud est décrit par un vecteur de caractéristiques tout au long de l'architecture. Si chaque couche peut modifier les caractéristiques d'un nœud, voire les dimensions de ces caractéristiques ; un nœud est toujours représenté par son vecteur. Ceci est illustré dans la Figure III.3 qui montre que la forme du vecteur en sortie de chaque couche est $(N_{max} \times F)$ où F est le nombre (variable) de caractéristiques décrivant les nœuds, tandis que N_{max} reste fixe sur toutes les couches. La dimension N_{max} représente le nombre de nœuds pendant l'entraînement et sera détaillée peu après. Un des avantages de cette conception est qu'elle permet facilement de gérer des graphes de taille variable. Un des défauts notoires des réseaux de neurones profonds est leur inflexibilité sur le format des données en entrée. Toute l'architecture du réseau se basant sur une taille spécifique d'entrée pour calculer en cascade la taille des matrices de poids nécessaires à l'apprentissage du modèle, gérer des entrées de tailles variables se fait la plupart du temps à partir de déformation de la donnée en entrée pour la mettre au format attendu par le réseau. En faisant le choix de cette conception préservant un vecteur pour chaque nœud tout au long de l'architecture, il devient plus facile d'utiliser notre modèle pour des tailles de graphes variables. Toutefois, cela ralentit également l'entraînement du modèle. Aussi, nous n'utilisons pas cette possibilité pour l'entraînement du modèle, mais bien pour ses prédictions une fois entraîné.

III.3.2 Convolutions de Graphes Spectrales

Comme décrit dans la Section III.3.1, notre méthode adapte des architectures standard de CNN aux graphes en remplaçant les convolutions par des convolutions de graphes. Cependant, il existe plusieurs types de convolutions de graphes (voir Section II.2.2). Dans ce travail, nous utilisons les convolutions de graphes basées sur les arêtes : chaque nœud est décrit par un vecteur de caractéristiques et est convolué avec ses voisins suivant une structure de données fournie au réseau et encodant la topologie du graphe. Ce type de convolution rejoint la philosophie des Réseaux de Neurones par Transfert de Message [132] (MPNN pour « Message Passing Neural Networks »), les MPNN étant une famille générique de réseaux décrivant une méthode d'apprentissage basée sur l'agrégation d'information entre les entités d'une donnée (ici, les nœuds). L'idée de cette méthode est que même avec une structure topologique très simple (*e.g.*, n'encodant que le voisinage direct des éléments), l'information de chaque élément se propage à tous les autres, avec un poids dépendant de sa position au sein de la topologie de la donnée. Par exemple, dans la Figure III.4, bien que les nœuds A et C ne soient pas directement connectés, l'information contenue dans le nœud C (*i.e.*, du *rouge*) finit par atteindre le nœud A,

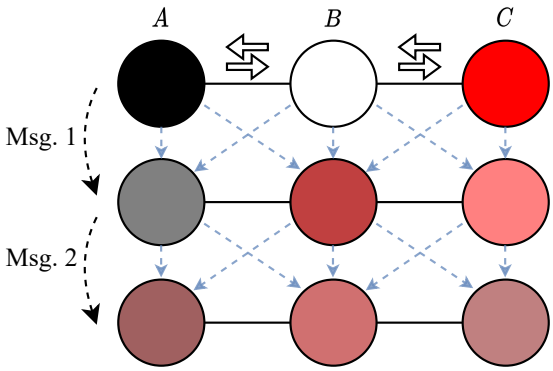


FIGURE III.4 : Illustration du concept de Transfert de Message [132] où les flèches bleues symbolisent un transfert de caractéristiques (ici la couleur). Dans cet exemple, les nœuds A et C ne sont pas directement connectés, et la topologie utilisée (*i.e.*, les arêtes noires) n'indique pas de transfert d'information entre eux. Cependant, en répétant l'opération plusieurs fois, l'information du nœud C (*i.e.*, du rouge) finit par atteindre le nœud A, et inversement.

et inversement.

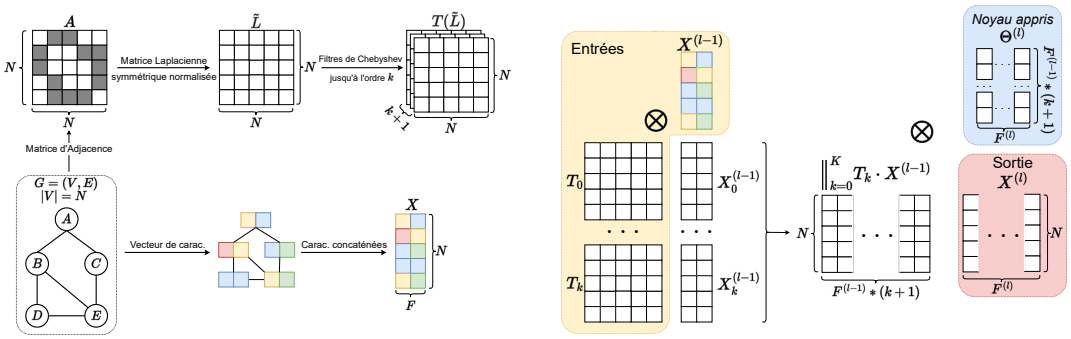
Dans la plupart des cas, une convolution de graphe implémentant le concept des MPNN suit une forme générique. Une couche de convolution l transforme un vecteur de caractéristiques $X_i^{(l-1)}$ représentant un nœud en un nouveau vecteur $X_i^{(l)}$ suivant la formule :

$$X_i^{(l)} = X_i^{(l-1)} \cdot \Theta^{(l)} + \sum_{j \in V} W_{ij} \cdot X_j^{(l-1)} \cdot \Theta'^{(l)} \tag{III.1}$$

où $X_i^{(l)}$ est le vecteur de caractéristiques du nœud v_i à la couche l. $W \in \mathbb{R}^{N \times N}$ est la structure de données qui pondère l'agrégation d'informations entre les nœuds et est typiquement générée à partir de la topologie du graphe. Θ et Θ' sont des matrices de poids qui seront appris pendant l'entraînement dans le but d'optimiser une fonction de coût. Ce sont ces paramètres dont les valeurs sont affinées pendant l'entraînement pour satisfaire la fonction de coût. La majorité des convolutions de graphe suivent cette formule, et leur originalité diffère sur la définition des matrices de poids Θ et de la structure de donnée W encodant les pondérations de l'agrégations d'information entre les nœuds. La matrice d'adjacence ou celle de distance inversée peuvent par exemple être utilisées comme structure de donnée topologique W. Ainsi, le premier terme de la formule permet à la couche d'apprendre à extraire de la connaissance des caractéristiques d'un nœud *courant*, tandis que le deuxième nœud permet d'extraire de la connaissance de ses voisins. L'agrégation des deux permet alors de mettre à jour les caractéristiques de chaque nœud en fonction des siennes et de celles de ses voisins.

(DNN)^2 suit l'Équation III.1 en implémentant les *convolutions de graphes spectrales* définies par Kipf et Welling [123] et illustrées dans la Figure III.5. Dans cette convolution, les caractéristiques des nœuds X, sont considérées comme le *signal* du graphe, et sont convoluées en fonction du *spectre* du graphe (W dans l'Équation III.1). Le *spectre* du graphe est défini comme la décomposition en vecteurs propres de la matrice Laplacienne symétrique normalisée $\tilde{L}$:

$$\begin{aligned} L &= D - A, \\ L^{sym} &= I_N - D^{-\frac{1}{2}} L D^{-\frac{1}{2}}, \\ \tilde{L} &= \frac{2}{\lambda_{max}} L^{sym} - I_N \end{aligned} \tag{III.2}$$



(a) Mise en forme des données en entrée (b) Opération de convolution de graphe

FIGURE III.5 : Illustration de (a) la génération des inputs nécessaires au calcul d'une convolution de graphe spectrale à partir du graphe lui-même ; et (b) l'opération de convolution de graphe. Dans (b), le tenseur $X^{(l-1)}$ en entrée de la première couche est constitué des caractéristiques initiales générées pour chaque nœud. Pour les convolutions suivantes, le tenseur en entrée d'une couche est celui résultant de la précédente couche.

où D et A sont respectivement les matrices de degré et d'adjacence du graphe, I_N est la matrice identité, et toutes les trois sont de taille $N \times N$. λ_{max} est la plus grande valeur propre de L^{sym}.

La décomposition en vecteurs propres de $\tilde{L}$ (*i.e.*, le *spectre* du graphe) étant coûteuse à calculer, elle est approximée via les polynômes de Chebyshev [195] jusqu'à un ordre K :

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x), \forall k \geq 2 \end{aligned} \tag{III.3}$$

où la matrice d'adjacence A est utilisée comme paramètre en entrée x. L'ordre K peut ainsi être considéré comme la taille du noyau de la convolution. Enfin, une couche de convolution de graphe spectrale H^l peut être définie par :

$$H^l(X) = \left(\Big\Vert_{k=0}^{K^l} T_k\left(\tilde{L}\right) \cdot X \right) \cdot \Theta^l \tag{III.4}$$

où le symbole $||$ représente l'opération de *concaténation* sur les tenseurs $T_k(\tilde{L}) \cdot X$, et $X \in \mathbb{R}^{N \times F^{l-1}}$ est le tenseur des caractéristiques des nœuds (*i.e.*, le signal du graphe) où chaque nœud est représenté par F^{l-1} caractéristiques. $T_k\left(\tilde{L}\right)$ note la pondération de l'agrégation d'information entre les nœuds dans la convolution (*i.e.*, filtres de Chebyshev, le spectre du graphe) et correspond à W dans l'Équation III.1. Enfin, $\Theta^l \in \mathbb{R}^{(F^{(l-1)} * (K+1)) \times F^{(l)}}$ est la matrice de poids de la couche l qui est apprise pendant l'entraînement, où $F^{(l)}$ est le nombre de caractéristiques désirées pour chaque nœud en sortie de la convolution.

III.3.3 Fonction de Coût Non-supervisée

Si des méthodes telles que DeepDrawing [42] utilisent des dessins de graphes résultant d'algorithmes de la littérature comme vérité terrain, nous pensons que cette approche n'est pas satisfaisante. En effet, elle présente plusieurs défauts, les principaux étant (i) le coût de génération des dessins de vérité terrain, et (ii) le fait que le modèle apprenne à copier un algorithme,

y compris ses défauts. De plus, il n'est pas tout à fait juste de considérer qu'il existe une *vérité terrain* pour un dessin de graphe car des dessins du même graphe optimisant différents critères esthétiques (*e.g.*, préservation du voisinage, préservation des distances) peuvent tous être considérés corrects (voir Figure III.1).

L'entraînement de *(DNN)²* est *non-supervisé* : la qualité des dessins produits ne dépend pas d'un dessin de graphe connu. *(DNN)²* est entraîné pour optimiser une fonction dérivable capable d'évaluer la qualité du dessin produit en fonction de la topologie du graphe. Par exemple, Ahmed *et al.* [40] a proposé des expressions dérivables des principales mesures de qualité esthétique de la littérature (voir Section II.2.1) pour les optimiser avec une descente de gradient.

Dans *(DNN)²*, nous proposons d'optimiser la divergence de Kullback-Leibler comme proposé par Kruiger *et al.* [37] dans leur méthode *tsNET*. Cette fonction de coût est une adaptation aux graphes de l'algorithme t-SNE [35] et est définie par :

$$\begin{aligned} C_{comp} &= \frac{1}{2N}\sum_i ||X_i||^2, \\ C_{rep} &= \frac{1}{2N^2}\sum_{i,j} \log\left(||X_i - X_j|| + \varepsilon_r\right), \\ C_{tsNET} &= \lambda_{KL} C_{KL} + \lambda_c C_{comp} - \lambda_r C_{rep} \end{aligned} \tag{III.5}$$

où C_{KL} (III.6) est le terme principal (discuté ci-après) de la fonction de coût et est relié à la topologie du graphe. Le second terme, C_{comp} est un facteur de *compression* qui minimise l'échelle du dessin (*i.e.*, le domaine de valeur des positions des nœuds). Ce composant est également connu pour accélérer la convergence de t-SNE. Le dernier terme, C_{rep}, est une *répulsion* qui contre-balance les effets indésirables du terme précédent. $(\lambda_{KL}, \lambda_c, \lambda_r)$ sont des poids utilisés pour ajuster le comportement de la fonction de coût pendant l'optimisation. $\varepsilon_r = \frac{1}{20}$ est une constante de régularisation.

Dans les premières expérimentations de *(DNN)²* [25], nous avons reproduis le processus d'optimisation en deux étapes défini dans tsNET [37]. Ce processus vise à accélérer la convergence du modèle en donnant un poids fort au terme de compression dans une première étape $(\lambda_{KL} = 1, \lambda_c = 1.2, \lambda_r = 0)$ avant de contre-balancer ses effets indésirables avec une deuxième étape $(\lambda_{KL} = 1, \lambda_c = 0.01, \lambda_r = 0.6)$. Cependant, nous avons observé que l'introduction des deux termes C_{comp} et C_{rep} a tendance à significativement occulter l'importance du terme principal lié à la topologie du graphe C_{KL}. En outre, il devient difficile de mettre en relation des variations de performances du modèle avec des propriétés du graphe en entrée, tant le poids des termes secondaires est important. La principale motivation pour intégrer ces termes étant l'accélération de la convergence de l'algorithme, nous avons décidé de nous en passer pour l'optimisation de *(DNN)²*. En effet, tsNET effectuant une optimisation pour chaque dessin de graphe à produire, il est nécessaire d'optimiser son temps d'exécution afin qu'il soit compétitif envers les méthodes de l'état de l'art. Ce problème n'a pas lieu d'être avec *(DNN)²* qui est une approche par réseau de neurones profond car nous pouvons accorder un temps d'entraînement plus long au modèle sans que cela ait une influence sur son temps d'exécution une fois mis en production. Ainsi, *(DNN)²* optimise uniquement C_{KL} (*i.e.*, fonction de coût de l'Équation III.5 avec $\lambda_{KL} = 1, \lambda_c = 0, \lambda_r = 0$).

C_{KL} mesure mesure la (dis)similarité entre deux ensembles de probabilités avec la divergence de Kullback-Leibler (KL) :

$$C_{KL} = \sum_{i,j \in V, i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{III.6}$$

où p_{ij} et q_{ij} sont des ensembles de probabilités que chaque paire de nœuds $(v_i, v_j) \in V^2$ soit connectée dans le graphe, en fonction de leur position dans celui-ci (*i.e.*, zone dense ou non) et

de la distance qui les sépare. La différence entre les deux ensembles de probabilités est l'espace depuis lequel les zones de densité du graphe et les distances sont définies. p_{ij} est calculé à partir de l'espace topologique du graphe, alors que q_{ij} est défini à partir de l'espace projeté en 2D. Évaluer C_{KL} revient donc à étudier si « *la probabilité que les nœuds soient voisins en fonction de leur emplacement* ***dans le graphe*** » est similaire à « *la probabilité que les nœuds soient voisins en fonction de leur emplacement* ***dans le dessin du graphe*** ». Si tel est le cas, le dessin est une bonne représentation du graphe. La définition formelle de p_{ij} et q_{ij} est reportée à la Section III.5.1 dans laquelle nous discutons les défauts manifestes de l'optimisation de C_{KL} observés durant l'évaluation de la Section III.4.

III.3.4 Entrées du modèle

Pendant l'entraînement, une taille fixe (en nombre de nœuds) N_{max} est fixée. Les tenseurs en entrée du modèle sont donc composés de caractéristiques provenant des N nœuds réels, et de $N_{max} - N$ nœuds fictifs. Ce fonctionnement permet que tous les tenseurs qui composent le modèle aient une dimension commune fixe, ce qui accélère grandement l'apprentissage de modèle. Ce type de remplissage est également courant dans certains domaines tels que le Traitement du Langage Naturel où les phrases données aux réseaux de neurones sont complétées avec des mots fictifs pour toujours être composées du même nombre de mots [196]. La définition de cette taille maximale n'est effectuée que pendant la phase d'apprentissage du modèle et n'est pas nécessaire en dehors. En revanche, cette phase nécessite l'intégration d'un masque des nœuds réels/fictifs dont l'objectif est d'éviter que les caractéristiques des nœuds fictifs soient prises en compte et *biaisent* l'apprentissage. Ce masque peut être défini par $Mask \in \{0;1\}^N$ où $Mask_i = 0$ si v_i est un nœud fictif, 1 sinon. En le multipliant avec le tenseur qui contient les caractéristiques des nœuds après chaque block résiduel, les caractéristiques des nœuds fictifs sont systématiquement remises à 0. Pour une meilleure lisibilité, la suite de cette section continue d'utiliser N pour signifier la taille d'une dimension encodant le *nombre de nœuds* dans un tenseur ; les nœuds fictifs (et N_{max}) étant uniquement mis en œuvre pendant l'entraînement.

Puisque *(DNN)²* implémente les convolutions de graphe *spectrales*, la seconde entrée du modèle est le *spectre* du graphe, approximé par les filtres de Chebyshev jusqu'à l'ordre K comme présenté dans la Section III.3.2. Cette structure encode l'information de la topologie du graphe qui guide les transferts de caractéristiques entre les nœuds dans les convolutions. Ainsi, la structure est fournie à toutes les couches de convolutions et n'est jamais modifiée pendant l'entraînement.

La dernière entrée du modèle est le *signal*, le tenseur qui contient les caractéristiques initiales des nœuds. En outre, ce sont les informations encodées dans ce tenseur qui sont convoluées en fonction du *spectre* du graphe tout au long des couches composant le modèle, puis régressées vers un plongement 2D (voir Figure III.3). Il existe plusieurs méthodes de génération de caractéristiques initiales dédiées aux GNN [116, 118, 119] (voir Section II.2.2). Cependant, la plupart des techniques de dessin de graphe (voir Section II.2.1) se basent uniquement sur la topologie du graphe pour produire son plongement (et incluent parfois une part d'aléatoire). Dans notre approche, nous préférons rester proches du comportement de ces algorithmes de dessin et n'utilisons pas les techniques de génération automatique de caractéristiques. De plus, cela nous permet (i) de réduire le temps de pré-traitement, et (ii) de rester en adéquation avec notre volonté de proposer une implémentation simple de notre méthodologie, même si ces choix de conception ne sont pas optimaux. En outre, il serait difficile, voire impossible de relier des changements de performances de nos modèles aux caractéristiques initiales des nœuds si celles-ci étaient calculées par des méthodes automatiques dont les résultats ne sont évidemment pas

interprétables par un humain. Ainsi, les caractéristiques initiales des nœuds que nous proposons d'utiliser sont : un identifiant unique pour chaque nœud (de 0 à $N-1$ pour chaque graphe) et une valeur aléatoire. L'intuition est que l'utilisation de l'identifiant unique peut aider le réseau à dissocier des nœuds ayant des voisinages similaires dans la topologie du graphe, ce qui conduirait à les positionner au même endroit. Ce comportement est d'ailleurs connu et est une limitation des convolutions de graphes [25, 124, 125] (voir Section II.2.2). La valeur aléatoire des caractéristiques initiales permet aussi d'aider le modèle à différentier les nœuds, et il est assez commun pour les algorithmes de dessin de graphe d'introduire une composante aléatoire (*e.g.*, GEM [106]) pour éviter certains blocages dans des minima locaux lors de l'optimisation. Les caractéristiques initiales des nœuds sont donc définies par un tenseur $X \in \mathbb{R}^{N\times 2}$. Par ailleurs, il est également commun (*e.g.*, tsNET* [37], DeepGD [26], SmartGD [142]) pour des algorithmes de dessin de graphe à base d'optimisation d'initialiser les positions des nœuds avec le résultat d'un algorithme rapide mais pas nécessairement efficace de la littérature (*e.g.*, PivotMDS [60]). C'est une alternative au positionnement initial aléatoire des nœuds qui permet d'harmoniser le comportement de l'algorithme entre plusieurs exécutions. Nous proposons donc également une variante de notre algorithme prenant les positions 2D résultantes de l'algorithme PivotMDS [60] en entrée (*i.e.*, $X \in \mathbb{R}^{N\times 4}$).

Enfin, chaque entrée est normalisée en fonction des valeurs du jeu de données d'apprentissage pour leur attribuer la même importance et améliorer la généralisabilité du modèle en atténuant les problématiques de domaine de valeur des caractéristiques. Cela permet par exemple d'éviter que les identifiants de nœuds (dans les caractéristiques initiales des nœuds) aient une valeur trop élevée (*i.e.*, un poids trop important) simplement parce qu'un graphe a un nombre de nœuds plus élevé que d'autres.

Dans ce travail, nous considérons que les graphes ne sont pas pondérés, bien qu'il serait assez simple de les gérer avec l'architecture proposée par *(DNN)*^{2}. Par exemple, un poids sur les nœuds pourrait être encodé dans le tenseur de caractéristiques initiales des nœuds. Une pondération sur les arêtes pourrait également être encodée dans la structure de topologie fournie aux convolutions et utilisée pour l'agrégation d'information entre les nœuds. Ici, nous utilisons le *spectre* du graphe (*i.e.*, les filtres de Chebyshev) qui est difficilement paramétrable. Toutefois, d'autres structures telles qu'une matrice d'adjacence pondérée, pourraient être expérimentées. En fonction du choix de cette structure de topologie, les graphes orientés pourraient également être gérés dans notre approche. Ces idées ne sont pas expérimentées dans ce travail et constituent des pistes de travaux futurs.

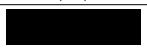
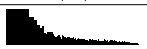
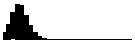
III.4 Évaluation

Dans cette section, nous présentons le protocole et les résultats de l'évaluation de *(DNN)*^{2}.

III.4.1 Jeux de Données

Certaines propriétés des jeux de données utilisés pour l'évaluation sont reportées dans la Table III.1. Tous ces jeux sont scindés en au moins deux parties à des fins d'apprentissage : *entraînement* et *validation*. La partie d'*entraînement* sert naturellement à entraîner le modèle et les données de *validation* sont utilisées à la fin de chaque époque d'apprentissage pour contrôler certains aspects des performances du modèle, par exemple pour mesurer la généralisabilité et prévenir le sur-entraînement. Tous les jeux de données sauf HeR et HoR sont à nouveau scindés en une partie *test*. Cette partie est constituée de données que le modèle n'a jamais vues pendant

TABLE III.1 : Distribution du nombre de nœuds $|V|$, nombre d'arêtes $|E|$, densité ρ_2 et distances dans les jeux de données présentées dans la Section III.4.1. La distribution de distances représente la somme de toutes les distances entre toutes les paires de noeuds dans les graphes d'un jeu de données. Les graphiques reportant $|E|$ ont une échelle logarithmique sur l'axe Y. Tous les graphiques d'une colonne ont le même intervalle sur l'axe X.

Jeu de données	|V|	|E|	Densités	Distances
HoR				
HeR				
Arbres				
Grilles				
Pseudo-Grilles				
Planaires				
Commu.				
Rome				

l'entraînement et est donc utilisée pour l'évaluer. Tous les graphes de ces jeux de données sont connexes, non-orientés, non-pondérés et sont constitués d'entre 2 et 128 nœuds.

Tous les jeux de données suivants ont été générés pour cette étude à l'exception de **Rome**.

HoR [25] est un jeu de données aléatoires homogènes (HoR pour « Homogeneous Random ») constitué de 1 000 instances de graphes connexes aléatoires pour chaque taille de graphe entre 2 et 128 nœuds. La densité de chaque graphe est aléatoirement choisie et le nombre d'arêtes correspondant est calculé avec la formule de densité $\rho_1 = \frac{2|E|}{|V|*(|V|-1)}$. Des graphes isomorphes peuvent être générés, notamment avec les nombres de nœuds faibles. En Apprentissage Profond, il est préférable d'éviter d'avoir plusieurs fois la même donnée dans un jeu d'entraînement car cela augmente son poids et risque de biaiser le modèle. Dans notre cas, nous considérons que la présence de graphes isomorphes, notamment les petits graphes, n'est pas un problème. Au contraire, puisque tous les graphes peuvent être décomposés en sous-graphes de plus petite taille, nous pensons que donner plus de poids aux petits graphes aide le modèle à traiter les plus grands.

HeR (pour « Heterogeneous Random ») est un jeu de données aléatoires hétérogènes qui, comme HoR, est constitué de 1 000 graphes de chaque taille entre 2 et 128 nœuds. Il comprend deux principales différences avec HoR. La première est la fonction de densité utilisée pour associer un nombre d'arêtes à chaque graphe $\rho_2 = \sqrt{\frac{2|E|}{|V|*(|V|-1)}}$. La deuxième réside dans le fait que le tirage aléatoire des densités est contrôlé de telle sorte que celles-ci soient plus uniformément distribuées au travers du jeu de données (voir la colonne *Densités* des deux premières lignes de la Table III.1). Cela permet d'avoir une plus grande diversité de topologies de graphes et mène à un jeu de données plus hétérogène.

Arbres est un jeu de 15 245 graphes ayant une structure arborescente. La moitié de ces arbres sont larges avec un degré maximum élevé (voir Figure III.8b troisième ligne), tandis que ceux de l'autre moitié sont profonds (voir Figure III.8b quatrième ligne). La profondeur des arbre est paramétrée dans le générateur de Tulip [197].

Grilles est l'énumération de toutes les grilles de degré 4, 6 et 8 avec $N \leq 128$, $largeur \geq 2$ and $hauteur \geq 2$.

Pseudo-Grilles est un jeu de graphes généré avec le générateur *Grid Approximation* de Tulip [197] en utilisant les paramètres par défaut. Dans ce générateur, les nœuds sont projetés

dans un espace 2D, et les nœuds proches (*i.e.*, plus proche qu'un seuil manuellement défini) sont connectés. Cet espace 2D n'est qu'un intermédiaire nécessaire pour la génération de la topologie du graphe.

Planaires est constitué de 19 050 graphes planaires aléatoires générés par Tulip [197] avec 2 à 128 nœuds. Tulip générant des graphes planaires de densité maximum (*i.e.*, avec $3N-6$ arêtes), nous enlevons aléatoirement des arêtes dans chaque graphe jusqu'à ce qu'il en reste entre $N-1$ et $3N-6$ tout en s'assurant de préserver la connectivité du graphe.

Commu. est un jeu de 18 161 graphes de communautés suivant une répartition gaussienne. Ils sont générés avec l'algorithme de Brander *et al.* [198] avec les paramètres par défaut.

Rome est un jeu de données de 11 531 graphes mis à disposition par la communauté du symposium *Graph Drawing*[1]. La topologie de ces graphes est variable et ils ne sont d'aucune famille spécifique. Ils sont généralement peu denses (voir la haute concentration de densités faibles dans Table III.1). Ce jeu de données est largement établi dans la communauté de Dessin de Graphe pour l'évaluation de nouveaux algorithmes. Aussi, nous l'utiliserons pour l'évaluation de notre méthode *(DNN)*[2] (voir Section III.4.4 et III.4.7).

III.4.2 Paramètres d'Entraînement

Cette section décrit les hyper-paramètres utilisés pour entraîner les modèles étudiés dans les prochaines sections d'évaluation.

En ce qui concerne le modèle, le premier choix de conception est l'utilisation de ResNet50 [138] comme squelette (voir Section III.3.1). Nous avons sélectionné cette architecture car elle est largement étudiée et obtient de bonnes performances dans plusieurs domaines. La taille maximale des graphes N_{max} est fixée à 128, une valeur légèrement plus élevée que le plus grand graphe dans le jeu de données **Rome**. D'une part, la taille N_{max} doit être suffisamment grande pour permettre aux graphes générés d'avoir une large variété de topologies. D'autre part, N_{max} doit rester raisonnablement petit car l'évaluation de la fonction de coût nécessite le calcul de la matrice de distances de chaque graphe. $N_{max} = 128$ permet d'obtenir un bon compromis entre ces deux critères. La taille des noyaux de convolutions de graphe (*i.e.*, l'ordre des filtres de Chebyshev) est fixée à $K = 4$ pour toutes les couches de convolution, sauf les 10 dernières. Cette taille de noyau peut être interprétée comme la distance de voisinage jusqu'à laquelle chaque nœud sera convolué. Par exemple, avec $K = 4$, les caractéristiques des nœuds sont mises à jour à partir de leurs voisins jusqu'à une distance 4. Pour les 10 dernières couches de convolution (*i.e.*, celles des derniers blocks résiduels) nous fixons $K = 2$. Ce changement résulte de l'intuition que réduire l'ordre des filtres de Chebyshev dans les convolutions va donner plus d'importance aux caractéristiques du voisinage proche des nœuds. Il est donc effectué dans les dernières couches de la partie d'extraction de caractéristiques du réseau afin que la conception de son architecture rajoute du poids à la préservation du voisinage direct des nœuds.

Pour l'entraînement sur les petits jeux de données (*e.g.*, Grilles, voir Section III.4.1), les parties *entraînement* et *validation* sont suréchantillonnées par répétition jusqu'à obtenir une taille d'au moins 8800 et 1760 éléments respectivement. Cela signifie que le modèle verra certains graphes plusieurs fois pendant l'entraînement. Chaque jeu de données concerné est entièrement répété afin de préserver la distribution de ses caractéristiques (voir Table III.1). Cette augmentation de données est mise en place pour permettre l'entraînement de nos modèles qui, étant des réseaux profonds, nécessitent une certaine quantité de données pour converger. Cependant, la considération même d'une partie de *validation* pour ces jeux de données pourrait être remise en cause. En effet, y a-t-il un réel bénéfice à empêcher le sur-entraînement d'un

[1] Graphes Rome : `http://www.graphdrawing.org/data.html`

modèle si son jeu d'entraînement est constitué de l'énumération de tous les graphes possibles ayant une certaine topologie ? Dans ce genre de cas peu commun, la notion de *capacité du modèle à généraliser ses performances à des données qu'il n'a jamais vues* n'a pas de sens puisqu'il est entraîné sur l'ensemble exhaustif des données du contexte défini. Nous n'étudions pas cette question dans ce document mais il serait intéressant d'effectuer des expérimentations complémentaires sur ce cas particulier dans des travaux futurs.

En ce qui concerne les hyper-paramètres de l'entraînement du modèle, nous avons utilisé l'optimiseur Adam [77] avec un taux d'apprentissage initial de 10^{-3}. La durée maximale des entraînements est fixée à 200 époques, avec un arrêt anticipé si le score de la fonction de coût sur le jeu de validation ne s'améliore pas pendant 20 époques consécutives. Ce choix a été réalisé après avoir observé que laisser les modèles s'entraîner pendant 1000 époques n'offrait pas d'amélioration significative des performances, par rapport aux coûts engendrés par la durée d'entraînement. En pratique, la plupart des instances de *(DNN)²* ont convergé en moins de 100 époques. Les entraînements sur les jeux HeR et HoR ont été effectués avec une taille de lots (*i.e.*, *batch size*) de 400 sur un cluster CPU avec 20 exécuteurs. Le temps d'entraînement était de quelques heures pour chacun de ces deux jeux de données. Pour tous les autres jeux, la taille de lot est fixée à 32 et les entraînements ont été conduits sur un GPU NVIDIA 2080 Super Max-Q (Mobile), durant environ une heure chacun. Pour ces jeux de données, cela montre que le coût d'entraînement de *(DNN)²* pour obtenir des performances satisfaisantes n'est pas très élevé. Le temps de prédiction du modèle hors entraînement n'est pas mesuré car il n'est pas significatif comparé au temps nécessaire pour pré-traiter les graphes (*e.g.*, chargement mémoire, calcul des filtres de Chebyshev).

III.4.3 Protocole et Métriques d'Évaluation

Cette section décrit les métriques de qualité utilisées pour comparer les techniques de dessin de graphe. Celles notées avec une * ont été inversées par rapport à leur implémentation courante dans la littérature afin que toutes soient lisibles tel qu'un score plus faible signifie une meilleure qualité (*i.e.*, *lower is better*).

Aspect Ratio* [40] est défini par le plus mauvais ratio entre la largeur et la hauteur du dessin parmi une série de rotations de tout le plongement. Le meilleur score est obtenu lorsque la boîte englobante du dessin produit est carrée.

$$aspect\_ratio = 1 - \min_{\theta} \frac{\min(w_\theta, h_\theta)}{\max(w_\theta, h_\theta)} \tag{III.7}$$

où $\theta \in \{\frac{2\pi t}{T}, t \in [0, 1, ..., T-1]\}$ est une des T rotations du dessin et w_θ et h_θ sont respectivement la largeur et la hauteur du dessin après la rotation d'angle θ.

Angular Resolution* [40] est le ratio de l'angle minimum formé par deux arêtes autour d'un nœud avec l'angle formé par les arêtes autour du nœud de degré maximal si elles étaient réparties uniformément autour de lui.

$$angular\_resolution = 1 - \frac{\min_{(i,j),(j,k)\in E} \theta_{ijk}}{\theta_G} \tag{III.8}$$

où $\theta_G = \frac{2\pi}{d_{max}}$ est l'angle optimal que devraient former les arêtes, d_{max} étant le degré maximum du graphe.

Edge Crossings Number [40, 56] est le nombre de croisement d'arêtes dans le dessin.

$$crossing\_number = \sum_{e1,e2\in E, e1\neq e2} \mathbb{1}\{crossing(e1, e2)\} \tag{III.9}$$

où *crossing* est une fonction géométrique vérifiant si deux segments s'intersectent.

Cluster Overlap [199] correspond à la somme des distances normalisée entre les nœuds qui sont à une distance inférieure à r mais *pas* dans le même cluster de nœuds. La métrique nécessite la spécification du rayon seuil r et d'un algorithme de groupement en clusters. Ici, nous utilisons $r = 0.2$ et MCL [200] comme algorithme de groupement dont l'implémentation, tirée de Tulip [197], est déterministe.

$$cluster\_overlap = \sum_{i \in V} \frac{\sum_{u \in U_i} (1 - ||X_u - X_i||) * \mathbb{1}\{MCL_i \neq MCL_u\}}{\sum_{u \in U_i} (1 - ||X_u - X_i||)} \tag{III.10}$$

où MCL_i est le numéro du cluster du nœud i, U_i est l'ensemble des nœuds v_u tels que $||X_i - X_u|| < r$ avec $r = 0.2$.

Neighborhood Preservation* [37] est la somme, pour chaque nœud, de la taille de l'intersection de son k-voisinage U dans l'espace topologique avec ses $|U|$ plus proches voisins dans le dessin, sur l'union de ces deux ensembles. Conceptuellement, cette métrique est la plus proche de l'objectif d'optimisation de la fonction de coût optimisée par *(DNN)²* dans cet article, C_{KL} (voir Section III.3.3).

$$neighborhood\_preservation = 1 - \frac{1}{|V|} \sum_{i \in V} \frac{|U_i \cap Y_i|}{|U_i \cup Y_i|} \tag{III.11}$$

où U_i est le k-voisinage du nœud v_i dans la topologie du graphe, et Y_i est l'ensemble des $|U_i|$ plus proches voisins de v_i dans le dessin.

Stress [40, 59] est une mesure largement établie dans la communauté de Dessin de Graphe. Elle mesure à quel point une distribution de distances dans un espace projeté est représentative d'une distribution de distances dans un autre espace, généralement en plus hautes dimensions. Ici, le *Stress* mesure à quel point les distances entre les nœuds dans le dessin sont représentatives de la longueur des plus courts chemins dans la topologie du graphe. Le score du Stress est ensuite normalisé pour que des graphes de tailles différentes aient le même poids dans l'évaluation.

$$Stress(X) = \frac{1}{N} \sum_{i,j} w_{ij} (||X_i - X_j|| - \delta_{ij})^2 \tag{III.12}$$

où w_{ij} est une pondération communément fixée à $w_{ij} = \delta_{ij}^{-2}$.

Validation statistique. Étant donné que nous évaluons la performance de techniques de dessin de graphe sur des milliers de données, nos évaluations doivent être soutenues par une validation statistique. Pour ce faire, nous suivons le protocole défini par Purchase [184]. Pour vérifier que les différences de performances entre les différentes techniques comparées soient significatives, nous conduisons d'abord un test de Friedman [201], avec un seuil d'acceptabilité de $\alpha = 0.05$. Ce test est une version non-paramétrique (*i.e.*, pour des données qui ne suivent pas une distribution normale) d'analyse de variance (ANOVA). Si ce test passe, c'est que les variations de performances entre les groupes (ici, les techniques de dessins) sont significatives, et des tests plus approfondis peuvent être conduits. Si ce test ne passe pas, aucune interprétation ne devrait être faite des résultats car il n'est pas certain que les différences entre les méthodes ne soient pas dues au hasard. Dans le cas où le test de Friedman passe avec succès, des tests *post-hoc* entre toutes les paires de méthodes sont conduits pour vérifier la significativité de la différence entre leurs performances. Pour cela, nous utilisons le test de Nemenyi [202] avec un seuil d'acceptabilité de $\alpha = 0.05$. Pour chaque paire de méthodes évaluée, si le test passe, nous pouvons affirmer que la méthode avec le score le plus faible est significativement plus efficace.

III.4.4 Hétérogénéité du Jeu de Données et Optimisation de KL

L'équilibre de la distribution dans le jeu d'entraînement est crucial avec les réseaux de neurones profonds. Si la distribution est trop hétérogène, le modèle peut échouer à apprendre une stratégie qui résout la tâche car celle-ci nécessiterait un ajustement à chaque échantillon. À l'inverse, si un jeu de données est trop homogène, le modèle peut rapidement devenir sur-entraîné et être incapable de généraliser ses résultats à des données sortant de la distribution homogène sur laquelle il a appris à résoudre la tâche. Il est donc important de garder un équilibre dans l'hétérogénéité du jeu d'entraînement afin de tirer le meilleur parti d'un modèle de réseau de neurones profond.

Cette section présente la comparaison des performances d'instances de *(DNN)*^{2} entraînées sur deux jeux de données de graphes aléatoires dont la principale différence est l'hétérogénéité des topologies des graphes qui les composent : HeR et HoR (voir Section III.4.1). Les modèles sont ensuite évalués sur le jeu de données **Rome** afin de comparer leurs performances et capacités de généraliser à d'autres structures de graphes. Pour rappel, la Table III.1 présente les différentes distributions de certains paramètres des graphes composant ces jeux de données. Nous profitons de cette première comparaison pour évaluer une autre hypothèse : optimiser la fonction de coût C_{KL} (voir Équation III.6) est plus efficace que C_{tsNET} (voir Équation III.5). Comme mentionné dans la Section III.3.3, nous avons observé pendant nos expérimentations que l'introduction des termes de compression et de répulsion dans C_{tsNET} (accélérant la convergence) avait tendance à occulter l'optimisation de la divergence de Kullback-Leibler (KL). Nous souhaitons donc vérifier si cette hypothèse est juste. Pour optimiser C_{tsNET}, nous reproduisons le protocole de l'implémentation originelle de tsNET [37] en entraînant le modèle en deux phases avec les différents poids $(\lambda_{KL}, \lambda_c, \lambda_r)$. L'optimisation de C_{KL} se fait en un seul entraînement avec les poids $(\lambda_{KL} = 1, \lambda_c = 0, \lambda_r = 0)$. Les performances de *(DNN)*^{2} optimisant C_{KL} ne sont reportées que sur le jeu de données HeR puisque nous verrons qu'entraîner le modèle sur ce jeu de données mène à de meilleurs résultats.

Nous comparons les performances de ces modèles sur le jeu de données **Rome** qui est communément utilisé pour l'évaluation dans la communauté de Dessin de Graphe. Il nous sert ici de base commune pour l'évaluation de la généralisabilité de modèles entraînés sur des jeux de données différents. Ces performances sont présentées dans la Figure III.6. Les tests statistiques (voir Section III.4.3) sont passés avec succès : les différences entre les modèles sont significatives sur toutes les métriques de qualités.

Tout d'abord, nous comparons *(DNN)*^{2} entraîné sur HeR ou HoR pour optimiser C_{tsNET} (respectivement HeR-C_{tsNET} et HoR-C_{tsNET}, premières et deuxièmes barres dans la Figure III.6) afin d'évaluer l'impact de l'hétérogénéité du jeu d'entraînement sur les performances du modèle. Nous pouvons observer que le modèle HoR-C_{tsNET} est uniquement meilleur (avec une faible marge) sur les métriques *Aspect Ratio* et *Angular Resolution*. À l'inverse, HeR-C_{tsNET} est meilleur (avec une forte marge) sur toutes les autres métriques, notamment celles mesurant la préservation des distances (*e.g.*, *Stress*) et du voisinage (*e.g.*, *Cluster overlap*, *Neighborhood preservation*). Ces résultats confirment que l'entraînement sur le jeu de données le plus hétérogène permet au modèle d'obtenir de meilleures performances et de mieux généraliser à de nouvelles structures de graphe.

Les deux dernières barres de chaque graphique représentent les modèles de *(DNN)*^{2} entraînés sur HeR pour optimiser C_{tsNET} ou C_{KL}. À l'exception de *Angular Resolution*, le modèle optimisant C_{KL} (*i.e.*, HeR-C_{KL}) obtient systématiquement de meilleures performances. Par définition, nous pouvons être certain que HeR-C_{KL} optimise la préservation de voisinage définie dans la méthode de visualisation initiale [35], alors qu'il est plus difficile de savoir quel terme a le

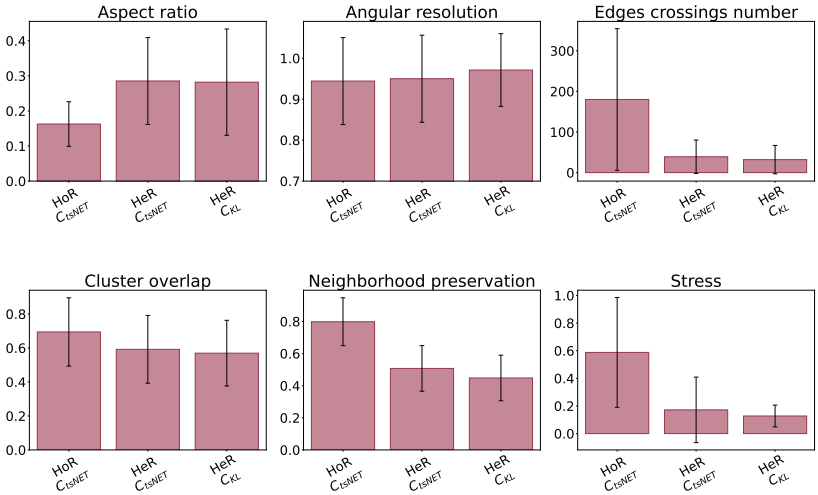


FIGURE III.6 : Performances d'instances de *(DNN)*^{2} entraînées sur le jeu de données HeR ou HoR et évaluées sur Rome. Les deux premiers modèles optimisent C_{tsNET} (III.5) tandis que le dernier optimise C_{KL} (III.6). Tous les tests statistiques sont passés, ce qui implique que les différences entre les trois méthodes sont toutes significatives.

plus influé les performances du modèle HeR-C_{tsNET}. Nous pouvons en conclure qu'optimiser directement C_{KL} est plus cohérent avec un réseau de neurones profond puisque (i) nous n'avons pas besoin de chercher à accélérer la convergence du modèle, (ii) le modèle obtient de meilleurs résultats, et (iii) nous savons exactement ce qu'il cherche à optimiser. Ce dernier point est probablement le plus important : une des grandes problématiques des réseaux de neurones profonds est de comprendre leur fonctionnement et ce qui a conduit à leur prise de décision afin de pouvoir faire confiance à leurs prédictions (voir Section I.3). Il est donc préférable, pour l'interprétation des résultats et les futures re-conceptions, que le modèle optimise une fonction de coût dont le comportement est le moins ambigu possible.

Dans notre première version de *(DNN)*^{2} [25], nous avons conclu qu'affiner les modèles à partir de poids pré-entraînés améliorait leur performance. Cette conclusion était tirée d'une comparaison sur le jeu HoR. Étant donné que nous venons d'observer que les performances étaient meilleures sur un jeu de données plus hétérogène (*i.e.*, HeR), nous allons re-évaluer cette hypothèse dans la suite de cette étude. Dans le même temps, nous pensons que n'importe quel modèle pré-entraîné peut tirer des bénéfices d'un affinage sur un jeu de données spécifique. En effet, les graphes de familles spécifiques (*e.g.*, grille, arbre) sont composés de structures topologiques singulières. Un modèle entraîné ou affiné sur des graphes d'une famille spécifique devrait obtenir de bonnes performances en apprenant à reconnaître et utiliser ces structures topologiques particulières. Ces deux hypothèses font écho à des approches de la littérature de dessin de graphe (voir Section II.2.1) dans lesquelles certains algorithmes génériques sont conçus pour dessiner n'importe quels graphes, alors que d'autres sont dédiés à des types de graphes possédant une ou plusieurs propriétés topologiques singulières. Ces derniers tirent parti de ces propriétés pour produire des dessins de meilleure qualité. À l'inverse, les algorithmes génériques produisent des dessins optimisant certains critères esthétiques, quelles que soient les propriétés du graphe. La prochaine section étudie donc le comportement de l'approche par réseau de neurones profond de *(DNN)*^{2} sur des familles de graphes spécifiques avec ou sans transfert d'apprentissage, et de quelle catégorie d'algorithme elle se rapproche le plus.

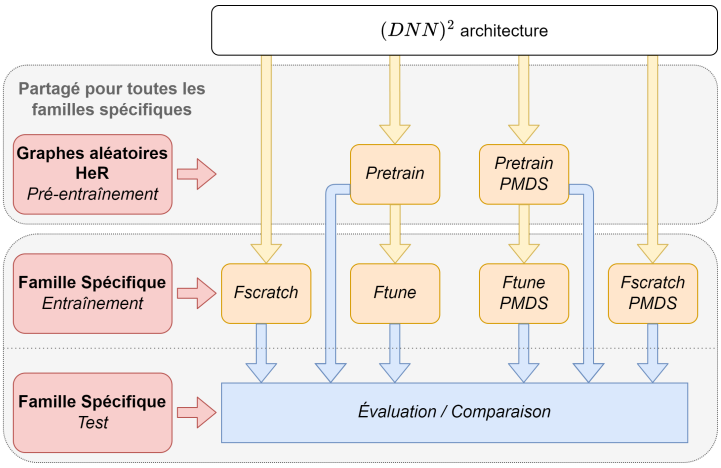


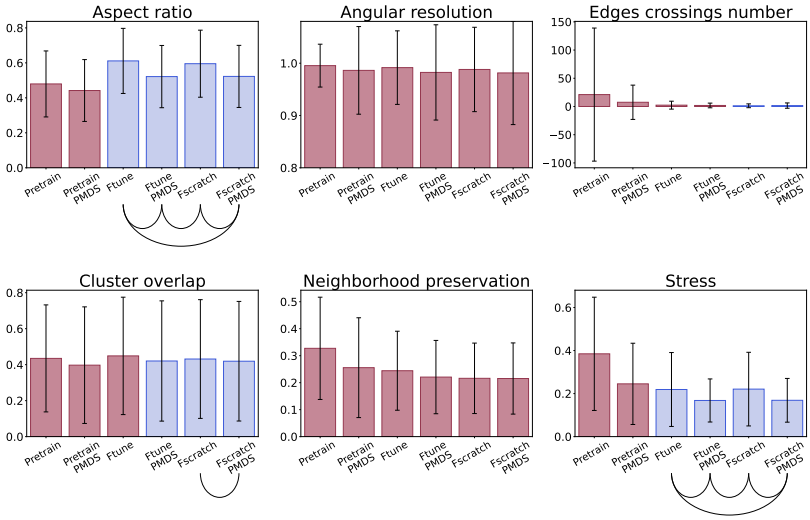
FIGURE III.7 : Récapitulatif des modèles entraînés pour l'évaluation sur des familles de graphes spécifiques. Les blocks rouges sont des jeu de données. Les blocks oranges sont des instances de modèle *(DNN)*^{2}, entraînées sur les jeux de données en ligne. Une seule instance de *Pretrain* et *Pretrain PMDS* est entraînée pour toute l'évaluation. Une instance *Ftune* et *Fscratch* est entraînée pour chaque famille spécifique.

III.4.5 Familles de Graphes Spécifiques

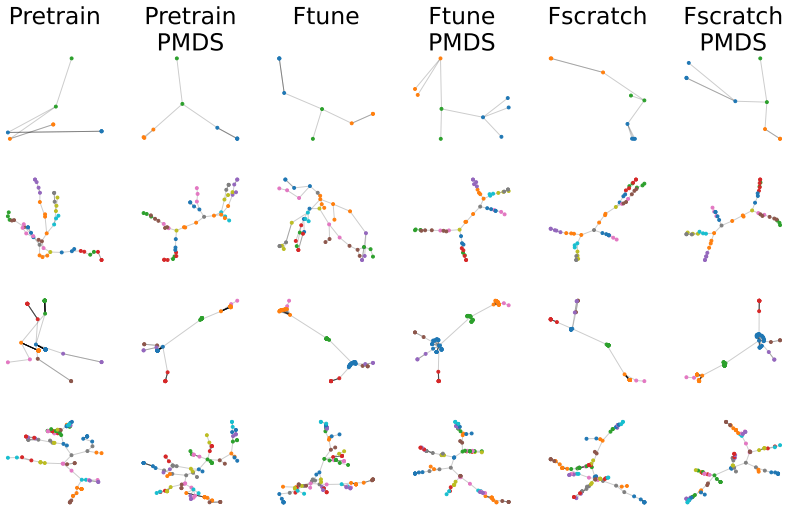
Cette section présente l'évaluation de trois types d'entraînements de *(DNN)*^{2} sur des familles de graphes spécifiques. Chaque famille est représentée par un jeu de données décrit en Section III.4.1 : Arbres, Grilles, Pseudo-Grilles, Planaires et Communautés. Ces familles de graphes sont sélectionnées parce qu'elles couvrent une large variété de topologies. Comme vu dans la Section III.4.4, l'approche par réseau de neurones profond de *(DNN)*^{2} pourrait convenir aux deux types d'approches au dessin de graphe dans la littérature : les algorithmes *dédiés* à des familles de graphes, et les *génériques*. Nous pensons que la même architecture peut être adaptable aux deux approches en fonction du jeu d'entraînement du modèle. Par exemple, nous nous attendons à ce qu'un modèle entraîné sur des grilles reconnaisse des structures topologiques (*e.g.*, régularité) de ce type de graphe et en tire parti pour produire des dessins de meilleure qualité.

Les trois approches d'apprentissage étudiées ici sont celles définies dans la première présentation de *(DNN)*^{2} [25] : entraînement à partir de poids aléatoires sur des graphes aléatoires (dit *pretraining*), à partir de poids aléatoires sur des graphes spécifiques (dit *fromscratch*), et affinage des poids appris de *pretraining* sur des graphes spécifiques (dit *finetuning*). Ces approches sont nommées *Pretrain*, *Fscratch* et *Ftune*. La Figure III.7 récapitule les instances de *(DNN)*^{2} entraînées et leur nommage. Une seule instance de *Pretrain* est entraînée pour l'évaluation, tandis qu'une instance *Ftune* et *Fscratch* est entraînée pour chaque famille.

La seconde hypothèse étudiée est le bénéfice pour *(DNN)*^{2} de connaître les positions des nœuds issues d'une autre technique de dessin. Plus généralement, c'est l'évaluation des gains pour le modèle d'être nourri avec des caractéristiques initiales (voir Section III.3.4) qui nous semblent sémantiquement utiles. Plusieurs algorithmes de dessin par optimisation de la littérature (*e.g.*, tsNET* [37]) initialisent les positions du dessin à partir de techniques rapides mais peu efficaces. Puisqu'il n'y a pas d'initialisation des positions dans une approche par réseau de neurones profond, nous tentons de reproduire l'injection de cette connaissance en l'ajoutant aux caractéristiques initiales des nœuds et souhaitons vérifier si cette étape est effectivement utile. Dans notre cas, nous ajoutons les positions 2D produites par l'algorithme PivotMDS [60]. Nommée *(DNN)*^{2*} dans notre premier travail [25], les variantes ayant ces caractéristiques additionnelles en entrée seront notées avec l'inscription *PMDS* dans ce document.



(a) Comparaison des instances Pretrain, Ftune et Fscratch avec et sans les caractéristiques initiales PMDS sur le jeu Arbres. Les graphiques présentent les moyennes et écarts types de chaque métrique. La couleur des barres et les arcs encodent la significativité des différences de performance (voir Section III.4.5.1).



(b) Exemples de dessins du jeu Arbres avec $N \approx \{10, 50, 75, 100\}$.

FIGURE III.8 : (a) Performances et (b) exemples visuels des instances de *(DNN)*^{2} Pretrain, Ftune et Fscratch sur le jeu **Arbres** avec et sans les caractéristiques initiales PMDS.

III.4.5.1 Lecture des Résultats

Pour chaque jeu de données et métrique de qualité, la performance moyenne et l'écart type des 6 instances de modèles sont présentés dans un graphique à barres. Le protocole de validation statistique défini dans la Section III.4.3 est appliqué sur les données de chaque graphique. Pour rappel, un test de Friedman [201] est d'abord conduit pour rejeter l'hypothèse nulle, vérifiant si les différences entre les techniques de dessin ne sont pas dues au hasard. Si ce premier test passe, des tests de significativité (ici, Nemenyi [202]) vérifient si la différence entre chaque paire de techniques de dessin de graphe est significative.

Initialement, chaque barre du graphique est bleue et un arc est dessiné entre les labels de

chaque techniques entre lesquelles les performances sont significativement différentes. Si une méthode est significativement différente de toutes les autres, nous retirons ses arcs incidents et colorions sa barre en rouge pour alléger la visualisation. Par exemple, dans la Figure III.8a sur la métrique *Aspect Ratio*, les modèles *Pretrain* et *Pretrain PMDS* ont des performances significativement différentes de tous les autres. À l'inverse, les performances du modèle *Ftune* sont significativement différentes de celles avec qui un arc le relie (*i.e.*, *Ftune PMDS* et *Fscratch PMDS*), et les méthodes avec une barre rouge (*i.e.*, *Pretrain* et *Pretrain PMDS*). Cela implique que la différence n'est *pas* significative entre *Ftune* et *Fscratch* sur cette métrique.

Pour illustrer les performances des six instances sur chaque famille, quatre exemples visuels de dessin (avec $N \approx \{10, 50, 75, 100\}$) provenant du jeu de données de *test* sont aussi présentés.

III.4.5.2 Arbres

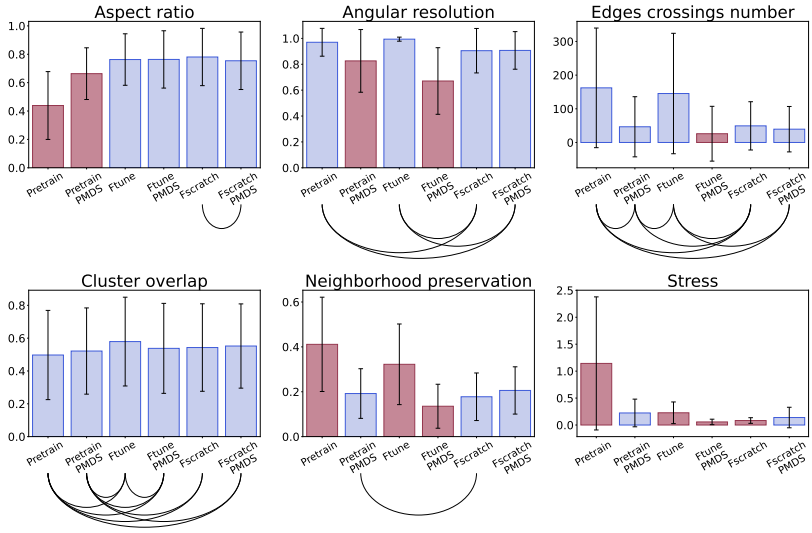
Les performances de nos instances de *(DNN)*^{2} sur le jeu de données Arbres sont présentées dans la Figure III.8a, et des exemples visuels proposés dans la Figure III.8b. Nous pouvons observer que l'apprentissage spécifique sur les graphes de ce jeu de données (*i.e.*, les instances *Ftune* et *Fscratch*) mène à de meilleures performances sur les métriques de *Stress*, *Edge Crossing Number* et *Neighborhood Preservation*, et produit un effet moindre sur *Angular Resolution* et *Cluster Overlap*. Puisqu'il n'y a que peu de différence entre les instances *Ftune* et *Fscratch*, nous pouvons en conclure qu'utiliser les connaissances issues d'un pré-entraînement sur des graphes aléatoires n'améliore pas sensiblement les performances du modèle. L'aspect visuel des graphes dans la Figure III.8b confirme qu'il est très bénéfique pour le modèle d'avoir des arbres dans son jeu d'entraînement pour apprendre à les dessiner. Nous concluons donc qu'il est nécessaire d'apprendre sur des arbres pour en produire des dessins de bonne qualité, et que les connaissances issues d'un entraînement sur des graphes aléatoires ne sont ni nécessaires, ni suffisantes.

Nous observons une amélioration systématique de la qualité des dessins obtenus sur toutes les métriques avec l'ajout des positions 2D de l'algorithme PivotMDS [60] aux caractéristiques initiales. Visuellement, les modèles avec PMDS semblent nettement plus efficaces. Les améliorations de résolution angulaire sont nettement visibles et les dessins ont tendance à moins souffrir de chevauchements de nœuds.

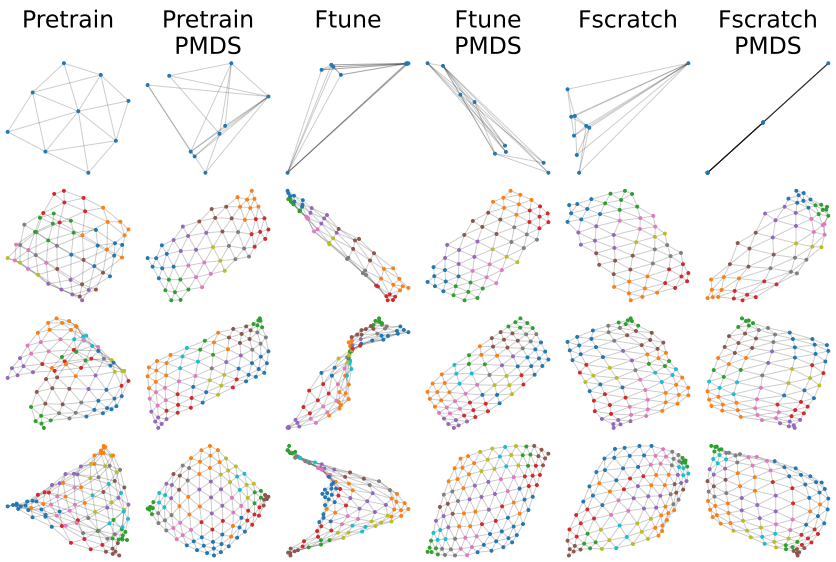
III.4.5.3 Grilles

Les performances et des exemples visuels sur le jeu **Grilles** sont reportés dans la Figure III.9a et III.9b respectivement. Entraîner le modèle sur le jeu de données spécifique améliore les performances sur les métriques *Neighborhood Preservation*, *Edge Crossing Number* et *Stress*. Dans l'ensemble, l'instance *Ftune* est moins efficace alors que *Ftune PMDS* est la meilleure sur toutes les métriques. Visuellement, *Pretrain* est la seule instance à dessiner la petite grille correctement. Cependant, elle a tendance à complètement replier les grandes grilles sur elles-mêmes si elle n'a pas les caractéristiques initiales PMDS. Les instances entraînées sur le jeu Grilles produisent des dessins satisfaisants, sauf *Ftune*. Nous observons que *Fscratch* dessine correctement les grilles même sans les caractéristiques initiales PMDS, à l'exception de la plus petite grille. Étant donné que les grilles ont une topologie très particulière, nous nous attendions à ce qu'un entraînement sur des graphes de cette famille améliore grandement les performances du modèle, ce qui ne semble pas être le cas (ou du moins, pas à l'ampleur escomptée).

La présence des caractéristiques initiales PMDS améliore les performances de *Ftune PMDS* mais détériore celles de *Fscratch PMDS*. En outre, tant visuellement qu'au regard des métriques



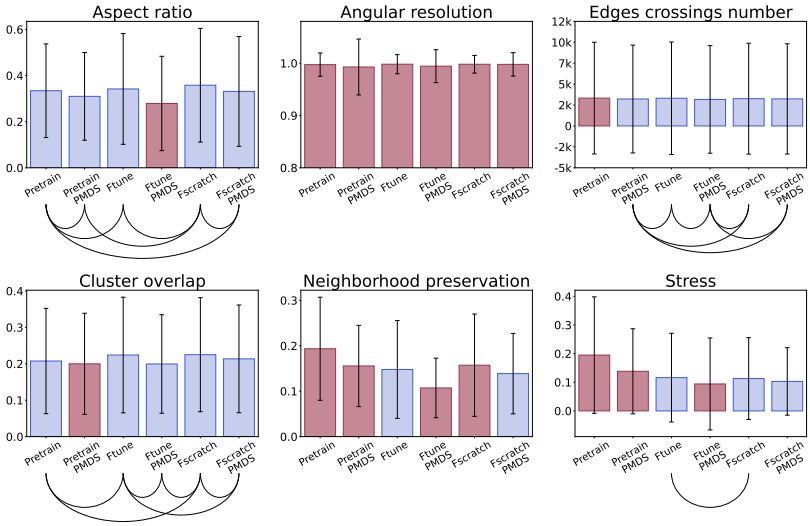
(a) Comparaison des instances Pretrain, Ftune et Fscratch avec et sans les caractéristiques initiales PMDS sur le jeu Grilles. Les graphiques présentent les moyennes et écarts types de chaque métrique. La couleur des barres et les arcs encodent la significativité des différences de performance (voir Section III.4.5.1).



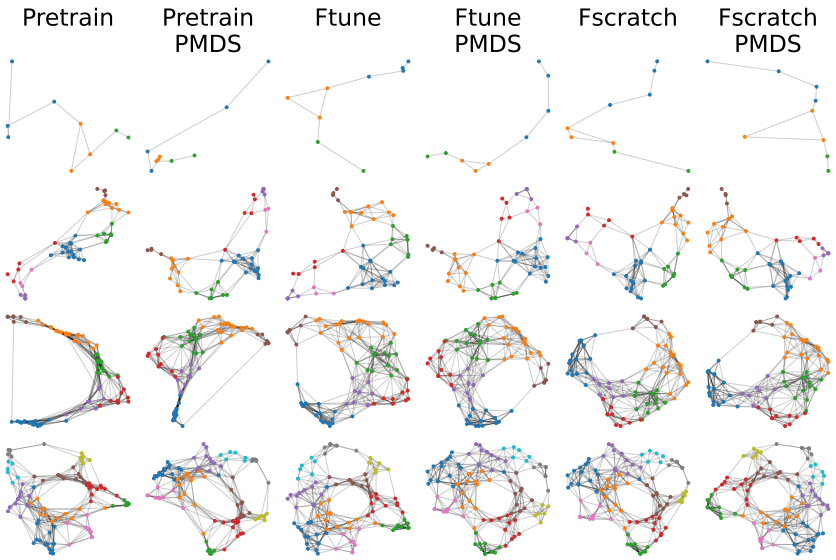
(b) Exemples de dessins du jeu Grilles avec $N \approx \{10, 50, 75, 100\}$.

FIGURE III.9 : (a) Performances et (b) exemples visuels des instances de *(DNN)*^{2} Pretrain, Ftune et Fscratch sur le jeu **Grilles** avec et sans les caractéristiques initiales PMDS.

de qualité, *Ftune PMDS* est meilleur que *Ftune* tandis que *Fscratch PMDS* est moins performant que *Fscratch*. Nous nous attendions à ce qu'ajouter les caractéristiques initiales PMDS améliore significativement les résultats étant donné que PivotMDS [60] est une technique de dessin qui s'adapte bien aux grilles. Les résultats semblent également signifier que le transfert d'apprentissage de *Pretrain PMDS* à *Ftune PMDS* aide à tirer partie des caractéristiques PMDS, alors que l'apport du transfert d'apprentissage sans ces caractéristiques (*i.e.*, de *Pretrain* à *Ftune*) influe moins sur les résultats. Comme vu précédemment, il ne semble pas y avoir de bénéfice significatif à disposer des caractéristiques PMDS ou non pour l'instance *Fscratch*. Les deux instances *Fscratch* proposent des dessins satisfaisants des grandes grilles et montrent qu'elles peuvent obtenir de meilleurs résultats que *Pretrain* dans ce cas.



(a) Comparaison des instances Pretrain, Ftune et Fscratch avec et sans les caractéristiques initiales PMDS sur le jeu Pseudo-Grilles. Les graphiques présentent les moyennes et écarts types de chaque métrique. La couleur des barres et les arcs encodent la significativité des différences de performance (voir Section III.4.5.1).

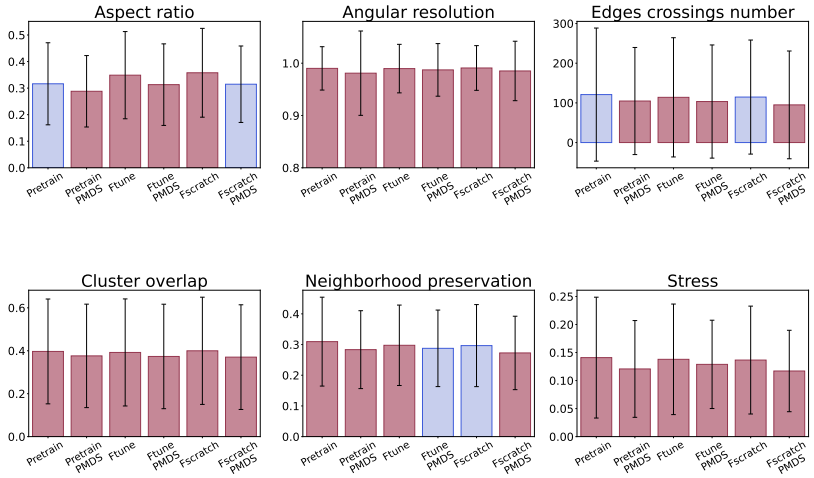


(b) Exemples de dessins du jeu Pseudo-Grilles avec $N \approx \{10, 50, 75, 100\}$.

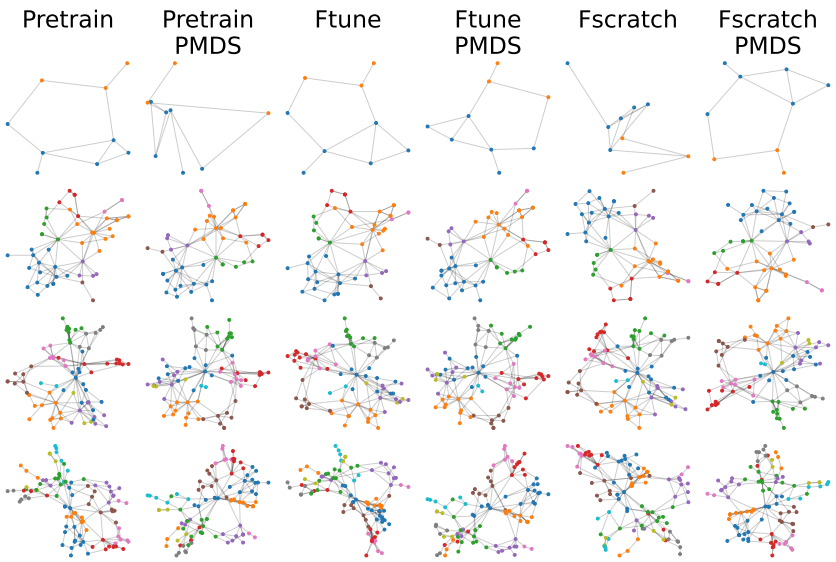
FIGURE III.10 : (a) Performances et (b) exemples visuels des instances de *(DNN)*^{2} Pretrain, Ftune et Fscratch sur le jeu **Pseudo-Grilles** avec et sans les caractéristiques initiales PMDS.

III.4.5.4 Pseudo-Grilles

Les Figures III.10a et III.10b présentent respectivement les performances et des exemples visuels des dessins produits par les modèles sur le jeu **Pseudo-Grilles**. Nous constatons que l'entraînement sur des graphes Pseudo-Grilles (*i.e.*, *Ftune* et *Fscratch*) n'affecte positivement que les métriques *Neighborhood Preservation* et *Stress*. Là encore, *Ftune PMDS* est la meilleure instance d'une manière générale tandis que *Ftune* est meilleure que *Pretrain* et *Fscratch*. Visuellement, tous les dessins produits par les instances *Ftune* et *Fscratch* avec ou sans PMDS sont satisfaisants. À l'inverse, *Pretrain* ne semble pas efficace pour dessiner ces graphes, surtout les plus grands.



(a) Comparaison des instances Pretrain, Ftune et Fscratch avec et sans les caractéristiques initiales PMDS sur le jeu Planaires. Les graphiques présentent les moyennes et écarts types de chaque métrique. La couleur des barres et les arcs encodent la significativité des différences de performance (voir Section III.4.5.1).



(b) Exemples de dessins du jeu Planaires avec $N \approx \{10, 50, 75, 100\}$.

FIGURE III.11 : (a) Performances et (b) exemples visuels des instances de *(DNN)*^{2} Pretrain, Ftune et Fscratch sur le jeu **Planaires** avec et sans les caractéristiques initiales PMDS.

Avoir les caractéristiques initiales PMDS améliore les métriques *Aspect Ratio*, *Cluster Overlap*, *Neighborhood Preservation* et *Stress*. Pourtant, le gain apporté par ces caractéristiques ne semble pas sensiblement améliorer la qualité visuelle des résultats.

III.4.5.5 Planaires

Les performances et exemples visuels sur le jeu **Planaires** sont présentés dans les Figures III.11a et III.11b. Par conception, une des propriétés principales des graphes planaires est qu'ils peuvent être dessinés sans croisement d'arêtes sur le plan ou sur une sphère. Les algorithmes de dessins dédiés aux graphes planaires doivent offrir cette garantie (*e.g.*, [100]). Étant constitué d'un réseau

de neurones, l'approche probabiliste de *(DNN)*^{2} ne permet évidemment pas d'offrir cette garantie. Cependant, sachant qu'il existe un dessin sans croisement d'arêtes de ces graphes, il est tout de même intéressant d'observer les dessins produits par les instances de *(DNN)*^{2}. Cette idée à par ailleurs été étudiée par van Wageningen et Mchedlidze [203] qui utilisent un réseau de neurones pour dessiner des graphes *quasi-planaires* avec un style de dessin planaire. Si la planarité ainsi que la propriété de dessin sans croisement sont encodées dans la topologie même du graphe, nous devrions observer des améliorations sur la métrique *Edge Crossing Number* pour les instances ayant appris sur ce jeu de données. Néanmoins, cette hypothèse semble peu vraisemblable étant donné que la fonction de coût ne quantifie que la préservation du voisinage.

Au regard des métriques de qualité, il n'y a pas de bénéfice particulier à entraîner le modèle sur les graphes du jeu Planaires. Les dessins produits par *Pretrain* sont visuellement très proches de ceux produits par *Ftune* et *Fscratch*, sinon meilleurs. Ainsi, malgré quelques modestes améliorations sur *Edge Crossing Number*, nous pouvons en conclure que le modèle n'a pas été capable d'extraire de connaissance qui améliore significativement ses performances à partir des topologies spécifiques de graphes planaires.

L'ajout des caractéristiques initiales PMDS améliore légèrement les scores des métriques de qualité, mais cette amélioration ne semble pas perceptible dans les exemples visuels proposés.

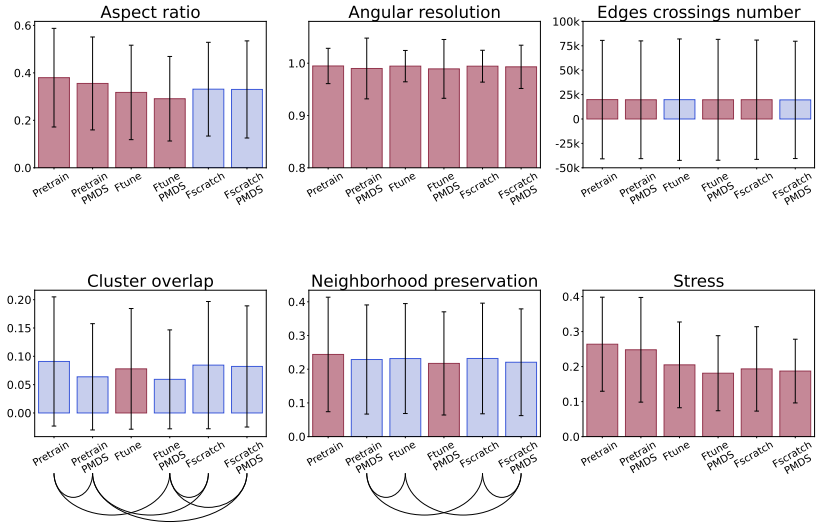
III.4.5.6 Communautées

Commu. est le dernier jeu de données que nous étudions dans cette section. Les performances et exemples visuels sont présentés dans les Figures III.12a et III.12b. Nous pouvons constater qu'entraîner les modèles sur ce jeu de données améliore *Aspect Ratio*, *Cluster Overlap* et *Stress*, mais n'a pratiquement aucun effet sur les autres métriques. Cependant, *Cluster Overlap* est une métrique très importante sur ce jeu de données constitué de graphes de communautés. Nous observons dans les exemples visuels que toutes les instances sont capables d'identifier les communautés et les regrouper dans les dessins qu'elles produisent. La principale différence entre l'instance *Pretrain* et les autres est la capacité à représenter les structures internes des clusters. Là où *Pretrain* projette pratiquement tous les nœuds d'une même communauté sur le même point (créant beaucoup de chevauchements), *Ftune* et *Fscratch* mettent mieux en valeur la structure interne de ces communautés.

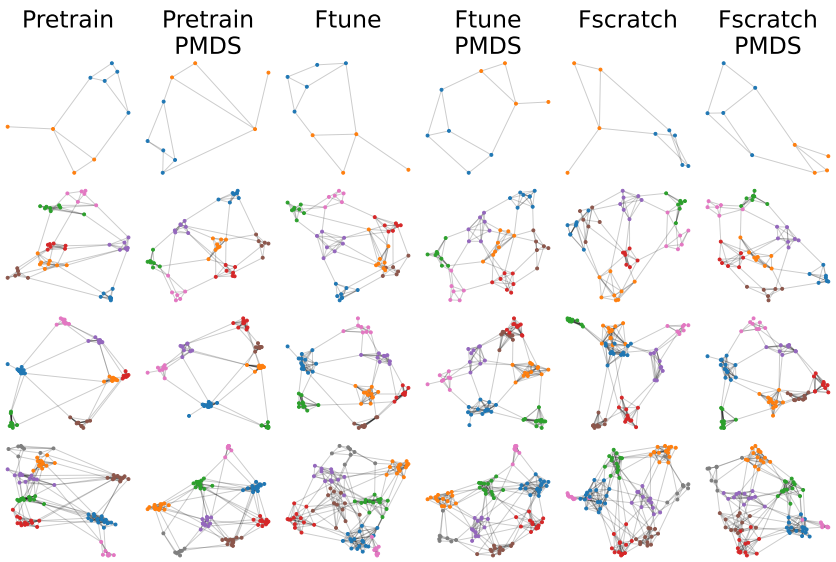
Pretrain et *Ftune* bénéficient des caractéristiques initiales PMDS supplémentaires, alors que celles-ci ont un effet bien plus léger sur les performances de *Fscratch*. Visuellement, les dessins des méthodes utilisant les caractéristiques PMDS ne semblent pas nettement meilleurs que leur homologue qui ne les utilise pas.

La Table III.2 récapitule *à titre indicatif* les performances relatives des différents modèles étudiés tout au long de cette section qui nous a permis d'étudier deux hypothèses.

La première est qu'entraîner un modèle sur une famille de graphes améliorerait la qualité de leurs dessins en apprenant à tirer parti de l'information spécifique à leur topologie. De plus, cette hypothèse nous permet d'étudier si l'approche par réseau de neurones proposée par *(DNN)*^{2} est plus adaptée à dessiner des graphes quelconques ou des graphes partageant des propriétés singulières. D'une manière générale, nous avons observé que l'entraînement sur ces familles spécifiques a amélioré les capacités des modèles à les dessiner. Sur les familles avec une topologie très particulière (*e.g.*, Arbres, Grilles), l'entraînement sur un jeu de données dédié est presque obligatoire pour obtenir des dessins satisfaisants. Sur les autres familles étudiées ici, les différences avec ou sans cet entraînement dédié sont moins prononcées. Nous souhaitons également rappeler que les instances *Pretrain* et *Pretrain PMDS* utilisées tout au long



(a) Comparaison des instances Pretrain, Ftune et Fscratch avec et sans les caractéristiques initiales PMDS sur le jeu Commu. Les graphiques présentent les moyennes et écarts types de chaque métrique. La couleur des barres et les arcs encodent la significativité des différences de performance (voir Section III.4.5.1).



(b) Exemples de dessins du jeu Commu. avec $N \approx \{10, 50, 75, 100\}$.

FIGURE III.12 : (a) Performances et (b) exemples visuels des instances de *(DNN)*^{2} Pretrain, Ftune et Fscratch sur le jeu **Commu.** avec et sans les caractéristiques initiales PMDS.

de cette étude sont les mêmes pour tous les jeux de données. Elles ont été entraînées sur des graphes aléatoires hétérogènes (*i.e.*, HeR) et évaluées dans chaque section sur un jeu de données spécifique. En outre, nous avons observé que ces deux instances obtiennent de bons résultats d'une manière générale, bien qu'en deçà de ce que peuvent atteindre les modèles entraînés sur les jeux de données spécifiques. Lorsque l'entraînement sur un jeu dédié présente un intérêt, la qualité des dessins de *Ftune* et *Fscratch* est nettement meilleure que celle de *Pretrain*.

La seconde hypothèse étudiée est que l'ajout des positions 2D résultantes de l'algorithme PivotMDS [60] améliorerait les résultats. D'une manière plus générale, cette hypothèse revient à étudier si l'ajout de caractéristiques initiales que nous pouvons considérer *sémantiquement utiles* à la tâche est vraiment bénéfique pour le modèle. Nous avons observé que tous les modèles

TABLE III.2 : Tableau récapitulatif des performances des instances de *(DNN)²* entraînées sur chaque jeu de données représentant une famille de graphes spécifiques. Les symboles indiquent *à titre indicatif* l'efficacité relative de ces instances, et sont basés sur nos appréciations des différences entre leurs métriques de qualité et exemples visuels.

	Pretrain	**Pretrain PMDS**	**Ftune**	**Ftune PMDS**	**Fscratch**	**Fscratch PMDS**
Arbres	✗	✗	~	✓	~	✓
Grilles	✗	~	~	✓	✓	✗
Pseudo-Grilles	✗	✗	✓	✓	~	✓
Planaires	~	✓	~	✓	~	✓
Communautées	✗	~	✓	✓	~	~

ayant bénéficié de transfert d'apprentissage (*i.e.*, *Ftune*) ont obtenu de meilleurs performances avec les caractéristiques PMDS (*Ftune PMDS* étant également la meilleure variante comme indiqué dans la Table III.2). Cela signifie que l'ajout de ces caractéristiques permet au modèle de produire des dessins de meilleure qualité, mais nécessite une phase d'entraînement plus conséquente. À l'inverse, bien que *Fscratch PMDS* soit meilleur que *Fscratch* sur la plupart des familles considérées, la différence entre eux est bien moins significative. A minima, l'ajout des caractéristiques initiales PMDS permet d'éviter en partie les chevauchements de nœuds dans les dessins produits par *(DNN)²*. D'une manière générale, tant que les positions 2D produites par PivotMDS n'ont pas de chevauchement, ces caractéristiques sont un moyen supplémentaire pour que le modèle différencie les nœuds ayant un voisinage identique, aidant ainsi à surmonter un des défauts typiques des convolutions de graphes (voir Section II.2.2).

III.4.6 Comparaison à PivotMDS

Les instances notées *PMDS* de *(DNN)²* ayant les positions des nœuds produites par l'algorithme PivotMDS [60] comme caractéristiques initiales, nous nous attendons à ce que la qualité des dessins produits par ces instances soient meilleurs que ceux de PivotMDS. Dans ce contexte, ces instances de *(DNN)²* sont une sorte de post-traitement des positions produites par PivotMDS bien qu'il n'y ait pas d'évidence que le réseau de neurones profond ait interprété ces caractéristiques comme des positions dans un plan euclidien orthonormé, tel que nous le faisons.

La Table III.3 présente les scores des métriques de qualité de PivotMDS et une instance de *(DNN)²* ayant les positions de PivotMDS dans ses caractéristiques initiales (Ftune PMDS) sur le jeu de données **Rome**. Les résultats dans cette table démontrent que *(DNN)²* améliore systématiquement la qualité des dessins de PivotMDS sur toutes les métriques de qualité, excepté le Stress pour lequel les scores restent néanmoins très proches.

Des exemples visuels des deux algorithmes de dessin sont présentés dans la Figure III.13. Pour le *Dodécaèdre* et la *Grille*, nous observons que les dessins de PivotMDS et *(DNN)²* sont relativement proches. Au vu des bases du Dodécaèdre et des angles dans la Grille, les dessins de *(DNN)²* souffrent de quelques défauts de régularité. Pour rappel, cette instance de *(DNN)²*, Ftune PMDS, a été pré-entraînée sur des graphes aléatoires puis affinée sur le jeu **Rome**. Il y a donc peu de chance que ce modèle ait appris à dessiner des structures régulières.

Les trois graphes suivants sont extraits du jeu **Rome**. Dans ces exemples, nous observons que les dessins produits par *(DNN)²* sont globalement plus satisfaisants que ceux de PivotMDS. Sur *Rome A*, le dessin de PivotMDS présente un aspect très orthogonal et certaines parties (*e.g.*, en haut à gauche) du dessin sont presque illisibles tant les nœuds sont proches et les résolutions

TABLE III.3 : Performances de PivotMDS [60] et *(DNN)*^{2} (Ftune PMDS) sur le jeu **Rome**. Pour chaque métrique, le score de la meilleure méthode est en gras. *(DNN)*^{2} obtient de meilleurs scores sur toutes les métriques excepté Stress. Les différences entre les scores de deux méthodes sur toutes les métriques sont significatives d'après les tests post-hoc de Conover [204] avec un seuil d'acceptance $\alpha = 0.05$.

	Aspect ratio	**Angular resolution**	**Edges Cross. number**	**Cluster overlap**	**Neighb. preserv.**	**Stress**
PivotMDS	0.298 ± 0.125	0.978 ± 0.088	38.7 ± 43.6	0.623 ± 0.202	0.49 ± 0.17	$\mathbf{0.104 \pm 0.035}$
***(DNN)*^{2} w/ PMDS**	$\mathbf{0.229 \pm 0.105}$	$\mathbf{0.917 \pm 0.138}$	$\mathbf{30.6 \pm 34.8}$	$\mathbf{0.541 \pm 0.206}$	$\mathbf{0.409 \pm 0.136}$	0.115 ± 0.046

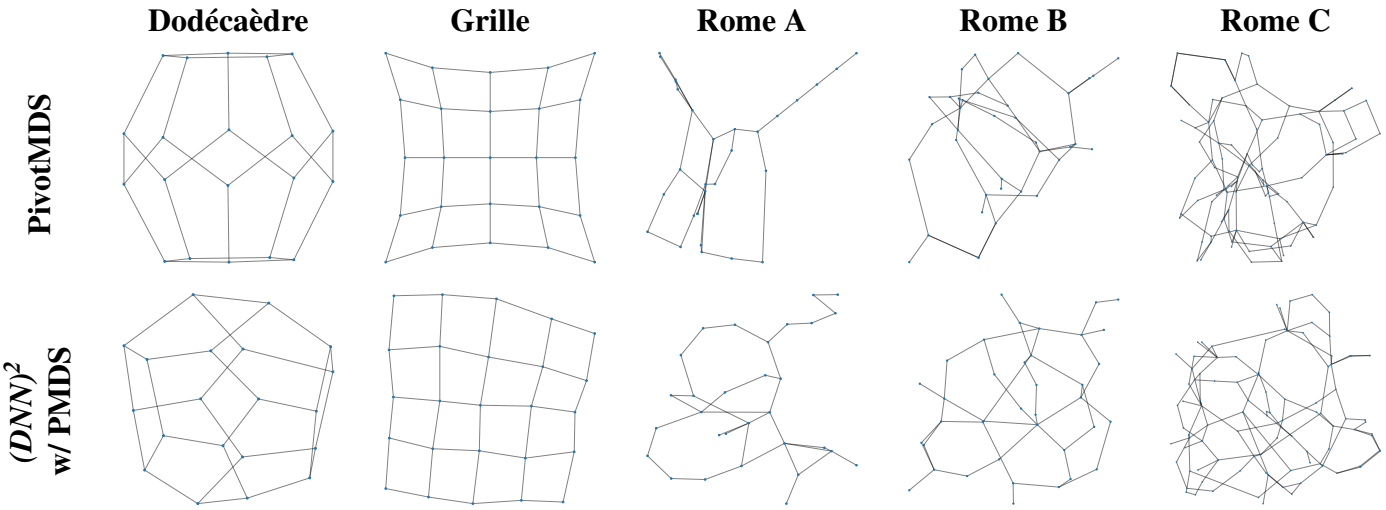


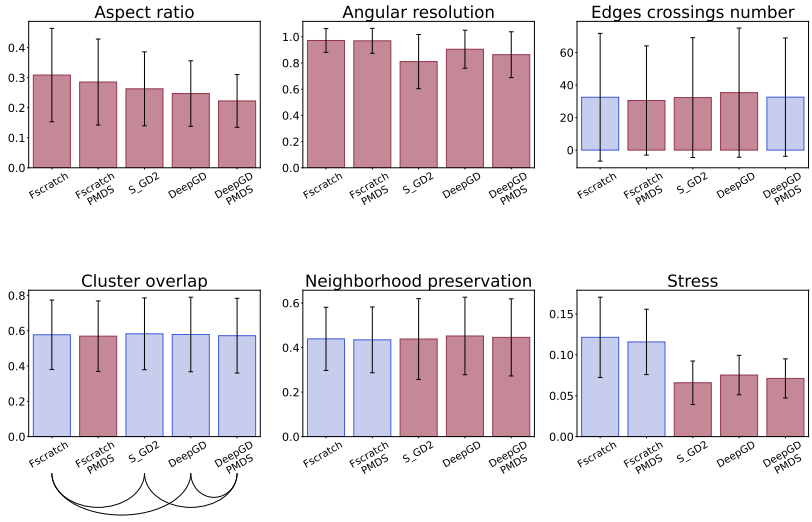
FIGURE III.13 : Exemples visuels de dessins de graphes produits par PivotMDS [60] et *(DNN)*^{2} (Ftune PMDS). Les deux premiers graphes dessinés sont respectivement un dodécaèdre et une grille régulière, tandis que les trois suivants sont extraits du jeu **Rome**.

angulaires faibles. A contrario, le dessin de *(DNN)*^{2} sur *Rome A* est plus arrondi et les zones denses du graphe restent lisibles. Les mêmes observations peuvent être faites pour les dessins des graphes *Rome B* et *Rome C*. Il semble également que PivotMDS ait tendance à superposer les nœuds ayant un voisinage similaire (*e.g.*, en bas à gauche du dessin de *Rome B*). Sachant que ce défaut est aussi présent dans les convolutions de graphe (voir Section II.2.2), avoir les positions produites par PivotMDS comme caractéristiques initiales n'aide probablement pas le réseau de neurones à résoudre ces cas autant que nous l'espérions. Néanmoins, les dessins produits par *(DNN)*^{2} montrent que ces nœuds ne sont pas exactement superposés, signifiant que le modèle est capable de mieux les dissocier que PivotMDS.

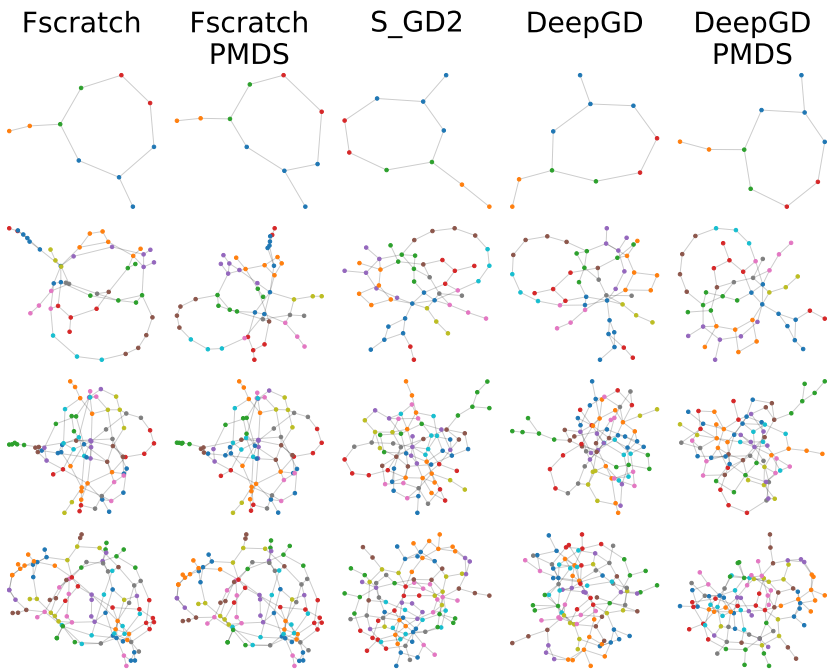
Nous pouvons conclure de cette évaluation quantitative et de ces quelques exemples visuels que *(DNN)*^{2} améliore en effet les dessins de PivotMDS quand ceux-ci lui sont donnés en caractéristiques initiales.

III.4.7 Comparaison aux Algorithmes de la Littérature

Dans cette section, nous comparons les performances de *(DNN)*^{2} avec deux méthodes proches dans la littérature : *S_GD*^{2} [39] et *DeepGD* [26]. *S_GD*^{2} effectue une descente de gradient stochastique pour optimiser une fonction de *Stress* pour chaque graphe à dessiner. Son originalité est de traiter les déplacements de nœuds par paires indépendantes, ce qui simplifie grandement la



(a) Comparaison de *(DNN)*^{2} Fscratch, *S_GD*^{2} et *DeepGD* avec et sans caractéristiques initiales PMDS sur le jeu Rome. Les graphiques présentent les moyennes et écarts types de chaque métrique. La couleur des barres et les arcs encodent la significativité des différences de performance (voir Section III.4.5.1).



(b) Exemples de dessins du jeu Rome avec $N \approx \{10, 50, 75, 100\}$.

FIGURE III.14 : (a) Performances et (b) exemples visuels des instances de *(DNN)*^{2} Fscratch, *S_GD*^{2} et *DeepGD* sur le jeu **Rome** avec et sans les caractéristiques initiales PMDS.

complexité de l'optimisation du Stress et accélère la converge de l'algorithme. *DeepGD* est une méthode de dessin de graphes par réseau de neurones profond. C'est probablement la méthode la plus proche de *(DNN)*^{2} (voir Section II.2.3). Les implémentations de *S_GD*^{2}[1] et *DeepGD*[2] proviennent de leurs dépôts de code source respectifs fournis par leurs auteurs.

Les performances et exemples visuels des dessins produits par ces méthodes sur le jeu **Rome**

[1]Implémentation de *S_GD*^{2} : `www.github.com/jxz12/s_gd2`
[2]Implémentation de *DeepGD* : `www.github.com/yolandalalala/deepgd`

sont présentés dans les Figures III.14a et III.14b. Pour comparer *(DNN)*^{2} aux méthodes de la littérature, nous avons choisi d'utiliser la stratégie d'entraînement *Fscratch* car c'est celle qui est également employée par *DeepGD*. La fonction de coût optimisée par notre instance *Fscratch* est C_{KL} (III.6). Le modèle *DeepGD* est entraîné pendant 1000 époques pour optimiser la fonction de *Stress* (III.12). *(DNN)*^{2} *Fscratch* et *DeepGD* sont donc tous les deux entraînés à partir de poids aléatoires sur la partie *entraînement* du jeu de données Rome, avec et sans les caractéristiques initiales PMDS.

En ce qui concerne les métriques de qualité, *(DNN)*^{2} (*i.e.*, *Fscratch* dans la Figure III.14a) est légèrement meilleur sur *Edge Crossing Number*, *Cluster Overlap* et *Neighborhood Preservation*. Les meilleures performances sur ces deux dernières peuvent être expliquées par la fonction de coût optimisée par *(DNN)*^{2}, C_{KL}, car celle-ci est explicitement dédiée à la préservation de voisinage. À l'inverse, nous constatons que *(DNN)*^{2} est moins efficace sur *Stress*. Nous nous attendions à ce résultat étant donné que *S_GD*^{2} et *DeepGD* optimisent directement le *Stress*. Ces deux techniques sont également meilleures que *(DNN)*^{2} sur *Aspect Ratio* et *Angular Resolution*.

Nous observons dans la Figure III.14b que tous les exemples visuels sont corrects, bien que ceux produits par *(DNN)*^{2} semblent moins satisfaisants. En effet, la longueur d'arête entre les nœuds des régions clairsemées des graphes est disproportionnée par rapport à la distance entre les nœuds des zones denses. Les dessins de *(DNN)*^{2} tendent à mettre en évidence les structures inter-communautaires au détriment des structures intra-communautaires. Bien que ce comportement puisse s'expliquer par la préservation induite par la fonction de coût du modèle, il peut être problématique et a été observé dans plusieurs exemples que nous avons étudiés. Par conception, C_{KL} assume que la distribution des distances autour de chaque nœud suit une loi normale centrée sur 0. Cependant, cette supposition mène souvent à la détérioration du voisinage proche des nœuds, comme nous le détaillerons dans la Section III.5.1.

Pour conclure, l'approche par réseau de neurones profond proposée par *(DNN)*^{2} et *DeepGD* ne produit pas encore de meilleurs résultats que des méthodes établies dans la littérature de dessin de graphes (*e.g.*, *S_GD*^{2}). Cependant, les premiers résultats que nous observons sont encourageants, d'autant plus que notre objectif n'était pas d'optimiser les performances de notre modèle en cherchant les meilleurs valeurs d'hyper-paramètres. Ainsi, il a été possible de mettre en relation des comportements observés avec les éléments constitutifs de *(DNN)*^{2}. Ces méthodes constituent des avancées notoires vers des applications efficaces de réseaux de neurones profonds pour le dessin de graphes.

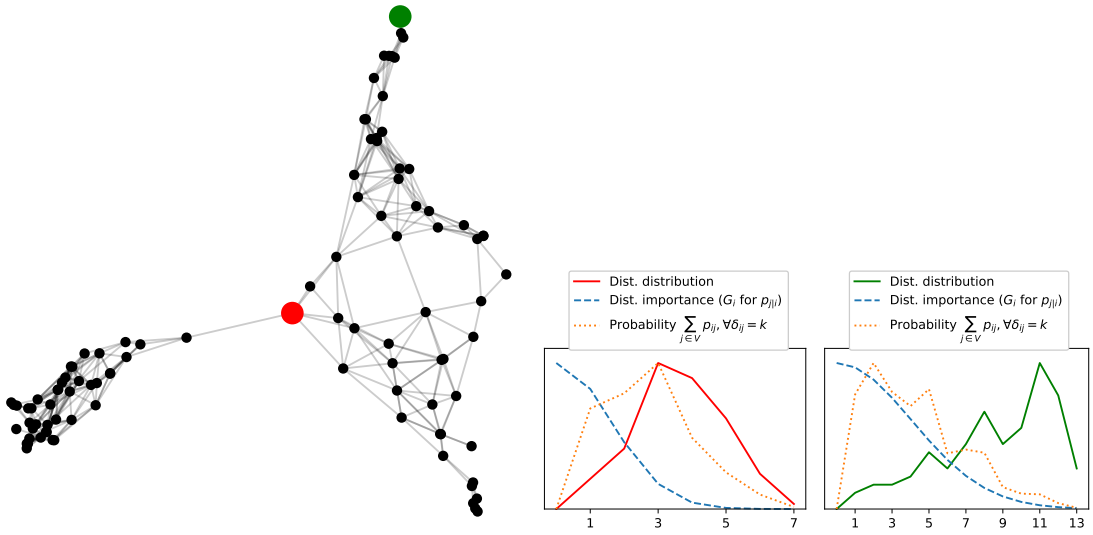
III.5 Discussion

Cette section discute certaines limitations observées dans les précédentes sections à propos du concept, de l'implémentation et de l'évaluation de *(DNN)*^{2}.

III.5.1 Optimisation de KL pour le Dessin de Graphe

Ici, nous discutons des limitations induites par l'optimisation de la divergence de Kullback-Leibler C_{KL} définie par Kruiger *et al.* [37]. Cette fonction a été présentée une première fois dans la Section III.3.3 et nous la détaillons ici. C_{KL} est définie par :

$$C_{KL} = \sum_{i,j \in V, i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{III.6}$$



(a) (b) Nœud central (c) Nœud excentré

FIGURE III.15 : (a) Un dessin de graphe du jeu Pseudo-Grilles par *(DNN)*^{2} *Ftune PMDS* avec la mise en valeur d'un nœud central (rouge) et excentré (vert). (b) (resp. (c)) présente la distribution des distances autour d'un nœud central (resp. excentré) v_i, la loi Gaussienne G_i de l'importance des distances autour de v_i (*i.e.*, le numérateur de l'Équation III.14) et la somme de probabilités résultante p_{ij} (III.13) autour de v_i pour chaque distance k. La courbe orange est une agrégation qui représente ce vers quoi un modèle optimisant C_{KL} devrait converger en terme de préservation des distances autour du nœud v_i. L'échelle des trois courbes de chaque graphique est différente, celles-ci ont été normalisées pour que les tendances soient plus facilement comparables.

où p_{ij} (III.13) et q_{ij} (III.15) sont les probabilités que chaque paire de nœuds du graphe soit connectée. p_{ij} est définie à partir des plus courts chemins dans le graphe tandis que q_{ij} est définie à partir des distances euclidiennes entre les nœuds dans le plongement. Ainsi, un modèle qui optimise C_{KL} cherche un plongement dans lequel la proximité des nœuds (convertie en probabilités) q_{ij} satisfait l'ensemble de probabilités d'appariement p_{ij} définit par la topologie.

$$p_{ij} = p_{ji} = \frac{p_{i|j} + p_{j|i}}{2N}, p_{ii} = 0 \tag{III.13}$$

où $p_{j|i}$ est définie dans l'Équation III.14.

$$p_{j|i} = \exp\left(-\frac{\delta_{ij}^2}{2\sigma_i^2}\right) / \sum_{k\in V, k\neq i} \exp\left(-\frac{\delta_{ik}^2}{2\sigma_i^2}\right), p_{i|i} = 0 \tag{III.14}$$

où σ_i est trouvé par recherche dichotomique pour que la perplexité $\kappa_i = 2^{\sum_{j\in V} p_{j|i}\log_2 p_{j|i}}$ se rapproche d'une valeur souhaitée. Pour rappel, δ est l'ensemble des paires de distances entre les nœuds dans l'espace topologique du graphe (*i.e.*, les longueurs des plus courts chemins).

Enfin, q_{ij} est définie par :

$$q_{ij} = q_{ji} = \frac{(1 + ||X_i - X_j||^2)^{-1}}{\sum_{k,l\in V k\neq l} (1 + ||X_k - X_l||^2)^{-1}}, q_{ii} = 0 \tag{III.15}$$

Notre première problématique à propos de l'optimisation de C_{KL} a été levée par l'effet *Fisheye* que nous avons observé dans les dessins produits par *(DNN)*^{2}. Les parties centrales des

graphes semblent dessinées convenablement, alors que les zones périphériques sont écrasées sur les bords. Cela peut typiquement être observé dans l'exemple de la Figure III.15a et dans les dessins de Grilles dans la Figure III.9b où les coins sont contractés. Cette distorsion était déjà observée par Kruiger *et al.* [37], mais ils ne l'avaient pas mise en relation directe avec la fonction de coût C_{KL}. Nous pensons que ce comportement vient de la combinaison de deux limitations liées à l'optimisation d'une fonction de coût basée sur t-SNE [35] pour le dessin de graphe. Notre seconde problématique concerne les capacités d'un modèle de réseau de neurones profond à optimiser une telle fonction puisque nous verrons que l'évaluation de C_{KL} est étroitement liée à la topologie de la donnée en entrée. Dans la suite, nous détaillons les raisons pour lesquelles nous pensons que C_{KL} n'est pas adapté au dessin de graphes par réseau de neurones profond.

Tel que défini par Kruiger *et al.* [37], C_{KL} est une adaptation de l'algorithme t-SNE [35] au contexte du dessin de graphes. t-SNE est une méthode de réduction de dimensions qui fonctionne en transformant les distances dans l'espace hautement dimensionnel en un ensemble de probabilités modélisant les (dis)similarités entre les données en fonction de leur distance aux autres et de leur position dans l'espace. La probabilité que deux points X_i et X_j soient similaires est choisie en supposant que la probabilité que X_i soit similaire à n'importe quel autre point suive une loi Gaussienne G_i centrée sur X_i. Plus précisément, la probabilité que X_i et X_j soient voisins est relative à $G_i(\delta_{ij})$ et $G_j(\delta_{ij})$ où δ_{ij} est la distance entre X_i et X_j dans l'espace en hautes dimensions. L'adaptation aux graphes est effectuée en considérant des *nœuds* plutôt que des *points de données* et en fixant δ_{ij} au plus court chemin entre les nœuds v_i et v_j. Pour chaque nœud v_i, l'adaptation au *dessin de graphe* de $G_i(k)$ peut être interprétée comme « à quel point le k-voisinage de v_i devrait-être préservé dans le dessin ». En centrant la Gaussienne G_i sur 0 et sachant qu'il n'y a aucun nœud à distance 0, la plus grande importance est donnée à la distance $k = 1$ et l'importance des distances $k > 1$ décroît en fonction de la pente de la Gaussienne (*i.e.*, en fonction de son écart type). Ce phénomène est illustré par la courbe bleue dans les Figures III.15b et III.15c.

Notre première problématique concerne l'importance effective donnée à la préservation des distances autour de chaque nœud en fonction de sa position dans le graphe. Nous pensons que cette problématique est à l'origine de l'effet *Fisheye* observé dans les dessins produits par les méthodes optimisant la fonction C_{KL}. Cela pourrait résulter de deux limitations relatives à la conception de cette fonction. La première limitation vient de la manière dont l'écart type σ_i de G_i est choisie. Telle que définie dans l'Équation III.14, σ_i est trouvé par recherche dichotomique pour qu'un nœud central (resp. excentré) obtienne une forte (resp. faible) valeur. D'une manière générale, cela revient à dire au modèle que la préservation des distances longues est plus importante autour des nœuds excentrés qu'autour des nœuds centraux. Cependant, autour des nœuds excentrés, les distances longues sont également plus nombreuses (voir la distribution des distances dans la Figure III.15c). Augmenter σ_i revient donc à donner plus de poids à la préservation de distances qui sont déjà plus représentées que les autres. Puisque la probabilité $p_{j|i}$ (III.14) est le ratio de $G_i(\delta_{ij})$ avec $\sum_{k \in V} G_i(\delta_{ik})$, augmenter σ_i revient également à réduire toutes les probabilités associées aux distances autour des nœuds excentrés. Dans l'exemple proposé dans la Figure III.15, la somme des probabilités p_{ij} (*i.e.*, la somme cumulée de la courbe orange) autour du nœud central est 0.014 contre 0.007 (soit deux fois moins) pour le nœud excentré. Si nous revenons à la formule de C_{KL} (III.6), p_{ij} est également un facteur de pondération pour le logarithme du ratio entre p_{ij} et q_{ij}. Cela signifie donc que la préservation des distances autour du nœud excentré est globalement deux fois moins importante que la préservation des distances autour du nœud central. Le cas idéal voudrait que l'importance soit $\frac{1}{N}$ pour chaque nœud (*i.e.*, 0.01 dans l'exemple de la Figure III.15 qui a 100 nœuds). La seconde limitation est que donner d'avantage d'importance aux plus nombreuses longues distances va

à l'encontre de la volonté initiale de donner la plus grande importance aux distances $k = 1$ et de diminuer l'importance des distances $k > 1$ en fonction de la pente de la Gaussienne G_i. Par exemple, avec le nœud central dans la Figure III.15b, bien que la distance la plus importante devrait être $k = 1$ (*cf.*, la courbe bleue G_i), les probabilités associées aux distances $k = 2$ et $k = 3$ sont plus fortes (*cf.*, la courbe orange). Enfin, nous observons que les tendances des courbes p_{ij} (orange) du nœud central et excentré sont affectées par la distribution des distances, ce qui nous amène à notre deuxième problématique.

Cette seconde problématique concerne les capacités d'un modèle de réseau de neurones profond à correctement apprendre à minimiser C_{KL}. Contrairement aux méthodes d'optimisation standards qui opèrent sur une seule donnée à la fois (*e.g.*, un graphe), un DNN est entraîné pour optimiser sa fonction de coût sur tout un jeu de données. Il doit donc apprendre une stratégie (*i.e.*, affiner ses poids) pour transformer l'information contenue dans le graphe en entrée afin de minimiser la fonction de coût, quel que soit ce graphe. Au regard de ce que nous avons observé plus tôt, nous voyons au moins deux risques qu'un DNN échoue à correctement dessiner des graphes en optimisant C_{KL}. Le premier risque concerne l'importance générale donnée aux nœuds centraux et excentrés. Comme observé dans l'exemple de la Figure III.15, la somme des p_{ij} était 0.014 pour le nœud central, contre 0.007 pour l'excentré, rendant ce dernier moins important dans l'estimation générale de C_{KL}. Sachant qu'un DNN doit minimiser sa fonction de coût sur tout un jeu de données, il y a une probabilité élevée qu'une bonne stratégie pour résoudre la tâche en optimisant C_{KL} soit d'ignorer les nœuds excentrés, puisqu'ils pèsent moins dans l'évaluation. Ceux-ci ayant un impact mineur, l'objectif du réseau serait alors de maximiser la qualité du plongement des nœuds centraux, au détriment des nœuds excentrés. Cela pourrait également expliquer l'effet *Fisheye* observé précédemment. Le second risque que nous identifions découle de notre première problématique. Comme évoqué plus tôt, les conséquences de l'augmentation de σ_i quand les nœuds deviennent excentrés casse l'hypothèse initiale : la préservation des distances autour d'un nœud v_i doit suivre une loi Gaussienne centrée sur 0. Comme nous l'avons observé, p_{ij} est sensible à la distribution des distances autour de chaque nœud. Ce comportement rend l'évaluation du positionnement de chaque nœud dépendante de la distribution des distances dans le graphe. Par extension, cela signifie que l'évaluation de chaque graphe dépend de ses propres propriétés, qui ne sont pas spécifiées au modèle. Il devient alors compliqué pour un DNN de trouver une stratégie générique, applicable à n'importe quel graphe en entrée. Nous nous attendons toutefois à ce que le modèle soit en partie capable d'apprendre à s'adapter à cette évaluation. Cependant, force est de constater que cela rajoute une inconnue supplémentaire à l'apprentissage qui peut nuire aux performances du réseau. Dans ce contexte, nous pouvons par exemple opposer l'optimisation de C_{KL} à celle du *Stress* (voir Équation III.12) qui met complètement de côté les propriétés topologiques des graphes pour ne mesurer que les différences de distances entre deux espaces (*i.e.*, projeté et hautes dimensions).

Visuellement, les problématiques que nous venons de mentionner mènent à des chevauchements, principalement entre les nœuds excentrés. Ce phénomène peut être observé dans la Figure III.15a où le centre du graphe est correctement dessiné alors que les trois parties excentrées souffrent de chevauchements. Il peut également être observé dans les dessins de Grilles (voir Figure III.9b) où les coins des grilles sont contractés.

III.5.2 Limitations

Notre évaluation a montré que *(DNN)*^{2} est capable d'apprendre à produire des dessins de graphes à partir de leur topologie, et de manière non-supervisée en optimisant la divergence de Kullback-Leibler (KL) malgré les défauts que nous venons de mentionner (voir Section III.5.1). Cependant,

d'autres limitations sont apparues et ont été en partie discutées tout au long de l'évaluation. Cette section présente les limitations qui ont été observées mais pas encore discutées.

Comme nous l'avons présenté dans la Section III.5.1, nous pensons que les chevauchements de nœuds dans les dessins produits par *(DNN)*^{2} sont principalement dus à la conception de la fonction de coût utilisée. Cependant, nous ne pouvons pas ignorer que *(DNN)*^{2} est composé de convolutions. Comme présenté dans des travaux précédents [25,124,125] et dans la Section II.2.2, ces opérations distinguent difficilement les nœuds ayant un voisinage similaire et ont tendance à produire le même plongement pour tous ces nœuds. L'architecture du modèle ne dispose pas de beaucoup d'information pour apprendre à dissocier ces nœuds. L'utilisation d'un identifiant unique et d'une valeur aléatoire ne suffit probablement pas à résoudre ce problème. Bien que cet effet ait été légèrement atténué par l'ajout des caractéristiques initiales PMDS (voir Section III.3.4), nous avons constaté que l'algorithme PivotMDS [60] souffrait lui-même de ce défaut et ne peut donc pas pleinement y pallier.

Une autre limitation est que les performances de *(DNN)*^{2} sont sensibles aux paramètres d'entraînement. Pendant la phase d'expérimentation, nous avons fixé certains choix de conception bien que nous nous attendions à ce qu'ils soient sous-optimaux (*e.g.*, utilisation de convolution de graphes, fonction de coût C_{KL}, caractéristiques initiales minimalistes) pour limiter l'effet *boîte noire* de notre modèle et être capable de relier des changements de performances à des critères observables et interprétables. Pourtant, certains hyper-paramètres ont tout de même dû être recherchés pendant une phase expérimentale d'essais-erreurs, ce qui est commun et problématique avec les réseaux de neurones profonds. Parfois, ajuster un hyper-paramètre (*e.g.*, diviser le taux d'apprentissage par 10) a plus d'impact sur les performances que changer une structure de données plus essentielle au fonctionnement du modèle. Évidemment, nous avons présenté une implémentation de *(DNN)*^{2} où les choix de ces hyper-paramètres ont mené aux meilleurs résultats que nous avons obtenus expérimentalement. Toutefois, nous ne pouvons être certains que ces choix soient les plus efficaces. Bien que l'apprentissage profond soit une approche prometteuse pour le dessin de graphe, elle comporte également ses propres limitations telles que l'opacité des modèles et les difficultés de conception, de paramétrage et d'interprétation. L'aspect positif de cette limitation est qu'avec cette approche, tous ces choix de conception sont faits *a priori*, avant l'entraînement du modèle. D'une manière générale, nous pouvons considérer que toute l'optimisation et l'apprentissage des modèles sont effectués par des experts sur des bancs d'essais. Les utilisateurs finaux du modèle entraîné n'ont alors qu'à manipuler les données d'entrée et sortie du modèle, sans plus avoir aucun paramétrage à effectuer. Ceci est un net avantage comparé aux approches plus classiques d'optimisation qui demandent parfois une expertise directe sur la technique d'optimisation afin de la paramétrer pour obtenir des résultats satisfaisants.

Enfin, la capacité de *(DNN)*^{2} à généraliser ses performances à des graphes de tailles différentes est compliquée. Le temps d'exécution des réseaux de neurones profonds, une fois entraînés, est un de leurs avantages majeurs. En effet, obtenir l'estimation d'un de ces modèles nécessite simplement d'appliquer les transformations polynomiales définies par les opérations et les poids du réseau sur l'entrée de celui-ci, ce qui se fait généralement très efficacement sur du matériel GPU. Cependant, l'entraînement de modèle sur des graphes de tailles significativement différentes devient difficile puisque augmenter la taille des graphes accroît considérablement les variations possibles dans les topologies de ceux-ci. Dès lors, il devient encore plus difficile pour le modèle de trouver une stratégie générique capable de dessiner efficacement n'importe quel graphe, quelle que soit sa topologie, sa taille ou ses caractéristiques. Par ailleurs, l'utilité de la diversité des données du jeu d'entraînement peut parfois être remise en question. Par exemple, le concept de généralisation des performances n'a pas d'application avec des *Cliques* ou des

Grilles puisqu'il est possible d'énumérer toutes leurs topologies possibles jusqu'à une certaine taille de graphe. La notion même de séparation du jeu de données en parties *entraînement-validation* pour éviter le surentraînement perd alors de son sens. Dans ce travail, nous avons étudié les effets de certaines propriétés des jeux de données d'entraînement sur les performances de *(DNN)*^{2}. De nombreuses recherches sont encore nécessaires pour comprendre comment construire correctement un jeu de données adapté au Dessin de Graphe.

III.6 Conclusion

Ce chapitre a présenté *(DNN)*^{2} : une approche d'apprentissage profond pour le dessin de graphes. Elle propose de tirer parti des architectures de réseaux de neurones profonds établis dans la littérature pour les adapter au contexte de graphes en transposant certaines opérations clefs. Ici, nous avons évalué l'approche en adaptant un réseau de neurones convolutif au contexte des graphes en remplaçant les convolutions standards par des convolutions de graphe. Le modèle apprend par lui-même à extraire les caractéristiques du graphe nécessaires à la résolution de la tâche. Les dessins produits optimisent une fonction de coût non-supervisée. Au travers de plusieurs évaluations, nous avons démontré les forces et les limitations de la méthode *(DNN)*^{2}, et plus généralement de l'apprentissage profond pour le dessin de graphes.

Les recherches dans cette thématique sont récentes et bénéficient d'un intérêt croissant dans la communauté, qui se traduit par une augmentation du nombre de publications reliant le domaine de l'apprentissage profond et des graphes. En ce qui concerne l'implémentation de *(DNN)*^{2} utilisée dans cette évaluation, le paramétrage global pourrait être amélioré. Nous avons fixé certains choix de conception potentiellement sous-optimaux afin de limiter l'effet *boîte noire* connu des réseaux de neurones profonds dans le but de pouvoir relier des changements de performances à des éléments structurants de la conception. Ainsi, il reste une large marge d'amélioration sur les choix d'implémentation concernant plusieurs hyper-paramètres. Par exemple, utiliser une technique de génération de caractéristiques initiales de nœuds, changer la fonction de coût, ou l'architecture du modèle, pourrait améliorer les performances. La principale préoccupation avec cette piste d'amélioration est d'être capable de résoudre les problématiques de généralisabilité de l'approche en la rendant moins sensible aux topologies ou tailles de graphes qui sortent du domaine défini par le jeu de données d'entraînement.

Enfin, d'autres pistes de travaux sont de mettre en œuvre de nouvelles méthodes d'entraînement d'apprentissage profond pour le dessin de graphes. Par exemple, puisque nous avons vu que le modèle apprenait différemment en fonction des topologies de graphes de son jeu d'entraînement, il serait intéressant de concevoir un *meta-model* agrégateur apprenant à combiner des dessins produits par plusieurs modèles *dessineurs*, chacun spécialisé sur une famille spécifique de graphes. De cette manière, la méthode pourrait bénéficier des avantages de l'extraction de caractéristiques des modèles spécialisés sur des familles de graphes tout en conservant une bonne capacité à généraliser ses performances en dehors du domaine défini par son jeu d'entraînement. Cette approche se rapprocherait d'une adaptation de TopoLayout [104] utilisant les capacités des réseaux de neurones profonds. Une dernière piste consisterait à explorer le dessin de graphe hiérarchique avec de l'apprentissage profond [45]. Il s'agirait alors d'entraîner un réseau à décomposer le graphe en sous-graphes, dessiner chaque composant, puis les recombiner. Chacune de ces trois étapes pourrait être réalisée par un modèle spécifiquement dédié à sa tâche restreinte, et le tout permettrait de gérer plus efficacement des graphes de grande taille.

Chapitre IV

Suppression de Chevauchements par Descente de Gradient Stochastique

Sommaire

IV.1 Problématique

La projection de données multidimensionnelles est devenue fréquente dans de nombreux domaines (*e.g.*, biologie [205, 206], science des données [34], traitement de graphe [37, 207]). Avec l'essor de domaines tels que l'Intelligence Artificielle et l'Internet des Objets, le volume et la complexité des données que nous traitons ne cessent d'augmenter. Cette complexification impacte également l'efficacité des techniques de visualisation, en particulier le plongement de données en 2D. Par exemple, il est courant de projeter les prédictions d'un modèle de réseau de neurones profond en 2D afin d'identifier des clusters et/ou trouver des intrus (*i.e.*, erreur de classification) [208–211]. La lisibilité de la projection ainsi construite revêt une importance capitale pour que la tâche visuelle puisse être résolue.

Pour produire une visualisation de nuage de points en 2D efficace, des algorithmes de réduction de dimensions tels que t-SNE [35] ou Umap [36] sont désormais couramment employés. Cependant, ces algorithmes traitent les données comme des points dans l'espace sans considérer que ceux-ci puissent avoir une taille non nulle (*i.e.*, une surface). Or, ces données sont la plupart

du temps représentées par une forme géométrique (*e.g.*, cercle, carré) lors de leur affichage à l'écran. Ces formes et leur taille permettent alors d'encoder de l'information supplémentaire dans la représentation même de la donnée (*e.g.*, étiquette textuelle, couleur catégorielle). L'ajout de ces formes *après* le plongement 2D modifie les distances relatives (*i.e.*, l'espace vide) entre les éléments d'une manière qui n'est pas prise en compte par l'algorithme de positionnement. Cela peut créer des *chevauchements* entre les éléments graphiques, lesquels chevauchements sont susceptibles d'occulter une partie, voire la totalité de l'information dans la visualisation.

Pour atténuer l'impact de ces chevauchements, un utilisateur peut expérimentalement modifier certains paramètres de la visualisation (*e.g.*, changer la taille des éléments, ajouter de l'opacité). Cette méthode n'est pas satisfaisante car (i) elle nécessite que l'utilisateur ait accès aux paramètres de la visualisation, (ii) elle est expérimentale, peut s'avérer coûteuse en temps (*e.g.*, si le re-calcul de la visualisation est long) et n'offre aucune garantie sur la visualisation produite. Par exemple, ajouter de la transparence aux éléments n'est efficace que jusqu'à une certaine quantité de chevauchements. Une autre approche parfois utilisée est le passage de la représentation en trois dimensions, avec la possibilité d'interagir en déplaçant la caméra[1]. Dans la plupart des cas, cela permet effectivement de résoudre les problèmes créés par les chevauchements, sans pour autant les supprimer. Cependant, ces visualisations 3D étant généralement représentées sur un écran en 2D, chaque angle de caméra contient la plupart du temps des chevauchements [212]. Il est alors nécessaire de visualiser les données sous plusieurs angles, ce qui complexifie la formation et la préservation de la *carte mentale* [213, 214] des données chez l'utilisateur [1, 156]. Pour résoudre ces chevauchements, nous nous intéressons ici aux algorithmes de la thématique Suppression de Chevauchements (OR pour « Overlap Removal »). Plusieurs approches d'OR existent, la plus répandue étant la dispersion des données. Cette approche consiste à déplacer les éléments dans le plan 2D de façon à supprimer les chevauchements, mais elle induit en contrepartie des déformations du plongement initial qui devrait idéalement être préservé. Nous nous concentrons ici sur les chevauchements dans les représentations 2D issues de plongements de données hautement dimensionnelles. Cela concerne donc les plongements d'algorithmes tels que t-SNE [35] pour des nuages de points mais également tous les algorithmes de dessin de graphe en nœud-lien, étant donné qu'un dessin est une projection de la topologie du graphe en 2D (voir Chapitre III). Or, les déformations induites par les algorithmes OR sur des données de ce contexte ne sont pas pénalisantes pour la visualisation puisque l'information principale à utiliser pour explorer ces représentations est la distribution des distances (ou densités). Au contraire, ces déformations endommagent fortement les visualisations 2D où les axes X et Y ont un sens sémantiques (*e.g.*, carte avec symboles). Par exemple, si l'axe X d'une visualisation encode des *années*, une inversion sur cet axe crée de la fausse information qui peut induire l'utilisateur en erreur.

Dans la littérature, un grand nombre de méthodes ont été proposées pour résoudre ces problèmes de chevauchements par déplacements des données [144–146, 154, 156–160, 162, 163]. Nous divisons ces méthodes en trois catégories principales en fonction du type de données qu'elles s'autorisent à manipuler.

Espace des Données. Les méthodes travaillant dans l'espace des données manipulent la donnée originale, brute (en hautes dimensions). Ces approches sont typiquement utilisées pour *agréger* ou *échantillonner* [147–150] les données efficacement, en fonction de leur distribution réelle dans leur propre espace (avant projection).

Espace Géométrique. Ces méthodes (*e.g.*, [27, 144, 154, 156–160, 162]) travaillent à partir des données déjà projetées, par exemple en 2D : $\{\{x, y, s\}, x, y, s \in \mathbb{R}\}$ où x et y sont les coordonnées des éléments, et s leur taille. Elles produisent un nouveau plongement qui ne modifie

[1] `https://projector.tensorflow.org/`

que les positions ($f : \{x, y, s\} \rightarrow \{x', y'\}$) et qui propose une solution *exacte* au problème de chevauchements. L'objectif est donc de déplacer les éléments jusqu'à ce que la somme des aires des chevauchements soit égale à zéro. Par la suite, l'Espace Géométrique sera abrégé GS (pour « Geometric Space »).

Espace Visuel. Cette dernière famille de méthodes regroupe les algorithmes qui produisent une visualisation ayant conscience de certains paramètres du rendu final [145, 146, 163]. Là où les approches GS produisent des plongements 2D des données, nous pouvons considérer que les méthodes de l'Espace Visuel (VS pour « Visual Space ») produisent des *images*. Ces méthodes peuvent déplacer les éléments, mais aussi modifier certaines propriétés du rendu final (*e.g.*, résolution de l'image, taille des éléments, opacité). Elles ne proposent pas nécessairement de solution exacte au problème de chevauchements car l'objectif est de produire une visualisation fidèle aux données d'origines, et où l'information est visible.

Ce chapitre est dédié à la présentation de deux algorithmes de suppression de chevauchements, FORBID [27] (pour « Fast Overlap Removal By stochastic gradIent Descent ») et son extension GIST [65] (pour « Guaranteed Visibility in Scatterplots with Tolerance ») publiés respectivement dans les conférences *Graph Drawing and Network Visualization 2022* et *VIS 2023*. FORBID est une méthode de suppression de chevauchements dans l'Espace Géométrique dont le code source est ouvert [66], tandis que GIST en est d'extension à l'Espace Visuel et dont une démonstration est disponible en ligne [67]. L'objectif de ces deux algorithmes est de produire des plongements sans chevauchements (ou presque, pour GIST) qui proposent un compromis entre (i) la visibilité des données et (ii) la préservation du dessin initial.

FORBID et GIST sont comparés aux algorithmes de la littérature. Nos deux méthodes ne travaillant pas dans le même espace (GS pour FORBID, VS pour GIST), leur description et évaluation seront distinctes dans les prochaines sections. Le reste de ce chapitre est organisé comme suit. La Section IV.2 rappelle les travaux principaux menés sur la suppression de chevauchements dans la littérature. La Section IV.3 présente FORBID et sa comparaison aux algorithmes de la littérature d'OR dans l'Espace Géométrique. La Section IV.4 présente GIST, l'extension à l'Espace Visuel de FORBID, ainsi que son évaluation. Enfin, la Section IV.5 synthétise les contributions de ces algorithmes et propose des pistes de travaux futurs.

IV.2 Rappel de l'État de l'Art

Cette section rappelle les algorithmes de Suppression de Chevauchements (OR) principaux de l'état de l'art (voir Section II.3). La description de ces algorithmes distingue les approches travaillant dans l'Espace Géométrique (GS) de l'Espace Visuel (VS). Toutes ces méthodes reposent sur le principe de déplacement des éléments graphiques représentant les données. Dans la suite, ces *éléments* sont appelés *nœuds*.

IV.2.1 Espace Géométrique et Dispersion de Nœuds

Les algorithmes d'OR travaillant dans le GS prennent un ensemble de positions et tailles en entrée, et ont pour objectif de supprimer entièrement les chevauchements. Pour y parvenir, ils se basent sur la Dispersion de Nœuds. Dans cette approche, le plongement initial est considéré comme une distribution de nœuds *de référence* dans GS, au sein de laquelle des contraintes (*i.e.*, chevauchements) doivent être relaxées. Pour cela, les nœuds sont déplacés tout en essayant de minimiser les déformations de cette distribution *référence*.

Une revue de la littérature des algorithmes de Dispersion de Nœuds dans GS a été présentée

par Chen *et al.* [49, 155]. PFS [156], PFS' [157], FTA [158], RWordle-L [159] sont des algorithmes itératifs se basant sur *scan-line* [154] pour identifier les chevauchements et déplacer les nœuds de manière orthogonale. VPSC [154, 160] et Diamond [161] sont deux algorithmes qui modélisent explicitement les chevauchements comme des contraintes à satisfaire. Ils utilisent ensuite un algorithme de relaxation de contraintes pour résoudre ces chevauchements et inférer une nouvelle position aux nœuds. Enfin, PRISM [144] et GTree [162] modélisent les distances idéales entre toute paire de nœuds en fonction de leur chevauchement ou non. Le déplacement des nœuds s'effectue ensuite pour optimiser une fonction de *Stress*. Ce Stress, communément noté σ, peut être défini comme :

$$\sigma(X) = \sum_{i,j} W_{ij}(||X_i - X_j|| - \delta_{ij})^2 \tag{IV.1}$$

où $||X_i - X_j||$ est la distance euclidienne entre i et j, δ_{ij} est la distance idéale entre ces nœuds et W_{ij} une pondération de l'importance de la paire (i, j). La suppression de chevauchements est satisfaite en modélisant les distances idéales entre chaque paire de nœuds par une valeur telle qu'ils ne puissent plus se chevaucher. La préservation du plongement initial est atteinte en utilisant la distance entre les nœuds dans le plongement initial comme distance idéale. Cette modélisation des chevauchements pour optimiser une fonction de Stress est l'approche utilisée par nos algorithmes FORBID et GIST. Nous verrons par la suite ce qui les distingue des algorithmes de la littérature.

Le principal défaut de ces méthodes est le coût de leur exécution. Leur complexité est quadratique sur le nombre de nœuds et la contrainte de supprimer *strictement* les chevauchements les rend incapables de gérer de gros volumes de données efficacement. Comme nous le verrons par la suite, c'est également le cas de notre algorithme FORBID, malgré ses temps d'exécution compétitifs.

IV.2.2 Espace Visuel

Les algorithmes traitent, au moins en partie, l'espace de rendu final de la visualisation, et peuvent même modifier certains de ses paramètres (*e.g.*, résolution de l'image, encodages visuels tels que l'opacité). En outre, si les algorithmes GS doivent *strictement* supprimer les chevauchements, la contrainte est plus souple pour les algorithmes VS. En effet, l'objectif de ces derniers est de produire une visualisation dans laquelle *toute l'information encodée est visible*. Il est donc possible de simplifier la contrainte de suppression de chevauchements avec des approximations, ce qui permet de mieux préserver le plongement initial ou gérer de gros volumes de données.

ScatterplotUnfold (SU) [146] est une illustration des algorithmes VS. Son objectif est de produire une visualisation qui préserve visuellement la distribution des *densités* du plongement initial. Pour atteindre cet objectif, l'algorithme déplace les nœuds, modifie leur diamètre et peut agréger des nœuds. La visualisation produite contient alors des nœuds de diamètres variables et ne comporte plus de chevauchement, tout en optimisant la fidélité du rendu de la distribution des densités.

Nous considérons également que les algorithmes de positionnement en grille (gridification) sont des algorithmes de suppression de chevauchements dans VS. Par conception, ils positionnent les nœuds dans une grille dont la taille peut être interprété comme une résolution minimale pour l'image de la visualisation produite. DGrid [163] et HaGrid [145] sont deux exemples d'algorithmes de positionnement en grille. Par principe, ils affectent une position exclusive à chaque nœud dans la grille, résultant ainsi en une visualisation sans chevauchement. Ces méthodes ont une complexité faible en $\mathcal{O}(Nlog(N))$, mais produisent des plongements qui déforment significativement l'initial.

Enfin, nous pouvons également inclure les algorithmes de visualisation compacte (*e.g.*, [164, 165]) dans la famille des OR VS. Ceux-ci ont pour objectif de produire la visualisation la plus compacte possible et assignent une position exclusive à chaque nœud. Il ne peut donc plus y avoir de chevauchement dans le plongement en sortie. Ces algorithmes se distinguent de ceux de gridification car ils cherchent bien à *minimiser* la taille de la grille pour la visualisation produite. Ce n'est pas forcément le cas avec les algorithmes de gridification qui peuvent au contraire être paramétrés avec une résolution suffisamment grande pour ne pas produire une visualisation compacte.

Nous considérons que GIST est une méthode d'OR dans VS car il manipule certains paramètres de rendu tels que la résolution de l'image, le diamètre des nœuds et une tolérance aux chevauchements (*i.e.*, approximation d'OR). Néanmoins, il se base sur FORBID, notre algorithme d'OR dans GS conçu pour résoudre *strictement* les chevauchements.

IV.3 Chevauchements dans l'Espace Géométrique

FORBID (pour « Fast Overlap Removal By stochastic gradIent Descent ») est dédié à l'Espace Géométrique, et sera par la suite comparé avec des algorithmes de ce contexte. Ces derniers sont communément dédiés à la suppression de chevauchements dans des graphes dont les nœuds sont rectangulaires. Nous nous plaçons donc dans ce contexte et définissons les notions suivantes pour cette section.

Notations

Soit $G = (V, E)$ un graphe avec $V = \{v_1, v_2, ..., v_N\}$ son ensemble de $N = |V|$ nœuds et $E \subseteq V \times V$ son ensemble d'arêtes. Un plongement (ou dessin) de graphe est défini par un vecteur $X \in \mathbb{R}^{N\times 2}$ où X_i est la position de v_i en 2D. Un nœud v_i est défini par un rectangle de largeur et hauteur (w_i, h_i) centré sur X_i. Deux nœuds se chevauchent si l'aire de l'intersection des rectangles les représentant est non nulle. Par commodité, nous définissons également l'ensemble des paires de nœuds se chevauchant par $O \subseteq V \times V$. Le plongement sans chevauchement d'un dessin initial X^0 est noté X'. La distance euclidienne entre deux nœuds v_i et v_j est notée $||X_i - X_j||$.

IV.3.1 Algorithme FORBID

Cette section présente FORBID [27], un algorithme qui résout les chevauchements en combinant (i) une recherche dichotomique d'un facteur d'échelle idéal, et (ii) des mouvements de nœuds optimisant une fonction de Stress via une simulation de Descente de Gradient Stochastique (SGD pour « Stochastic Gradient Descent »). Un aperçu du fonctionnement de FORBID est illustré dans la Figure IV.1 et sa complexité est en $\mathcal{O}(s(N^2 + N\log N))$ où s sera défini dans une prochaine section.

IV.3.1.1 Modélisation du Stress pour la Suppression de Chevauchements

Préliminaires. Dans la littérature du Dessin de Graphe, il est courant que les algorithmes optimisent une fonction de *Stress* pour produire leur plongement (*e.g.*, [39,59,215,216]). Comme vu précédemment, le Stress peut être défini comme :

$$\sigma(X) = \sum_{i,j} W_{ij}(||X_i - X_j|| - \delta_{ij})^2 \tag{IV.1}$$

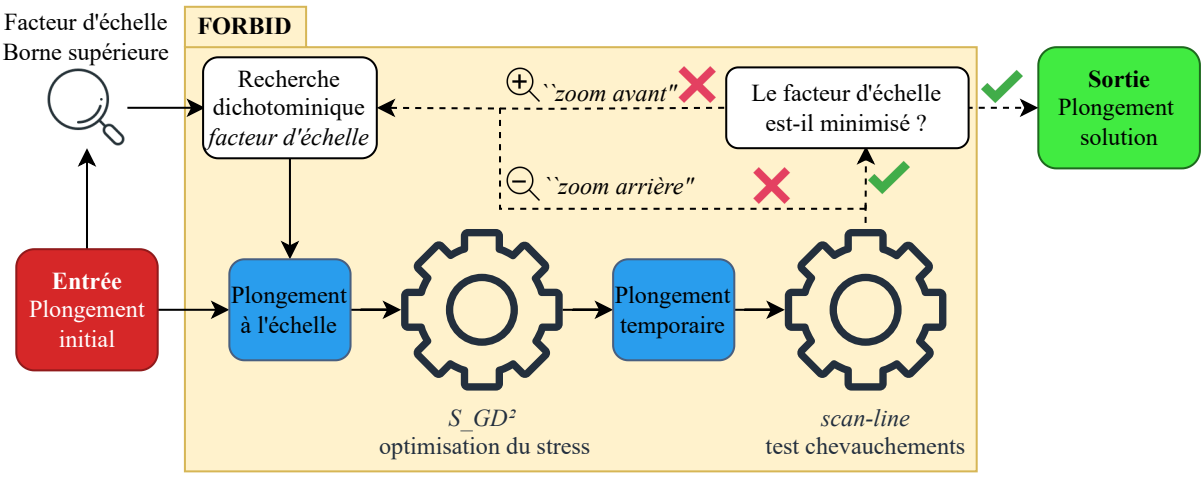


FIGURE IV.1 : Schéma simplifié de FORBID représentant l'organisation de la recherche dichotomique du facteur d'échelle idéal avec l'optimisation du Stress déplaçant les nœuds. Les engrenages représentent les algorithmes sur lesquels FORBID repose (*i.e.*, *S_GD²* [39] et *scan line* [154]). Les boîtes colorées représentent des états du plongement des nœuds, tandis que les boîtes blanches représentent les blocks relatifs à la recherche dichotomique.

où δ_{ij} est une *distance idéale* que le plongement du graphe doit préserver, et W_{ij} est une pondération typiquement fixée à δ_{ij}^{-2} (*i.e.*, plus des nœuds sont proches, plus il est important de respecter leur distance dans le plongement). Dans le contexte de Dessin de Graphe, δ_{ij} est fixé aux distances théoriques dans le graphes (*i.e.*, longueur des plus courts chemins entre les nœuds). De cette manière, les distances entre les nœuds dans le plongement du graphe en 2D doivent être représentatives des distances théoriques dans le graphe.

Comme proposé par Gansner et Hu [144], le *Stress* est également un bon critère pour résoudre le problème de suppression de chevauchements. En effet, optimiser le Stress revient à faire correspondre une distribution de distances d'un espace en basses dimensions (*e.g.*, plongement 2D) à une distribution de distances dans un espace hautement dimensionnel (*e.g.*, topologie de graphe) considéré comme idéal mais trop complexe pour être représenté tel quel. Dans le contexte d'OR, la définition des *distances idéales* est en deux partie. L'idée est de permettre la préservation de certaines distances, tout en autorisant une certaine quantité de mouvement nécessaires pour supprimer les chevauchements. Si une paire de nœuds $p_{ij} = (v_i, v_j)$ n'est pas un chevauchement (*i.e.*, $p_{ij} \notin O$), la distance idéale est fixée à la distance qui sépare ces nœuds dans le plongement initial $\delta_{ij} = ||X_i^0 - X_j^0||$. Cela permet à l'algorithme d'optimisation de converger vers un plongement préservant les distances entre les nœuds qui ne se chevauchent pas. En revanche, si $p_{ij} \in O$, la distance idéale entre les nœuds est fixée de telle sorte que les nœuds ne puissent plus se chevaucher. Cette valeur doit être suffisamment grande pour ne plus que les nœuds se chevauchent, mais suffisamment petite pour ne pas engendrer de déformation excessive du plongement initial. Par exemple, PRISM [144] fixe cette valeur à $\delta_{ij} = r_{ij}||X_i^0 - X_j^0||$ où r_{ij} est le facteur d'expansion minimal du vecteur $\overrightarrow{X_i^0 X_j^0}$ tel que les nœuds ne se chevauchent plus.

Cette modélisation des distances idéales revient à créer un espace K-dimensionnel où K est inconnu mais dans lequel *toutes* les distances entre les nœuds sont préservées, sauf celles entre les nœuds se chevauchant. L'optimisation du *Stress* tend alors à converger vers un plongement 2D dans lequel la distribution des distances correspond à cet espace en K dimensions.

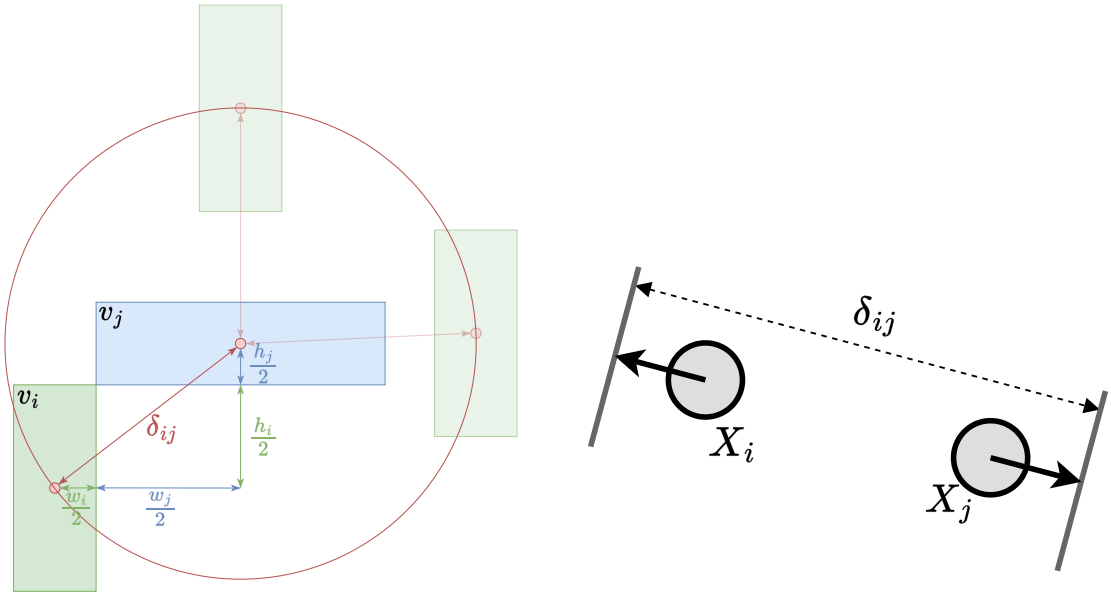
Modélisation du Stress dans FORBID. Contrairement à PRISM, la distance idéale entre des nœuds qui se chevauchent est fixée uniquement en fonction de leur *taille* dans FORBID.

Entre deux nœuds rectangulaires v_i et v_j, elle est définie par la distance qui les séparerait s'ils étaient tangents en un sommet comme illustré dans la Figure IV.2a. Formellement, la distance idéale dans FORBID s'exprime comme :

$$\delta_{ij} = \begin{cases} ||X_i^0 - X_j^0||, \text{si } (v_i, v_j) \notin O \\ \sqrt{\left(\frac{w_i+w_j}{2}\right)^2 + \left(\frac{h_i+h_j}{2}\right)^2}, \text{si } (v_i, v_j) \in O \end{cases} \tag{IV.2}$$

où X^0 est le plongement initial, tandis que w_i et h_i sont la largeur et hauteur du nœud v_i. Avoir une définition *resserrée* de la distance idéale comme celle de PRISM signifie qu'il n'y a qu'une seule direction de mouvement pour la paire de nœuds qui permet de satisfaire à la fois la distance idéale et la suppression de chevauchements. Avec la distance idéale que nous définissons dans FORBID, deux nœuds qui respectent cette distance ne peuvent plus se chevaucher *quelle que soit leur position relative*. Cette définition sera alors plus facile à satisfaire pour l'algorithme d'optimisation et devrait faciliter sa convergence. En revanche, elle est propice à l'apparition d'inversions dans l'ordre orthogonal des nœuds, ce qui est commun avec les approches par optimisation de stress.

Enfin, le facteur de pondération entre les paires de nœuds W_{ij} est une puissance ajustable de δ_{ij} qui permet de maîtriser la balance entre l'importance de la préservation des distances courtes ou longues, et l'importance de la préservation des distances entre les nœuds qui se chevauchent



(a) Distance idéale entre nœuds en chevauchement. (b) Mouvement de nœuds dans *S_GD²*.

FIGURE IV.2 : (a) Distance idéale δ_{ij} de FORBID pour une paire de nœuds qui se chevauche dans le plongement initial. Cette distance est uniquement définie en fonction de la taille des deux nœuds. À cette distance, ils ne peuvent plus se chevaucher, quelle que soit leur position relative. (b) Illustration du mouvement d'une paire de nœuds dans *S_GD²* [39]. Les nœuds sont déplacés dans des directions opposées le long du vecteur séparant leur centre. L'amplitude du mouvement dépend de la différence entre la distance qui les sépare et la distance idéale δ_{ij}.

ou entre ceux qui ne se chevauchent pas :

$$W_{ij} = \begin{cases} \delta_{ij}^{\alpha}, \text{si } (v_i, v_j) \notin O \\ \\ \delta_{ij}^{\alpha * \omega}, \text{si } (v_i, v_j) \in O \end{cases} \tag{IV.3}$$

où $\alpha \in \mathbb{R}$ est un facteur relatif à la distance entre toutes les paires de nœuds (typiquement fixée à $\alpha = -2$ pour que les distances courtes aient plus de poids), et $\omega \in \mathbb{R}$ est un poids permettant de paramétrer l'importance donnée aux distances entre les paires de nœuds qui se chevauchent.

IV.3.1.2 Optimisation du Stress

Pour minimiser la fonction de Stress définie dans la Section IV.3.1.1, FORBID se base sur l'algorithme *S_GD²* [39]. L'originalité de *S_GD²* est de proposer une méthode de relaxation de contraintes qui mime une descente de gradient stochastique. Dans les algorithmes classiques d'optimisation de Stress, chaque mouvement de nœud est calculé à partir de la somme des forces d'attraction et de répulsion qu'exercent tous les autres nœuds sur lui. Pour alléger ce calcul, *S_GD²* utilise une approche de programmation par contrainte [115, 217] pour modéliser l'optimisation du Stress comme un ensemble de contraintes (entre chaque paire de nœuds) à satisfaire *individuellement* [218]. Chaque mouvement d'une paire de nœuds s'effectue en fonction de la direction du vecteur entre leurs centres, et de l'amplitude du gradient du Stress (*i.e.*, $\frac{\partial \sigma_{ij}(X)}{\partial X_i}$ et $\frac{\partial \sigma_{ij}(X)}{\partial X_j}$, où σ est définie dans l'Équation IV.1). Dans l'exemple de la Figure IV.2b, deux nœuds v_i et v_j sont déplacés dans des directions opposées le long du vecteur qui sépare leur centre, en fonction de la différence entre la distance qui les sépare actuellement $||X_i - X_j||$ et la distance idéale δ_{ij}. Le mouvement peut aussi être négatif pour rapprocher les nœuds. L'algorithme parcourt ensuite toutes les paires de nœuds un nombre fixé de fois (*i.e.*, nombre d'itérations), en utilisant un pas d'apprentissage dégressif η pour réduire l'amplitude des mouvements à chaque itération et converger vers une solution.

L'optimisation du Stress est censée faire correspondre une distribution de distance dans un espace en basses dimensions à une distribution de distances d'un espace hautement dimensionnel. Cependant, avec notre modélisation de la tâche de suppression de chevauchements, les modifications dans l'espace en basses dimensions (*i.e.*, les mouvements de nœuds) peuvent affecter l'espace hautement dimensionnel. En effet, la distribution des distances idéales δ dépend des chevauchements. Or, les mouvements de nœuds peuvent supprimer et recréer des chevauchements. Ainsi, il devient nécessaire de re-évaluer les distances idéales δ et leurs pondérations correspondantes W pour que les prochains déplacements de nœuds soient toujours calculés de façon à converger vers la bonne distribution de distances.

IV.3.1.3 Recherche du Facteur d'Échelle Optimal

Nous appelons *modification du facteur d'échelle* l'augmentation (ou la réduction) de la taille de la boîte englobante du dessin. Cet effet peut être obtenu en multipliant les positions de tous les nœuds par une constante. Cela revient alors à éloigner ou rapprocher tous les nœuds uniformément, ce qui préserve toutes les distances relatives.

Pour les algorithmes d'OR, le changement d'échelle du plongement initial est une préoccupation majeure. En effet, l'utilisation de ces algorithmes est nécessaire lorsque la visibilité de l'information encodée par les nœuds est importante. Or, lorsque la représentation de la donnée est rendue dans une fenêtre de taille constante, augmenter le facteur d'échelle du plongement revient à réduire la taille de tous les nœuds dans la fenêtre. Cet effet doit donc être limité pour ne

pas devenir contre-productif en rendant les nœuds moins visibles qu'ils ne l'étaient malgré les chevauchements.

Cependant, il n'est pas toujours possible de supprimer les chevauchements sans augmenter le facteur d'échelle, par exemple lorsque la somme de l'aire des nœuds dépasse l'aire de la boîte englobante du dessin. De plus, cette somme de l'aire des nœuds n'est pas non plus une borne inférieure satisfaisante, car la satisfaire revient à produire une visualisation compacte qui ne préserve pas nécessairement le plongement initial.

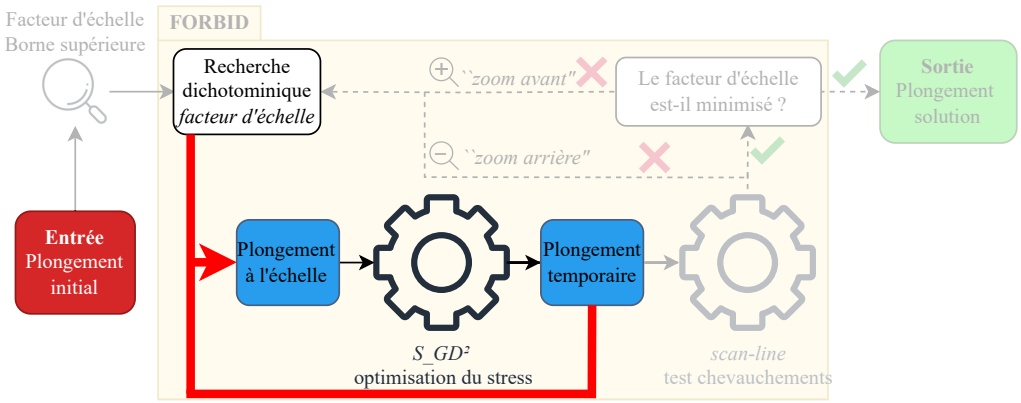
L'augmentation du facteur d'échelle du plongement initial devrait donc (i) rester suffisamment limité pour ne pas que les nœuds deviennent trop petits ; et (ii) devrait être suffisante pour qu'il y ait assez d'*espaces vides* pour produire un nouveau plongement sans chevauchements qui préserve l'aspect du plongement initial. Dans la plupart des algorithmes de la littérature, cette augmentation nécessaire du facteur d'échelle est un effet de bord des déplacements de nœuds et est mesurée comme une métrique de qualité permettant l'évaluation de l'algorithme.

Dans FORBID, la recherche du facteur d'échelle optimal fait partie intégrante de l'algorithme. Il n'est pas optimisé en même temps que les déplacements de nœuds puisqu'il est nécessaire de le fixer afin d'identifier les chevauchements dans le plongement courant et définir les distances idéales pour les déplacements de nœuds. Ici, ce facteur est trouvé via une recherche dichotomique entre 1 (*i.e.*, l'échelle du plongement initial) et une borne supérieure s_{max} qui peut être définie de plusieurs façons. Pour nous assurer que FORBID converge vers un plongement sans chevauchements, s_{max} est défini comme le facteur d'échelle de l'algorithme Scaling [49]. En outre, ce facteur spécifique permet de supprimer tous les chevauchements sans modifier les distances relatives entre les nœuds (sous réserve qu'il n'y ait pas de chevauchement *parfait*, *i.e.*, plusieurs nœuds avec exactement les mêmes coordonnées). La recherche se termine lorsque la différence entre deux étapes consécutives est plus faible qu'un seuil de précision s_p paramétrable. La profondeur de la recherche dichotomique est donc $s \leq \log\left(\frac{s_{max}-1}{s_p}\right)$.

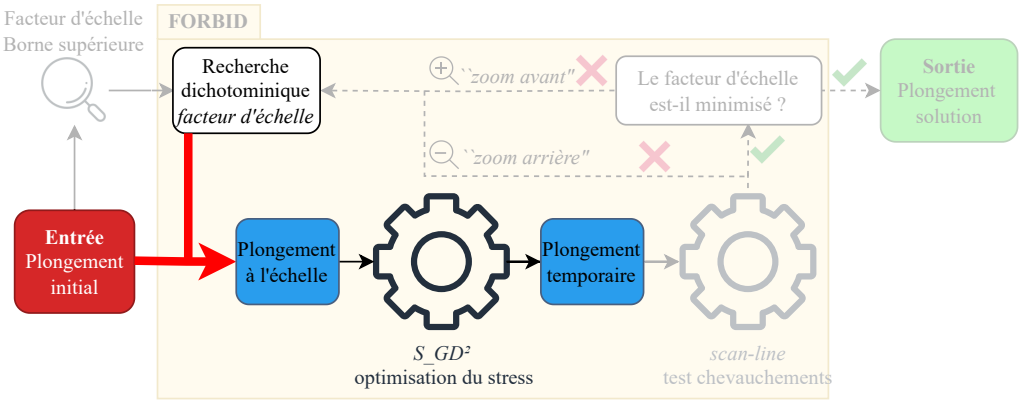
IV.3.1.4 FORBID et sa variante FORBID'

Comme décrit dans la Figure IV.1, la première étape de FORBID est de définir un facteur d'échelle via la recherche dichotomique (voir Section IV.3.1.3). Chaque étape dans cette recherche forme ce que nous appelons une *passe* dans l'algorithme. Dans chaque *passe P*, FORBID déplace les nœuds pour optimiser une fonction de Stress (voir Section IV.3.1.1) pour supprimer les chevauchements et préserver le plongement initial. Cette optimisation est faite via une simulation de descente de gradient stochastique pour un nombre d'itérations fixé, qui permet de donner une sorte de *budget* de mouvements à l'algorithme. Une *passe* peut également s'arrêter avant que le *budget* ne soit entièrement consommé si aucun mouvement n'est produit au cours d'une itération. S'il reste des chevauchements à la fin d'une passe P après ce *budget* de mouvements, nous estimons que résoudre tous les chevauchements en déplaçant les nœuds à ce facteur d'échelle déformerait trop le plongement initial. Dans ce cas, la prochaine *passe* $P+1$ se fera avec un facteur d'échelle plus grand (*i.e.*, plus d'espaces vides entre les nœuds) dont la valeur sera donnée par la recherche dichotomique. En revanche, il y a deux possibilités si une passe P ne contient plus de chevauchements après les déplacements de nœuds. Si la différence de facteur d'échelle avec la passe précédente est plus grande que le seuil de précision s_p, l'algorithme continue son optimisation et la prochaine passe $P+1$ de la recherche dichotomique réduit le facteur d'échelle pour essayer de trouver une autre solution sans chevauchements, plus compacte. Dans le deuxième cas, si la différence de facteur d'échelle avec la passe précédente est inférieure à s_p, l'algorithme se termine et le plongement sans chevauchement est retourné.

Le nombre de passes dans l'algorithme est déterminé par la profondeur de la recherche



(a) FORBID, chaque nouvelle passe $P+1$ commence à partir du plongement de la passe précédente X^P.



(b) FORBID', chaque nouvelle passe $P+1$ commence à partir du plongement initial X^0.

FIGURE IV.3 : Différences entre les variantes FORBID (a) et FORBID' (b).

dichotomique $s \leq \log\left(\frac{s_{max}-1}{s_p}\right)$. La détection de l'existence des chevauchements est réalisée avec l'algorithme *scan-line* [154] en $\mathcal{O}(N\log(N))$, et la complexité de l'algorithme d'optimisation *S_GD²* [39] est en $\mathcal{O}(N^2)$. Ainsi, la complexité de FORBID est $\mathcal{O}(s(N^2+N\log N))$.

Puisque chaque passe P prend un plongement X en entrée pour produire un nouveau plongement X^P, deux plongements initiaux sont envisageables pour $P+1$. La première possibilité est de commencer $P+1$ à partir du plongement en sortie de P (*i.e.*, X^P), comme montré dans la Figure IV.3a. Cela permet à l'algorithme d'augmenter la quantité de mouvement de nœuds totale à mesure que P augmente. Nous pensons qu'augmenter la quantité de mouvement autorisée de cette manière peut permettre à l'algorithme de converger plus rapidement, au coût d'une plus grande déformation du plongement initial. L'autre possibilité pour démarrer une passe $P+1$ est de repartir du plongement initial X^0, mis à l'échelle en fonction du facteur courant donné par la recherche dichotomique (voir Figure IV.3b). Cette approche revient à n'autoriser qu'une seule passe de déplacement de nœuds pour chaque facteur d'échelle. Nous pensons que cela peut ralentir la convergence de l'algorithme et aboutir à des plongements solutions moins compacts, mais avec une meilleure préservation du dessin initial.

Nous appelons la première variante (commençant $P+1$ à partir de X^P) FORBID, tandis que la seconde (commençant $P+1$ à partir de X^0) est nommée FORBID'. La différence entre ces deux variantes est illustrée dans la Figure IV.3.

TABLE IV.1 : Nombre de graphes, nœuds par graphe et chevauchements pour chaque jeu de données de l'évaluation. Les chevauchements sont décrits par les nombres minimum, maximum et la moyenne ± écart type du nombre de chevauchements dans chaque jeu de données.

	Générés	**Graphviz**
# Graphes	840	14
# Nœuds	[10 ; 1 000]	[36 ; 1 463]
# Chevauchements	[0 ; 36 537] 3 196 ± 8 717	[4 ; 11 582] 2 119 ± 4 080

IV.3.2 Protocole d'Évaluation

Dans un soucis de reproductibilité, l'évaluation de FORBID reprend le protocole expérimental défini par Chen *et al.* [49, 155]. Cette section présente les jeux de données et les métriques de qualité utilisées pour comparer différents algorithmes d'OR par déplacement de nœuds dans l'Espace Géométrique.

IV.3.2.1 Jeu de données

Les jeux de données sélectionnés dans cette évaluation sont ceux utilisés par Chen *et al.* [49, 155], disponibles en ligne[1].

Générés est un jeu de 840 graphes synthétiques générés pour l'évaluation d'algorithmes d'OR. Il est constitué de 120 graphes de chaque taille $N = 10, 20, 50, 100, 200, 500, 1\,000$. Les nœuds sont positionnés par l'algorithme FM^3 [110].

Graphviz est un jeu de 14 graphes réels de la suite Graphviz [219]. Ils ont entre 36 et 1 463 nœuds positionnés par l'algorithme *SFDP* [220].

Un récapitulatif des principales propriétés de ces jeux de données est présenté dans la Table IV.1.

IV.3.2.2 Métriques

Dans les algorithmes de suppression de chevauchements (OR), nous supposons que le plongement initial en entrée a été calculé pour optimiser certains critères et produire une représentation ayant une signification particulière pour l'utilisateur final. Un algorithme d'OR efficace devrait alors supprimer les chevauchements tout en préservant les caractéristiques de ce plongement initial.

Puisque les algorithmes qui seront comparés produisent tous un plongement sans chevauchement, leur performance est uniquement évaluée en fonction de leur préservation du plongement initial. Ce plongement pouvant être détérioré sous plusieurs aspects, nous reprenons cinq métriques sélectionnées dans l'étude de Chen *et al.* [49] pour capturer différentes formes de déformation. Toutes sont orientées de telle sorte qu'un score plus faible signifie une meilleure qualité du plongement.

Ordre orthogonal des nœuds. Nommée *oo_nni*, cette métrique compte le nombre de fois que l'ordre orthogonal des nœuds du plongement initial est violé dans le plongement produit.

$$oo\_nni = \frac{\sum_{(v_i,v_j)\in V^2, i\neq j} \begin{array}{r} \mathbb{1}\{x_i^0 > x_j^0 \wedge x_i' < x_j'\}, \\ +\mathbb{1}\{y_i^0 > y_j^0 \wedge y_i' < y_j'\} \end{array}}{N(N-1)} \tag{IV.4}$$

[1]Graphes Générés et Graphviz : `https://github.com/agorajs/agora-dataset`

où $X_i^0 = (x_i^0, y_i^0)$ (resp. $X_i' = (x_i', y_i')$) sont les coordonnées de v_i dans le plongement initial X^0 (resp. sans chevauchements X'). La métrique est normalisée pour permettre la comparaison de sa valeur sur des graphes de différentes tailles (en nombre de nœuds).

Aire de l'enveloppe convexe. *sp_ch_a* est le ratio de l'enveloppe convexe du plongement initial sur l'aire de l'enveloppe du plongement sans chevauchements. Cette métrique permet donc de mesurer une forme *d'expansion* du plongement et capture l'augmentation du facteur d'échelle.

$$sp\_ch\_a = \frac{\mathcal{A}(\text{ConvexHull}(X'))}{\mathcal{A}(\text{ConvexHull}(X^0))} \tag{IV.5}$$

où *ConvexHull* calcule l'enveloppe convexe contenant tous les nœuds d'un plongement en prenant en compte leur position, forme et taille ; et $\mathcal{A}$ calcule l'aire d'une enveloppe convexe.

Aspect ratio. *gs_bb_iar* mesure la déformation du rapport hauteur/largeur entre le plongement initial et celui sans chevauchements.

$$gs\_bb\_iar = \max\left(\frac{W' * H^0}{H' * W^0}, \frac{H' * W^0}{W' * H^0}\right) \tag{IV.6}$$

où W^0 et H^0 (resp. W' et H') sont la largeur et la hauteur de la boîte englobante rectangulaire du plongement initial (resp. sans chevauchements).

Mouvements de nœuds. *nm_dm_imse* mesure le déplacement moyen des nœuds de leur position dans le plongement initial jusqu'à leur position dans la solution sans chevauchements. Puisque la modification du facteur d'échelle est déjà capturée par une autre métrique (*i.e.*, *sp_ch_a*), *nm_dm_imse* est rendue insensible au changement d'échelle et au déplacement global des plongements initiaux et sans chevauchements. Cela permet alors de mesurer les déformations qu'ont pu engendrer certains algorithmes et qui ne sont pas simplement liées au facteur d'échelle.

$$nm\_dm\_imse = \frac{1}{N}\sum_{i<N} ||X_i' - \text{shift}(\text{scale}(X_i^0))||^2 \tag{IV.7}$$

où les fonctions $shift$ et $scale$ projettent les positions des nœuds du plongement initial vers le domaine de valeur du plongement sans chevauchements.

Préservation des longueurs d'arêtes. *el_rsdd* mesure l'uniformité de la préservation des distances. Par opposition à *nm_dm_imse* qui pénalise tous les mouvements de nœuds, *el_rsdd* mesure la déviation moyenne des longueurs des arêtes de la Triangulation de Delaunay [75] du plongement initial. L'intuition est que si une certaine quantité de mouvement est nécessaire pour supprimer les chevauchements, ces mouvements devraient s'opérer sur des nœuds proches (susceptibles de se chevaucher) et être répartis uniformément. Pour une quantité de mouvement donnée, cette métrique privilégie la répartition de celle-si sur tous les nœuds, et pénalise donc les cas où peu de nœuds se déplacement fortement.

$$\begin{aligned} \rho_{uv} &= \frac{||X_u' - X_v'||}{||X_u^0 - X_v^0||}, \forall (u,v) \in E_{DT}, \\ \overline{\rho} &= \frac{1}{|E_{DT}|}\sum_{(u,v)\in E_{DT}} \rho_{uv}, \\ el\_rsdd &= \frac{\sqrt{\frac{1}{|E_{DT}|}\sum_{(u,v)\in E_{DT}}(\rho_{uv} - \overline{\rho})^2}}{\overline{\rho}} \end{aligned} \tag{IV.8}$$

où E_{DT} est l'ensemble des arêtes de la Triangulation de Delaunay du plongement initial X^0, et ρ_{uv} représente le facteur de déviation de la longueur de l'arête (u,v).

IV.3.2.3 Algorithmes de Référence

Les algorithmes de la littérature auxquels nous comparons FORBID sont en partie ceux sélectionnés par Chen *et al.* [49,155]. Tous sont des algorithmes d'OR par déplacement de nœuds dans GS. Nous avons sélectionné ces algorithmes pour couvrir les différentes approches de la résolution de la tâche. Ainsi, nous comparons FORBID à PFS [156] et PFS' [157] qui ont une approche par traitement séquentiel orthogonal des nœuds. PRISM [144] et GTREE [162] implémentent des approchent par optimisation de Stress. Enfin, Diamond [161] est un algorithme de programmation par contraintes. Ces algorithmes sont présentés plus en détail dans la Section II.3.2.

IV.3.3 Évaluation Quantitative

Cette section présente les résultats de l'étude des performances de l'algorithme FORBID comparé aux algorithmes de la même catégorie dans la littérature (voir Section IV.3.2.3), sur les jeux de données et métriques sélectionnées (voir Section IV.3.2.1 et IV.3.2.2 respectivement). Toutes les métriques dans cette section doivent être minimisées (*i.e.*, un score plus faible signifie une meilleure qualité).

Sur le jeu de données **Générés** (voir Figure IV.4), chaque algorithme a minimisé *oo_nni* avec succès. Nous pouvons toutefois noter que Diamond et FORBID ont des scores légèrement plus élevés sur les plus mauvais cas (Q3). Sur *sp_ch_a* et *gs_bb_iar*, FORBID et FORBID' obtiennent les meilleurs scores avec une certaine avance, surtout sur Q3 où les autres algorithmes perdent significativement en performance. Les résultats sur ces deux métriques démontrent les capacités de FORBID et FORBID' à limiter l'augmentation de l'échelle du plongement produit. De plus, nos deux variantes minimisent les quantités de mouvements relatives des nœuds (*i.e.*, *nm_dm_imse*) mieux que n'importe quel algorithme de la littérature de cette évaluation. Plus particulièrement, nous observons que FORBID' ne produit presque aucun mouvement de nœuds, même dans les pires cas (*nm_dm_imse* = 50.23 en Q3). Enfin, FORBID et sa variante ont tous les deux des scores élevés sur *el_rsdd*. Comme défini dans la Section IV.3.2.2, cette métrique mesure la préservation des longueurs des arêtes de la Triangulation de Delaunay du plongement initial. Par conception, nous savons que FORBID peut mener à des déformations locales du plongement initial. En modélisant le Stress entre toutes les paires de nœuds (voir Section IV.3.1.1), notre algorithme favorise la préservation de la forme *globale* du dessin, parfois au détriment de la préservation de structures *locales*.

		FORBID	*FORBID'*	*PFS*	*PFS'*	*PRISM*	*GTREE*	*Diamond*
oo_nni	*Q1*	0,00	0,00	0.00	0.00	0.00	0,00	0,04
	Median	0,00	0,00	0.00	0.00	0.01	0,02	0,06
	Q3	0,06	0,03	0.00	0.00	0.02	0,03	0,09
sp_ch_a	*Q1*	1,00	1,00	1.04	1,00	1.01	1,12	1,16
	Median	1,03	1,03	1.69	1,22	1.12	1,49	1,96
	Q3	2,86	3,21	17.63	5,04	4.22	5,94	6,94
gs_bb_iar	*Q1*	1,00	1,00	1.01	1,00	1.00	1,01	1,04
	Median	1,01	1,00	1.19	1,04	1.04	1,04	1,07
	Q3	1,04	1,01	1.65	1,14	1.23	1,08	1,12
nm_dm_imse	*Q1*	3,90	3,90	6.50	1,65	9.82	28,22	535,65
	Median	34,69	25,03	694.83	116,06	131.57	292,56	1971,15
	Q3	419,88	50,23	7899.2	1782,80	688.1	2221,70	16571,10
el_rsdd	*Q1*	0,11	0,11	0,05	0,05	0,13	0,13	0,33
	Median	0,37	0,32	0,25	0,21	0,31	0,28	0,46
	Q3	0,70	0,46	0,33	0,26	0,38	0,36	0,89

FIGURE IV.4 : Quartiles des métriques de qualité de chaque algorithme sur le jeu de données **Générés** (voir Section IV.3.2.1).

Les performances sur le jeu de données **Graphviz** sont présentées dans la Figure IV.5. Nous pouvons observer que les tendances sont les mêmes que sur le jeu de données précédent. Chaque algorithme minimise correctement *oo_nni*, et FORBID et FORBID' ont toujours une large avance sur *gs_bb_iar*. Cependant, nous pouvons voir que nos algorithmes ne sont plus les meilleurs sur *sp_ch_a* et seul FORBID' maintient sont excellent score sur *nm_dm_imse*. Ces résultats montrent clairement les différences de conception entre FORBID et FORBID'. Comme nous nous y attendions, FORBID mène à des plongements avec moins d'augmentation du facteur d'échelle (*sp_ch_a*) mais plus de mouvements de nœuds (*nm_dm_imse*), à l'inverse de FORBID'. PRISM obtient également de très bons scores sur ces métriques, surtout comparé à ses performances sur le jeu **Générés**. Enfin, le score de FORBID sur *el_rsdd* est peu satisfaisant, tandis que celui de FORBID' est plus proche de ceux des autres méthodes.

Temps d'exécution. La Figure IV.6 présente les temps d'exécution de FORBID, FORBID' que nous comparons avec les algorithmes ayant obtenu les meilleures performance sur les métriques de qualité : PFS', GTREE et PRISM. Nous mesurons les temps d'exécution sur les données du jeu **Graphviz** dans lequel nous identifions trois catégories de difficulté. FORBID, FORBID' et PFS' sont instantanés sur les données *faciles*, prenant moins de *20ms*, tandis que GTREE et PRISM sont plus lents. Sur les données de difficulté *moyenne*, PFS' obtient encore les meilleures performances. La tendance sur les autres méthodes est la même peu importe le graphe de cette catégorie : FORBID est plus rapide que GTREE, puis vient FORBID', et enfin PRISM qui est au moins deux fois plus lent que les précédents. Enfin, PFS' est toujours quasi-instantané sur les données *difficiles*. FORBID est plus rapide que les autres méthodes, tandis que FORBID' est plus lent que GTREE. Sur ces données complexes, PRISM devient significativement plus

	FORBID	*FORBID'*	*PFS*	*PFS'*	*PRISM*	*GTREE*	*Diamond*
oo_nni	0,05	0,02	0,00	0,00	0,02	0,02	0,05
sp_ch_a	3,71	5,39	210,28	6,92	2,18	4,02	30,63
gs_bb_iar	1,05	1,02	1,97	1,22	1,33	1,17	1,10
nm_dm_imse	43795,43	2697,03	9768157,94	63594,74	42919,66	37331,83	1348426,00
el_rsdd	0,55	0,42	0,39	0,28	0,37	0,35	0,46

FIGURE IV.5 : Performances moyennes des algorithmes sur les métriques qualités pour le jeu de données **Graphviz** (voir Section IV.3.2.1).

	$\|V\|$	$\|E\|$	$\|O\|$	***Graphe***	*FORBID*	*FORBID'*	*PFS'*	*GTREE*	*PRISM*
Facile	36	107	4	***dpd***	0	0	0	29	34
	41	49	20	***unix***	0	0	0	46	54
	43	64	9	***rowe***	0	0	0	41	46
	47	55	33	***size***	1	1	0	49	91
	50	99	13	***ngk104***	0	0	0	40	46
	76	104	19	***nan***	1	1	0	58	115
	79	181	33	***b124***	4	5	0	77	229
	135	366	53	***b143***	13	19	0	126	275
Moyen	213	269	1105	***mode***	166	420	1	270	1040
	302	611	268	***xx***	305	682	2	540	1450
	302	611	282	***b102***	268	824	2	540	1750
Difficile	1054	1083	11582	***root***	3097	12989	22	7256	37734
	1235	1616	10540	***badvoro***	4577	18013	34	9694	27514
	1463	5806	5691	***b100***	11827	22880	48	13074	63877

FIGURE IV.6 : Temps d'exécution (en ms) de FORBID, FORBID', GTREE et PRISM sur le jeu de données **Graphviz** (voir Section IV.3.2.1). Les lignes sont triées par la « complexité » du plongement initial mesuré en nombre de nœuds $|V|$, d'arêtes $|E|$ et de chevauchements $|O|$.

lent que les autres algorithmes.

Nous pouvons en conclure que FORBID passe relativement bien à l'échelle pour un algorithme d'OR par déplacement de nœuds dans GS, malgré sa complexité quadratique. En revanche, la scalabilité de FORBID' est moindre. Comme nous nous y attendions, recommencer chaque *passe* à partir du plongement initial rallonge nécessairement la durée de l'optimisation (voir Section IV.3.4.2). Comparé à PRISM et GTREE (*i.e.*, d'autres approches basées sur l'optimisation de Stress) nous observons tout de même que les temps d'exécution de FORBID et sa variante sont compétitifs, voire meilleurs. Enfin, le temps d'exécution de PFS' reste significativement meilleur que ceux des algorithmes considérés dans cette étude.

IV.3.4 Discussion

IV.3.4.1 Exemples Visuels

Dans cette section, nous comparons visuellement les plongements produits par FORBID, FORBID', PFS', PRISM et GTREE sur trois données du jeu **Graphviz** : *mode*, *badvoro* et *root*. Les plongements initiaux et sans chevauchements produits par les algorithmes sont présentés dans la Figure IV.7 et les scores des métriques de qualité qui leur sont associés sont reportés dans la Table IV.2.

Sur *mode*, FORBID et PRISM ont tous les deux endommagé le plongement initial pour produire des dessins compacts. À l'inverse, les autres algorithmes préservent la structure du plongement initial, FORBID' étant le plus efficace visuellement. Les métriques de qualité sur cette donnée corroborent ces observations. FORBID et PRISM ont un meilleur score de *sp_ch_a* (mesurant l'échelle) tandis que FORBID', PFS' et GTREE ont de meilleurs scores sur *nm_dm_imse* (déformation du plongement). Comme observé dans l'évaluation quantitative (voir Section IV.3.3), FORBID' atteint un nettement meilleur score que les autres algorithmes sur *nm_dm_imse*.

Les tendances visuelles sont similaires sur les deux données *badvoro* et *root*. PRISM produit

TABLE IV.2 : Métriques de qualité des dessins présentés dans la Figure IV.7. Pour chaque donnée et chaque métrique, le meilleur score est mis en avance en gras.

		FORBID	FORBID'	PFS'	GTREE	PRISM
mode	*oo_nni*	0.20	**0.03**	0.54	0.05	0.06
	sp_ch_a	**2.98**	10.35	9.08	7.77	3.72
	gs_bb_iar	1.02	**1.01**	1.20	1.23	1.52
	nm_dm_imse	20 417	**1657**	8655	10 238	17 442
	el_rsdd	0.95	0.70	**0.39**	0.54	0.62
badvoro	*oo_nni*	0.01	**0.00**	0.48	0.02	0.02
	sp_ch_a	11.42	12.40	17.92	13.45	**6.87**
	gs_bb_iar	**1.00**	**1.00**	1.25	1.17	1.85
	nm_dm_imse	979	**451**	57 267	67 296	47 708
	el_rsdd	0.32	0.23	**0.20**	0.40	0.39
root	*oo_nni*	**0.01**	**0.01**	0.53	0.05	0.04
	sp_ch_a	19.17	29.99	48.34	14.29	**6.2**7
	gs_bb_iar	**1.01**	**1.01**	1.34	1.30	2.04
	nm_dm_imse	24 502	**12 151**	662 185	356 632	450 287
	el_rsdd	0.90	0.72	**0.43**	0.67	0.73

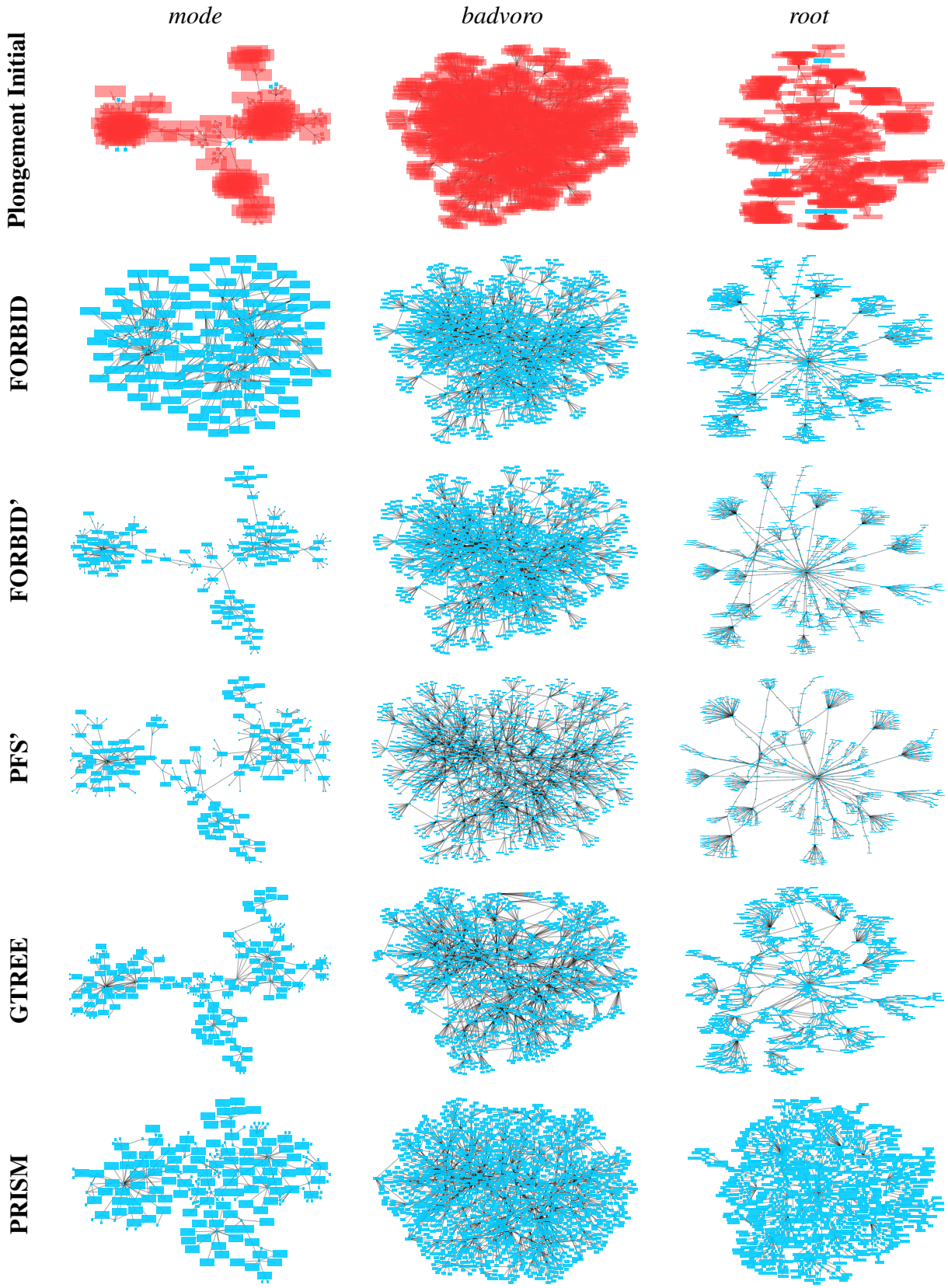


FIGURE IV.7 : Exemples visuels de données issues du jeu **Graphviz** par FORBID, FORBID', PFS', GTREE et PRISM. Les nœuds rouges chevauchent au moins un autre nœud.

des plongements si compacts qu'il en devient difficile de reconnaître le plongement initial, ou même de visualiser les arêtes du graphe. Nous pensons que ses dessins sont une bonne illustration de la raison pour laquelle la génération de plongements trop compacts, bien que sans chevauchements, n'est pas adaptée au dessin de graphe où la visualisation des communautés de nœuds et des arêtes est primordiale. GTREE produit des dessins moins compacts, mais déforme plusieurs structures du plongement initial. Enfin, FORBID, FORBID' et PFS' produisent des plongements sans chevauchement satisfaisants, bien que ceux de PFS' semblent moins compacts. Ces observations sont corroborées par les scores des métriques de qualité (voir Table IV.2) : PRISM est meilleur sur *sp_ch_a* (*i.e.*, plus compact), GTREE n'est le plus performant sur aucune métrique, et les trois méthodes restantes ont de meilleurs scores de *nm_dm_imse* (*i.e.*, préservation du plongement initial). PFS' a également un score de *sp_ch_a* plus grand que FORBID et FORBID', démontrant une augmentation plus forte du facteur d'échelle.

D'une manière générale, FORBID et FORBID' produisent des plongements équilibrés qui optimisent à la fois la préservation du plongement initial et la compacité du plongement sans chevauchement.

IV.3.4.2 Analyse de Convergence

Dans cette section, nous discutons l'impact de la re-initialisation du plongement à chaque *passe* dans l'optimisation de FORBID', notamment sur sa vitesse de convergence. La Figure IV.8 présente l'évolution de plusieurs critères tout au long de l'optimisation de FORBID et FORBID' sur les données *mode*, *badvoro* et *root* du jeu **Graphviz**. Chaque graphique reporte l'évolution du Stress, du nombre de chevauchements et du facteur d'échelle du plongement en fonction de l'avancée de l'optimisation (*i.e.*, nombre d'itérations).

Sur ces données, le nombre de passes dans FORBID et FORBID' n'excède jamais 10. Le nombre d'itérations maximum (*i.e.*, le *budget* de mouvements) est fixé à 30. Un grand nombre de passes se termine avant d'avoir utilisé les 30 itérations grâce à l'arrêt anticipé conditionné par la mesure de la quantité de mouvement. Pour rappel, lorsqu'une quantité trop faible de mouvements est mesurée dans une itération, la passe est arrêtée car les itérations suivantes ne peuvent que générer encore moins de mouvements (voir Section IV.3.1.2). Sur chaque graphique, nous pouvons observer que la courbe du Stress est fortement liée à celle du nombre de chevauchements. Cela confirme qu'optimiser notre modélisation du Stress permet bien de supprimer des chevauchements. La différence entre FORBID et FORBID' est également clairement visible. Dans FORBID, la plupart des chevauchements sont supprimés dès les premières passes, les passes suivantes étant dédiées à la recherche du facteur d'échelle optimal et comportent peu de déplacement de nœuds. Nous observons également distinctement l'impact de recommencer à partir du plongement initial au début de chaque passe dans les graphiques montrant la convergence de FORBID'. Notamment sur les données *badvoro* et *root*, le Stress et le nombre de chevauchements augmentent drastiquement au début de chaque nouvelle passe. La *hauteur* légèrement différente de chaque pic successif est due à la variation du facteur d'échelle entre chaque passe, qui recrée ou supprime une partie des chevauchements. En outre, cela illustre bien que dans FORBID', la tâche est de trouver le facteur d'échelle le plus petit qui permet de supprimer tous les chevauchements en seulement 30 itérations d'optimisation du Stress, ce qui est foncièrement différent de l'approche de l'autre variante. Comme nous pouvions nous y attendre, FORBID' a systématiquement besoin de plus d'itérations que FORBID pour produire une solution (approximativement deux fois plus dans les exemples de la Figure IV.8). Ces résultats illustrent pourquoi FORBID' est plus lent que FORBID mais préserve mieux le plongement initial (voir Section IV.3.3).

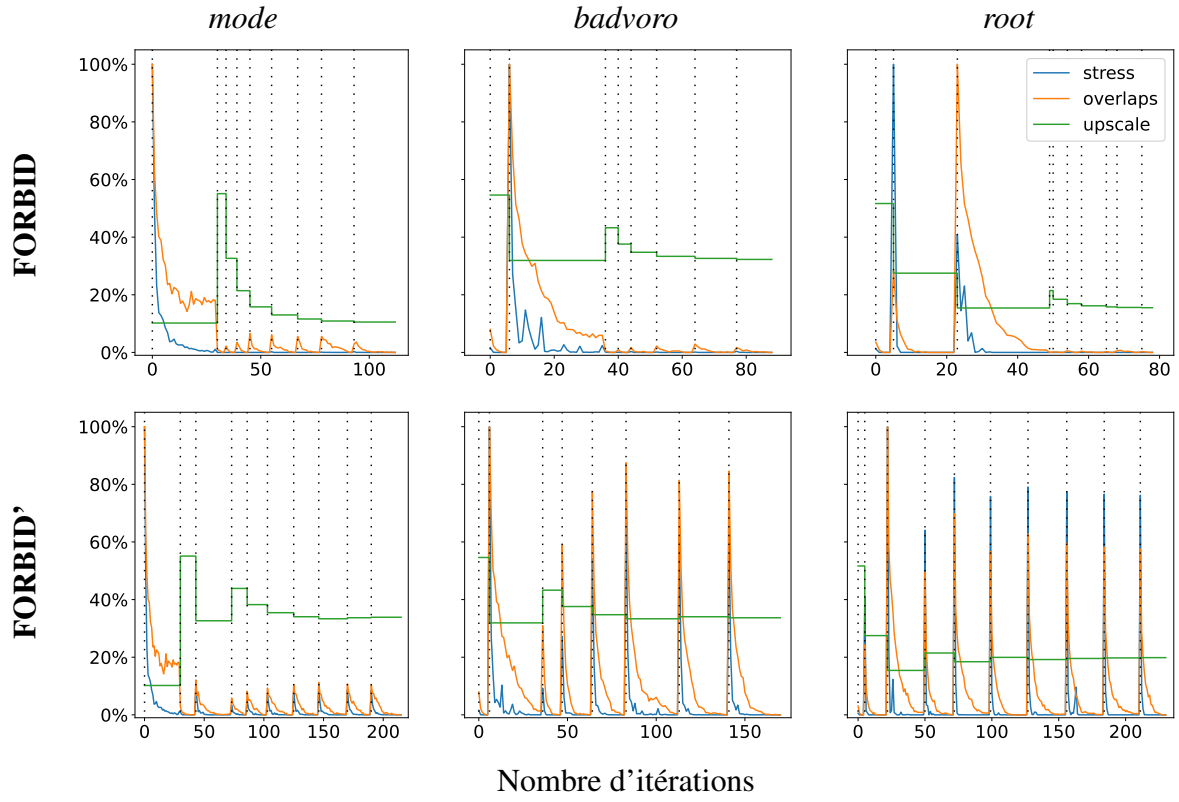


FIGURE IV.8 : Graphiques de convergence de FORBID et FORBID' sur trois données du jeu **Graphviz**. Chaque graphique reporte l'évolution du Stress, du nombre de chevauchements et du facteur d'échelle du plongement en fonction de l'avancée de l'optimisation (*i.e.*, nombre d'itérations). Les barres verticales en pointillés représentent le début de nouvelles *passes* (*i.e.*, changement de facteur d'échelle). Le Stress et le nombre de chevauchements sont normalisés par leur valeur maximum respective, tandis que le facteur d'échelle est normalisé par la borne supérieure définie dans la recherche dichotomique (voir Section IV.3.1).

Enfin, nous souhaitons discuter le choix de la borne supérieure de la recherche dichotomique s_{max}. Dans ce travail, nous l'avons fixée au facteur d'échelle minimum qui permet de supprimer tous les chevauchements sans autre mouvement de nœud (voir Section IV.3.1.3). Cela permet de garantir que l'algorithme converge vers un plongement sans chevauchement, puisque même si l'optimisation du Stress ne permet jamais de tous les supprimer, l'algorithme augmente le facteur d'échelle jusqu'à ce qu'il n'y en ait plus aucun. Une telle définition de s_{max} peut cependant être très élevée lorsque deux nœuds se chevauchent quasi parfaitement. En pratique, les facteurs d'échelle des plongements produits par FORBID restent éloignés de cette borne supérieure. Cependant, il serait possible de fixer cette borne à une valeur plus faible pour forcer l'algorithme à produire des plongements plus compacts, ou accélérer son temps d'exécution en évitant de tester des facteurs d'échelle inutilement élevés. Par exemple, s_{max} pourrait être fixé à un multiple de la somme de l'aire des nœuds. Cela permettrait de paramétrer le facteur d'échelle maximal en fonction de la surface nécessaire pour dessiner tous les nœuds, et permettrait de gérer explicitement la portion d'espaces vides dans le plongement produit. En revanche, nous ne pourrions pas garantir qu'il existe une solution sans chevauchements atteignable avec un tel espace restreint. En conclusion, ces choix dépendent du plongement initial et de l'aspect désiré du plongement sans chevauchements. Bien que notre définition de s_{max} puisse ralentir l'exécution de l'algorithme, ou l'amener à produire des plongements avec un facteur d'échelle

trop élevé, nous pensons qu'elle est bien adaptée dans le cas général.

Un autre axe qui permettrait d'accélérer la convergence de l'algorithme serait d'alléger la contrainte des chevauchements. En effet, si la définition du chevauchement est une intersection non nulle entre les formes représentant les nœuds, il est toutefois tolérable qu'il reste une portion d'intersection tant que celle-ci n'endommage pas la lisibilité de la représentation. Nous étudions cette autre approche dans la section suivante qui présente GIST [65], notre algorithme de suppression de chevauchements dédié à l'Espace Visuel.

IV.4 Chevauchements dans l'Espace Visuel

Les algorithmes de Suppression de Chevauchements dans l'Espace Géométrique ont pour objectif d'enlever la totalité des chevauchements entre les nœuds. Cependant, il n'est parfois pas nécessaire de les supprimer complètement pour obtenir une visualisation efficace. En effet, les représentations étant visualisées sur un écran, la discrétisation de l'espace géométrique vers un espace visuel dans lequel est fait le rendu de l'image (*i.e.*, **rastérisation**) induit une perte d'information, principalement sur le contour des éléments graphiques. La Figure IV.9 illustre ce principe. Dans ce cas, même s'il reste un faible chevauchement entre deux nœuds, il est fort probable que celui-ci ne soit pas visible après la rastérisation du plongement, menant ainsi à une représentation sans chevauchement dans l'espace visuel. Nous pouvons donc *tolérer* une certaine quantité de chevauchements tout en s'assurant que celle-ci ne gène pas la visibilité des éléments.

Dans cette section, nous présentons GIST [65] (pour « Guaranteed vIsibility in Scatterplots with Tolerance », voir Figure IV.10), un algorithme basé sur l'idée de FORBID adaptée à l'espace visuel et incluant une tolérance aux chevauchements. GIST étant une méthode d'OR dans VS, il optimise le déplacement des nœuds en ayant connaissance de certains paramètres de rendu de l'image issue du plongement qu'il produit. Plus précisément, GIST calcule un plongement optimisé pour une certaine *résolution cible* d'image, avec une *tolérance* aux chevauchements en nombre de pixels, et modifie la *taille des nœuds* en pixels dans la résolution cible (*i.e.*, l'équivalent du facteur d'échelle dans l'espace visuel). L'objectif de GIST est d'optimiser trois critères : (i) les nœuds doivent avoir une taille supérieure à 1 pixel dans la résolution cible R, pour garantir leur visibilité ; (ii) les nœuds doivent être le plus grand possible, pour maximiser leur visibilité ; et (iii) le plongement initial doit être préservé. L'objectif de GIST est également d'être utilisable sur des données de plus grande échelle (dizaines de milliers de nœuds) que les algorithmes travaillant dans GS gèrent difficilement.

Dans ce travail, nos données ne sont plus des graphes, mais des nuages de points. Les représentations des nœuds sont des cercles, comme présenté dans la Figure IV.10. Ce changement est opéré pour s'adapter au contexte du traitement des données volumineuses dans l'espace

(a) Espace Géométrique (b) Espace Visuel (rastérisé)

FIGURE IV.9 : Illustration d'un chevauchement dans (a) l'Espace Géométrique. Ce chevauchement n'est plus visible une fois rastérisé dans une grille de pixels (b). L'objectif de GIST est de tirer parti de cet effet.

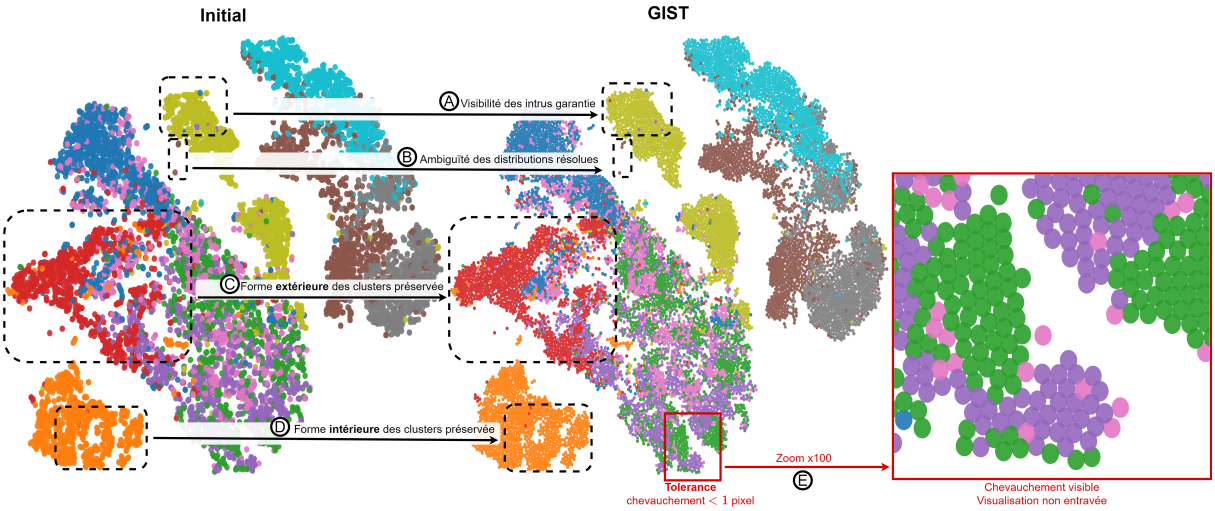


FIGURE IV.10 : Résultat de GIST [65] sur un jeu de données de 10 000 points pour une résolution 2000×2000. L'algorithme garantit que les nœuds soient visibles dans la résolution cible ⒶⒷ et maximise à la fois la taille des nœuds et la préservation du plongement initial ⒸⒹ. Sa convergence est facilitée par l'inclusion d'une *tolérance* aux chevauchements Ⓔ qui lui permet de gérer ces 10 000 nœuds en moins de 400*ms*.

visuel. Les jeux de données utilisés pour évaluer ces méthodes et les méthodes elles-mêmes sont conçus pour des données de type nuage de points.

IV.4.1 Algorithme GIST

Dans cette section, nous décrivons le fonctionnement de GIST. Inspiré de FORBID, il est constitué de deux composants : des mouvements de nœuds, et la recherche du diamètre optimal de la représentation des nœuds dans la résolution cible. L'objectif est de garantir la visibilité des nœuds tout en essayant de la maximiser, et de préserver les structures du plongement initial. Trouver un compromis entre ces différents critères est ce que nous appellerons *résoudre la tâche* dans la suite. Puisque GIST peut tolérer en partie les chevauchements, nous considérons qu'il ne produit pas des plongements *sans chevauchement* mais des plongements *solution* de la tâche.

IV.4.1.1 Recherche du Diamètre Optimal avec Tolérance aux Chevauchements

L'entrée de GIST est un plongement où les positions et le diamètre des nœuds sont considérés être dans l'espace géométrique. Pour produire un plongement qui garantisse la visibilité de tous les nœuds (*i.e.*, au moins 1 pixel) dans la résolution cible R, GIST manipule le plongement dans le GS (positionnement des nœuds) pour optimiser son rendu dans le VS (visibilité/taille des nœuds). La valeur de R donnée en entrée peut être modifiée si elle n'est pas adaptée pour produire une solution à la tâche. Par exemple, R est augmentée automatiquement quand $R^2 < N$ car il n'y aurait pas assez de pixels dans l'image pour que chaque nœud soit représenté avec au moins 1 pixel.

Comme présenté dans l'Algorithme 1, le composant principal de GIST est la recherche dichotomique (ligne 21) qui cherche le diamètre optimal des nœuds dans l'espace visuel de résolution R. Pour faciliter sa convergence, l'algorithme tolère les chevauchements tant que (i) ceux-ci ne sont pas plus grands que T pixels, et (ii) que le diamètre des nœuds reste supérieur à 1 pixel même une fois amputé de cette tolérance, ce qui garantie leur visibilité. Les mouvements

Algorithme 1 Pseudo-code de GIST. Les variables en **bleu** (resp. **vert**) correspondent à des valeurs dans l'espace **géométrique** (resp. **visuel**).

Methods
  visualToGeometric(v, R, X) : retourne la valeur correspondante à v de l'espace visuel R dans l'espace géométrique X
  geometricToVisual(v, R, X) : inverse de visualToGeometric
  tolerance(D, R, T, X) : retourne la valeur de la tolérance de GS telle qu'elle ne dépasse pas T pixels et que geometricToVisual($D-2*t$)>1 dans VS
  overlaps(X, D, t) : retourne l'ensemble des chevauchements avec t de tolérance
  moveNodes(X, O) : déplace les paires de nœuds qui se chevauchent en optimisant le Stress par SGD [39]
end Methods

Input Variables
  X^0 : Plongement initial, position des nœuds dans GS
  D : Diamètre initial des nœuds
  R : Résolution cible
  T : Tolérance en pixels dans la résolution cible
end Input Variables

procedure GIST (X^0, D, R, T)
  $minDiam \leftarrow 1$
  $maxDiam \leftarrow$ geometricToVisual(D, R, X)
  $stopBS \leftarrow false$
  $X' \leftarrow X^0$
  while not $stopBS$ **do**
    $diam \leftarrow (minDiam + maxDiam)/2$
    $X \leftarrow X'$
    $t \leftarrow$ tolerance($diam$, R, T, X)
    $O \leftarrow$ overlaps(X, visualToGeometric($diam$), t)
    $X \leftarrow$ moveNodes(X, O)
    $fail \leftarrow$ overlaps(X, visualToGeometric($diam$), t) $\neq \emptyset$
    if $fail$ **then**
      $maxDiam \leftarrow diam$ ▷ réduire le prochain diamètre
    else
      $X' \leftarrow X$ ▷ stocker la solution courante
      $minDiam \leftarrow diam$ ▷ augmenter le prochain diamètre
    if $maxDiam$ - $minDiam < \varepsilon$ **then**
      if $X' = X^0$ **then** ▷ recommencer avec un plus grand R
        $R \leftarrow R*2$
        $minDiam \leftarrow 1$
        $maxDiam \leftarrow$ geometricToVisual(D, R, X)
      else
        $stopBS \leftarrow true$
  return X'

de nœuds étant calculés dans le GS, une tolérance comprise entre 0 et 1 pixel dans le VS peut être utilisée par l'algorithme, car une fois convertie dans le GS, elle atténue tout de même les contraintes qui pèsent sur les mouvements de nœuds. C'est même un cas idéal dont GIST essaie de tirer parti : il permet de faciliter l'optimisation du Stress et des déplacements de nœuds tout en utilisant une tolérance qui, étant inférieure à 1 pixel, ne sera probablement pas visible dans la représentation finale rastérisée. Comme définie dans les lignes 25 et 27 de l'Algorithme 1, la tolérance est utilisée pour identifier les chevauchements. Deux nœuds i et j se chevauchent si $||X_i - X_j|| < r_i + r_j - t$ où t est la valeur de tolérance dans l'espace géométrique, adaptée pour s'assurer que les nœuds restent visibles.

A chaque passe P dans la recherche dichotomique (*i.e.*, pour chaque valeur de diamètre dans le VS de résolution R), les nœuds qui se chevauchent sont déplacés pour optimiser une fonction de Stress (ligne 26). La modélisation du Stress et les mouvements qui en découlent sont largement inspirés de FORBID (voir les Sections IV.3.1.1 et IV.3.1.2) et nous en détaillerons les différences dans la Section IV.4.1.2. S'il y a toujours des chevauchements après les mouvements de nœuds de la passe P, alors la passe $P+1$ commencera avec des nœuds ayant un diamètre plus petit, ce qui simplifie la tâche en créant de plus grands espaces vides où déplacer les nœuds (ligne 29). S'il n'y a plus de chevauchements à la fin de la passe P, le plongement courant est sauvegardé (ligne 31) et la passe $P+1$ essaiera de résoudre la tâche avec des nœuds plus grands (ligne 32). L'algorithme cherche donc le plus grand diamètre de la représentation des nœuds qui permet de résoudre la tâche

Enfin, lorsque la différence entre la borne supérieure et inférieure de la recherche dichotomique est inférieure à une valeur seuil ε (fixée à 10^{-3} dans nos expériences), la recherche est stoppée. À ce moment là, si la tâche n'est pas résolue (ligne 33), nous considérons qu'elle ne peut pas l'être avec la résolution cible R courante et nous recommençons l'algorithme avec une plus grande résolution (ici doublée, voir ligne 34–37). Nous reconnaissons que relancer entièrement l'algorithme du début est coûteux en calculs. En pratique, cela n'arrive que très rarement tant que la résolution cible initiale donnée à l'algorithme est pertinente compte tenu de la complexité de la donnée en entrée (*e.g.*, nombre de nœuds, de chevauchements). Cela permet également de converger vers une solution qui résout la tâche telle que nous la définissons, même si l'utilisateur n'a pas une bonne connaissance de la donnée manipulée. Finalement, si la tâche est résolue (lignes 33 et 38), la recherche est stoppée (ligne 39) et le plongement solution est retourné (ligne 40).

Puisqu'une des ambitions de GIST est de résoudre, au moins en partie, les problèmes de scalabilité de FORBID, nous n'en implémentons que la variante dans laquelle chaque passe P travaille à partir du plongement de la passe $P-1$. Il n'y a donc pas d'équivalent de FORBID' avec GIST.

IV.4.1.2 Déplacement de Nœuds

Comme vu tout au long de ce chapitre, les mouvements de nœuds doivent permettre de (i) supprimer les chevauchements et (ii) limiter les déformations du plongement initial. Les mouvements de nœuds dans GIST sont inspirés de FORBID : ils sont calculés pour optimiser une fonction de Stress (voir les Sections IV.3.1.1 et IV.3.1.2). Comme la représentation des nœuds dans les données utilisées dans cette section sont des cercles, la distance idéale de GIST entre des nœuds qui se chevauchent est la somme de leurs rayons. Comme dans FORBID, cette distance idéale est telle que les nœuds ne peuvent plus se chevaucher quelle que soit leur position relative.

Soit $O \in N \times N$ l'ensemble des chevauchements dans un plongement, où $(i, j) \in O$ si i et j se

chevauchent malgré la *tolérance* autorisée. Le changement principal opéré dans la fonctionnalité de déplacement des nœuds de GIST est l'ensemble sur lequel il opère. Dans FORBID, toutes les (N^2) distances entre les nœuds sont optimisées dans le Stress. Ici, pour améliorer la scalabilité de l'algorithme, GIST ne déplace que les nœuds qui se chevauchent (*i.e.*, toutes les paires de nœuds de O). Le Stress est là encore optimisé via descente de gradient stochastique [39] avec un nombre fixe d'itérations, correspondant au *budget* de mouvements autorisé. Malgré le choix de ne réaliser des mouvements que pour les paires de nœuds dans O, la complexité de cette optimisation reste quadratique puisque tous les nœuds se chevauchent dans le pire des cas ($|O| \leq N^2$). En pratique, nous verrons par la suite (voir Section IV.4.3) que cette optimisation est tout de même très efficace.

Formellement, le Stress optimisé par GIST peut être défini par :

$$\delta_{ij} = r_i + r_j, (i,j) \in O \tag{IV.9}$$

$$\sigma(X) = \sum_{i,j} W_{ij}(||X_i - X_j|| - \delta_{ij})^2 \tag{IV.1}$$

où W_{ij} est fixée à δ_{ij}^{-2}. Il est important de noter que la distance idéale entre les nœuds ne tient pas compte de la tolérance aux chevauchements. C'est cette différence entre la distance idéale pour l'algorithme d'optimisation des mouvements (qui ne tient pas compte de la tolérance), et la distance à laquelle nous considérons que les nœuds se chevauchent (qui tient compte de la tolérance), qui fait que notre utilisation de la tolérance ne revient pas simplement à réduire la taille des nœuds.

IV.4.2 Protocole d'Évaluation

IV.4.2.1 Jeu de données

Le jeu de données de cette évaluation est composé de 48 nuages de points provenant de deux sources. Tout d'abord, nous reprenons les données de ScatterplotUnfold [146]. Comme certains algorithmes de référence n'étaient pas capables de traiter les volumes de données de certains nuages de points, ils étaient évalués sur des sous-échantillons de taille $N = 3000$ dans leur article. Nous préférons évaluer tous les algorithmes sur les même données afin d'être en mesure de comparer de manière équitable leur efficacité sur une base commune. Nous ne conservons donc que les données de ScatterplotUnfold [146] où $N < 150\,000$, car tous les algorithmes de référence que nous considérons sont capables de les traiter.

Pour compléter le jeu de données, et contre-balancer la suppression des données avec $N > 150\,000$ du jeu précédent, nous y ajoutons 10 nuages de points avec $N \in [1024; 58\,509]$ provenant des domaines du Positionnement Multidimensionnel (MDS) et de l'Apprentissage Automatique (ML) [221–223] et dans lesquels les plongements 2D sont réalisés par l'algorithme t-SNE [35].

La Figure IV.11 présente la distribution des 48 données en fonction du nombre de nœuds et de chevauchements qu'elles contiennent. Dans tous ces nuages de points, le diamètre initial des nœuds passé en entrée aux algorithmes est fixé à $D = 2$. Les domaines de valeur des positions et de la taille des nœuds dans les plongements initiaux sont considérés être dans le GS.

IV.4.2.2 Métriques

Les métriques de qualité considérées dans cette évaluation diffèrent de celles utilisées pour FORBID. Elles sont toutefois toujours inspirées de l'étude de Chen *et al.* [49, 155], mais

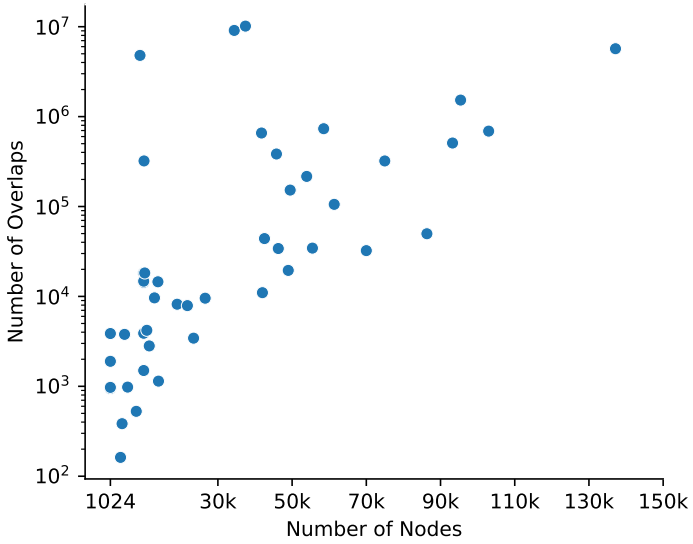


FIGURE IV.11 : Nombre de chevauchements en fonction du nombre de nœuds dans les 48 données de l'évaluation. L'axe Y a une échelle logarithmique. Une donnée avec $N = 10\,000$ n'est pas représentée car elle ne contient aucun chevauchement.

également de celle de Li *et al.* [146]. Certaines métriques ont été ajustées pour que leur résultat reflète mieux leur objectif. Puisque dans le VS, certains algorithmes (dont GIST) ne suppriment pas complètement les chevauchements, nous concevons également deux métriques pour évaluer la visibilité des nœuds dans le plongement solution produit par les algorithmes étudiés.

Mouvements de nœuds normalisés Cette métrique s'inspire de *nm_dm_imse* [49, 155] (voir Section IV.3.2.2) et quantifie la quantité de mouvement relative réalisée pour obtenir le plongement solution à partir du plongement initial. Avant de calculer cette quantité de mouvement, les deux plongements sont donc *superposés* pour ne pas que le domaine de valeur ou l'échelle des plongements ne soit quantifié. Contrairement à *nm_dm_imse*, nous projetons le plongement solution sur le plongement initial, et non pas l'inverse. En effet, la définition de *nm_dm_imse* pose problème lorsque l'on souhaite comparer des algorithmes dont les plongements solutions ne sont pas au même facteur d'échelle. Si deux algorithmes ont produit des plongements sans chevauchement X'_A et X'_B avec un facteur d'échelle de $\times 10$ et $\times 5$ respectivement, le score *nm_dm_imse* de chaque plongement est calculé dans le domaine de valeur de la solution de l'algorithme qui l'a produit. Or, ces domaines de valeurs sont différents ($\times 10$ et $\times 5$), ce qui signifie que les scores de *nm_dm_imse* pour X'_A et X'_B sont difficilement comparables. Dans l'évaluation de GIST, nous projetons le plongement solution dans le domaine de valeur du plongement initial qui sert alors de base commune et permet de comparer équitablement les plongements produits par différents algorithmes. Le résultat est également normalisé par le plus grand mouvement possible : la longueur de la diagonale de la boîte englobante rectangulaire du plongement.

Formellement, la métrique peut être définie comme :

$$NNM(X', X^0) = \frac{\frac{1}{N}\sum_{i=1}^{N} ||X_i^0 - \text{shift}(\text{scale}(X'_i))||}{\sqrt{BB_{width}^2 + BB_{height}^2}} \tag{IV.10}$$

où X' est un plongement solution et BB_{width} (resp. BB_{height}) est la largeur (resp. hauteur) de la

boîte englobante rectangulaire du plongement initial X^0. Les opérations $shift$ et $scale$ permettent de superposer les deux plongements et de les projeter dans le même domaine de valeur.

Préservation de forme Inspirée de Li *et al.* [146], cette métrique mesure la préservation de la forme globale du plongement initial. Elle consiste à dessiner plusieurs cercles concentriques de rayons croissants, centrés sur le centre de gravité du plongement initial, noté $\mu(X^0)$. Nous tirons uniformément 36 points $\left\{p_{i,k}, 1 \leq k \leq 36\right\}$ sur chaque cercle C_i. Pour chaque point $p_{i,k}$ de chaque cercle C_i, nous capturons le nœud le plus proche $X^0_{i,k}$ dans le plongement initial, sauf si $||X^0_{i,k} - C_i|| > T$ où T est une valeur seuil fixée à 10 comme proposé par Li *et al.* [146]. Dans ce cas, aucun nœud n'est associé au point $p_{i,k}$ et le cercle C_i est ignoré s'il n'a pas permis de capturer au moins 2 nœuds. Pour chaque nœud capturé, nous calculons le ratio de sa distance au centre de gravité dans le plongement solution $l'_{i,k} = ||X'_{i,k} - \mu(X')||$, sur sa distance au centre de gravité dans le plongement initial $l_{i,k} = ||X^0_{i,k} - \mu(X^0)||$. Enfin, nous calculons *Préservation de forme* comme la moyenne des coefficients de variation des ratios de distances aux centres de gravités sur tous les cercles.

$$SP(X', X^0) = \frac{1}{C} \sum_{i=1}^{C} \frac{std\left(\left\{\frac{l'_{i,k}}{l_{i,k}}, X^0_{i,k} \in C_i\right\}\right)}{\mu\left(\left\{\frac{l'_{i,k}}{l_{i,k}}, X^0_{i,k} \in C_i\right\}\right)} \tag{IV.11}$$

où C est le nombre de cercles initialement fixé à 20, mais qui peut être plus petit si certains cercles sont ignorés car ils n'ont pas permis de capturer au moins 2 nœuds.

Similarité d'ordre Empruntée à Li *et al.* [146], cette métrique mesure la préservation de l'ordre des nœuds suivant différents angles. Pour chaque angle, une droite est tracée par dessus le plongement et chaque nœud est projeté orthogonalement sur cet axe. Ce processus est réalisé sur le plongement initial et solution, nous donnant ainsi les séquences de l'ordre des nœuds sur l'axe dans chaque plongement. Un coefficient de corrélation de Kendall [224] est alors calculé et associé à l'axe. Nous obtenons alors le score de similarité entre l'ordre des nœuds dans le plongement initial et celui dans le plongement solution, en fonction de l'angle de l'axe. Comme proposé par Li *et al.* [146], nous réalisons cette évaluation pour 30 angles différents et le score final de la métrique est la moyenne des scores de similarité des axes.

Préservation des k-voisins Capture la stabilité du voisinage. C'est la moyenne, pour chaque nœud, de la portion de l'ensemble de ses k plus proches voisins dans le plongement initial qui sont parmi ses k plus proches voisins dans le plongement solution.

$$KNP(X', X^0) = \frac{1}{N} \sum_{i=1}^{N} \frac{|\mathrm{KNN}(X_i, k) \cap \mathrm{KNN}'(X'_i, k)|}{k} \tag{IV.12}$$

où KNN (resp. KNN′) retourne le k-voisinage d'un nœud dans le plongement initial (resp. solution). Soit $d(k)$ la distance entre un nœud et son k^{eme} voisin, il est possible qu'il y ait plus que k nœuds à distance au plus $d(k)$. Dans ce cas, nous choisissons dans les deux plongements l'ensemble de voisins le plus grand. Comme proposé par Li *et al.* [146] k est fixé à 10.

Préservation de densité Également tirée de Li *et al.* [146], cette métrique mesure la préservation de la densité de nœuds dans le plongement. Chaque nœud se voit associé une mesure de densité locale en utilisant un ensemble KNN. La mesure de densité locale est fixée à $\rho_i = \frac{1}{\frac{1}{k}\sum_{j \in \text{KNN}_i} ||X_i - X_j||}$. Les nœuds sont triés par leur valeur de densité locale, et se voient associés leur position q_i dans cet ordre. Le score de *Préservation de densité* est alors la moyenne sur tous les nœuds de la différence entre leur position dans l'ordre issu du plongement initial, et celui issu du plongement solution.

$$DP(X', X^0) = \frac{1}{N} \sum_{i=1}^{N} |q_i^0 - q_i'| \tag{IV.13}$$

Échelle Cette métrique mesure explicitement la préservation du facteur d'échelle du plongement initial. Le score de densité globale du plongement est calculé avec le ratio de la somme des aires des nœuds sur l'aire de la boîte englobante rectangulaire du plongement. La métrique est alors définie comme le ratio du score de densité du plongement solution sur celui du plongement initial.

$$Scale(X', X^0) = \frac{\sum_{i=1}^{N} {r_i'}^2 * \pi}{BB'_{area}} \Big/ \frac{\sum_{i=1}^{N} r_i^2 * \pi}{BB_{area}} \tag{IV.14}$$

où r_i (resp. r_i') est le rayon du nœud i dans le plongement initial (resp. solution), et BB_{area} (resp. BB'_{area}) est l'aire de la boîte englobante du plongement initial (resp. solution).

Nombre minimal de pixel visible de nœud Cette métrique quantifie le nombre de pixels exclusivement dédiés au nœud le moins visible dans le plongement solution une fois rendu dans une image de résolution $R \times R$ (fixée à 2000×2000 dans cette évaluation). La quantification des pixels *exclusivement dédiés* aux nœuds signifie que tous les pixels correspondant à une zone de chevauchement ne sont pas comptabilisés.

$$NMVP(X', X^0) = \min_{1 \leq i \leq N} |P_i| \tag{IV.15}$$

où $P_i = \left\{ p_{j,k}, 1 \leq j,k \leq R, n_i \text{ est le seul nœud dans le pixel } p_{j,k} \right\}$. L'intuition de cette métrique est que maximiser la surface du nœud le moins visible revient à améliorer la visibilité de toute l'information dans la représentation, puisque tous les nœuds d'un plongement sont au moins aussi visible que le moins visible d'entre eux.

IV.4.2.3 Algorithmes de Référence

Dans cette évaluation, nous comparons GIST à des méthodes d'OR dans l'espace visuel (compact ou non) et géométrique. Plus précisément, les algorithmes de référence sont ScatterplotUnfold (SU) [146], DGrid [163], HaGrid [145], PFS' [157], GTree [162] et PRISM [144]. Ces algorithmes sont décrits dans la Section II.3.2.

Pour l'évaluation, nous utilisons notre propre implémentation de GIST et PFS'. L'implémentation de ScatterplotUnfold est tirée du dépôt de code source de ses auteurs[1]. Nous utilisons l'implémentation de DGrid et HaGrid provenant du dépôt de HaGrid[2]. Ces 5 algorithmes ont été exécutés sur un processeur Intel Core i9-12900KF CPU. Les implémentations de PRISM

[1] www.github.com/diyike/scatterplotUnfold

[2] www.github.com/saehm/hagrid

et GTree sont tirées de la bibliothèque Microsoft Automatic Graph Layout (MSAGL) [225] et ont été exécutées sur un processeur Intel Core i7-9700K. Nous n'intégrons pas FORBID à cette évaluation car l'algorithme n'a pas été en mesure de fournir une solution sur les nuages de points avec $N > 50000$, ou a produit des plongements ayant sévèrement dégradé le plongement initial. Ce résultat était attendu puisque GIST est en partie une extension de FORBID qui s'attelle à améliorer la scalabilité de la méthode.

Tous les algorithmes ont été exécutés avec des paramètres fixes sur les 48 nuages de points du jeu de données présenté dans la Section IV.4.2.1. Les paramètres sont les suivants pour chaque algorithme :

GIST. La résolution cible R est fixée à 2000 et la tolérance dans l'espace visuel à 1 pixel. Pour l'optimisation du Stress, W_{ij} est fixée à δ_{ij}^{-2} (voir Équation IV.1). Le nombre d'itérations dans la descente de gradient stochastique est fixée à 200, avec un arrêt anticipé si une itération produit moins de 10^{-4} quantité de mouvement.

HaGrid. Nous utilisons les paramètres de la courbe de Hilbert définis dans [146], avec le paramètre recommandé $level = l_{min} = log_4(N)$ [145]. Le paramètre $keep\_aspect\_ratio$ est laissé à sa valeur par défaut définie par les auteurs.

DGrid. Le paramètre $aspect\_ratio$ est fixé à 1.

PRISM et GTree. Nous utilisons les paramètres par défaut de PRISM et GTree définis dans la bibliothèque Microsoft Automatic Graph Layout (MSAGL) [225]. Le paramètre *node separation* est fixé à 0.

SU et PFS'. Les paramètres de ces algorithmes sont fixés à leur valeur par défaut.

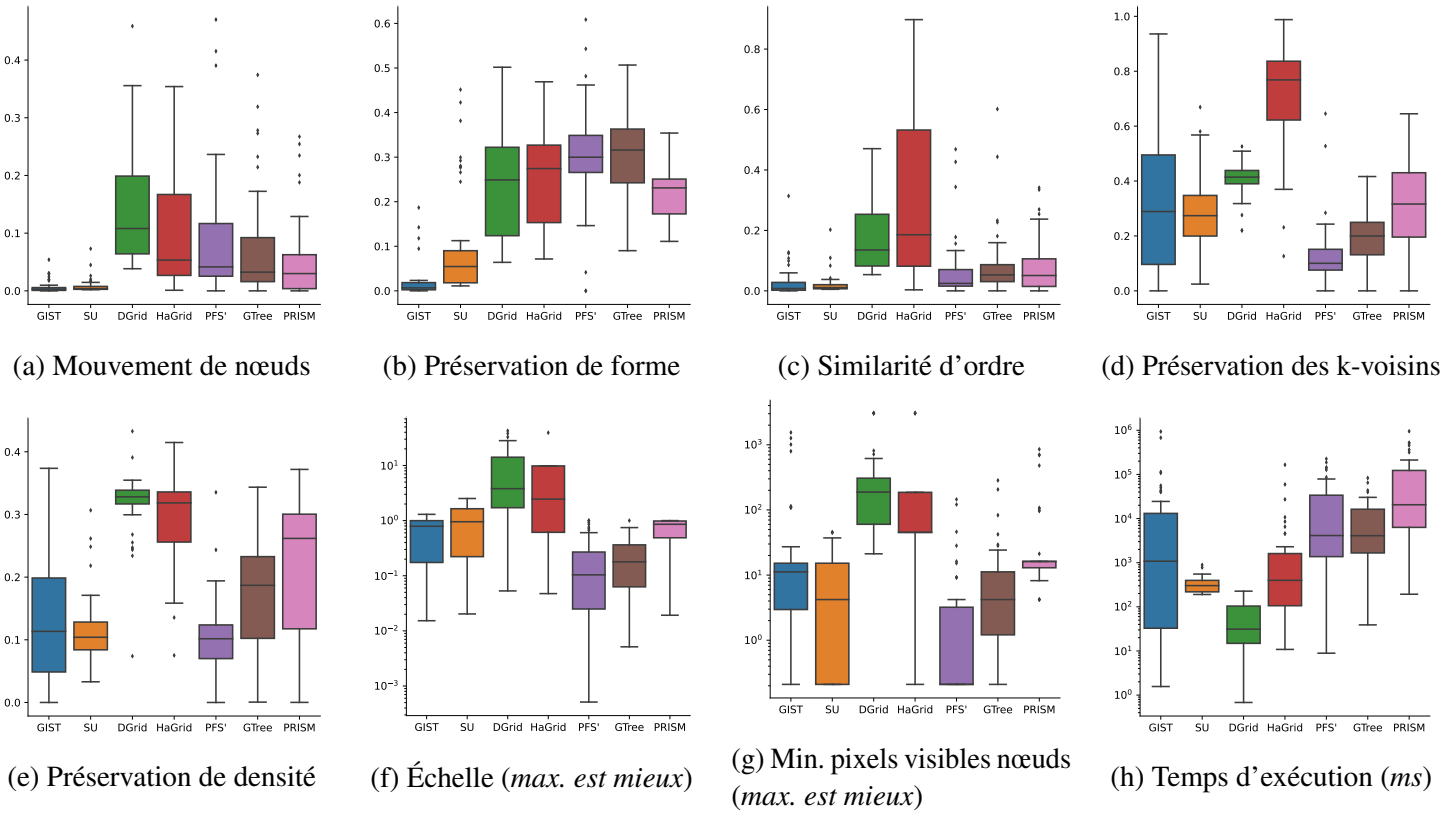
IV.4.3 Évaluation Quantitative

Les scores de métriques des algorithmes sélectionnés sur le jeu de données de 48 nuages de points (voir Section IV.4.2) sont présentés dans la Figure IV.12 et discutés ci-après. Les résultats sont étudiés en fonction de trois critères : la préservation du plongement initial, la visibilité des nœuds et le temps d'exécution.

Minimiser la déformation du plongement initial. Pour mesurer à quel point le plongement solution préserve les structures du plongement initial, nous nous concentrons sur les 5 premières métriques de la Figure IV.12. Nous pouvons observer que GIST surpasse les algorithmes de référence sur *Mouvements de nœuds* (IV.12a), *Préservation de forme* (IV.12b) et *Similarité d'ordre* (IV.12c). Ces trois métriques capturent la préservation globale du plongement initial, ce qui démontre les bonnes capacités de GIST sur ce critère. La différence est moins grande avec ScatterplotUnfold (SU) sur *Similarité d'ordre* et *Mouvement de nœuds*, mais les résultats restent en faveur de GIST sur ces métriques d'une manière générale.

Sur *Préservation de densité* (IV.12e) et *Préservation des k-voisins* (IV.12d), la performance médiane de GIST est au même niveau que SU, mais la variation de son score est plus grande. Ces deux métriques sont plutôt dédiées à la mesure de la préservation des structures locales du plongement initial. Ces résultats ne semblent pas étonnants. En effet, le choix des distances idéales entre les nœuds (voir Section IV.4.1.2) autorise leur déplacement dans toutes les directions, sans contrôle. De plus, puisque GIST n'optimise que les distances entre les paires de nœuds qui se chevauchent, il perd petit à petit la connaissance de l'aspect du plongement initial. Si ceci n'impact pas la structure globale du plongement initial (comme vu précédemment), cela semble bien avoir un effet sur les structures locales.

D'une manière générale, les performances de HaGrid et DGrid ne sont pas satisfaisantes, et ils n'optimisent pas la préservation du plongement initial. Ces résultats étaient attendus puisque tous les deux sont conçus pour produire des visualisations compactes, maximisant la visibilité



(a) Mouvement de nœuds (b) Préservation de forme (c) Similarité d'ordre (d) Préservation des k-voisins

(e) Préservation de densité (f) Échelle (*max. est mieux*) (g) Min. pixels visibles nœuds (*max. est mieux*) (h) Temps d'exécution (*ms*)

FIGURE IV.12 : Scores des métriques de qualité sur les 48 nuages de points du jeu de données. Un score plus faible signifie une meilleure qualité pour toutes les métriques sauf (f) et (g). Les trois derniers graphiques ont une échelle logarithmique sur l'axe Y. Les métriques (a), (b) et (c) capturent la préservation de la forme globale du plongement initial. (d) et (e) se concentrent sur la préservation des structures locales du plongement initial. Enfin, (f) et (g) capturent la visibilité des nœuds dans le plongement solution.

des nœuds (comme nous allons le voir dans la suite). Les résultats de PFS', GTree et PRISM sur ce critère sont mitigés. Aucun d'entre eux n'obtient de bon résultat sur aucune métrique, et la tendance est relativement la même entre ces trois méthodes sur chaque métrique. Ces tendances corroborent d'ailleurs les résultats de l'évaluation de Chen *et al.* [49], à savoir : PFS' est sensiblement meilleur que GTree, lui même légèrement meilleur que PRISM. GIST est principalement en concurrence avec SU sur ce critère de préservation du plongement initial. La Figure IV.13 illustre les mouvements de nœuds relatifs produits par GIST sur deux exemples. Chaque ligne représente le mouvement relatif d'un nœud de sa position dans le plongement initial jusqu'à sa position dans le plongement solution. Ces exemples démontrent que les mouvements de nœuds préservent la forme globale du plongement initial, et qu'il peut y avoir des déformations des structures locales puisque c'est là que se concentre la majorité des mouvements.

Maximiser la visibilité des nœuds. Ce critère est évalué au regard des métriques *Échelle* (IV.12f) et *Min. pixels visibles nœuds* (IV.12g). À l'inverse des métriques étudiées précédemment, celles-ci doivent être maximisées (*i.e.*, une valeur plus élevée signifie une meilleur qualité) et capturent la visibilité des nœuds.

Échelle mesure à quel point la visibilité des nœuds dans le plongement solution est représentative de leur visibilité dans le plongement initial. Cette mesure est réalisée en fonction de la surface des nœuds par rapport à la taille de la boîte englobante du dessin. Les scores peuvent être supérieurs à 1 si le plongement solution est plus compact que l'initial. Comme

FIGURE IV.13 : Mouvements relatifs de nœuds dans deux plongements produits par GIST. Le plongement solution est superposé avec le plongement initial, puis une ligne est tracée de la position de chaque nœud dans le plongement initial à sa position dans le plongement solution. Nous observons que les mouvements de nœuds sont locaux, uniformes et préservent les structures des nuages de points.

nous nous y attendions, cela est notamment visible dans les plongements produits par DGrid et HaGrid puisqu'ils organisent leur solution en grille relativement (voire totalement) compacte. Le score d'*Échelle* de GIST est proche de celui de SU et PRISM, tous les trois étant proches de 1. Ce résultat signifie qu'ils produisent des plongements solution dans lesquels les nœuds sont proportionnellement aussi visibles que dans le plongement initial. Ces scores montrent la bonne capacité de ces algorithmes à utiliser les espaces vides existants pour résoudre la tâche.

Les scores de *Min. pixels visibles nœuds* quantifient le nombre de pixels exclusivement réservés à l'affichage du nœud le moins visible dans une résolution 2000×2000. S'assurer que le nœud le moins visible soit représenté par au moins $p_{min} = 1$ pixel qui lui est dédié signifie que *tous* les nœuds sont visibles. Maximiser p_{min} permet alors d'améliorer la visibilité de tous les nœuds, puisqu'ils ont tous nécessairement une surface supérieure ou égale à p_{min}. Là encore, DGrid et HaGrid obtiennent les meilleurs scores puisqu'ils produisent des visualisations compactes. Comme attendu, dans la majorité des plongements produits par GIST, le nœud le moins visible est représenté par au moins 1 pixel, cette condition étant explicitement optimisée par l'algorithme. Les seules exceptions sont les cas où GIST n'a pas trouvé de solution dans la résolution cible $R = 2000$ qui lui a été demandée. GIST a obtenu un score $p_{min} < 1$ sur 8 données, contre 14 pour SU, 4 pour HaGrid, 28 pour PFS' et 10 pour GTree. SU et PFS' ont également échoué sur les 8 cas où GIST obtient $p_{min} < 1$. GTree a trouvé une solution pour l'un de ces 8. DGrid, HaGrid et PRISM ont réussi à trouver une solution pour ces 8 cas, mais au prix de sévères déformations du plongement initial (voir la Figure IV.14). Malgré ces cas problématiques, la distribution des scores de GIST sur cette métrique montre que $p_{min} > 1$ sur la grande majorité des données. Les résultats de SU et GTree sont légèrement en deçà de ceux de GIST, et PFS' produit les plongements où les nœuds les moins visibles ont la moins grande surface.

Temps d'exécution. Étant donné que GIST est construit sur la même idée que FORBID, sa complexité hérite de la sienne : $\mathcal{O}(s(N^2 + N\log(N))$ où s est la profondeur de la recherche dichotomique. La partie la plus coûteuse de cette complexité correspond aux N^2 mouvements calculés par FORBID entre toutes les paires de nœuds. Bien que GIST allège fortement cette contrainte en ne calculant un mouvement qu'entre les paires de nœuds qui se chevauchent, sa

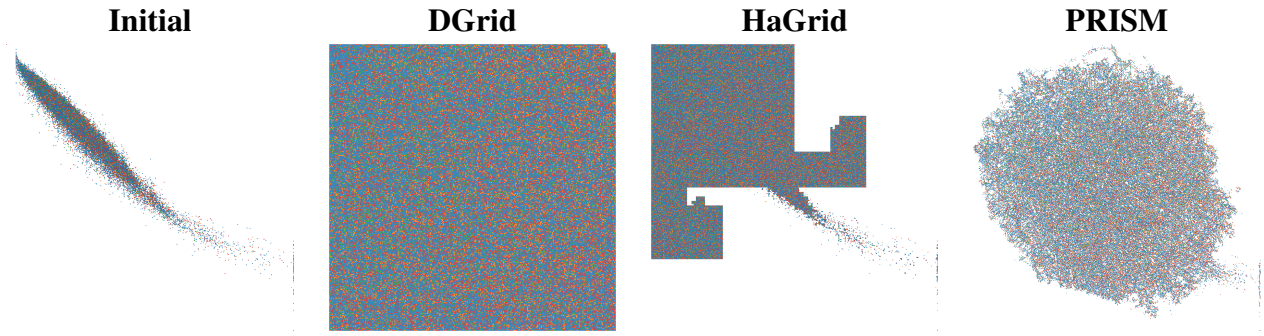


FIGURE IV.14 : Exemple d'un des 8 cas problématiques de GIST auquel DGrid, HaGrid et PRISM trouvent une solution où les nœuds sont visibles (*i.e.*, diamètre > 1 pixel dans une résolution 2000×2000). Comme nous l'observons, ces *solutions* sont trouvées au prix de sévères déformations du plongement initial.

complexité dans le pire des cas reste quadratique sur le nombre de nœuds. En effet, dans le pire cas, tous les nœuds se chevauchent et il y a donc $|O| = N^2$ mouvements à calculer. La complexité réelle de GIST est donc $\mathcal{O}(s(|O| + N\log(N))$ où $|O|$ est le nombre de chevauchements.

Néanmoins, la Figure IV.12h montre que le temps d'exécution de GIST est meilleur que les autres algorithmes quadratiques de la littérature (*i.e.*, PFS', GTree et PRISM). Cependant, il perd face aux temps d'exécution de SU, DGrid et HaGrid qui ont tous les trois une complexité linéarithmique. Les cas problématiques pour GIST sont les nuages de points où la distribution des nœuds est très dense dans certaines régions, tandis que d'autres sont très clairsemées. La ligne du haut de la Figure IV.15 illustre un cas problématique. Dans cet exemple, la solution de GIST a été trouvée à une résolution cible $R = 4000$. Puisque GIST essaie de minimiser la déformation du plongement initial, l'algorithme ne déplace les nœuds que pour un certain *budget* de mouvements. Si les chevauchements sont confinés dans une petite zone très dense, le budget de mouvements fixe peut être insuffisant pour occuper les espaces vides et résoudre les chevauchements. Ces cas problématiques étaient déjà identifiés par Li *et al.* [146] et la plupart des algorithmes d'OR sont affectés par ce type de distribution. À l'inverse, nous pouvons observer que GIST est très efficace sur des nuages de points où les nœuds sont plus uniformément distribués dans la boîte englobante du plongement initial. Un exemple de cas facile est présenté dans la ligne du bas de la Figure IV.15. En pratique, nous n'avons pas rencontré beaucoup de cas problématiques puisqu'une bonne partie de notre jeu de données est issue de domaines en lien avec la Réduction de Dimension. Dans ce type de données, l'utilisation de l'espace est optimisé pour mettre en évidence la proximité des nœuds qui sont souvent organisés en clusters étalés. Les cas problématiques apparaissent plutôt lorsqu'une zone très dense de nœuds est confinée dans une petite zone de la boîte englobante du dessin à cause d'autres zones clairsemées. Puisque beaucoup de méthodes ont pour objectif de préserver la forme globale du plongement initial, il est nécessaire de respecter cette distribution de nœuds déséquilibrée, et il devient alors complexe de résoudre la tâche. Nous pensons qu'il est possible d'anticiper ces cas problématiques en mesurant *a priori* les distributions de distances ou densité de nœuds dans l'espace du plongement initial. Par exemple, le score de Sparsity [226] des nuages de points dans la Figure IV.15 sont 894 pour la donnée du haut contre 146 pour celle du bas.

Dans cette évaluation, nous avons identifié trois catégories d'algorithmes. (i) Les méthodes telles que GIST et SU sont les mieux adaptées à la préservation des formes du plongement initial. Ces méthodes sont en concurrence avec (ii) les algorithmes d'OR par déplacement de

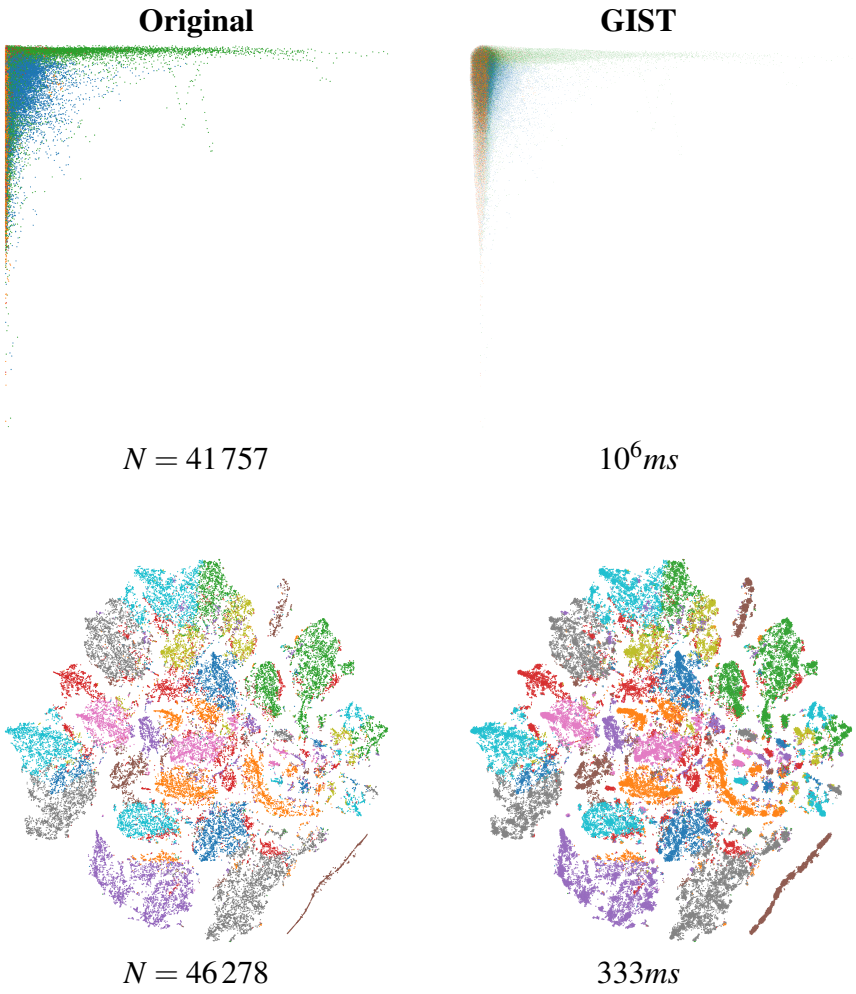


FIGURE IV.15 : Exemple d'un cas *problématique* (en haut) et d'un cas *facile* (en bas) pour GIST. Alors que les deux nuages de points ont sensiblement la même taille ($N = 41\,757$ contre $N = 46\,278$), GIST est plus rapide ($333ms$) sur l'exemple du bas que sur celui du haut ($10^6 ms$).

nœuds standards tels que PFS', GTree et PRISM. Pour ceux-ci, les résultats de notre évaluation corroborent ceux obtenus dans l'étude de Chen *et al.* [49, 155] et dans l'évaluation de FORBID (voir Section IV.3.3) : PFS' est meilleur que PRISM et GTree, ces deux derniers obtenant des résultats assez similaires. La similarité entre les tendances observées pour PRISM et GTree était attendue puisque GTree est une extension de PRISM. Enfin, (iii) DGrid et HaGrid illustrent la dernière catégorie d'algorithmes en produisant des plongements extrêmement compacts où les nœuds sont les plus visibles mais où le plongement initial est sévèrement déformé. Cette troisième catégorie d'algorithmes ne devrait alors être utilisée que si une visualisation compacte est en effet la représentation souhaitée. En revanche, cette évaluation a démontré que des approches comme GIST ou SU sont préférables aux algorithmes standards de déplacement de nœuds dans le GS pour préserver le plongement initial tout en maximisant la visibilité des nœuds. GIST et SU ont aussi montré une meilleure capacité à s'adapter aux données volumineuses. Enfin, les tendances des scores de métriques proposés par GIST suggèrent que ses plongements solutions sont plus appréciables que ceux de SU, ce que nous vérifierons dans la Section IV.4.4.1 sur des exemples visuels.

IV.4.4 Discussion

Cette section compare GIST avec les algorithmes de référence sur des exemples visuels, et discute de l'impact de la variation de la *tolérance*, et sa relation avec la *résolution cible*.

IV.4.4.1 Exemples Visuels

La Figure IV.16 présente les plongements solutions de GIST et des algorithmes de référence considérés dans l'évaluation quantitative (voir Section IV.4.3). Ces exemples couvrent la quasi-totalité des tailles de nuages de points du jeu de données (*i.e.*, de $N = 1024$ à $N = 93239$). Ils ne contiennent pas de cas problématique comme ceux identifiés plus tôt (voir Section IV.4.3). Dans cette section, nous discutons l'aspect des plongements *réussis* par les différents algorithmes.

News popularity. Nous observons que *News popularity* n'a pas été dessiné de manière satisfaisante par PFS' ni GTree. Dans leur dessin, la taille des nœuds dans l'image est significativement réduite et les structures du plongement initial sont à peine reconnaissables. Bien que le dessin de PRISM soit légèrement meilleur, il n'est pas non plus satisfaisant au regard des dessins produits par les algorithmes restants. DGrid produit une visualisation compacte où tous les nœuds sont visibles. Bien que nous puissions retrouver la structure « bleue » du plongement initial, la topologie de la donnée est sévèrement détériorée. Pour les trois algorithmes restants (*i.e.*, GIST, SU et HaGrid), nous considérons que le plongement solution préserve le plongement initial, à la fois dans ses structures locales et globales. Le dessin de GIST contient les nœuds les plus visibles (*i.e.*, les plus grands), suivi par HaGrid puis SU.

MoCap et GaAsH6. Contrairement à *News popularity*, tous les algorithmes produisent un dessin relativement interprétable de *MoCap* et *GaAsH6*. Les nœuds dans le dessin de PFS' sont très petits, mais les structures des plongements initiaux sont reconnaissables malgré quelques déformations. Les plongements de GTree ne sont pas très satisfaisants non plus. La visibilité des nœuds est légèrement meilleure que dans les dessins de PFS', mais le plongement initial est plus fortement détérioré. PRISM produit un dessin très compact de *MoCap* qui déforme plus sévèrement le plongement initial. En revanche, son dessin de *GaAsH6* est satisfaisant. En effet, les nœuds sont plus grands que dans les dessins des autres algorithmes, et les structures du plongement initial sont bien préservées. Nous observons que les dessins de HaGrid induisent de larges déformations, surtout dans les zones denses de ces deux nuages de points. Nous constatons aussi distinctement l'apparition d'artefacts du positionnement en grille de HaGrid. La même observation peut être faite des dessins de DGrid. Son dessin compact de *GaAsH6* est plutôt satisfaisant car les différents clusters sont bien délimités et imbriqués. À l'inverse, son dessin de *MoCap* rend difficile la lecture des agencements de clusters du plongement initial, la distribution des clusters donnant un effet de désordre. Enfin, les dessins de GIST et SU ont préservé le plongement initial. Nous considérons que le dessin de GIST est plus satisfaisant car les nœuds sont plus grands et les clusters plus faciles à identifier.

Elec board. Ce nuage de points illustre une limitation majeure des méthodes qui produisent des visualisations compactes. Au regard des dessins produits par DGrid et HaGrid, il est évident que cette approche ne peut pas produire de résultat satisfaisant sans l'utilisation d'autres variables d'encodage permettant d'identifier les nœuds et les structures topologiques (*e.g.*, couleur). PRISM et GTree ont sévèrement déformé le plongement initial. Bien que PFS' ait réussi à le préserver, les nœuds dans son dessin sont trop petits pour que la visualisation soit lisible, en particulier

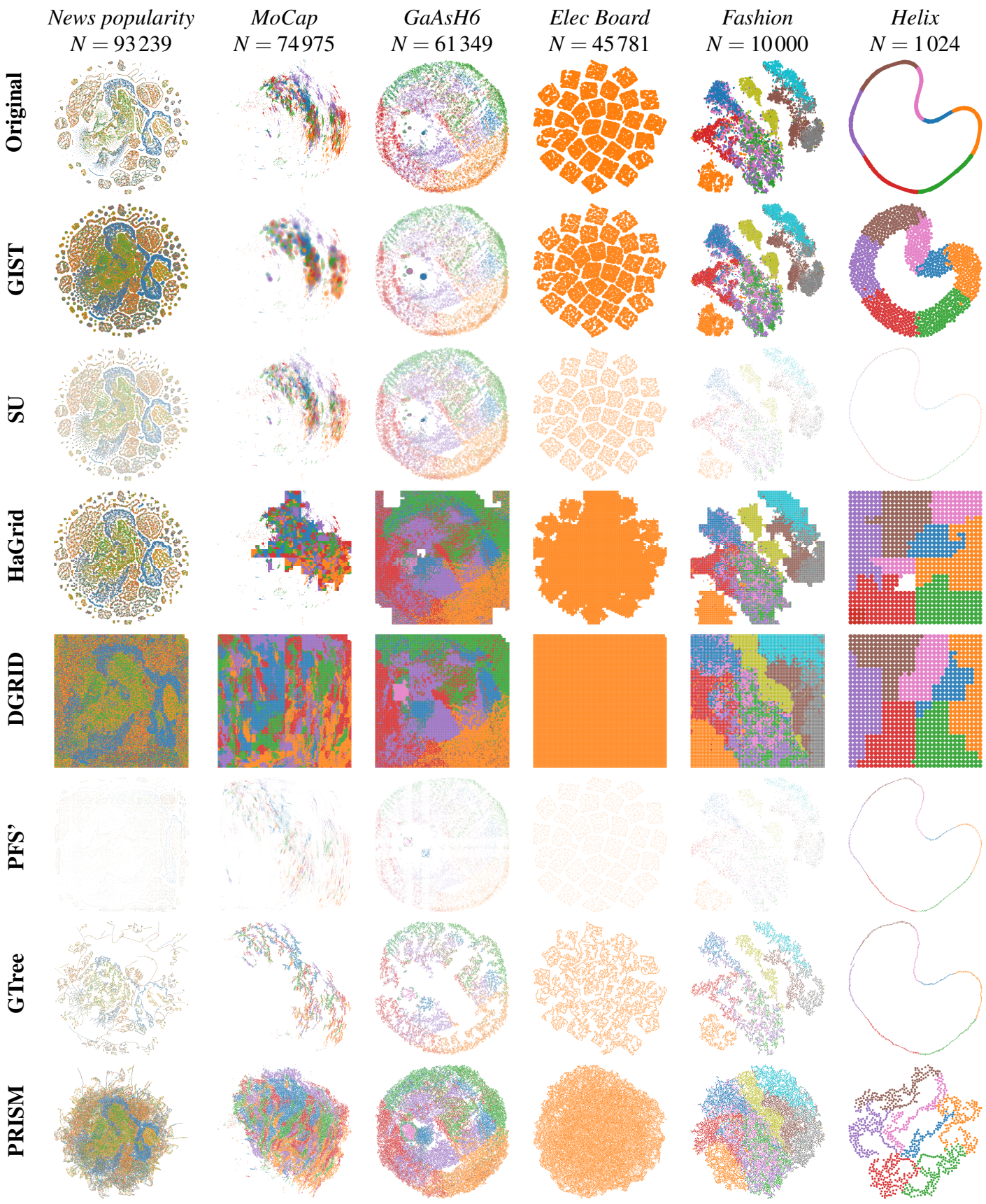


FIGURE IV.16 : Exemples de plongements solutions de GIST, ScatterplotUnfold (SU), DGRID, HaGrid, PFS', GTree et PRISM. Les images ont une résolution de 2000 × 2000px sans anti-aliasing, et les nœuds ont une opacité de 90%. Elles sont rendues avec la bibliothèque Python Matplotlib [227], basée sur la bibliothèque de rendu C++ Anti-Grained Geometry [228]. Il est recommandé de lire cette figure dans un environnement numérique capable de zoomer (jusqu'à 6000%).

comparé aux dessins des algorithmes restants. Ici encore, GIST et SU proposent les dessins les plus satisfaisants, préservant l'agencement des clusters du plongement initial. Les nœuds sont toutefois nettement plus grands dans le plongement de GIST que dans celui de SU.

Fashion et Helix. Sur ces deux derniers nuages de points, les plus petits en nombre de nœuds, PRISM a significativement endommagé le plongement initial. Les déformations visibles dans *Helix* mettent d'ailleurs bien en évidence l'utilisation du *Graphe de proximité* utilisé par l'algorithme (voir Section II.3.2). Bien que les dessins de GTree, PFS' et SU aient préservé le plongement initial, ils ont également significativement réduit la taille des nœuds à tel point que les représentations sont difficilement lisibles. Les dessins de DGrid et HaGrid sont peu satisfaisants. Malgré la *gridification* des dessins, nous reconnaissons les formes des clusters de données de *Fashion*. Sur ce dernier, Le plongement produit par HaGrid est plus satisfaisant que celui de DGrid car la compacité de la grille ne semble pas nécessaire à la bonne visibilité des éléments. *Helix* faisant seulement 1024 nœuds, la gridification de DGrid et HaGrid n'est probablement pas nécessaire et laisse penser que certains clusters sont adjacents (*e.g.*, rose et rouge pour DGrid, rose et bleu pour HaGrid). Ce type d'agencement peut provoquer une mauvaise interprétation des données par l'utilisateur final et devrait uniquement être privilégié lorsqu'il est nécessaire. Enfin, les dessins de GIST sur ces deux exemples sont sans aucun doute les meilleurs, ayant à la fois quasi-parfaitement préservé les structures des plongements initiaux tout en maximisant la taille des nœuds.

D'une manière générale, nous avons observé que PRISM, GTree et PFS' (*i.e.*, les algorithmes d'OR dans GS) ne produisent pas des dessins satisfaisants sur des nuages de points volumineux (en nombre de nœuds), même lorsque nous n'étudions que des cas qui ne sont pas problématiques. Les visualisations compactes produites par DGrid et HaGrid peuvent également être satisfaisantes, dans la mesure où elles sont nécessaires. Ces méthodes déformant sévèrement le plongement initial, leur utilisation ne remplit un objectif de lisibilité que si la compacité de la visualisation est réellement nécessaire. Finalement, les dessins produits par GIST et SU proposent une meilleure optimisation de la préservation du plongement initial, et ceux de GIST maximisent la visibilité des nœuds.

IV.4.4.2 Augmenter la Tolérance, Réduire la Résolution

Comme présenté dans la Section IV.4.1, la *tolérance* aux chevauchements a été fixée à 1 pixel dans l'espace visuel (VS) de résolution $R = 2000$ dans notre expérience. Ce choix nous a semblé évident pour faciliter la convergence de l'algorithme tout en demeurant inoffensif pour la lisibilité de la visualisation.

Cependant, il est trivial dans l'implémentation de GIST de gérer des tolérances plus élevées. Cela peut par exemple être utile pour produire des visualisations plus compactes. Cependant, il est important de s'assurer que les nœuds aient toujours au moins 1 pixel exclusivement dédié à leur propre représentation. Formellement, il faut toujours que $D - 2 * t > 1$ où D et t sont respectivement le diamètre des nœuds et la tolérance dans l'espace visuel. S'assurer que la *surface* du nœud soit supérieure à 1 pixel ne suffit pas à garantir sa visibilité car la phase de rastérisation (qui n'est pas gérée dans GIST) pourrait ne pas l'afficher s'il ne fait pas au moins 1 pixel de large. En réalité, ce paramètre de *tolérance* permet de gérer la compacité du plongement produit et la vitesse de la convergence de l'algorithme.

La Figure IV.17 présente le plongement solution de GIST sur le nuage de points *Fashion* avec les valeurs de tolérance $t \in \{1, 3, 6, 10\}$ dans une résolution cible $R = 2000$. Il n'est pas nécessaire

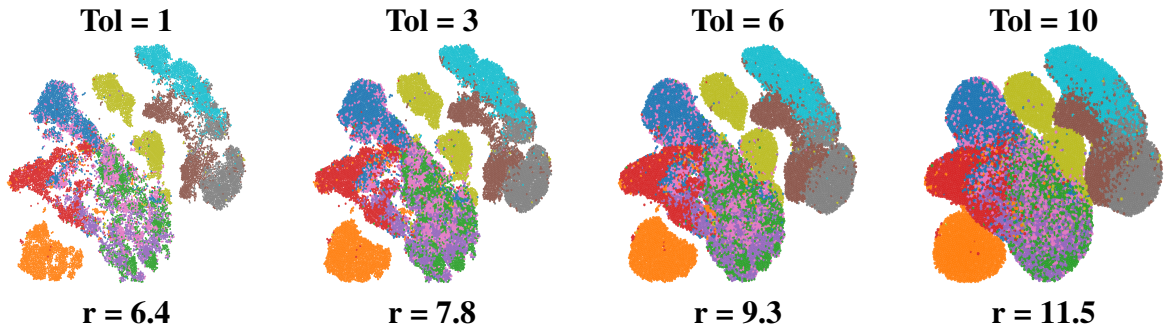


FIGURE IV.17 : Plongements solutions de GIST sur le nuage de points *Fashion* en faisant varier la tolérance. La résolution cible est fixée à $R = 2000$ et le rayon des nœuds dans l'espace visuel (VS) des plongements solutions sont reportés en dessous des figures. Ces rayons sont des nombres réels après reconversion des valeurs de l'espace géométrique utilisé pour calculer les déplacements de nœuds. La rastérisation est effectuée *a posteriori* par la génération de l'image.

d'expérimenter une tolérance plus élevée car GIST ne peut pas s'en servir en garantissant que $D - 2 * t > 1$. En dessous de chaque dessin est reporté le rayon des nœuds dans le plongement solution. Ces figures montrent qu'augmenter la tolérance n'est pas nécessairement bénéfique. En effet, notre objectif étant de garantir la *visibilité* des nœuds, nous allons considérer le *pire* scénario. Dans celui-ci, la tolérance t est entièrement utilisée, de sorte que le diamètre visible des nœud est exactement $D - 2 * t$. Ainsi, lorsque le rayon d'un nœud est 6.4 et avec une tolérance $t = 1$, le rayon minimum visible du nœud est $r_{min} = 6.4 - 1 = 5.4$ pixels. Avec en plus la rastérisation de l'image, dans le pire cas, ce rayon pourrait être diminué à $r_{min} = \text{floor}(5.4) = 5$ pixels. En suivant ce raisonnement, nous observons que le rayon visible garanti par GIST diminue $r_{min} \in \{5,4,3,1\}$ à mesure que la tolérance augmente $t = \{1,3,6,10\}$. Bien qu'il ne soit pas visuellement bénéfique d'augmenter cette tolérance, cela peut toutefois être un moyen efficace pour améliorer le temps d'exécution de l'algorithme et l'inclure dans des scénarios d'exploration interactive (voir la démonstration de GIST [67]).

Dans nos expériences, la résolution cible R était également fixée à $R = 2000$. Faire varier ce paramètre en fixant la valeur de la tolérance a pratiquement le même effet que faire l'inverse (vu précédemment). En effet, la combinaison de la résolution cible avec la tolérance est ce qui détermine la quantité d'espace vide que l'algorithme peut utiliser pour déplacer ou modifier la taille des nœuds. Avoir une tolérance $t = 1$ dans un espace de résolution $R = 1000$ produit le même effet qu'avoir $t = 2$ dans $R = 2000$. La seule différence qui peut en résulter dépend de l'uniformité de la rastérisation de l'algorithme de rendu d'image. En pratique, nous recommandons de fixer la résolution R en fonction du cas d'application ou de l'environnement d'affichage désiré, puis de faire varier la tolérance t pour gérer la vitesse d'exécution de l'algorithme et l'aspect visuel du plongement solution.

IV.4.4.3 Limitation

Une limitation de l'algorithme qui n'a pas encore été abordée est le comportement de GIST lorsque la recherche dichotomique se termine sans qu'une solution n'ait été trouvée. Ce cas arrive majoritairement sur les nuages de points complexes où l'algorithme converge difficilement vers une solution, ou quand la résolution cible est bien trop faible par rapport à la complexité du plongement initial (*e.g.*, nombre de nœuds, distribution des nœuds dans le plan). Dans ces cas, il est peu satisfaisant de recommencer systématiquement l'algorithme en doublant simplement la résolution cible. Cela revient à appeler plusieurs fois GIST en changeant automatiquement le

paramètre en entrée. Cette stratégie est très coûteuse en calcul et n'est probablement pas idéale. Un pré-traitement plus intelligent, consistant à déterminer une résolution cible raisonnable en fonction de certains critères du plongement initial, permettrait de préemptivement raffiner la résolution cible et constituerait une approche plus approprié.

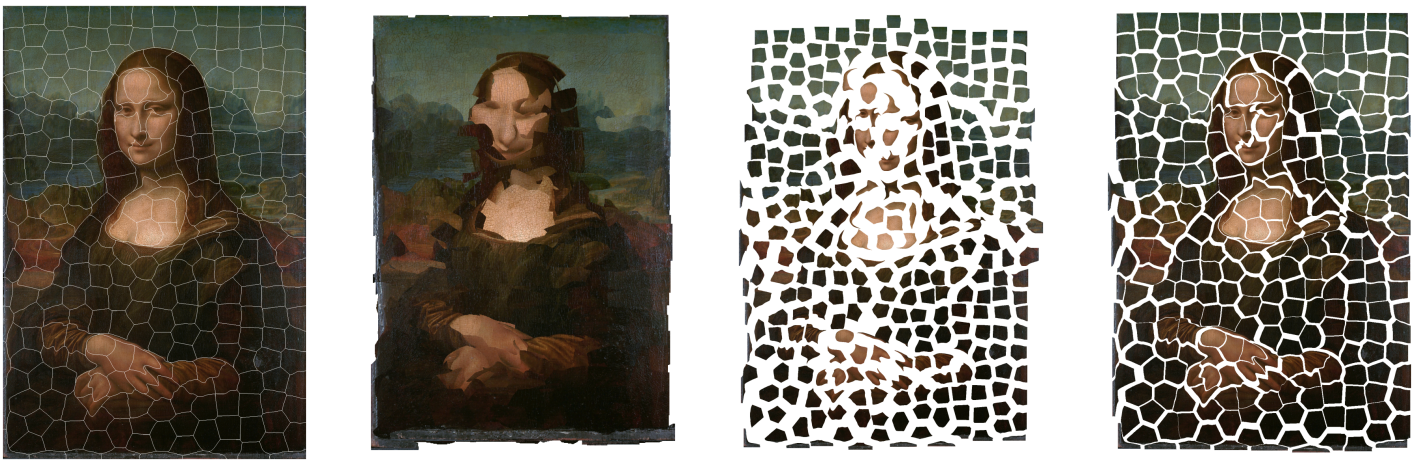
Bien que nous percevions ce comportement comme une limitation à cause de son coût calculatoire, il est le résultat d'un choix de conception que nous avons fait pour nous assurer que GIST converge vers un plongement solution préservant l'initial. Nous aurions pu décider d'autoriser les déplacements de nœuds jusqu'à ce que l'algorithme converge. Cependant, un tel choix aurait signifié de déformer le plongement initial et ne permettrait pas de remplir l'objectif de GIST. Il existe déjà des algorithmes d'OR qui autorisent de telles déformations, par exemple pour produire des visualisations compactes. Nous pensons que GIST est une alternative novatrice qui propose un compromis explicite entre la visibilité des nœuds et la préservation du plongement initial.

IV.5 Conclusion

Dans ce chapitre, nous avons présenté FORBID et GIST, deux algorithmes de suppression de chevauchements basés sur le déplacement de nœuds optimisant une fonction de Stress via descente de gradient stochastique. L'idée de ces algorithmes est de combiner (i) une recherche dichotomique pour gérer uniformément la quantité d'espaces vides utilisable dans le plongement, et (ii) des déplacements de nœuds pour supprimer les chevauchements. Cette conception permet aux deux algorithmes de produire un plongement où les nœuds sont visibles en proposant un compromis unique entre la préservation du plongement initial et la taille des nœuds (ou échelle) dans le plongement produit. La différence majeure entre ces deux algorithmes est l'espace dans lequel ils travaillent. FORBID est une approche dans l'Espace Géométrique, ayant pour objectif la suppression stricte des chevauchements. GIST est une adaptation de FORBID à l'Espace Visuel qui inclut une tolérance aux chevauchements. Travaillant dans l'Espace Visuel, GIST utilise et optimise des propriétés de rendu du plongement (*i.e.*, résolution cible de l'image, taille des nœuds en pixels) pour améliorer la scalabilité de la méthode, désormais capable de traiter des dizaines de milliers de nœuds en temps raisonnable, voire interactif (voir démonstration en ligne de GIST [67]). L'efficacité et la différence entre ces deux approches ont pu être observées au travers d'évaluations quantitatives et qualitatives avec des algorithmes de référence de la littérature.

La conception et l'évaluation de ces deux algorithmes nous a mené à considérer différentes pistes d'évolution. Une piste serait de trouver une manière de réellement améliorer la complexité quadratique des deux algorithmes. Nous pensons par exemple qu'il est possible de concevoir une version hiérarchique de notre algorithme. Puisque les chevauchements apparaissent dans la plupart des cas entre des nœuds adjacents, une approche hiérarchique permettrait de résoudre les chevauchements entre groupes du même niveau, puis à l'intérieur de chaque groupe ; et aurait une complexité nécessairement moindre. Une autre approche pour améliorer la complexité de la résolution de cette tâche pourrait être d'utiliser des réseaux de neurones. Puisque nous savons qu'il est possible de *produire* des plongements de graphes en 2D avec des réseaux de neurones profonds (*e.g.*, *(DNN)*^{2}, voir Chapitre III), il devrait être possible d'entraîner un tel réseau à calculer un plongement sans chevauchements qui satisferait certains critères.

D'autres pistes d'amélioration consisteraient à optimiser différents critères visuels ou perceptuels. Par exemple, la gestion de la tolérance pourrait être dynamique et complétée par une optimisation de l'opacité ou de l'ordre de rendu des nœuds dans la représentation finale. Cela



(a) Initial et super-pixels (b) Altération taille doublée (c) Boîte englobante (d) Forme quelconque

FIGURE IV.18 : Illustration de l'application d'un algorithme de suppression de chevauchements forme-agnostique, par opposition à l'approche standard considérant la boîte englobante d'une forme quelconque. (a) La célèbre peinture « *La Joconde* » (1506) de Leonardo Da Vinci découpée en super-pixels avec l'algorithme SLIC [229]. (b) Version altérée de (a) pour créer des chevauchements en doublant la taille des super-pixels. (c) Résultat de FORBID' sur (b) en modélisant chaque nœud par sa boîte englobante rectangulaire. (d) Résultat de l'adaptation FORBID aux formes quelconques sur (b).

permettrait de baser l'évaluation de la visibilité des nœuds sur d'autres critères que leur surface. Optimiser des critères perceptuels [167] serait également une nouvelle étape vers des visualisations centrées sur l'interprétation par des humains. Il est évident que garantir qu'un élément est *visible* avec au moins un pixel est nécessaire. Cependant, il y a un risque que ce pixel (et l'information qu'il encode) ne soit pas perçu car noyé dans une visualisation complexe. Il serait donc particulièrement utile de s'intéresser au système perceptif humain pour déterminer certains critères qui permettraient de garantir que l'information encodée dans la visualisation est bien perçue et correctement interprétable par l'utilisateur final.

Enfin, une piste d'amélioration que nous avons explorée est l'adaptation de FORBID à tout type de formes de nœuds. Cette adaptation se fait via la Triangulation de Delaunay des formes de nœuds quelconques. Les chevauchements sont ensuite calculés entre les paires de triangles pour améliorer la *finesse* (*i.e.*, l'utilisation des espaces vides) des résultats. La Figure IV.18 illustre un exemple de rendu de cet algorithme adapté aux formes de nœuds quelconques. Cette contribution est incluse dans la version étendue de FORBID conditionnellement acceptée avec révisions mineures dans le journal *IEEE Transactions on Visualization and Computer Graphics*.

Chapitre V

Évaluation Automatique de Visualisations par Apprentissage Profond

Sommaire

V.1 Problématique

L'objectif principal des visualisations d'information est de permettre l'exploration efficace de données [1, 184, 230]. Elle devient notamment nécessaire pour faire de l'analyse exploratoire lorsque ces données sont complexes. Les visualisations sont alors produites en représentant les données au travers de concepts abstraits tels que des nuages de points [34], des tableaux de

bords [31] ou toute autre forme de représentation visuelle.

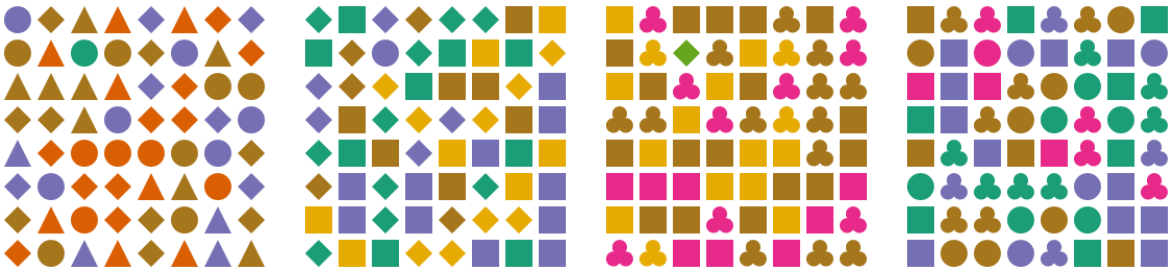
Efficacité des Visualisations : Encodage avec des Attributs Visuels

Pour concevoir des visualisations, les experts doivent encoder la donnée de manière efficace (*i.e.*, choisir les bons *attributs visuels* et leurs *caractéristiques*). Plusieurs études [1, 180, 230] ont été menées par la communauté de Visualisation d'Information pour mettre en place des lignes directrices décrivant des bonnes pratiques pour concevoir des représentations efficaces, et dont les fondements sont basés sur la théorie de la Perception humaine [1, 184]. Depuis lors, la communauté continue de mener des études sur l'encodage visuel d'information et d'actualiser ces lignes directrices [181, 182, 231–233].

La *Couleur* [181, 234, 235] et la *Forme* [236, 237] sont deux attributs visuels largement utilisés et souvent combinés pour représenter de l'information (*e.g.*, nuages de points [238], cartes géographiques [230], graphes [239], coordonnées parallèles [240]). Cependant, il n'est pas certain que la combinaison de ces deux attributs visuels soit efficace et le demeure alors que la complexité des données augmente. Des recherches en Perception [170] ont montré que la *Couleur* est une *caractéristique pré-attentive*, ce qui signifie que toutes les couleurs d'une représentation sont traitées en même temps par notre système perceptif. Pourtant, certaines représentations utilisant la couleur pour encoder leurs stimuli ont tendance à surcharger, voire saturer notre système de perception. La question est donc de savoir jusqu'à quel *point*, en terme d'hétérogénéité, certains attributs visuels sont efficaces. Ce *point* est connu comme la *capacité limite d'attention* [181] et varie en fonction de l'attribut visuel considéré. Pour la couleur, cette limite est supposée être autour de 7 ± 2, bien que nous n'ayons pas de référence pour soutenir cette supposition. Cette valeur semble provenir d'une mauvaise interprétation assez courante du *nombre magique de Miller* [241]. Ce nombre magique est en réalité défini comme le nombre de concepts qu'un humain est capable de conserver dans sa mémoire à court terme. Nous pensons que cette limite est trop optimiste et que la complexité de l'information encodée dans une visualisation hétérogène ne peut pas être contenue dans la mémoire à court terme, notamment quand d'autres paramètres (*e.g.*, formes, positions, densités) augmentent encore la complexité de la représentation.

Évaluation de l'Efficacité des Attributs Visuels

Dans ce chapitre, nous présentons une étude visant à vérifier l'hypothèse précédente, et plus généralement à mesurer la capacité limite d'attention dans des représentations où l'information est encodée avec les attributs visuels *Couleur* ***et*** *Forme*. Ce travail [30] a été publié dans la revue *Visual Informatics*. Notons que nous nous intéressons ici au *nombre maximum* de couleurs (ou formes) qui peut être utilisé dans une représentation, et pas aux couleurs (ou formes) elles-mêmes. Celles-ci peuvent évidemment avoir un impact sur la représentation, mais nous essaierons de le minimiser. Concrètement, la tâche étudiée est la *localisation d'intrus* dont la difficulté est paramétrée par l'hétérogénéité des éléments visuels (*i.e.*, stimuli). Les attributs visuels contrôlés sont la *Couleur* et la *Forme*, dont le nombre déterminera l'hétérogénéité de la représentation. Pour minimiser l'impact de la *Position* de l'intrus, tous les stimuli seront positionnés dans une grille régulière. De cette manière, *l'intrus* n'est pas encodé par sa position. Un stimulus peut donc être intrus par sa couleur et/ou sa forme uniquement. L'encodage de l'intrus (que nous appellerons *Type* d'intrus) peut alors avoir quatre valeurs possibles en fonction de la combinaison d'attributs visuels qui le rend unique (voir Figure V.1). Nos résultats montrent que la capacité limite d'attention est en effet plus basse que le *nombre magique de Miller* [241], et qu'elle



(a) Type *color* (b) Type *shape* (c) Type *red.* (d) Type *conj.*

FIGURE V.1 : Exemples d'objets expérimentaux avec les 4 valeurs possibles de *type* d'intrus. Dans chaque exemple, il y a 3 formes, 4 couleurs et l'intrus est à la position 10 (*i.e.*, deuxième ligne, troisième colonne en partant d'en haut à gauche). (a) Dans les images de type *color*, l'intrus a une couleur unique ; (b) dans type *shape*, l'intrus a une forme unique ; (c) dans type *redundant*, la forme et la couleur de l'intrus sont uniques ; et (d) dans type *conjunction*, l'intrus a une combinaison forme-couleur unique.

dépend fortement de l'encodage de l'intrus [169, 177].

Pour identifier la capacité limite de l'attention dans le contexte que nous avons défini (et détaillée dans la Section V.3), nous devons conduire une évaluation utilisateur. Cependant, la taille de l'espace de paramètres de notre contexte est beaucoup trop grand pour être exploré exhaustivement au cours d'une évaluation utilisateur. Nous devons donc sous-échantillonner cet espace de paramètres avant de faire passer cette évaluation à nos participants. L'originalité de notre étude est de baser cet échantillonnage sur un réseau de neurones profond (DNN pour « Deep Neural Network ») dont les performances sont considérées comme une mesure quantitative de la difficulté de la tâche.

Évaluation Automatique de Visualisations

Dans un précédent travail présenté dans *Visualization Meets AI Workshop* de la conférence *Pacific Visualization Symposium (PacificVis)* et paru dans la revue *Visual Informatics* [68], nous avons montré qu'il était possible d'utiliser des DNN pour comparer automatiquement des techniques de visualisation. L'idée était de retrouver les résultats d'études de la littérature [54, 55] comparant les techniques de dessin de graphe *nœud-lien* et *matrice d'adjacence*, en utilisant une approche automatique par réseau de neurones profond. Pour arriver à cela, notre méthode consiste à fixer une tâche à réaliser, un jeu de données, une architecture de DNN de l'état de l'art (en minimisant ses modifications) et un protocole d'évaluation de l'efficacité de la résolution de la tâche. Les techniques de visualisation sont ensuite utilisées pour générer des bases de données d'images distinctes. Nous entraînons alors un DNN sur chaque base de données en utilisant les conditions fixées. Le schéma de la Figure V.2 résume le processus. Après les entraînements, nous disposons d'un modèles de DNN par technique de visualisation, et nous avons pu mesurer leur performance pour résoudre la tâche. La seule différence entre l'entraînement des modèles est le type d'images qu'ils ont eu en entrée, issu des différentes techniques de visualisation. Nous pouvons alors conclure que le modèle ayant obtenu de meilleures performances a mieux réussi car la technique de visualisation utilisée pour représenter les données dans son entraînement était plus propice à la résolution de la tâche. Au regard des résultats de ce précédent travail, nous utilisons ici un DNN pour évaluer automatiquement la difficulté de la tâche de recherche d'intrus en fonction de l'hétérogénéité de la représentation. En outre, nous faisons varier les paramètres en entrée et certains attributs visuels utilisés pour la représenter, et étudions l'impact de ces variations

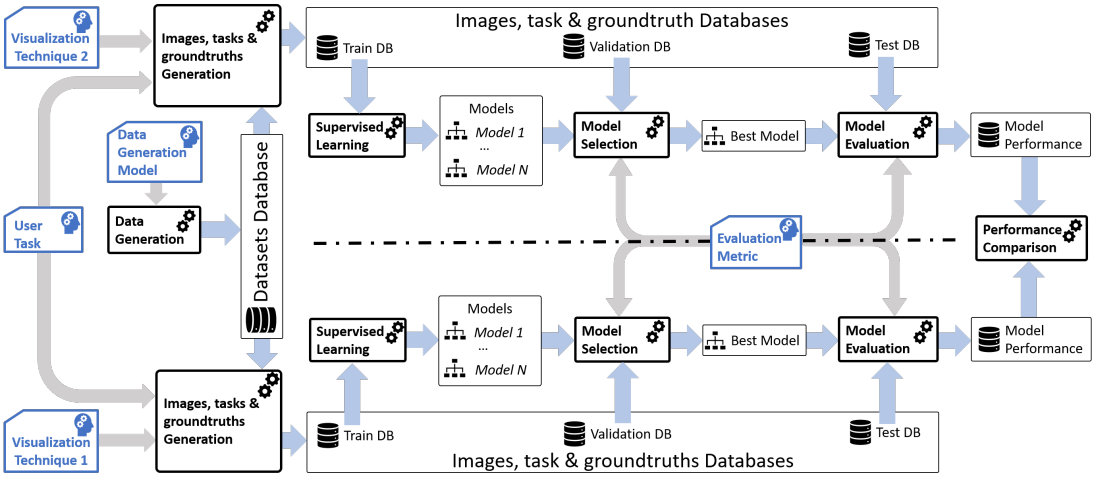


FIGURE V.2 : Schéma de la méthodologie de comparaison automatique de techniques de visualisation définie dans [68].

sur les performances d'un DNN. Ces résultats nous servent ensuite à raffiner nos hypothèses et sous-échantillonner l'espace de paramètres pour conduire une évaluation utilisateur qui permettra de conclure sur la question initiale : la capacité limite d'attention sur une tâche de détection d'intrus où l'information est encodée par des formes et des couleurs.

Nous pensons que l'utilisation d'un réseau de neurones pour sous-échantillonner l'espace de paramètres est une bonne alternative aux méthodes standards (*e.g.*, régulières, aléatoires) car elle se base sur un critère quantitatif et reproductible. En revanche, cette méthode repose sur l'hypothèse que les performances d'un réseau de neurones pour résoudre ce type de tâche de perception sont corrélées avec celles d'humains. Si plusieurs études dans la littérature semblent indiquer que cela peut être le cas [28, 29, 68], cette corrélation a pour l'instant été peu étudiée. Ainsi, ce chapitre présente aussi notre travail publié dans la conférence *International Conference Information Visualisation 2021* [69] dans lequel nous utilisons les performances du DNN et des participants humains récoltées pendant les expériences sur la tâche d'identification d'intrus pour étudier leurs corrélations.

Résumé des Contributions

Ce chapitre présente une évaluation de la capacité limite d'attention de la perception humaine sur une tâche d'identification d'intrus encodés par les attributs visuels *Couleur* et *Forme*. L'évaluation est réalisée en deux parties : (i) une pré-évaluation avec un réseau de neurones profond qui permet de mesurer quantitativement la difficulté de la tâche sur une large portion de l'espace de paramètres ; et (ii) une évaluation utilisateur qui étudie la capacité limite d'attention. Ainsi, les contributions apportées par ces travaux sont :

- Une méthodologie d'évaluation automatique de visualisations qui tire parti des capacité d'apprentissages des modèles de réseaux de neurones profonds. La méthodologie est définie et mise en œuvre dans notre expérience pour pré-évaluer la difficulté d'une tâche de perception.

- Une évaluation de l'efficacité de l'encodage de l'information avec les attributs visuels *Forme* et *Couleur* en fonction de variations de l'hétérogénéité de la représentation.

- Une analyse de corrélations entre les performances d'un réseau de neurones profond et celles d'humains sur une tâche de perception.

Le reste de ce chapitre est organisé comme suit. La Section V.2 rappelle des travaux de la littérature sur le système de perception humain étudié dans les communautés de Perception et de Visualisation, avec une emphase sur la tâche de détection d'intrus. La Section V.3 présente la tâche et l'espace de paramètres considéré en détail. La Section V.4 présente l'évaluation automatique de la difficulté de cette tâche avec un réseau de neurones profond, tandis que la Section V.5 en présente l'évaluation avec des participants humains. Enfin, la Section V.6 étudie les corrélations entre le réseau de neurones et les humains sur les données de nos évaluations, et la Section V.7 fait la synthèse des travaux réalisés et de leurs contributions.

V.2 Rappel de l'État de l'Art

Cette section rappelle les travaux principaux dans le domaine de la Recherche Visuelle des communautés de Perception et Visualisation d'Information, en mettant l'accent sur la recherche d'intrus. Pour plus de détails sur ces travaux, le lecteur peut se référer à la Section II.4.1.

V.2.1 Recherche Visuelle en Perception

La théorie fondatrice du domaine de Recherche Visuelle est la *Théorie de l'intégration des caractéristiques* (« Feature-Integration Theory » en anglais) de Treisman et Gelade [169]. Elle définit l'attention en un système à deux étapes, d'abord *pré-attentive* puis *attentive*. Elle distingue également la recherche de caractéristiques *simples* (*e.g.*, recherche d'une couleur) ; de la *conjonctive* (*e.g.*, recherche d'un objet défini par une couleur et une forme).

Depuis, cette modélisation de l'attention a été contestée et étendue dans différentes études. Duncan et Humphreys [176] ont basé leur théorie sur la similarité entre les stimuli et les modèles visuels. Cette étude met en avant l'importance de la similarité entre les cibles (T pour *target*) et les distracteurs (N pour *non-target*). D'une manière générale, la difficulté d'une tâche visuelle serait déterminée par la similarité T-N. Selon les auteurs, l'hétérogénéité de la représentation n'a d'importance que si elle a lieu dans un attribut visuel que les stimuli cibles partagent avec les distracteurs.

Quinlan et Humphreys [179] ont montré que la difficulté de la recherche d'une cible définie par sa *forme* augmentait linéairement avec le nombre total de stimuli, tandis qu'elle n'augmentait pas si la cible était définie par sa couleur. Cela corrobore les études définissant la *couleur* comme un attribut visuel pré-attentif [169, 170]. Ils démontrent également que la similarité T-N a plus d'impact en recherche conjonctive où les cibles sont définies par des combinaisons d'attributs visuels. D'une manière générale, ils en concluent que plus une cible partage d'attributs visuels avec les distracteurs, plus la difficulté de la tâche est grande et sensible à l'hétérogénéité.

Aujourd'hui, la théorie la plus consensuelle est *Guided Search* de Wolfe [171, 172] qui est régulièrement mise à jour (la plus récente étant la version 6.0 [173]). Selon cette théorie, le système d'attention peut suivre deux stratégies pour résoudre une tâche : *bottom-up* ou *top-down*. La stratégie *bottom-up* est centrée sur les stimuli : l'attention se dirige sur un stimulus, puis le cerveau essaie de l'interpréter pour le faire correspondre à un concept. À l'inverse, la stratégie *top-down* est centrée sur l'utilisateur. Généralement, cette deuxième stratégie est employée lorsque la cible est connue. Le cerveau part alors d'une vision du concept recherché et dirige son attention vers des stimuli susceptibles de lui correspondre.

V.2.2 Recherche Visuelle en Visualisation

Si le domaine de la Perception a mené des études théoriques du systèmes perceptif humain, ses recommendations ne sont pas toujours adaptées aux cas d'application en Visualisation d'Information. Par exemple, Treisman et Gelade [169] revendiquent que l' « *on ne peut pas détecter une cible d'un ensemble de distracteurs si on ne connait pas au moins l'attribut visuel qui le différencie* », mais leur évaluation est basée sur des cas expérimentaux présentés à des utilisateurs pendant seulement 3 secondes. Un tel cas d'application n'est pas représentatif des tâches réalisées sur les visualisations de nos jours : les plus complexes, malgré leur efficacité, nécessitent parfois un temps de déchiffrage important. Ce déchiffrage inclue la compréhension (i) de la technique d'encodage visuel de représentation des données, puis (ii) de la donnée elle-même. Healey et Enns [180] ont dressé un tableau de la littérature de la perception visuelle dédiée aux applications de la communauté de Visualisation. Depuis lors, cette communauté a continué de mener ses mesures de l'efficacité du système perceptif humain dans son contexte applicatif, en complément des recherches de la communauté de Perception.

Haroz et Whitney [181] ont étudié comment les *groupements de couleurs* et les *mouvements* influencent l'efficacité des visualisations pour la détection d'une cible. Les travaux de Gramazio *et al.* [182] ont étudié l'impact du nombre de stimuli total et de leur disposition dans le plan sur la difficulté de la tâche de détection de cible. Les résultats de ces études sont détaillées dans la Section II.4.1. Dans le travail présenté dans ce chapitre, nous étudions l'impact de la variation de l'hétérogénéité de la représentation (*i.e.*, nombre de couleurs et de formes) sur l'identification d'une cible inconnue (*i.e.*, un intrus).

Selon Mackinlay [233], la *position* d'un élément dans la visualisation peut être un des meilleurs paramètres pour le mettre en évidence. Par exemple, un élément isolé spatialement attire l'attention. Dans la culture occidentale où nous lisons de gauche à droite et de haut en bas, nous pouvons supposer que les stimuli en haut à gauche des visualisations sont naturellement traités en premier. Un biais de fixation centrale a également été mis en évidence par Tatler [242]. Cependant, la position des stimuli est déterminée par les données ou par un algorithme de positionnement dans la plupart des visualisations que nous produisons aujourd'hui. C'est alors la responsabilité de cet algorithme de capturer la nature intrinsèque de l'intrus pour le positionner de manière à ce qu'il soit mis en valeur. Or, ce n'est pas toujours leur objectif et de nombreuses visualisations (*e.g.*, erreurs de classification d'un réseau de neurones [34, 208–211]) définissent la notion d'intrus comme *un élément dont les propriétés ne correspondent pas à celles de ses voisins*, par exemple sa couleur/catégorie. La *position* est donc rarement utilisable à elle seule pour définir la notion d'intrus. Ainsi, nous ne souhaitons pas mesurer l'impact de ce paramètre dans notre évaluation. La disposition des stimuli dans nos visualisations est la plus neutre possible : une grille régulière. La position de l'intrus dans la grille est un paramètre que nous équilibrons dans le jeu de données pour limiter son impact (voir Section V.3).

V.2.3 Réseaux de Neurones pour la Visualisation

L'évaluation de la qualité de visualisations via les réseaux de neurones profonds est une approche récente [15, 16]. L'étude de Haehn *et al.* [28] a montré qu'il était possible de reproduire des évaluations de perception [187] avec une approche de traitement d'image de réseaux de neurones convolutifs. Haleem *et al.* [29] ont évalué automatiquement la qualité de dessins de graphes entraînant un DNN à prédire des métriques de qualité de ces dessins à partir de l'image de ceux-ci. Enfin, nous avons montré dans un précédent travail [68] qu'il était possible de reproduire des évaluations de visualisations de la littérature [54, 55] avec des DNN.

Ces différents travaux, regroupés dans les études AI4VIS [15], ML4VIS [16] et VIS+AI [8], démontrent l'intérêt croissant de la communauté pour les évaluations automatiques de visualisations par réseaux de neurones profonds. Pour une description plus détaillée des applications d'Intelligence Artificielle pour la Visualisation d'Information, le lecteur peut se référer aux Sections I.4 et II.4.2.

V.3 Description de la Tâche

Cette section détaille les paramètres sélectionnés pour faire varier la difficulté de la tâche de recherche d'intrus dans cette étude.

V.3.1 Tâche et Types d'Intrus

La tâche que nous étudions consiste à identifier un intrus dans une grille de 8×8 stimuli définis par une forme, une couleur et une position dans la grille (voir Figure V.1). Chaque forme colorée (*i.e.*, stimulus) est dessinée dans un espace de 32×32 pixels, menant donc à des images de résolution 256×256 pixels. Cette résolution est un bon compromis entre la lisibilité de l'information pour un humain, et la possibilité d'entraîner un réseau de neurones profond sur ces mêmes images. Dans ces images, un stimulus est considéré *intrus* si aucun autre ne lui est semblable (forme et/ou couleur). Les dimensions rendant l'intrus unique peuvent varier en fonction du paramètre que nous nommons *type*. Il y a toujours exactement un intrus par grille.

Type est le paramètre qui décrit la ou les dimensions rendant l'intrus unique dans la grille, et peut avoir quatre valeurs différentes illustrées dans la Figure V.1 :

- *color* : l'intrus est unique par sa *couleur*.
- *shape* : l'intrus est unique par sa *forme*.
- *redundant* (red.) pour redondant : l'intrus est unique par sa *forme* et sa *couleur* (cela se réfère à *l'encodage redondant* [243]).
- *conjunction* (conj.) pour conjonction : l'intrus est unique par sa conjonction *forme-couleur* (*i.e.*, une recherche conjonctive est nécessaire pour l'identifier).

V.3.2 Espace de Paramètres

Les objets expérimentaux de cette étude sont des images représentant des grilles de stimuli. Chaque objet est défini par les valeurs de six paramètres contrôlés et résumés dans la Table V.1 : la **forme**, **couleur** et **position** de l'intrus, ainsi que le **type**, **nombre de couleurs** et **nombre de formes** de l'image.

Formes. La forme des stimuli (intrus et distracteurs) est choisie parmi les formes définies dans cette expérience (voir Table V.1 colonne 1). Une forme peut être définie par de nombreuses sous-caractéristiques (*i.e.*, lignes droites, orientation, taille). Dans cette expérience, toutes les formes ont une orientation constante et leur taille est ajustée pour que leur boîte englobante rectangulaire occupe un espace de 32×32 pixels. Nous avons sélectionné 5 formes : Triangle ▲, Cercle ●, Carré ■, Trèfle ♣ et Losange ◆. Elles permettent de varier l'utilisation de lignes droites horizontales et verticales, diagonales et courbes.

Couleurs. La couleur des stimuli est sélectionnée parmi un ensemble de couleurs prédéfinies (voir Table V.1, colonne 2). Il existe déjà plusieurs méthodes pour définir un ensemble de couleurs pour représenter un ensemble de *cibles* (*e.g.*, [244], ou plus récemment Colorgorical [245]). Dans

TABLE V.1 : Espace de paramètres considéré dans cette évaluation. Les couleurs sont données en codes hexadécimaux. Les couleurs et formes définies peuvent être utilisées par tous les stimuli (intrus ou non).

Valeurs des Attributs Visuels			**Image**		
Forme	**Couleur**	**Position**	**Type**	**#couleurs**	**#formes**
▲	#1B9E77	0	color	1	1
●	#D95F02	⋮	shape	2	2
■	#7570B3	63	redondant (red.)	3	3
♣	#E7298A		conjonction (conj.)	4	4
◆	#66A61E			5	5
	#E6AB02			6	
	#A6761D			7	

notre expérience, nous considérons les couleurs comme des caractéristiques atomiques (*i.e.*, nous n'étudions pas la teinte et la saturation des couleurs). Concrètement, nous utilisons les *7 couleurs qualitatives* définies dans la palette *Dark2* de l'outil ColorBrewer[1] [246]. Nous choisissons une palette *qualitative* car nous les souhaitons indépendantes les unes des autres, comme ce serait le cas dans une visualisation de données catégorielles. *Dark2* est la palette de l'outil qui propose les couleurs les plus saturées. Cette palette n'étant pas adaptée aux daltoniens, nous exclurons ces derniers de l'évaluation utilisateur dans la Section V.5.

La **position** d'un stimuli est une valeur entre 0 et 63 déterminant son emplacement dans la grille 8×8 dans l'ordre *row-major* (ligne-majeur) ; $position = 8 * ligne + colonne$.

Le **type** de l'image définit la ou les dimensions rendant l'intrus unique dans la grille tel que décrit dans la Section V.3.1.

Nombre de couleurs (*#couleurs*) est le nombre total de couleurs dans un objet expérimental. Dans cette expérience, il varie entre 1 et 7. Notons que si un objet expérimental est de type *color* ou *redundant*, il n'est pas possible de créer un objet expérimental avec 1 couleur puisqu'il faut au moins 1 couleur unique pour l'intrus, et 1 couleur pour les distracteurs.

Nombre de formes (*#formes*) est le nombre total de formes dans un objet expérimental. Ici, il varie entre 1 et 5. Là non plus, la valeur 1 ne peut pas être attribuée aux objets expérimentaux de type *shape* et *redundant* car une forme doit être réservée pour l'intrus.

V.3.3 Génération des Données

Pour générer le jeu de données, les valeurs des six paramètres contrôlés sont équilibrées pour limiter les biais qui pourraient mener à un sur-apprentissage du modèle de réseau de neurones. Notre préoccupation principale est d'équilibrer les combinaisons forme-couleur-position de l'intrus (voir les Figures V.3d, V.3c et V.3a) pour éviter que le réseau de neurones apprenne à le reconnaître simplement car une position, forme ou couleur d'intrus apparaît plus souvent que d'autres dans le jeu de données.

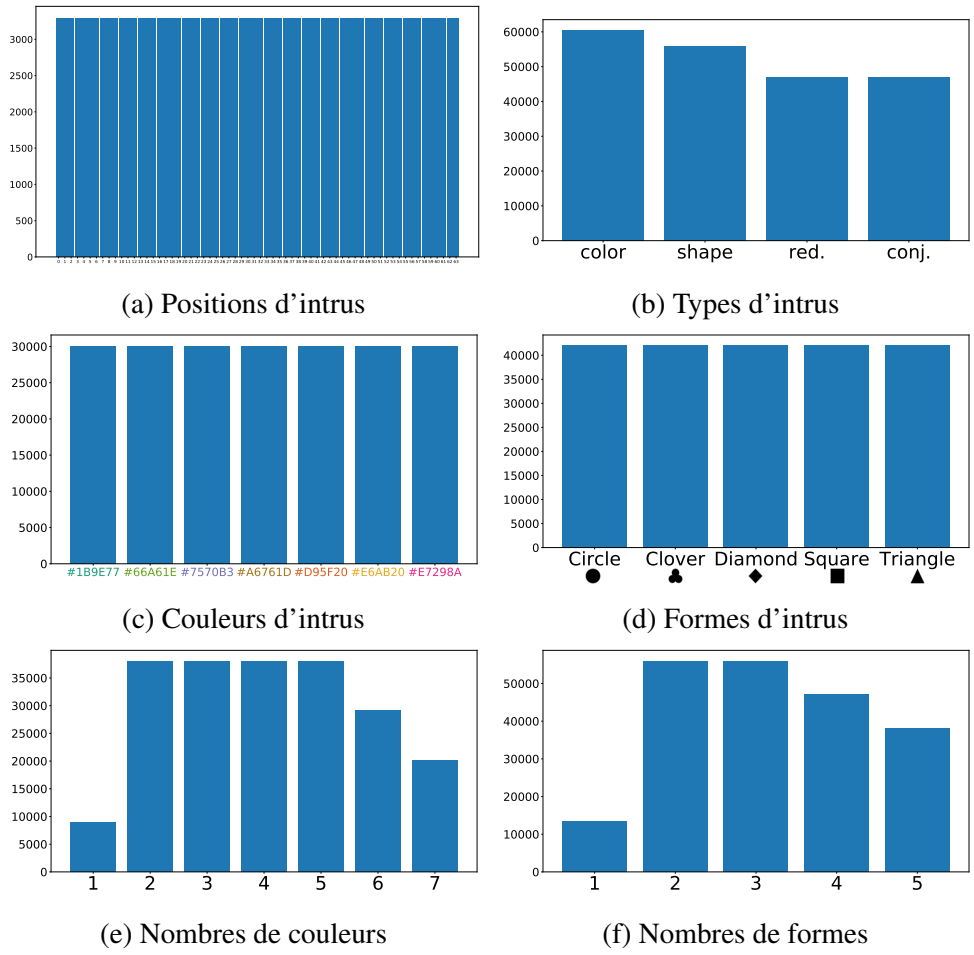
Idéalement, nous souhaiterions étudier la difficulté de la tâche sur l'ensemble de l'espace de paramètres que nous avons défini. Cependant, le produit cartésien de tous les paramètres que nous contrôlons est bien trop grand pour pouvoir être considéré exhaustivement. Néanmoins, le fait d'étudier automatiquement la difficulté de la tâche avec un réseau de neurones profond rend possible, voire nécessaire, l'étude d'une multitude de cas différents. Notre objectif est donc de réduire l'espace de paramètres (*i.e.*, des millions d'objets expérimentaux possibles) à un nombre plus raisonnable, mais toujours suffisamment grand pour être représentatif de l'espace

[1] `https://colorbrewer2.org/#type=qualitative&scheme=Dark2&n=7`

original (*i.e.*, quelques centaines de milliers). Les résultats de l'évaluation automatique du réseau de neurones nous permettra alors de significativement réduire cet espace pour l'évaluation utilisateur.

Le processus de génération des données suit également plusieurs contraintes. Évidemment, aucune image ne peut être générée avec 1 forme et 1 couleur, mais d'autres cas impossibles moins évidents existent aussi. Les images de type *redundant* ne peuvent pas être générées avec 2 couleurs et 1 forme, ni 1 couleur et 2 formes puisqu'une valeur de chaque dimension doit être réservée pour l'intrus, et qu'il en faut alors une de plus pour encoder les distracteurs. Pour les images de type *conjunction*, nous ne générons pas de donnée avec 1 couleur ou 1 forme car cela reviendrait à avoir des données de type *shape* ou *color*. De plus, nous ne pouvons pas non plus générer d'image de type *conjunction* avec 7 couleurs et 5 formes dans une grille 8×8. Puisqu'une des $5 * 7 = 35$ combinaisons forme-couleur est réservée pour l'intrus, cela signifie que les 34 combinaisons restantes doivent apparaître au moins deux fois. Cela nécessiterait alors de dessiner $34 * 2 + 1 = 69$ stimuli.

Pour réduire encore l'espace de paramètres et atteindre le nombre de données que nous voulons (*i.e.*, quelques centaines de milliers), nous ne générons pas d'image avec [4 formes, 7 couleurs] ni [5 formes, 6 couleurs]. Ceux-ci sont les cas les plus complexes restants et nous



(a) Positions d'intrus

(b) Types d'intrus

(c) Couleurs d'intrus

(d) Formes d'intrus

(e) Nombres de couleurs

(f) Nombres de formes

FIGURE V.3 : Distributions des valeurs des paramètres considérés dans l'évaluation. Le jeu de données est constitué de 210 560 images au total. La valeur 1 pour le nombre de couleurs et formes (e, f) est moins représentée car elle ne permet pas d'encoder des intrus de tous les types (*e.g.*, un intrus ne peut pas être unique par sa couleur dans une visualisation avec 1 seule couleur).

pensons fortement que la capacité limite d'attention que nous cherchons à étudier sera atteinte avec moins d'hétérogénéité.

Comme mentionné précédemment, nous n'étudions pas la difficulté de la tâche en fonction de la position de l'intrus que nous équilibrons pour éviter les biais et nous sert de paramètre de répétition pour créer de la variété dans les données. Ainsi, 3290 combinaisons de paramètres sont répétées 64 fois. Les distributions des paramètres contrôlés sont présentées dans la Figure V.3. En générant des images avec toutes les combinaisons de paramètres restantes, nous obtenons un jeu de 210 560 données.

Enfin, ce jeu de données est séparé aléatoirement en trois sous-ensembles *entraînement*, *validation* et *test* pour l'apprentissage du modèle de réseau de neurones avec une validation *Hold-Out* [76] (voir Section II.1.2).

V.4 Évaluation Automatique

Comme mentionné précédemment, l'espace de paramètres de la tâche considérée (voir Section V.3.1) est trop grand pour pouvoir étudier la capacité limite d'attention dans une évaluation utilisateur. Il faut donc sous-échantillonner cet espace de paramètres pour concevoir une évaluation réalisable par des humains et qui permet d'étudier des hypothèses en lien avec la tâche considérée. Dans ce travail, nous prenons le parti de baser le sous-échantillonnage en évaluant les performances d'un DNN entraîné pour résoudre la même tâche. L'idée est que même si le système de traitement de l'information de ce réseau est différent de celui des humains, les deux peuvent être impactés de la même manière par certaines propriétés de la visualisation.

Cette section présente la mise en place, l'entraînement et l'étude des résultats de ce DNN.

V.4.1 Métrique Basée sur un Réseau de Neurones Profond

Les méthodes classiques d'échantillonnage sont : (i) arbitraire, basée sur l'expérience et la littérature ; (ii) systémiques (*i.e.*, aléatoire, régulier) ; ou (iii) basée sur des métriques d'évaluation. Cette dernière semble préférable pour des raisons d'objectivité et de reproductibilité. Elle consiste a utiliser des algorithmes qui associent un score de qualité à chaque donnée, pour ensuite étudier statistiquement les cas similaires ou non. En fonction de la tâche considérée, cela permet alors de ne pas avoir à tester des cas qui mèneraient aux mêmes résultats, et d'insister sur des cas qui semblent essentiels pour l'étude d'hypothèses. Cependant, il existe rarement des métriques pour quantifier l'efficacité d'une visualisation. En effet, l'efficacité dépend de la tâche à réaliser et de plusieurs paramètres de rendu de l'image qui peuvent fortement varier d'une technique de visualisation à l'autre. En réalité, cette approche n'est possible que dans des domaines très spécifiques, où des métriques de qualité ont été développées pour une visualisation et validées par l'expérience (*e.g.*, les métriques de qualité de dessin de graphes [53, 57, 58]).

Inspirés de notre précédent travail [68], et suivant les recommendations de Haehn *et al.* [28] et Haleem *et al.* [29], nous proposons une nouvelle approche pour créer une méthode d'évaluation basée sur un réseau de neurones profond. L'idée est d'évaluer la difficulté de la tâche en fonction des performances du réseau sur sa résolution. Le réseau travaillant à partir de l'image d'un objet expérimental, nous pouvons alors remonter jusqu'aux propriétés de celui-ci pour associer un score de difficulté à ses paramètres. En répétant l'opération sur toutes les images du jeu de données, nous pouvons évaluer et comparer la difficulté qu'a le réseau à résoudre la tâche en fonction des différents paramètres de la visualisation. Pour atteindre cet objectif, nous utilisons une architecture générique de traitement d'image et ne l'adaptons pas spécifiquement à notre

tâche pour éviter d'introduire des biais. L'intuition est d'utiliser une architecture générique de la même manière que nous demanderions à des participants humains quelconques (*i.e.*, tirés aléatoirement dans la population) de résoudre la tâche. Nous n'avons pas besoin que le réseau ait d'autre capacité que celle de traiter des images, tant que celui-ci réussit à apprendre à résoudre la tâche. Les performances relatives de ce modèle sont ensuite évaluées et statistiquement étudiées pour pouvoir associer une difficulté à chaque valeur ou combinaison de valeurs de paramètres. En se basant sur ses résultats, l'espace de paramètres peut ainsi être réduit (voir Section V.4.4.3) pour l'évaluation utilisateur (voir Section V.5).

Les principaux avantages d'une telle évaluation sont : (i) elle s'adapte à n'importe quelle tâche qui peut être programmatiquement exprimée, et n'importe quelle technique de visualisation capable de générer une image statique, et (ii) elle ne requiert aucune connaissance préalable sur la tâche. Le modèle DNN apprend par lui-même à extraire l'information nécessaire à la résolution de la tâche dans la visualisation.

La principale préoccupation levée par cette approche est qu'elle considère que les système de traitement respectifs du DNN et d'humains peuvent être affectés de la même manière par certaines propriétés de la visualisation. Cela revient à considérer qu'une forme de corrélation entre les deux existe pour que cette approche soit fiable. Cette étude de corrélation est le résultat d'un autre de nos travaux [69] qui sera décrit ultérieurement (voir Section V.6). Elle montrera que les performances des humains et du DNN sont en effet fortement corrélées (jusqu'à 0.988, un score de corrélation parfaite valant 1). Ceci prouve que l'estimation de la difficulté par le DNN est représentative de la difficulté de la tâche.

V.4.2 Sélection de l'Architecture et Entraînement

Nous utilisons ici une architecture DNN générique pour ne pas biaiser le traitement de l'image. En prenant une architecture éprouvée de l'état de l'art, nous connaissons ainsi son fonctionnement et pourquoi elle pourrait échouer. Par exemple, nous savons qu'un réseau de neurones profond convolutif est basé sur de l'extraction de caractéristiques se basant sur la détection de contours dans une image. Cette extraction produit une série d'abstractions de la donnée en entrée jusqu'à la prédiction, modélisant ainsi un processus de traitement de l'information *bottom-up* (voir Section II.4.1) et favorisant probablement la détection de formes. Nos hypothèses sur le fonctionnement de l'architecture sélectionnée pour cette évaluation automatique seront plus amplement détaillées dans la Section V.6.2.2. Suivant la recommandation de Haehn *et al.* [28], nous avons essayé plusieurs architectures de réseaux de neurones convolutifs (CNN pour « Convolutional Neural Network ») pour traiter nos images. En l'occurrence, nous avons expérimenté *LeNet* [83], *VGG-16* et *VGG-19* [137] et avons finalement arrêté notre choix sur *ResNet50* [138]. L'objectif était de trouver une architecture capable de résoudre la tâche en lui apportant le moins de modifications possibles, ce que *ResNet50* réussit le mieux. Les poids initiaux de *ResNet* sont fixés à leur valeur pré-entraînée sur ImageNet [5]. Cette base de données ne contient a priori aucun échantillon ressemblant à nos images abstraites, mais cela donne rapidement au modèle une capacité de segmentation des éléments constitutifs de l'image. Ainsi, le modèle est très vite capable de segmenter efficacement l'information dans notre type d'images, et il ne lui reste qu'à apprendre à trouver l'intrus à partir de cette information.

Les seules modifications apportées à *ResNet* sont : l'ajout d'une couche d'entrée correspondant au format de nos images (*i.e.*, 256×256 pixels en 3 canaux *Rouge, Vert, Bleu RVB*), et l'ajout de deux couches entièrement connectées en sortie de *ResNet* pour régresser les caractéristiques extraites par le CNN vers le nombre de classes correspondant à notre tâche. Nous modélisons la tâche comme un problème de classification pour le réseau. Il y a 64 classes,

correspondant aux positions possibles de l'intrus dans la grille. La prédiction du modèle est donc la classe obtenant la plus forte probabilité dans son vecteur de prédiction. Cette modélisation de la tâche ne permet pas de mesurer la distance dans l'image entre la position de l'intrus prédite et sa position réelle lorsque le modèle se trompe. Cependant, il n'y a pas de raison de penser que lorsqu'un utilisateur se trompe de stimulus cela signifie que le véritable intrus soit proche de la position estimée.

L'optimiseur utilisé est Adam [77] avec ses valeurs par défaut dans la bibliothèque Tensorflow/Keras [247]. La taille des lots (*batch size* en anglais) est fixée à 64 pendant l'entraînement. Enfin, nous entraînons le modèle jusqu'à qu'il n'arrive plus à améliorer ses performances pendant 15 époques consécutives.

V.4.3 Résultats

À la fin de l'apprentissage, la meilleure époque atteint une exactitude (*accuracy* en anglais) de 74% et 76% sur les jeux de *validation* et *test*. Ces résultats montrent que le modèle n'a pas souffert de sur-apprentissage. Un coefficient de corrélation de Matthews [248] de 0.754 sur le jeu de *test* montre que le modèle a bien réussi à résoudre la tâche. Ainsi, nous pouvons penser que les mauvaises prédictions du modèle sont dues à des combinaisons de paramètres de la donnée en entrée, et non pas au hasard ou à de l'aléatoire de poids mal-appris.

Pour chaque paramètre de la tâche, un test de Kurskal-Wallis [249] est préalablement conduit pour vérifier si les variations de performances entre les valeurs de ce paramètre sont dues au hasard. Pour les paramètres ayant passé ce premier test, nous réalisons des tests de Wilcoxon sur les rangs pour vérifier si la différence entre chaque paire de valeurs de paramètres est significative. Comme nous le détaillerons par la suite, le paramètre *Type* influe grandement sur les performances du modèle. Aussi, nous étudions à la fois les résultats *en général* et *par valeur de type*. Le seuil de significativité des tests est fixé à $\alpha = 0.05$ pour l'étude générale, et à $\alpha = 0.025$ dans l'étude par valeur de type. Tous les paramètres ont montré qu'ils avaient un impact sur les performances du modèle dans au moins une des conditions étudiées, sauf la *Forme d'intrus* qui sera discutée dans la prochaine Section V.4.4.1.

Lecture des graphiques résultats. Les graphiques reportant les résultats du modèle encodent, pour chaque paramètre, le taux d'erreur moyen (*i.e.*, $1 - exactitude$) du modèle sur les données ayant ce paramètre *et* l'information de significativité. L'information sur la significativité se lit de la manière suivante. Si le graphique est entièrement estompé, c'est que le test de Kruskal-Wallis a échoué, et les variations de performances entre les valeurs de ce paramètre ne doivent pas être interprétées (*e.g.*, *#couleurs* dans le Type *shape* de la Figure V.5). Si ce test passe, la significativité des différences entre les paires de valeurs est encodée de deux manières. Initialement, la couleur des barres représentant le taux d'erreur pour chaque valeur de paramètre est bleue. Si la différence est significative entre deux valeurs de paramètre, un arc est tracé entre elles. Si une valeur est significativement différente de toutes les autres, sa barre correspondante est colorée en orange, ses arcs incidents sont transparents et un symbole * est ajouté sous le label. Par exemple, dans les performances *Générales* sur *#couleurs* dans la Figure V.5, les images avec 2 couleurs ont mené à des performances significativement différentes de toutes les autres valeurs. À l'inverse, les images avec 3 couleurs ne sont significativement différentes qu'avec les valeurs dont la barre est orange (*i.e.*, 1, 2, et 5) et avec les valeurs reliées par un arc (6 et 7). La différence des performances de *ResNet* n'est donc pas significative entre les images ayant 3 et 4 couleurs, dans le cas *Général*.

Type. Les taux d'erreur (ER pour « Error Rate ») de *ResNet* sur le jeu de données *test* en fonction des valeurs du paramètre *Type* sont présentés dans la Figure V.4. Comme nous nous y

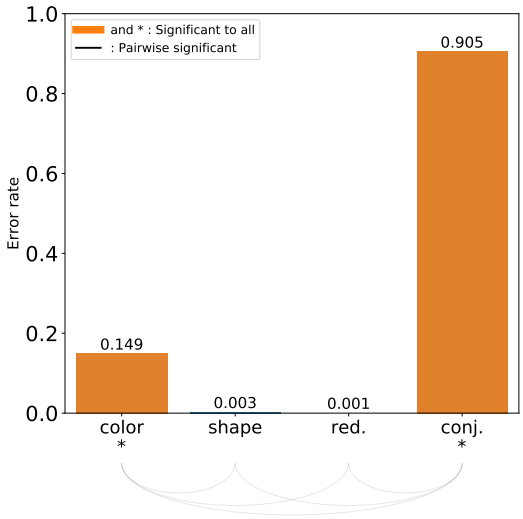


FIGURE V.4 : Taux d'erreur (ER pour « Error Rate ») de *ResNet* sur le jeu de *test* pour les valeurs de *Type*. La description des graphiques résultats ainsi qu'un exemple de lecture sont donnés dans la Section V.4.3.

attendions, ce paramètre revêt une importance prépondérante sur la difficulté de la tâche pour *ResNet*. Les données de type *conjunction* mènent significativement à plus d'erreurs que les autres valeurs. Le type *color* est également plus difficile que *shape* et *redundant*. Ces deux derniers ne sont pas significativement plus difficiles l'un que l'autre. Ce sont les forts écarts entre les performances de *ResNet* en fonction de ces valeurs de type qui ont motivé l'étude des autres paramètres *par valeur de type*. Le fait que les images de type *color* soient significativement plus difficiles que celles de type *shape* est surprenant au vu de la littérature en Recherche Visuelle (voir Section II.4.1). En effet, la *Couleur* est supposée être une caractéristique pré-attentive : ses valeurs peuvent être traitées en parallèle. Nous pensons que c'est ici l'illustration d'une différence fondamentale de fonctionnement du réseau de neurones convolutif qui sera discuté dans la Section V.4.4.1. Celui-ci étant construit pour détecter les contours, un intrus défini par sa forme est très facile à identifier pour lui. Il est donc important de garder cela à l'esprit lorsque nous interprétons les résultats de *ResNet*. Ce genre de différence entre le traitement de l'information du réseau et celle des humains illustre bien pourquoi il est important de prendre une architecture de DNN sur l'étagère dans notre approche. Cela nous permet de comprendre, voire anticiper certains comportements déviants.

Nombre de couleurs. Comme nous pouvons observer dans le cas général dans la Figure V.5, le taux d'erreur (ER) est quasi-linéaire sur l'augmentation de *#couleurs*. Nous observons une différence significative entre 1 et 2 *#couleurs* (et plus généralement, entre 1 et toutes les autres valeurs). Ce seuil peut être induit par un biais qui sera discuté dans la Section V.4.4.1. L'augmentation de la difficulté est encore plus prononcée lorsque *couleur* est la dimension qui rend l'intrus unique (*i.e.*, données de type *color*). Lorsque la couleur n'est pas une dimension servant à encoder l'intrus (*i.e.*, type *shape*), les variations de *#couleurs* n'ont presque pas d'impact significatif sur les performances du modèle. Enfin, la seule différence de performance significative dans les données de type *conjunction* est entre 2 et > 2 *#couleurs*. Lorsqu'il y a déjà plus de 2 couleurs, encore en augmenter le nombre ne semble pas rendre la tâche significativement plus difficile. Nous pensons que cela signifie que la difficulté de la tâche est seuillée et qu'au delà de 2 couleurs, les variations ne changent plus les performances du le modèle car la difficulté maximale est déjà atteinte.

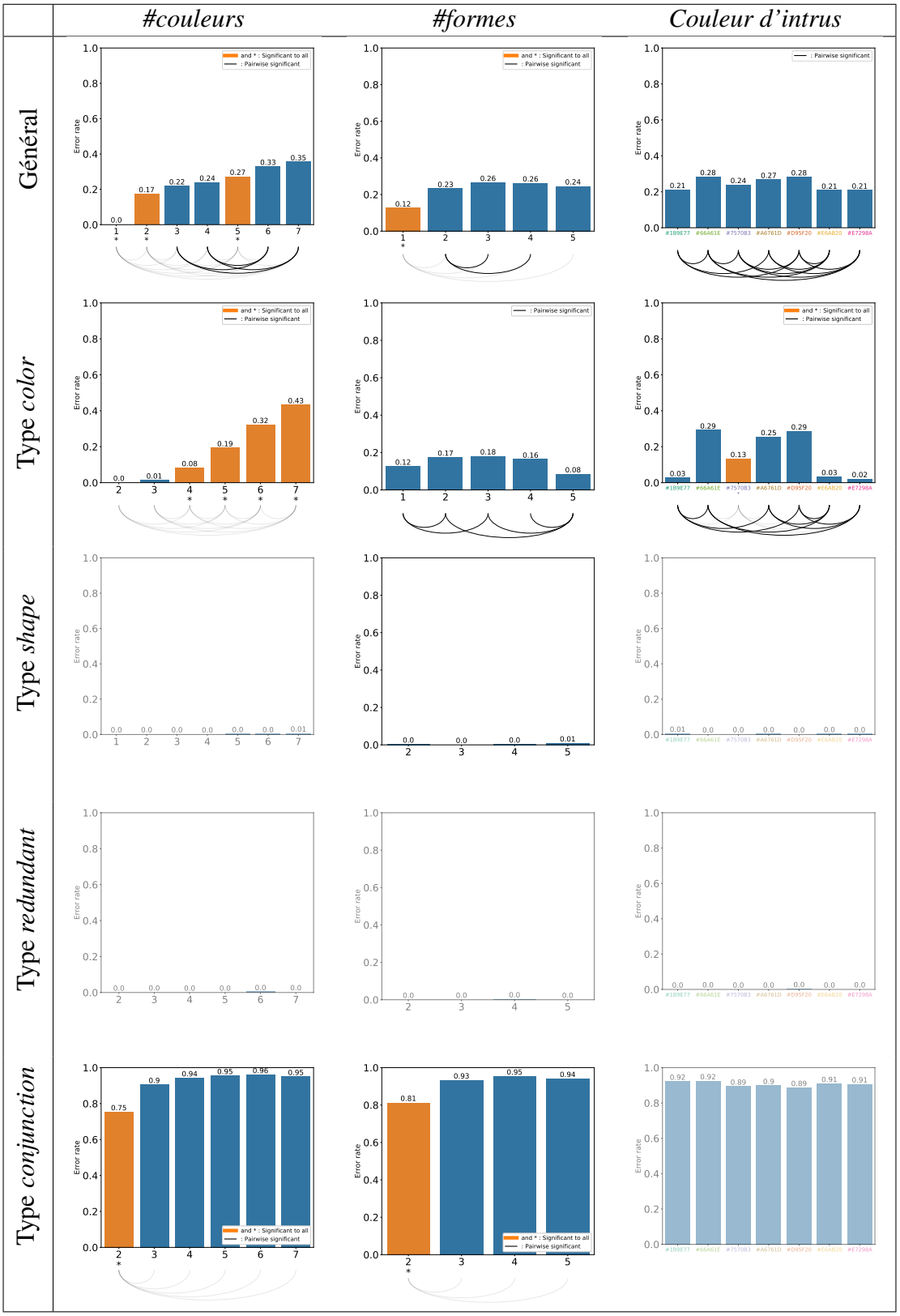


FIGURE V.5 : Taux d'erreur (ER pour « Error Rate ») de *ResNet* sur le jeu de *test*. La première ligne présente les résultats généraux, tandis que les lignes suivantes présentent les résultats séparément pour chaque valeur de *Type*. La description des graphiques résultats ainsi qu'un exemple de lecture sont donnés dans la Section V.4.3.

Nombre de formes. La Figure V.5 montre, dans le cas général, une différence de taux d'erreur significative entre 1 et > 1 *#formes*, de la même manière qu'avec *#couleurs* discuté plus tôt. Ce comportement sera également discuté dans la Section V.4.4.1. Cette valeur mise à part, il ne reste des différences significatives qu'entre 2 et 3 ou 4 *#formes*. Nous pouvons donc en conclure que dans le cas général, la difficulté de la tâche n'est pas linéairement corrélée à l'hétérogénéité des formes dans la visualisation. Quand l'intrus est unique par sa *couleur* (*i.e.*, type *color*), le test de Kruskal-Wallis indique que les différences de performances entre les valeurs de *#formes* sont significatives. Ce résultat est contre-intuitif dans la mesure où l'identification de l'intrus ne *peut pas* être réalisée en se fiant à la dimension *Forme*. Nous pensons que ce résultat est dû à un sur-apprentissage partiel du modèle convolutif qui oriente systématiquement sa stratégie vers une étude des formes pour résoudre la tâche. Lorsque la forme est la seule dimension rendant l'intrus unique (*i.e.*, type *shape*), le test de Kruskal-Wallis révèle que les différences de performances entre les valeurs de *#formes* sont significatives, avec un seuil d'acceptabilité de $\alpha = 0.025$. Cependant, aucune différence significative n'est trouvée entre des paires de valeurs par le test post-hoc. Puisque les taux d'erreur sont toujours inférieurs à 1%, nous pourrions supposer que la tâche est toujours très facile quel que soit le nombre de formes, tant que l'intrus est encodé par une seule dimension : sa forme. Cependant, cela va à l'encontre des hypothèses de la théorie de la perception à ce sujet (voir Section V.2). Puisque nous savons également que les réseaux de neurones convolutifs sont très efficaces pour détecter les contours, nous pensons que cette architecture favorise l'identification d'intrus déterminés par leur forme. Ainsi, nous considérons que c'est un point de divergence entre notre architecture de réseau de neurones (*i.e.*, *ResNet*) et le traitement de l'information des humains. Cette divergence sera étudiée dans l'analyse de corrélations entre les performances du DNN et des participants humains dans la Section V.6. Enfin, les taux d'erreur en fonction de *#formes* dans les données de type *conjunction* suivent la même tendance que pour les variations de *#couleurs* : la difficulté est seuillée entre *#formes* $= 2$ et *#formes* > 2.

Couleur d'intrus. La *couleur* de l'intrus, à l'inverse de la *forme* de l'intrus, impacte la difficulté de la tâche dans certaines conditions. Cependant, comme nous pouvons l'observer dans la Figure V.5, ce paramètre n'a d'impact significatif que dans le cas général et lorsque l'intrus est unique par sa couleur (*i.e.*, type *color*). Le cas général étant une agrégation des différents types, cela veut principalement dire que les différences sont significatives dans les données de type *color*. Ce résultat semble naturel étant donné que c'est la dimension qui encode l'intrus en tant que tel. Cependant, il ne nous semble pas possible de transposer la difficulté du réseau à reconnaître un intrus en fonction de sa couleur, à la difficulté que cela représenterait pour des humains. En effet, le modèle travaille sur des images dans l'espace de couleur RVB (Rouge, Vert, Bleu) et il est connu que cet espace de couleur n'est pas directement associé à la perception humaine [250]. Aussi, nous n'excluons pas qu'il puisse y avoir une corrélation entre l'humain et le modèle sur ce paramètre, mais cette hypothèse ne semble pas évidente. Dans le cas où la couleur ne sert pas à encoder l'intrus (*i.e.*, type *shape*), la couleur de l'intrus n'a pas d'effet significatif. Sur les données de type *redundant*, l'intrus peut être trouvé grâce à sa couleur ou sa forme (ou les deux). Comme mentionné précédemment, la *forme d'intrus* n'a jamais présenté d'impact significatif sur les performances du modèle. Puisque la *couleur d'intrus* n'a pas d'effet significatif non plus, nous en déduisons que la difficulté de la tâche ne dépend pas de la forme ou de la couleur de l'intrus dans les données de type *redundant*. Avec le même raisonnement, nous en concluons que la tâche est difficile à résoudre dans les données de type *conjunction* quelle que soit la couleur ou la forme de l'intrus.

Dans cette section, nous avons peu mentionné l'impact de la variation des paramètres sur les données de type *redundant*. Comme la Figure V.5 le montre, aucune variation de paramètre n'a

eu d'effet significatif sur les performances du modèle. Le taux d'erreur moyen sur les données de type *redundant* est $< 1\%$. Nous pouvons en conclure qu'il n'y a pas de condition univariée qui affecte la difficulté de la tâche lorsque l'intrus est à la fois unique par sa couleur et par sa forme.

V.4.4 Discussion

V.4.4.1 Limitations

Comme mentionné dans la Section V.4.1, les métriques de qualité ne peuvent jamais modéliser parfaitement l'efficacité de la perception humaine sur une tâche et une visualisation. Il existe toujours des conditions dans lesquelles le score de la métrique diverge de la capacité effective de la perception humaine. L'idéal est alors de pouvoir identifier, voire anticiper ces conditions de divergence pour éviter les mauvaises interprétations des résultats. Tout au long de la Section V.4.3, nous avons déjà identifié des comportements qui ne semblent pas correspondre au comportement qu'aurait notre perception sur cette tâche de recherche visuelle. Ces conditions sont discutées dans la suite de cette section, et nous en approfondirons certaines dans l'étude de corrélations de la Section V.6.

Dans la Section V.4.3, nous avons pu observer que le modèle est beaucoup plus efficace que ce à quoi nous nous attendions sur les données de type *shape*. Nous pensons que dans cette condition, sa stratégie est trop efficace pour représenter fidèlement les capacités de la perception humaine. Cette efficacité provient certainement de l'architecture du réseau basée sur les convolutions. Ces opérations sont conçues pour détecter les contours des objets, et donc en extraire de l'information. Il n'est donc pas surprenant que cette architecture favorise les cas où l'intrus est identifiable par sa forme. Un autre biais que le modèle pourrait avoir appris est de compter le nombre de pixels colorés par forme, notamment dans les données de type *shape* et *redundant*. Puisque toutes les formes ont une surface différente, l'intrus est celui dont le compte de pixels n'apparaît qu'une fois s'il est unique par sa forme. Ces deux biais combinés, il est probable que le réseau ait appris à détourer les stimuli et compter les pixels colorés qu'ils contiennent pour résoudre la tâche lorsque la forme de l'intrus est suffisante pour l'identifier. Cette condition sera examinée dans l'étude de corrélations (voir Section V.6) où nous verrons en effet que les résultats du modèle ne correspondent pas aux performances mesurées sur les participants humains.

Il est aussi important de noter que malgré les avantages qu'apporte l'approche par réseau de neurones profond (voir Section V.4.1), leur utilisation vient également avec leur défauts. L'effet *boîte noire* de ces réseaux rend parfois difficile l'interprétation de leur comportement, bien que nous ayons minimisé les modifications de l'architecture. Ici, nous ne pouvons pas expliquer pourquoi la *couleur d'intrus* a un impact significatif lorsque la couleur sert à encoder l'intrus (*i.e.*, type *color*) alors que la *forme d'intrus* n'est pas utile dans le cas correspondant (*i.e.*, type *shape*). Nous pensons également qu'il n'est pas sûr d'estimer l'efficacité d'une couleur pour encoder l'intrus à partir des performances du réseau. En effet, ces modèles sont communément entraînés sur des images dont l'espace de couleur est RVB et nous savons que cet espace n'est pas directement associé au système perceptif humain [250].

Enfin, comme mentionné dans la Section V.3.3, certaines configurations de paramètres ne peuvent pas être générées. Les taux d'erreur correspondants aux valeurs 1 *#couleurs* et 1 *#formes* n'agrègent chacun qu'un type de données (respectivement type *shape* et *color*). Concrètement, dans le cas général, le taux d'erreur avec 1 *#couleurs* (voir Figure V.5) est plus faible que n'importe quelle autre valeur dans ce graphique. Toutefois, les données qui mènent à ce taux d'erreur sont toutes du type *shape*, tandis que les taux d'erreur des autres valeurs sont agrégées des différents types. Bien que ce biais existe, nous pensons que son importance reste minime

pour notre étude car il est peu probable que le seuil de difficulté de la tâche soit atteinte avec une valeur d'hétérogénéité de 1.

V.4.4.2 Hypothèses

Dans ce qui suit, nous présentons les hypothèses que nous étudierons pendant l'évaluation utilisateur dans la Section V.5. Ces hypothèses sont basées sur nos connaissances de la littérature et des résultats de l'évaluation automatique.

H_{type} : **La difficulté en fonction du *type* devrait suivre l'ordre (du plus simple au plus difficile) : *redundant*, *shape* et *color*, *conjunction*.** Cette hypothèse se base sur les conclusions de Quinlan et Humphreys [179] qui ont montré que plus une cible partage d'attributs visuels avec des distracteurs, plus elle est difficile à identifier. Elle corrobore partiellement les résultats observés dans l'évaluation automatique.

H_{conj} : **Les données de type *conjunction* sont les plus difficiles. La difficulté dans cette condition augmente très rapidement avec *#couleurs* et *#formes* et atteint un seuil.** Nous pensons que ce type est le plus complexe et permettra d'observer la capacité limite d'attention le plus tôt. Comme observé durant l'évaluation automatique (voir Section V.4.3), la tâche atteint un seuil de difficulté au delà de 2 *#couleurs* ou *#formes*. Le fait que la recherche conjonctive soit plus compliquée que la recherche de caractéristique simple est déjà documenté dans la littérature [169, 176, 179].

H_{red} : **Les données de type *redundant* sont les plus faciles. La difficulté n'est affectée ni par *#couleurs*, ni par *#formes***, comme montré par Nothelfer *et al.* [243] et observé dans la Section V.4.3.

H_{color} : **Lorsque la couleur est le seul paramètre rendant l'intrus unique, la difficulté de la tâche augmente avec *#couleurs*. À l'inverse, si la couleur n'est pas une dimension utile pour identifier l'intrus, *#couleurs* n'a aucun effet sur la difficulté de la tâche.** Cette hypothèse corrobore à la fois les résultats de l'évaluation automatique et les conclusions de la littérature (voir Section II.4.1).

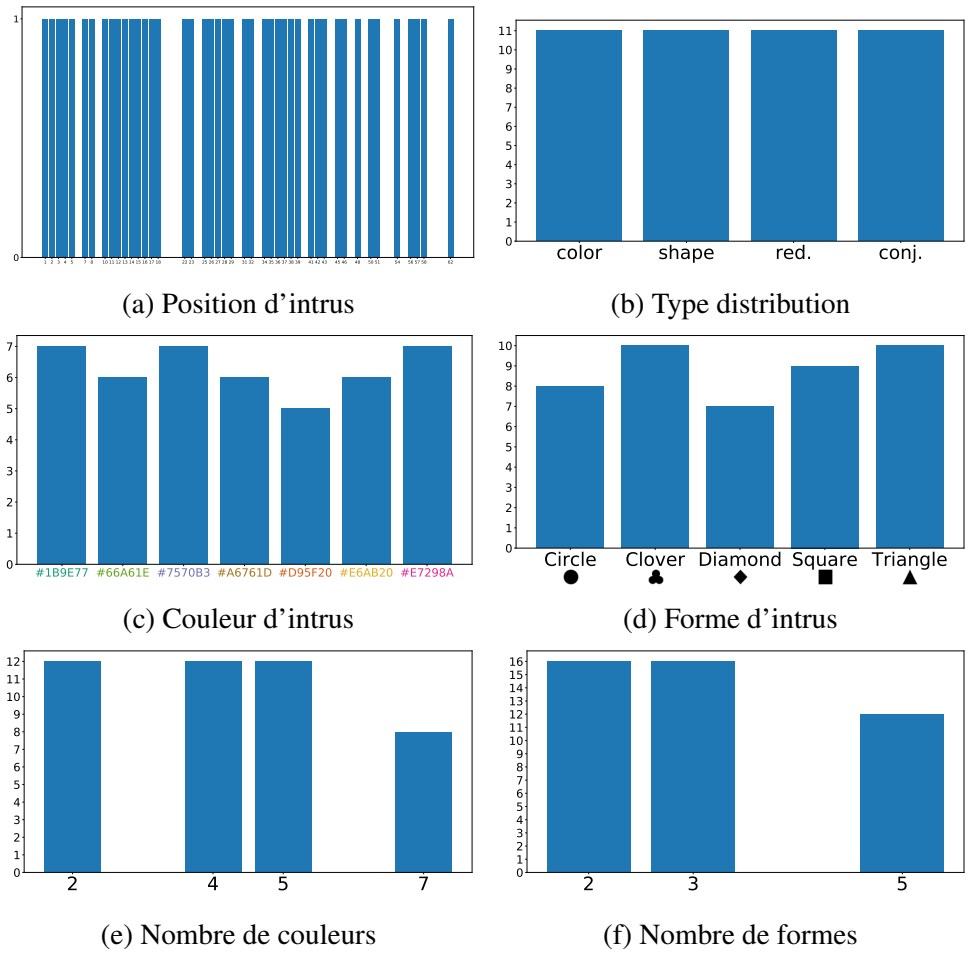
H_{shape} : **Lorsque la forme est le seul paramètre rendant l'intrus unique, la difficulté de la tâche augmente avec *#formes*. À l'inverse, si la forme n'est pas une dimension utile pour identifier l'intrus, *#formes* n'a aucun effet sur la difficulté de la tâche.** De la même manière que pour H_{color}, cette hypothèse se base sur les connaissances de la littérature (voir Section II.4.1). Celle-ci va à l'encontre des résultats observés dans la Section V.4.3. Néanmoins, nous avons discuté dans la Section V.4.4.1 du fait que ses performances sur les données de type *shape* pouvaient difficilement être une bonne représentation du système de perception.

V.4.4.3 Échantillonnage de l'Espace de Paramètres

L'objectif principal de l'évaluation automatique basée sur le réseau de neurones profond est d'obtenir une quantification de la difficulté de la tâche en fonction de certains des paramètres qui la composent pour pouvoir sous-échantillonner l'espace de paramètres. Ce sous-échantillonnage est nécessaire pour que l'évaluation puisse s'effectuer en un temps raisonnable pour ne pas introduire de biais relatifs à l'épuisement, ou à une sur-spécialisation [184]. Cependant, il doit également être suffisamment conséquent pour permettre une étude assez fine de la question de recherche posée (voir Section V.4.4.2). Cette approche est l'inverse d'études telles que celle de Haroz et Whitney [181], dans laquelle 5 participants ont réalisé 960 tâches d'une dizaine de secondes chacun. Pour notre expérience, nous souhaitons avoir un panel de participants plus large, mais que chacun d'entre eux ne réalise que quelques dizaines de tâches ; notamment

car nous savons que la tâche est fatigante à réaliser et peut nécessiter plusieurs dizaines de secondes. Cette section présente nos choix pour réduire l'espace de paramètres des cas à tester lors de l'évaluation utilisateur, et qui se basent principalement sur les résultats de l'évaluation automatique.

Notre étude visant à mesurer l'impact de l'hétérogénéité d'une visualisation sur la détection d'intrus, les paramètres *type*, *#formes* et *#couleurs* sont les plus importants car ils encodent l'hétérogénéité de la représentation. Ils doivent donc rester représentés équitablement dans l'espace de paramètres sous-échantillonné, et chacune de leur valeur doit avoir plusieurs occurrences. Les valeurs des autres paramètres sont uniformément distribuées à l'intérieur des combinaisons sélectionnées de *type-#formes-#couleurs*. Cependant, cette première sélection ne suffit pas à atteindre la taille de l'espace de paramètres recherchée. Pour continuer le réduire, certaines valeurs de *#formes* et *#couleurs* ne sont pas conservées. Premièrement, la valeur 1 est retirée pour *#formes* et *#couleurs*. Comme nous l'avons vu dans les Sections V.4.3 et V.4.4.1, il n'est pas utile d'étudier la valeur 1 et les résultats sur celle-ci pourraient même être biaisés. Pour retirer d'autres valeurs de *#formes* et *#couleurs*, notre stratégie est de supprimer des valeurs en minimisant la perte de différences significatives observées dans l'évaluation automatique (*i.e.*, supprimer le moins d'*arcs* possible dans la Figure V.5). Nous filtrons ainsi les valeurs 3 et 6 pour *#couleurs*, ce qui supprime 7 différences significatives dans le cas *Général*, 8 sur le type *color*, 0



(a) Position d'intrus

(b) Type distribution

(c) Couleur d'intrus

(d) Forme d'intrus

(e) Nombre de couleurs

(f) Nombre de formes

FIGURE V.6 : Distributions des valeurs de paramètres considérés dans les 44 objets expérimentaux de l'évaluation utilisateur. La stratégie de sous-échantillonnage de l'espace de paramètres est décrite dans la Section V.4.4.3.

sur les types *shape* et *redundant*, et 2 sur le type *conjunction* (voir les arcs dans la Figure V.5, une fois la valeur 1 déjà retirée). Pour *#formes*, nous filtrons la valeur 4, ce qui retire 1 différence significative dans le cas *Général*, 1 sur le type *color*, 0 sur les types *shape* et *redundant*, et 1 sur le type *conjunction*.

Ainsi, les valeurs de *#couleurs* sont réduites de {1, 2, 3, 4, 5, 6, 7} à {2, 4, 5, 7}, et celles de *#formes* sont réduites de {1, 2, 3, 4, 5} à {2, 3, 5}. Puisque nous excluons toujours la combinaison [7 *#couleurs*; 5 *#formes*] (voir Section V.3.3), nous arrivons à un total de $(|\#couleurs| * |\#formes| - 1) * |type| = 11 * 4 = 44$ cas de test pour l'évaluation utilisateur, ce qui correspond à l'objectif que nous avons fixé au début de la section. Les paramètres restants (*i.e.*, position, forme et couleur d'intrus) sont distribués uniformément dans les 44 cas sélectionnés suivant les critères précédents. Les distributions finales des paramètres dans les cas de tests sélectionnés pour l'évaluation utilisateur sont présentées dans la Figure V.6.

V.5 Évaluation Utilisateur

Cette section présente la mise en place, les choix de conception, et les résultats de l'évaluation utilisateur. Cette évaluation a pour objectif de mesurer la capacité limite d'attention d'utilisateurs sur une tâche de détection d'intrus dans une grille régulière et d'hétérogénéité variable de 8×8 stimuli définis par une couleur et une forme. De plus, nous étudions les hypothèses énoncées dans la Section V.4.4.2. Pour rappel, nous appelons *objet expérimental* une image et tous les paramètres qui la constituent, et sur laquelle les participants doivent résoudre la tâche.

V.5.1 Cadre Expérimental

V.5.1.1 La Tâche

Comme mentionné dans la Section V.3.1, la tâche consiste en l'identification d'un intrus dans une grille de 8×8 stimuli. Une grille est toujours exactement constituée d'un intrus unique et 63 distracteurs dont la forme-couleur est répétée. Dans cette évaluation utilisateur, une limite de temps de 30 secondes est fixée pour résoudre la tâche sur chaque objet expérimental. Elle vise à encourager les utilisateurs à rester suffisamment concentrés tout au long de l'évaluation en leur fixant un rythme. Cette limite de temps permet également d'obtenir un bon compromis entre le temps de réponse et les taux d'erreur mesurés. Lorsqu'un participant n'arrive pas à résoudre la tâche sur un objet expérimental en 30 secondes, son absence de réponse est enregistrée comme tel. Dans la suite, ces cas seront appelés *OOT* pour « Out Of Time ».

V.5.1.2 Jeu et Ordre des Objets Expérimentaux

Les objets expérimentaux utilisées dans cette évaluation sont extraits du jeu de *test* défini dans la Section V.3.3 pour correspondre à l'espace de paramètres réduit défini dans la Section V.4.4.3.

Dans ce type d'évaluation, l'ordre de présentation des objets expérimentaux aux utilisateurs est important. Par exemple, si le premier et le dernier objet sont toujours les mêmes, il est possible que les performances soient moins bonnes sur le premier car les utilisateurs gagnent en expérience sur la tâche tout au long de l'évaluation, et non pas parce que cet objet est réellement plus facile. Dans cette évaluation, nous générons un ordre aléatoire des objets expérimentaux, et nous créons un carré latin de cet ordre (*i.e.*, une matrice où chaque ligne représente le même ordre, mais avec un décalage à chaque fois). Chaque participant voit ainsi les objets expérimentaux dans

l'ordre déterminé par la ligne du carré latin qui lui est attribué. En outre, tous les participants voient les objets dans le même ordre, mais ne commencent pas par le même.

V.5.1.3 Protocole d'Évaluation

La première étape de l'évaluation consiste à lire l'énoncé du problème. Cet énoncé présente les couleurs, les formes et les types d'intrus pouvant apparaître pendant l'expérience, et montre un exemple de grille. Les participants sont libres de poser les questions qu'ils souhaitent et ils ne passent à l'étape d'après qu'une fois la tâche bien comprise. Ensuite, chaque participant suit un tutoriel constitué de 8 objets expérimentaux. Les 4 premiers sont présentés déjà solutionnés, et les 4 suivants sont à résoudre. Les participants ont un retour sur l'exactitude de leurs réponses dans le tutoriel, et ils peuvent recommencer ce dernier autant de fois que nécessaire. Lorsqu'il est prêt, un participant peut lancer l'évaluation.

Suivant les recommandations de Purchase [184], nous avons intégré 8 faux objets expérimentaux au début de l'évaluation. Tous les participants voient ces faux objets dans le même ordre, dès le début. L'objectif de ces faux cas est de préparer les participants et d'harmoniser leur niveau de concentration et d'expérience sur la tâche avant que l'évaluation ne commence effectivement. Les 44 objets expérimentaux sont ensuite présentées aux participants avec un délai de 3 secondes entre chaque objet. Pour chacune, la réponse et le temps de réponse des participants sont sauvegardés. Après le 26^{eme} objet, une pause optionnelle d'une minute est proposée aux participants. Lorsque les 44 objets sont traités, les participants remplissent un questionnaire leur demandant ce qu'ils ont trouvé facile, ou difficile, ainsi que leur stratégie pour résoudre la tâche dans ces deux cas.

L'évaluation dure approximativement 20 à 30 minutes. En comprenant le temps d'introduction, du tutoriel, et du questionnaire final, la durée total de l'expérience d'un participant varie entre 45 minutes et 1 heure. Cette durée est satisfaisante pour éviter l'*effet de fatigue* [184].

V.5.1.4 Interface Graphique de l'Évaluation

L'outil d'évaluation consiste en une page web dédiée à cette expérience. La page est affichée en plein écran sur un écran de résolution 1920×1080 pixels. Chaque objet expérimental est présenté avec un ratio 1 :1 (256×256 pixels) au centre de l'écran, avec une bordure noire pour le dissocier du fond. L'énoncé raccourci de la tâche, un compteur d'avancement et la durée restante

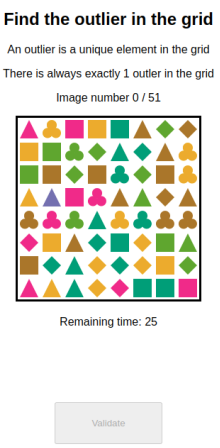


FIGURE V.7 : Capture d'écran de l'outil d'évaluation sur un exemple d'objet expérimental.

sur l'objet en cours sont également présentés à l'utilisateur sur cette page. Pour résoudre la tâche sur un objet, un participant doit cliquer sur le stimulus qu'il pense être l'intrus, ce qui crée un encadré autour sur ce stimulus dans l'image. Pour valider sa réponse, il doit cliquer sur un bouton dédié. Le bouton de validation est suffisamment éloigné de la grille pour ne pas perturber l'attention des participants pendant leur recherche, et suffisamment gros pour ne pas qu'ils aient besoin de diriger leur attention vers lui pour l'utiliser. Une capture d'écran de l'outil sur un objet expérimental présenté aux participants est présentée dans la Figure V.7.

V.5.1.5 Participants Impliqués

Les participants à cette évaluation sont 18 hommes et 6 femmes âgés de 21 à 50 ans pour une moyenne de 24.8 ans. Tous sont étudiants, doctorants, ingénieurs ou docteurs en informatique. Tous ont une vision parfaite ou parfaitement corrigée et aucun n'est daltonien.

V.5.2 Résultats

Pendant l'évaluation, les temps de réponse (RT pour « Response Time ») des participants ont été mesurés. À partir de leurs réponses, nous calculons également leurs taux d'erreur (ER pour « Error Rate ») agrégées en fonction de différents paramètres. Nous utilisons les résultats de 21 participants parmi les 24. Après avoir étudié leurs comportements, performances et réponses aux questionnaires, les résultats de 2 participants sont largement hors de la distribution du groupe conservé (*i.e.*, des performances déviant de plus de 1.5 fois l'écart type). La décision de retirer le troisième participant est basée sur ses performances, ses réponses au questionnaire et son comportement pendant l'évaluation qui laissent suggérer que la tâche n'a pas été correctement comprise ou sérieusement abordée.

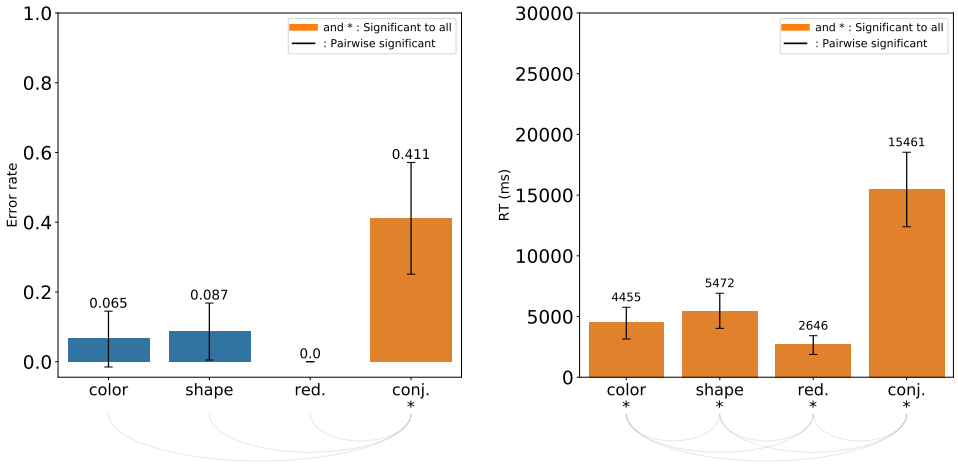
La présentation des performances des participants de cette évaluation est faite avec les mêmes graphiques que l'évaluation automatique de la Section V.4. La seule différence est que chaque barre représente la performance moyenne de plusieurs participants sur les mêmes cas. En conséquence, nous présentons également l'écart-type entre ces participants sur chaque barre. Une description de ces graphiques résultats ainsi qu'un exemple de lecture sont donnés dans la Section V.4.3.

V.5.2.1 Résultats Quantitatifs et Étude des Hypothèses

Dans cette section, nous étudions les résultats de l'évaluation utilisateur pour valider ou invalider les hypothèses construites dans la Section V.4.4.2.

Comme lors de l'évaluation automatique, nous conduisons d'abord un test de Kurskal-Wallis [249] pour vérifier si les variations de performances entre les valeurs de paramètres sont dues au hasard. Ce test est conduit indépendamment pour les taux d'erreur et les temps de réponse. Pour les RT, seules les réponses validées sont considérées (*i.e.*, celles où les participants ne sont pas arrivés à court de temps). En moyenne, cela est arrivé sur 5.5 cas par participant. La distribution de ces cas est présentée dans la Figure V.9. Nous étudions également les résultats dans le cas *Général* et *par valeur de Type* d'intrus. Le protocole d'évaluation de la significativité des différences de performances est le même que pour l'évaluation automatique (voir Section V.4.3). Les résultats des participants sont présentés dans les Figures V.8 et V.10.

Nous **validons** l'hypothèse $\boldsymbol{H_{type}}$ car les résultats la corrobore pleinement. Comme nous pouvons l'observer dans la Figure V.8, les ER et RT suivent les mêmes tendances en fonction des variations de valeur de Type. Le type *redundant* mène aux meilleurs performances : aucun participant n'a commis d'erreur sur les objets expérimentaux de ce type, avec un temps réponse



(a) Taux d'erreur (ER) par Type (b) Temps de réponse (RT) par Type

FIGURE V.8 : Taux d'erreur et temps de réponse des participants par Type d'intrus. La description des graphiques résultats ainsi qu'un exemple de lecture sont donnés dans la Section V.4.3.

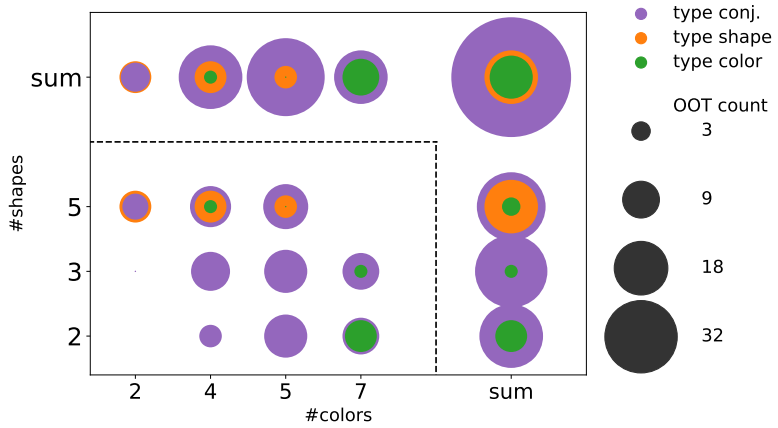


FIGURE V.9 : Nombre de cas pour lesquels les participants n'ont pas réussi à résoudre la tâche en 30 secondes (*i.e.*, OOT pour « Out Of Time »). Ces résultats sont étudiés par *type*, *#formes* et *#couleurs*. Il y a 116 cas OOT au total (5.5 par participant en moyenne), 74% desquels sont de type *conjunction*, 16% de type *shape* et 10% de type *color*.

moyen inférieur à 3 secondes. Bien que la différence de ER entre les objets de type *color* et *shape* ne soit pas significative (6.5% et 8.7% respectivement), les objets de type *color* sont significativement plus rapide à résoudre (4.5s) que ceux de type *shape* (5.5s). Enfin, les objets expérimentaux de type *conjunction* sont de loin les plus difficiles à résoudre, avec un ER de 41.1% pour un RT moyen de 15.5s. De plus, ces objets comptent aussi pour 74% des cas OOT (voir Figure V.9). Si ces cas avaient été résolus, ils auraient nécessité plus de 30s, ce qui rend les résultats de RT sur les objets de type *conjunction* plus optimistes qu'ils ne le sont vraiment.

Nous **validons** également $\boldsymbol{H_{conj}}$, bien que la lecture des résultats soit indirecte. Dans la Figure V.8, les objets expérimentaux de type *conjunction* ont distinctement mené à de moins bonnes performances sur ER et RT. De ce fait, nous pouvons en conclure que les objets de type *conjunction* sont les plus difficiles. Dans les graphiques A-5 et B-5 de la Figure V.10, nous pouvons observer que *#couleurs* a un effet significatif sur le type *conjunction*. L'effet est uniquement significatif entre *#couleurs* $= 2$ (11% ER ; 11.8 s RT) et *#couleurs* > 2 (>42% ER ;

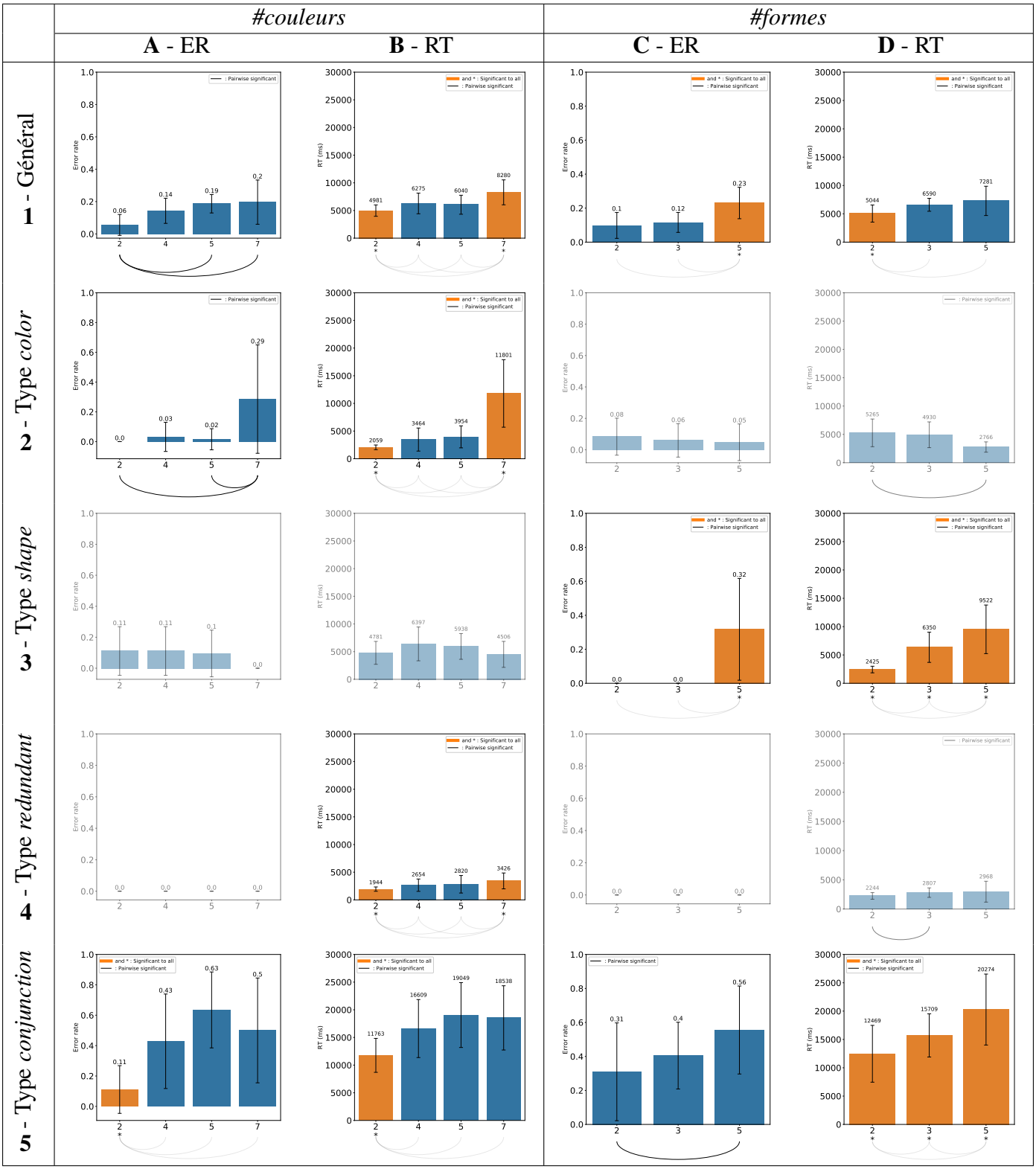


FIGURE V.10 : Taux d'erreur (ER pour « Error Rate ») et temps de réponse (RT pour « Response Time ») des participants. La première ligne présente les résultats généraux, tandis que les lignes suivantes présentent les résultats séparément pour chaque valeur de *Type*. La description des graphiques résultats ainsi qu'un exemple de lecture sont donnés dans la Section V.4.3. Les lignes et colonnes sont labellisées pour permettre une meilleure identification dans le texte.

>16.6 s RT), ce qui corrobore les résultats observés dans l'évaluation automatique. Cela confirme que la difficulté de la tâche augmente en même temps que *#couleurs* mais atteint un seuil très rapidement. Dans les graphiques C-5 et D-5 de la Figure V.10, nous n'observons pas directement d'effet seuil sur ER ni RT. Comme mentionné plus tôt, les cas que les participants n'ont pas réussi à résoudre dans le temps imparti (*i.e.*, OOT) ne sont pas inclus dans la moyenne des RT car cela aurait été incorrect de les considérer comme des *mauvaises réponses en 30 secondes*. Puisque 86 sur les 231 objets de type *conjunction* sont OOT (voir Figure V.9), cela signifie que plus d'un tiers des objets expérimentaux de ce type ne sont pas inclus dans la moyenne des RT. Ainsi, les performances RT dans cette condition sont amputées de cas dont le temps de réponse serait nécessairement supérieur à 30*s* et rend les performances observées dans graphique plus optimistes qu'elles ne le sont réellement.

Nous **acceptons** $\boldsymbol{H_{red}}$ avec une restriction dans notre contexte. La Figure V.10 montre que toutes les réponses sont correctes sur les objets expérimentaux de type *redundant*, et que ceux-ci sont significativement plus rapides à résoudre que les autres types. Ainsi, le type *redundant* est bien le plus facile à résoudre. Toutes les réponses étant correctes, nous pouvons en conclure que les variations de *#couleurs* et *#formes* n'impactent pas la difficulté de la tâche dans ce cas. Cependant, nous observons dans la cellule B-4 de la Figure V.10 que la difficulté impacte significativement le RT des participants. Néanmoins, ces variations de temps de réponse sont de très faible amplitude, et tous les objets expérimentaux ont été résolus correctement en 1.9 à 3.4 secondes par tous les participants. Bien que ces écarts soient statistiquement significatifs, nous pensons qu'ils ne sont pas assez grand pour rejeter l'hypothèse dans un contexte de visualisation où les tâches de recherche visuelle dans des représentations complexes peuvent nécessiter plusieurs dizaines de secondes.

Nous **validons** $\boldsymbol{H_{color}}$ car ses deux propositions sont validées par les performances des participants. Lorsque l'intrus est uniquement encodé par sa couleur (*i.e.*, type *color*), nous observons dans la cellule A-2 de la Figure V.10 que le ER reste faible jusqu'à $\#couleurs = 7$ où il augmente drastiquement. Le même comportement peut être observé sur le RT (cellule B-2). Lorsque l'intrus n'est pas encodé par sa couleur (*i.e.*, type *shape*, cellules A-3 et B-3), celle-ci n'a pas d'effet significatif sur les performances.

Enfin, nous **validons** $\boldsymbol{H_{shape}}$ de la même manière : ses deux propositions sont validées par les résultats de l'évaluation utilisateur. Lorsque l'intrus est uniquement encodé par sa forme (*i.e.*, type *shape*), nous observons dans les cellules C-3 et D-3 de la Figure V.10 que les performances sont significativement affectées par les variations de *#formes*, bien que ER et RT ne suivent pas tout à fait la même tendance. Les participants n'ont jamais échoué à résoudre la tâche lorsque l'intrus est encodé par sa forme et que $\#formes < 5$. Par ailleurs, les temps de réponse semblent croître linéairement avec *#formes*. Cependant, la Figure V.9 montre que les participants n'ont pas réussi à résoudre la tâche sur des objets expérimentaux de type *shape* 18 fois, systématiquement avec $\#formes = 5$. Ainsi, le temps de réponse aurait été supérieur à 30 secondes dans 18 cas sur $(|\#couleurs| - 1) * 21 = 63$, soit 28.6% des cas. Les résultats dans la barre 5 *#formes* du graphique de la cellule D-3 sont donc trop optimistes. Lorsque l'intrus n'est pas encodé par sa forme (*i.e.*, type *color*, cellules C-2 and D-2), *#formes* n'a pas d'impact significatif sur les performances.

V.5.2.2 Résultats Qualitatifs

Les résultats qualitatifs sont basés sur les réponses des participants au questionnaire de fin d'évaluation (voir Section V.5.1.3), qui traite de leur avis quant aux paramètres rendant la tâche facile ou difficile, et leur stratégie dans les deux cas.

La première question était d'ordonner, du plus facile au plus difficile, les différentes valeurs de *type*. Pour cette question, la grande majorité des participants (20/21) a classé le type *redundant* comme le plus facile, et *conjunction* comme le plus difficile. Plus de la moitié des participants (14/21) a classé *color* comme plus facile que *shape*. Ces résultats corroborent les performances observées dans l'évaluation.

La deuxième question était de signaler si une *couleur d'intrus* semblait plus facile à détecter que d'autres, et si oui, laquelle. Dix participants ont trouvé que **#E7298A** était plus facile à trouver que les autres couleurs. Quelques participants ont trouvé que la tâche était plus complexe lorsque **#66A61E** et **#1B9E77** étaient présents dans la même image. Un petit nombre de participants a considéré que les intrus encodés avec **#D95F02** étaient plus faciles à trouver. **#A6761D** et **#E6AB02** n'ont pas été mentionnés par les participants. Enfin, plusieurs participants ont signalé que la couleur de l'intrus ne leur paraissait pas avoir d'importance tant que le contraste avec les stimuli voisins était suffisamment prononcé. Bien que notre évaluation ne cherche pas à mesurer l'impact de la saturation de la couleur de l'intrus sur la difficulté de la tâche, ni de son voisinage, ceux-ci ont évidemment un effet que nous avons cherché à minimiser. Il serait néanmoins intéressant d'évaluer leur impact dans une prochaine étude en adaptant le réseau de neurones profond de l'évaluation automatique pour traiter les images dans un espace de couleur plus proche de la perception humaine. Cette adaptation rendrait également plus fiable l'interprétation des résultats du réseau en ce qui concerne l'efficacité des couleurs. Cette piste serait une extension aux travaux de Camgöz *et al.* [251, 252] sur les effets de la saturation sur l'attention.

La même question est ensuite posée aux participants sur la *forme d'intrus*. Les réponses ont été plus variées, et seuls le Carré ■ et le Cercle ● ont été régulièrement cités comme plus faciles à identifier. Losange ◆ n'a jamais été mentionné comme forme facile. Plus de la moitié des participants a trouvé que la difficulté à identifier l'intrus par sa forme était très dépendante de l'aspect des distracteurs. Plusieurs réponses donnaient des exemples similaires à : « trouver un triangle parmi des cercles est facile, alors que trouver un triangle parmi des losanges est difficile ».

La quatrième question demandait aux participants le nombre de couleurs et formes à partir duquel, selon eux, la tâche devenait difficile. Étant donné qu'ils n'avaient pas le nombre de couleurs/formes associé à chaque objet expérimental pendant l'évaluation et qu'ils n'ont pas particulièrement cherché à les compter, les participants n'avaient pas d'idée précise du nombre seuil qui avait pu rendre la tâche difficile. En revanche, la majorité des réponses donnait systématiquement un *#couleurs* supérieur au *#formes* (*e.g.*, (3 *#couleurs* ; 2 *#formes*), (4 ; 3) ou même (5 ; 4)). Cette tendance suggère qu'ils ont trouvé l'hétérogénéité induite par *#couleurs* moins dérangeante que *#formes*.

L'avant-dernière question était de présenter la stratégie qu'ils avaient employée pour résoudre la tâche. La réponse que nous avons le plus souvent reçue fait part de la stratégie décrite en suivant. Tout d'abord, regarder si l'intrus « ne saute pas aux yeux ». Si ce n'est pas le cas, identifier les formes et les couleurs présentes dans la visualisation. Pour chaque couleur, puis pour chaque forme, regarder rapidement s'il existe un doublon dans l'image. Enfin, si ces étapes n'ont pas abouti, examiner les stimuli un par un pour les comparer aux autres jusqu'à en trouver un incomparable. Cette stratégie est révélatrice d'un comportement typique de recherche visuelle avec un processus *top-down* puis *bottom-up* (voir les Sections I.4.2 et II.4.1).

La dernière question demandait aux participants ce qui, de manière générale, leur avait semblé complexifier la tâche. Deux facteurs principaux se sont distingués. Premièrement, la similarité entre l'intrus et les distracteurs dans l'image. Ce facteur corrobore la théorie de Duncan et Humphreys [176] à propos des hétérogénéités T-N et N-N où T représente l'intrus et N les

distracteurs (voir Section II.4.1). Le second facteur est le voisinage direct de l'intrus dans une image, qui se réfère sans doute aux *conjonctions illusoires* [169, 177].

V.5.2.3 Capacité Limite d'Attention des Dimensions *Couleur* et *Forme*

Pour rappel, ce que nous appelons la *capacité limite d'attention* de la couleur (ou forme) est le nombre maximum de différentes valeurs (*i.e.*, l'hétérogénéité) que peut prendre cet attribut dans une visualisation avant que celle-ci ne devienne trop complexe pour résoudre la tâche de recherche d'intrus.

L'étude des hypothèses H_{color} et H_{shape} nous a montré que *#couleurs* et *#formes* n'avaient pas d'impact significatif sur la difficulté de la tâche lorsque leur dimension respective ne servait pas à encoder l'intrus (*i.e.*, ne le rendait pas unique). Nous considérons uniquement la capacité limite d'attention d'une dimension lorsque le paramètre correspondant sert à encoder l'intrus. Avec H_{red}, nous avons aussi observé que l'hétérogénéité n'avait aucun effet sur la difficulté de la tâche lorsque l'intrus était encodé de manière redondante (*i.e.*, à la fois par sa couleur et sa forme). Ce résultat suggère qu'il n'existe pas de capacité limite d'attention dans ce cas, ou que nous ne l'avons pas atteint dans le contexte que nous avons défini (*i.e.*, avec une hétérogénéité allant jusqu'à 7 couleurs et 5 formes). Ces résultats corroborent des travaux de la littérature [243], bien qu'il ne soit pas toujours possible d'utiliser un encodage redondant de manière efficace dans les visualisations.

Couleur. Lorsque la couleur est la dimension qui rend l'intrus unique, nous observons (voir Figure V.10 A-2 et B-2) que toutes les valeurs de *#couleurs* mènent à un faible taux d'erreur ($<3\%$) et un temps de réponse court (<4s), à l'exception de la valeur 7 (29% ER et 11s RT). Nous pensons que cette baisse drastique des performances est due au dépassement du nombre de couleurs utilisable dans une visualisation, *i.e.*, un dépassement de la capacité limite d'attention. Nous en concluons que lorsque la couleur rend l'intrus unique, la capacité limite d'attention de la couleur est **strictement inférieure à 7**. Dans les cas de conjonction d'attributs (*i.e.*, type *conjunction*), la capacité limite d'attention est **strictement inférieure à 4** comme nous pouvons voir que les performances ne se dégradent plus au delà de 4 couleurs.

Forme. Lorsque la forme est la dimension qui rend l'intrus unique (*i.e.*, type *shape*), les participants ne font aucune erreur jusqu'à 5 formes, seuil à partir duquel le taux d'erreur augmente à 32%. Ce taux est proche de celui obtenu avec 7 couleurs lorsque la couleur sert à encoder l'intrus. Comme nous l'avons observé dans la Section V.5.2.1, bien que les mesures de temps de réponse semblent linéaires sur le nombre de formes, il y a en réalité de nombreux cas avec $\#formes = 5$ pour lesquels les participants n'ont pas eu le temps de résoudre la tâche en 30 secondes (voir Figure V.9). Nous en concluons que la capacité limite d'attention de l'attribut Forme est **strictement inférieure à 5** quand la forme est la dimension qui rend l'intrus unique. Pour la recherche conjonctive (*i.e.*, type *conjunction*), aucun effet seuil n'a pu être observé avec les variations de *#formes* dans cette évaluation. Le nombre de cas OOT (voir Figure V.9) est uniformément réparti entre les différentes valeurs de *#formes* pour les objets expérimentaux de type *conjunction*. Dans ce cas il y a trois possibilité : soit 2 est déjà au delà de la limite, soit 5 est toujours en deçà de la limite, soit la recherche d'intrus dans ce contexte est en effet linéaire sur le nombre de formes.

Ce résultat indique bien que la capacité limite d'attention de l'attribut Couleur est plus grande que celle de l'attribut Forme. Il corrobore les impressions reportées par les participants à l'étude lors du questionnaire (voir Section V.5.2.2). Cela se vérifie également sur la valeur d'hétérogénéité qu'ils ont en commun : 5. Avec 5 couleurs, lorsque la couleur rend l'intrus

unique, les performances des participants sont 1.5% ER ; 4s RT. Avec 5 formes, lorsque la forme rend l'intrus unique, les performances sont largement moins bonnes : 31% ER ; 9.5s RT. À valeur d'hétérogénéité égale, la perception humaine est donc moins sensibles aux variations de couleurs que de formes.

V.5.3 Robustesse des Résultats

Comme Demiralp *et al.* [183], nous testons la robustesse des résultats de notre évaluation à la suppression de données de participants. Des résultats plus robustes dans ce sens ont une plus grande capacité à être généralisés. Dans cette évaluation de robustesse, nous extrayons des sous-ensembles aléatoires de participants de différentes tailles (de 10% à 95% de l'ensemble total) et calculons les performances moyennes de ces sous-ensembles sur chaque paramètre. Nous calculons ensuite un coefficient de corrélation de Spearman [253] entre les performances obtenues par le sous-ensemble avec les performances de tous les participants. L'idée est que si les résultats de l'expérience sont robustes, enlever les données de quelques participants ne doit pas changer les conclusions de l'évaluation.

Pour réduire les incertitudes liées à l'échantillonnage, chaque sous-ensemble de participants de chaque taille est tiré aléatoirement 10 fois, et nous reportons les coefficients de corrélation moyens sur les 10 tirages dans la Figure V.11. Plus le score de corrélation est proche de 1, plus

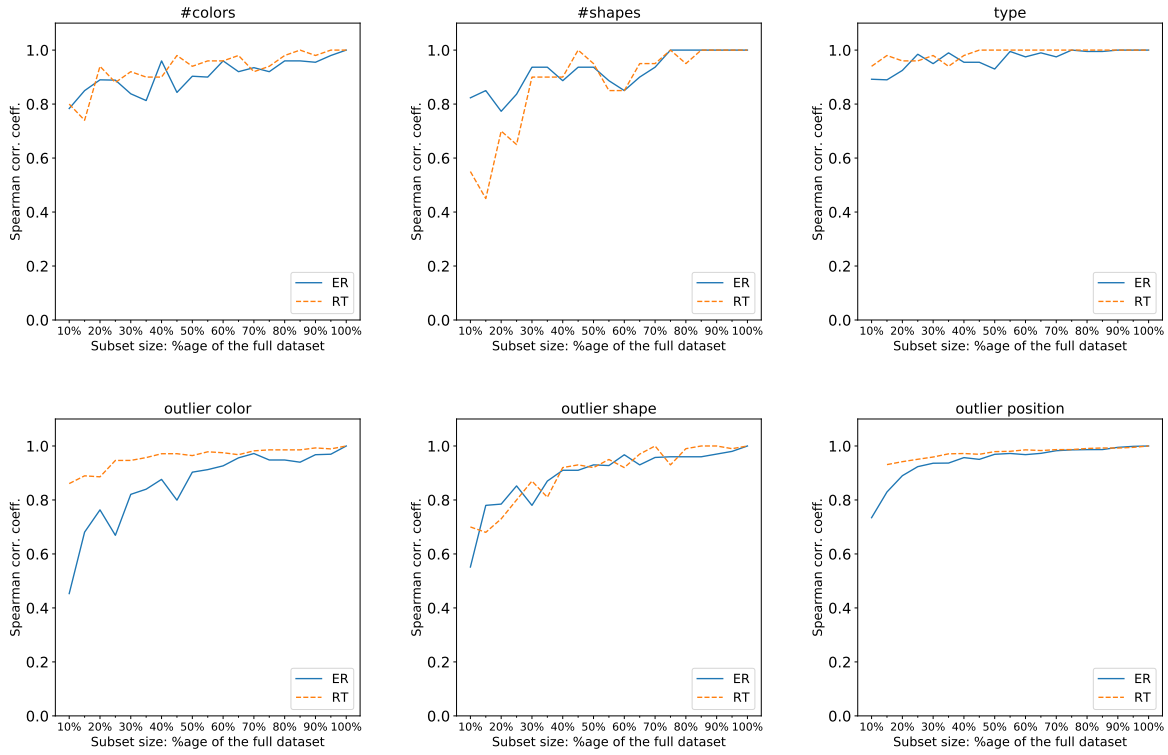


FIGURE V.11 : Coefficients de corrélation de Spearman [253] entre les performances des sous-ensembles aléatoires (de différentes tailles) de participants avec les performances de l'ensemble des participants. L'axe X correspond à la taille de chaque sous-ensemble (en pourcentage de la taille de l'ensemble total, de 10% à 95% avec un pas de 5%) avec lequel les performances sont comparées à l'ensemble total (100%). Les coefficients reportés correspondent à la moyenne sur 10 tirages de sous-ensembles aléatoires pour chaque taille.

les données sont corrélées. Nous observons que les corrélations sont au dessus de 0.5 avec 25% des données des participants, ce qui indique qu'une certaine quantité d'information est bien préservée. Avec 40% à 50% des données, les coefficients de corrélation sont tous entre 0.8 et 1, ce qui signifie qu'avec seulement la moitié de nos participants, nous obtenons les mêmes résultats. Un coefficient de corrélation aussi élevé avec seulement la moitié des participants montre que les résultats de notre évaluation sont suffisamment robustes pour être généralisés.

V.6 Étude des Corrélations entre DNN et Participants

Une des contributions du travail présenté dans ce chapitre est l'utilisation d'un réseau de neurones profond pour estimer automatiquement l'évolution de la difficulté de la tâche en fonction de divers paramètres. Cette évaluation automatique se base sur une hypothèse assez forte : les performances de DNN et d'humains peuvent être corrélées sur une tâche de perception. Quelques travaux ont abordé cette question, et leurs conclusions sont encourageantes [28, 192]. Même dans les cas où la stratégie de traitement de l'information du DNN est différente de celle des humains, certaines propriétés d'une visualisation peuvent affecter les performances des deux.

Dans cette section, nous étudions les corrélations entre les performances du DNN utilisé dans l'évaluation automatique de notre expérience (voir Section V.4) et celles collectées lors de l'évaluation utilisateur (voir Section V.5).

V.6.1 Données de Comparaison

Le terme *donnée* se réfère aux mesures de performance des participants ou du réseau de neurones profond collectées dans les évaluations précédentes. Le terme *objet expérimental* se réfère aux combinaisons de paramètres définis dans le contexte de la tâche et les images qui en résultent.

V.6.1.1 Performances des Participants

Les données des participants humains sont constitués des taux d'erreur et temps de réponse des 21 participants sur les 44 objets expérimentaux de l'évaluation utilisateur. Nous utilisons la moyenne et l'écart type des performances des participants pour mesurer la difficulté d'un objet expérimental. Il y a donc 4 mesures de performance pour chacun des 44 objets expérimentaux : ER moyen, RT moyen, écart type d'ER (noté ER STD) et écart type de RT (noté RT STD). C'est avec ces mesures des participants que nous calculons les corrélation avec le DNN.

V.6.1.2 Performances du Réseau de Neurones Profond

Les données du DNN que nous utilisons sont issues des prédictions du modèle *ResNet* entraîné pour l'évaluation automatique. Nous nous limitons ici uniquement au jeu de *test* (voir Section V.3.3) constitué de 21 056 images. Dans l'évaluation automatique de l'expérience, nous nous sommes basés sur les taux d'erreur de *ResNet* pour évaluer sa difficulté à résoudre la tâche. Pour cette étude de corrélation, nous considérons plusieurs métriques communes dans la communauté d'apprentissage automatique :

- *Confiance* : valeur la plus élevée dans le vecteur de probabilités en sortie du réseau de classification. L'index de cette valeur dans le vecteur de probabilités étant la classe prédite par le réseau, la valeur elle-même est la confiance avec laquelle le réseau prédit cette classe.

- *Entropie catégorielle croisée* : dite CCE pour « Categorical Cross Entropy », c'est une quantification de l'exactitude de la prédiction du réseau en fonction de l'ensemble de son vecteur de prédiction.
- *Aire sous la courbe* : dite AUC pour « Area Under the Curve », cette métrique quantifie à quel point le modèle est capable de distinguer la bonne classe à prédire des autres classes, en se basant sur la fonction d'efficacité du récepteur (ROC pour « Receiver Operator Characteristic »).
- *Marge catégorielle* : mesure à quel point la probabilité prédite par le réseau pour la position correcte de l'intrus est plus grande que la somme des probabilités associées aux autres positions.
- *Taux d'erreur* (ER) $= 1-(VP+VN)/(VP+FP+FN+VN)$
- $Rappel = VP/(VP+FN)$
- $Précision = VP/(VP+FP)$
- $Score\ F1 = 2*(Rappel*Précision)/(Rappel+Précision)$

où VP et VN sont respectivement des prédictions Vrai Positifs/Négatifs ; et FP et FN des Faux Positifs/Négatifs.

Puisque nous connaissons les performances de *ResNet* sur plus d'objets expérimentaux (*i.e.*, jeu de *test* de taille 21 056), que les performances des participants, nous agrégeons toutes les performances par valeur de certains paramètres pour les comparer à celles des participants. Les paramètres principaux de l'évaluation étant l'encodage de l'intrus (*i.e.*, Type) et l'hétérogénéité (*i.e.*, *#formes* et *#couleurs*), nous comparons les performances de *ResNet* et des participants pour toutes les combinaisons de *type-#formes-#couleurs* en agrégeant les données dont nous disposons. L'évaluation utilisateur ayant déjà été conçue sur un sous-ensemble de l'espace de paramètres, nous ne considérons que les combinaisons sélectionnées dans celle-ci. Ainsi, les performances de *ResNet* sont agrégées pour obtenir un score sur chacune des 44 combinaisons *type-#couleurs-#formes*, et seront comparées aux performances des participants sur la base de ces combinaisons.

V.6.2 Hypothèses sur les Systèmes de Traitement de l'Information Visuelle

Dans cette section, nous discutons et rappelons les stratégies de traitement de l'information visuelle probablement employées par *ResNet* et les participants humains pour résoudre la tâche. Cette connaissance nous informe sur les corrélations qui pourraient exister entre eux.

V.6.2.1 Stratégie des Participants : Perception et Recherche Visuelle

Les travaux de la littératures sur la Perception et de Recherche Visuelle proposent des théories primordiales pour comprendre le fonctionnement de l'attention humaine et la manière dont notre cerveau traite l'information, mêlant à la fois des processus visuels et de mémoire. Comme nous l'avons présenté dans la Section II.4.1, la stratégie employée dans le processus de recherche visuelle consiste à combiner une phase d'attention *bottom-up* (*e.g.*, attraction par les contrastes, couleur stressante pour l'œil) et d'attention *top-down* (basée sur le concept de la cible). D'après les réponses des participants au questionnaire (voir Section V.5.2.2), leur stratégie suivait bien ce processus. Leur première étape consistait en une phase d'attention *bottom-up* (*e.g.*, attraction par

les contrastes, couleur stressante pour l'œil). La deuxième étape est un traitement *top-down* : l'attention des sujets est dirigée sur des régions différentes et elle s'attarde sur les régions jugées intéressantes. Ce processus représente l'idée du *balayage du regard*. Enfin, si la tâche n'est toujours pas résolue, la dernière étape est un traitement *bottom-up* : des paires de stimuli sont séquentiellement comparées jusqu'à ce que l'intrus soit identifié.

V.6.2.2 Stratégie de *ResNet* : Réseau de Neurones Profond Convolutif

Le DNN utilisé dans nos travaux, *ResNet50* [138], est un réseau de neurones convolutif. Par conception, les CNN modélisent un traitement de l'information *bottom-up*. Les premières opérations de convolution opèrent sur les pixels de l'image, et chaque convolution abstrait les caractéristiques de la donnée d'un niveau, tout en injectant de la connaissance issue de l'apprentissage. Ainsi, les dernières opérations traitent des données dont le niveau d'abstraction est relativement élevé, notamment lorsque le réseau est profond (ce qui est le cas de *ResNet50*). Bien que l'architecture de *ResNet*, basée sur des blocks résiduels de convolution, soit capable de traiter plusieurs niveaux d'abstraction des caractéristiques en parallèle, notre modèle suit bien ce processus *bottom-up*. Ainsi, les performances de *ResNet* et des participants sont sans doute mieux corrélées sur les cas complexes, car dans ce cas leur stratégie respective est basée sur un traitement *bottom-up* de l'information.

Les convolutions sont des opérations conçues pour détecter les bordures dans un espace, ici l'espace de couleurs de l'image. comme nous l'avons observé dans la Section V.4.3, *ResNet* est meilleur lorsque l'intrus est identifiable par sa forme, à un point qui dépasse probablement les capacités des humains. Nous nous attendons donc à voir une faible corrélation entre les performances de *ResNet* et celles des participants sur les objets expérimentaux de type *shape*.

Pour les autres cas, bien que leurs stratégies soient différentes, nous pensons que les résultats qu'ils obtiennent peuvent être affectés de la même manière par certaines propriétés de la visualisation.

V.6.3 Analyse de Corrélations

La première approche pour étudier les corrélations entre *ResNet* et les performances des participants est le calcul de coefficients de corrélation standards, tels que celui de Pearson [189] qui capture les corrélations linéaires. Les scores de ce coefficient sont entre -1 et 1 où 1 est une corrélation forte, -1 une anti-corrélation forte, et 0 une absence de corrélation.

Puisque le coefficient de corrélation de Pearson (communément noté R) est plus fiable sur des données suivant une distribution normale [254], nous calculons les statistiques de Kurtosis et Skewness [255] pour quantifier la similarité entre la distribution de nos données et une loi normale. Lorsque ces deux statistiques sont comprises entre -2 et 2, nous pouvons considérer que la distribution des données suit une loi normale [184]. Dans cette section, tous les tests conduits sont passés, ce qui signifie que les corrélations de Pearson et leurs *p-values* correspondantes sont fiables.

Comme nous l'avons réalisé dans les précédentes évaluations (voir Sections V.4.3 et V.5.2.1), nous réalisons notre étude dans le cas *Général* et par valeur de *Type* car nous nous attendons à ce que les corrélations entre *ResNet* et les participants changent en fonction de la valeur de type d'intrus. Les seuils d'acceptation des tests sont $\alpha = 0.05$ dans le cas général et $\alpha = 0.025$ par type.

La Table V.2 présente les coefficients de corrélation dans le cas Général. Les métriques de

TABLE V.2 : Coefficients de corrélation de Pearson [189] entre les performances de *ResNet* et des participants à l'expérience. Les performances des participants sont mesurées en taux d'erreur moyen (ER) et son écart type (ER STD) ; et en temps de réponse moyen (RT) et son écart type (RT STD). Tous les tests de corrélation sont statistiquement significatifs ($p\text{-}value \ll 0.001$). Les données suivent une distribution normale selon les statistiques de Kurtosis et Skewness [255], rendant le coefficient de Pearson d'autant plus fiable. Les métriques de *ResNet* sont triées du plus haut coefficient de corrélation moyen au plus faible.

		Performances des participants			
		ER	**ER STD**	**RT**	**RT STD**
Métriques de *ResNet*	**Confiance**	-0.851	-0.77	-0.91	-0.7
	Aire sous la courbe	-0.846	-0.756	-0.916	-0.676
	Entropie catégorielle croisée	0.827	0.761	0.912	0.69
	Taux d'erreur	0.806	0.759	0.903	0.699
	Rappel	-0.806	-0.759	-0.902	-0.699
	Précision	-0.806	-0.759	-0.902	-0.699
	Marge catégorielle	0.785	0.753	0.894	0.7
	Score F1	-0.788	-0.75	-0.893	-0.687

ResNet sont ordonnées de la plus à la moins corrélée, en moyenne, avec les quatre mesures de performances des participants. Ici, nous ne nous intéressons qu'aux valeurs absolues des coefficients. Le fait qu'une valeur soit positive ou négative dépend uniquement de l'orientation des métriques (maximisation ou minimisation). Nous considérons donc *forts* les coefficients proches de -1 et 1 et *faibles* ceux proches de 0. Comme nous pouvons l'observer, les coefficients révèlent que des corrélations sont fortes entre *ResNet* et les participants sur toutes les paires de métriques, le coefficient le plus faible étant $|-0.676|$. D'une manière générale, les performances de *ResNet* mesurées avec les diverses métriques semblent être mieux représentatives des temps de réponse des participants que de leurs taux d'erreur, les coefficients étant toujours plus forts que 0.89. À titre de comparaison, la corrélation entre le taux d'erreur des participants et leur propre temps de réponse est de $0.912; p\text{-}value \ll 0.001$. Ce coefficient peut servir de référence, car il semble en effet logique que la difficulté de la tâche affecte conjointement le temps de réponse et le taux d'erreur des participants. Cela signifie que les forces des corrélations entre *ResNet* et le temps de réponse des participants est du même ordre de grandeur que la corrélation entre le ER et le RT des participants, qui nous sert de référence. La moyenne des valeurs absolues des corrélations avec le ER des participants étant 0.814, nous considérons que les performances de *ResNet* sont fortement corrélées avec le taux d'erreurs des participants. Enfin, les corrélations avec les écarts types des mesures de performance des participants sont plus faibles bien qu'elles restent acceptables ($coef. > |-0.676|$).

La métrique de *ResNet* la mieux corrélée avec les performances des participants est Confiance (voir Table V.2). Nous utilisons donc celle-ci pour étudier les corrélations avec les participants par valeur de Type, mais notons que toutes les métriques de *ResNet* suivent globalement la même tendance. Les coefficients de ces corrélations sont reportés dans la Table V.3. Sur les objets expérimentaux de type *color*, la Confiance de *ResNet* est fortement corrélée avec toutes les mesures de performance des participants ($R \leq -0.798$). Contrairement au cas Général, les corrélations entre Confiance et ER STD ne sont pas plus faibles qu'entre Confiance et ER. Nous n'observons qu'une faible diminution du coefficient entre la corrélation avec RT STD et celle avec RT. Sur le type *shape*, la Confiance de *ResNet* est fortement corrélée avec ER et ER

TABLE V.3 : Coefficients de corrélation de Pearson [189] entre la métrique Confiance de *ResNet* et des performances des participants *par valeur de Type*. Les coefficients en italique ne sont pas significatifs (*i.e.*, leur *p-value* correspondante est supérieure au seuil d'acceptation). Il n'est pas possible de calculer de corrélation avec ER et ER STD sur les objets expérimentaux de type *redundant* car les participants n'ont jamais commis d'erreur (*i.e.*, leur séquence d'erreurs est constante).

		Type			
	Général	***color***	***shape***	***redundant***	***conjunction***
ER	-0.851	-0.798	-0.86	–	-0.89
ER STD	-0.77	-0.806	-0.881	–	-0.847
RT	-0.91	-0.861	*-0.487*	-0.762	-0.913
RT STD	-0.7	-0.813	*-0.414*	-0.821	-0.722

STD. Cependant, la corrélation n'est pas significative avec les mesures de RT, et les coefficients sont faibles. Nous en concluons que les performances de *ResNet* ne sont pas corrélées avec les temps de réponse des participants. Sur les objets expérimentaux de type *redundant*, il n'est pas possible de calculer de corrélation avec ER et ER STD car les participants n'ont jamais commis d'erreur dans cette condition. Néanmoins, nous pouvons observer que la Confiance de *ResNet* est fortement corrélée avec RT et RT STD. Il est intéressant de noter que la corrélation avec RT STD est plus forte qu'avec RT, ce qui est l'opposé de la tendance sur le cas Général. Enfin, sur le type *conjunction*, tous les coefficients de corrélation sont forts, plus forts que dans le cas Général. Là encore, les corrélations avec les écarts types des performances des participants sont plus faibles qu'avec leurs performances moyennes.

Nous avons observé que les performances de *ResNet* et des participants sont fortement corrélées. Ces résultats sont largement encourageants et justifient l'utilisation de *ResNet* pour réaliser l'évaluation automatique dans la Section V.4. Cependant, nous venons de voir que la métrique de *ResNet* la mieux corrélée aux performances des participants pour quantifier la difficulté de la tâche est la *Confiance*. Or, nous avons utilisé le *Taux d'erreur* dans notre évaluation automatique. Néanmoins, la différence est négligeable et les métriques de *ResNet* suivent la même tendance, ce qui préserve la fiabilité des résultats de l'évaluation automatique. Toutefois, ces coefficients de corrélation ne nous disent pas *quelle est* la relation entre les performances de *ResNet* et celles des participants. Nous pensons qu'il est plus probable que la complexité des relations entre les deux entités (DNN et humains) ne puisse pas être convenablement capturée par un coefficient de corrélation cherchant une relation *un à un*. C'est pour cette raison que nous étudions les relations de type *un à plusieurs* qui permettraient une lecture plus fine des performances du DNN pour estimer la difficulté de la tâche pour un humain.

V.6.4 Analyse de Relations par Régression

La régression est une manière commune de trouver des relations de type *plusieurs à plusieurs* ou *un à plusieurs*. Notre hypothèse est que s'il existe une fonction telle que $f : X \rightarrow Y$ où X sont les performances de *ResNet* et Y celles des humains, alors il existe une relation entre eux. Ici, nous cherchons des *relations* (et non pas des *corrélations*) car trouver cette relation ne signifie pas nécessairement qu'elle admet une réciproque. Néanmoins, l'existence de cette seule relation est suffisante car, dans la majorité des cas, les méthodes d'apprentissage automatique sont utilisées

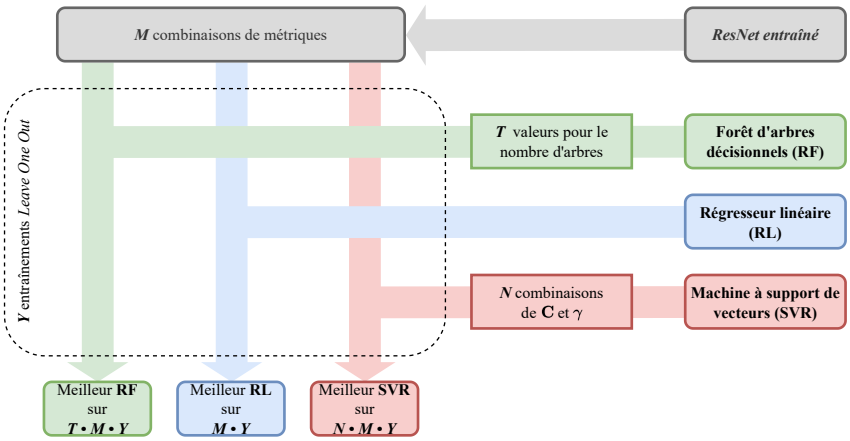


FIGURE V.12 : Organisation des apprentissages pour la sélection des meilleurs régresseurs. Les trois modèles sélectionnés sont entraînés à partir de toutes les combinaisons possibles de métriques de *ResNet*. De plus, certains hyper-paramètres sont recherchés en répétant les entraînements pour la Forêt d'arbres décisionnels et la SVR.

pour estimer les capacités de l'humain [15, 192] et non l'inverse.

Compte tenu des volumes de données que nous avons à disposition, l'utilisation de réseaux de neurones profonds ne semble pas appropriée. Nous employons donc des méthodes d'apprentissage automatique adaptées à des volumes de données plus faibles. Concrètement, nous utilisons trois modèles de régression : Régresseur Linéaire (RL), Forêt d'Arbres Décisionnels (RF pour « Random Forest ») [3], et Machine à Vecteurs de Support pour la Régression [81] (SVR pour « Support Vector Regression »). L'implémentation des trois modèles est tirée de la bibliothèque Scikit-Learn [256] en Python. Nous avons sélectionné d'une part le RL pour sa simplicité, et d'autre part la RF et la SVR car ce sont des modèles éprouvés de l'état de l'art. Nous utilisons une validation croisée « Leave One Out » [76] pour éviter certains biais dus au faible volume de données d'entraînement pour un modèle d'apprentissage automatique. Considérant qu'il y a N échantillons dans le jeu de données, une validation *Leave One Out* consiste à entraîner un modèle N fois avec $N-1$ échantillons. Dans chaque entraînement, l'échantillon retiré est différent.

Nous avons testé deux cadres d'entraînement : (i) entraîner un unique régresseur à prédire toutes les mesures de performance des participants, et (ii) entraîner un régresseur par mesure. Nous pensions que la deuxième approche donnerait de meilleurs résultats bien qu'elle soit plus coûteuse, puisque chaque modèle serait véritablement dédié à la mesure qu'il doit estimer. Cependant, la première méthode montre de meilleurs résultats, lesquels seront présentés dans la suite de cette étude. La Figure V.12 résume l'organisation des apprentissages pour la sélection du meilleur modèle de chaque régresseur. Comme montré dans la figure, nous affinons la RF et la SVR en cherchant exhaustivement les valeurs de certains hyper-paramètres qui mènent aux meilleurs résultats. Pour la RF, le nombre d'arbres est recherché entre 1 et 400. Pour la SVR, les deux hyper-paramètres C et γ varient comme proposé par Hsu *et al.* [257] : $C \in \{2^{-5}; 2^{-3}; \ldots; 2^{15}\}$ et $\gamma \in \{2^{-15}; 2^{-13}; \ldots; 2^{3}\}$. Aucun hyper-paramètre n'est modifié pour les modèles de Régresseur Linéaire. Tous les hyper-paramètres qui ne sont pas mentionnés sont laissés à leur valeur par défaut dans la bibliothèque Scikit-Learn [256]. Comme nous ne pouvons pas savoir quelles *combinaisons* de métriques de *ResNet* sont les mieux corrélées aux performances des participants, nous entraînons les modèles de régression avec toutes les $\sum_{k=1}^{8} \binom{8}{k} = 255$ combinaisons possibles des 8 métriques. Ces modèles sont entraînés à estimer les performances des participants en minimisant l'erreur absolue moyenne (MAE pour « Mean Absolute Error ») aux valeurs mesurées

TABLE V.4 : Performances des meilleurs modèles de régression sélectionnés. Pour chaque type de modèle, son score R^2, les valeurs des hyper-paramètres, et les métriques de *ResNet* utilisées en entrée sont reportées. Les dernières colonnes présentent les erreurs absolues moyennes (MAE) aux performances mesurées des participants de l'évaluation. Les taux d'erreur (ER) sont entre 0 et 1 tandis que les temps de réponse (RT) sont entre 0 et 30 (secondes). Le meilleur modèle (noté avec un symbole *) est la Forêt d'arbres décisionnels (RF).

				MAE entre estimations et mesures			
	R^2	**Hyper-paramètres**	**Métriques de *ResNet***	**ER**	**ER STD**	**RT**	**RT STD**
RL	0.715	–	Confiance, CCE, AUC, Rappel	0.06	0.085	1.865	1.516
RF*	0.956	9 arbres	Confiance, AUC, Rappel	0.044	0.059	1.418	1.166
SVR	0.753	$C = 2^9$; $\gamma = 2^3$	Confidence, CCE, AUC, Marge	0.059	0.121	2.586	1.962

pendant l'évaluation utilisateur (considérées comme des vérités terrain). Enfin, les modèles obtenus sont évalués sur la moyenne des MAE de tous leurs entraînements *Leave One Out*. Bien que cette organisation soit très coûteuse en calculs (nous entraînons des milliers de modèles), elle permet de se rapprocher au maximum d'un modèle idéal pour résoudre la tâche.

Les résultats de la meilleur instance de chaque modèle de régression sont présentés dans la Table V.4. Le coefficient de détermination R^2 est le carré du coefficient de Pearson [189] entre un ensemble d'estimations et un ensemble de vérités terrain. Ce score R^2 est communément utilisé pour estimer la capacité d'un modèle de régression à apprendre sa tâche. Comme nous pouvons l'observer, le meilleur modèle de régression que nous obtenons est la Forêt d'arbres décisionnels, avec un score quasi-parfait de $R^2 = 0.956$. Le modèle utilise les métrique de *ResNet* Confiance, AUC, et Rappel pour faire ses estimations. Il obtient de faibles scores de MAE avec les vérités terrain. En moyenne, le modèle se trompe de 4.4% sur son estimation des taux d'erreur et de 1.4s pour les temps de réponse. Au regard des performances des participants mesurées pendant l'évaluation, ces écarts aux vérités terrain sont suffisamment faibles pour considérer que le modèle trouve une relation entre les métriques de *ResNet* et les performances des participants.

Pour vérifier que cette approche par régression donne de meilleurs estimation des performances des humains que l'étude de corrélations présentée dans la Section V.6.3, nous calculons les coefficients de Pearson entre les performances des participants et l'estimation du meilleur modèle de régression (*i.e.*, la meilleure Forêt d'arbres décisionnels). Cela permet de comparer les résultats obtenus dans l'analyse de corrélations avec ceux obtenus dans la recherche par régression sur une métrique commune. Nous nous attendons à ce que ces corrélations soient plus fortes que celles présentées dans la Table V.3. Les résultats de cette étude sont présentés dans la Table V.5.

Dans le cas Général, nous observons que les estimations du modèle de régression sont très fortement corrélées aux performances des participants, et sont nettement supérieures aux coefficients observés dans la section précédente avec la Confiance de *ResNet* uniquement. Sur les objets expérimentaux de type *color*, les estimations sont presque parfaitement corrélées avec le ER ($R = 0.964$) et le RT ($R = 0.982$) des participants. Les corrélations avec les estimations des STD faites par le modèle d'apprentissage sont moins fortes, mais restent largement acceptable (0.892 et 0.925 pour ER STD et RT STD respectivement). Sur le type *shape*, les corrélations des estimations du régresseur avec les taux d'erreur (ER) et de leur écart type (ER STD) des participants sont du même ordre que celles observées dans l'analyse de corrélations (voir Table V.3). Aucune relation significative n'est non plus trouvée avec les temps de réponse (RT et RT STD) des participants. Là aussi, sur les objets expérimentaux de type *redundant*, nous ne pouvons pas calculer de corrélations avec les mesures de taux d'erreur des participants puisqu'ils n'ont jamais commis d'erreur dans cette condition. Néanmoins, nous avons constaté que les MAE du modèle

TABLE V.5 : Coefficients de corrélation de Pearson entre les estimations de performances humaines de la meilleure Forêt d'arbres décisionnels et les performances mesurées des participants à l'expérience. Les coefficients en italique ne sont pas significatifs.

		Type			
	Général	***color***	***shape***	***redundant***	***conjunction***
ER	0.914	0.964	0.885	–	0.983
ER STD	0.809	0.892	0.893	–	0.983
RT	0.934	0.982	*0.557*	0.841	0.988
RT STD	0.786	0.925	*0.312*	0.872	0.966

de régression pour ER et ER STD sur les objets de type *redundant* sont égales à 0, montrant que le modèle a bien appris que les participants ne se trompent pas dans ce cas. Les corrélations avec les temps de réponse sont plus fortes avec les estimations du régresseur que celles observées avec la Confiance de *ResNet*. Enfin, sur le type *conjunction*, les estimations de la Forêt d'arbres décisionnels sont très proches d'être parfaitement corrélées aux performances des participants ($R \geq 0.966$).

Au regard des performances (MAE et coefficients de corrélation) du meilleur modèle de régression obtenu, nous en concluons qu'il y a bien une relation entre les performances de *ResNet* et celles des participants dans le contexte de notre évaluation. Cette étude nous renseigne également sur les conditions dans lesquelles les corrélations sont plus ou moins fortes (en fonction du *Type* d'intrus). Par exemple, nous savons que les performances de *ResNet* sont très proches de celles des participants lorsque la tâche est difficile (*e.g.*, type *conjunction*). Ce résultat était attendu au vu de nos connaissances de leur système de traitement de l'information respectif (voir Section V.6.2). Nous avons également la confirmation qu'il n'y a pas de corrélation entre les performances de *ResNet* et des participants sur les objets expérimentaux de type *shape*. Grâce à ces résultats, nous pouvons mieux comprendre et interpréter les performances du réseau de neurones profond pour estimer la difficulté de la tâche pour des humains. Il aurait été préférable de connaître les résultats de cette étude de corrélations avant de faire l'interprétation des performances de *ResNet* dans l'évaluation automatique présentée dans la Section V.4, car nous aurions eu une vision plus fine des cas dans lesquels nous pouvions faire confiance au modèle pour estimer la difficulté de la tâche. Cela nous aurait également montré que le taux d'erreur de *ResNet* n'est pas la meilleure métrique pour estimer la difficulté de la tâche, bien que celle-ci est tout de même efficace. Cependant, faire ce constat n'est possible qu'après l'étude des corrélations qui nécessite de disposer des mesures de performances de participants humains. Ce point est problématique car il est à la fois nécessaire de disposer des mesures de performance humaines pour savoir comment interpréter celles d'un modèle d'apprentissage, et d'utiliser les performances du modèle d'apprentissage pour réduire l'espace de paramètres de l'évaluation collectant les performances humaines. Ce cycle de dépendance est une limitation de cette approche dont une proposition de solution sera proposée dans les pistes de travaux futurs.

V.7 Conclusion

Dans ce chapitre, nous avons présenté plusieurs travaux [30, 69] permettant d'étudier la capacité limite d'attention de la perception humaine sur une tâche de recherche visuelle d'intrus inconnu dans un contexte de visualisation d'information. La tâche consistait à identifier un stimulus

unique (*i.e.*, non répété) dans une grille de 8×8 stimuli organisés en grille régulière et définis par une couleur, forme et position dans la grille. L'objectif était d'étudier le degré d'hétérogénéité (mesurée en nombre de couleurs et de formes) auquel la tâche de recherche visuelle devenait trop compliquée.

En Visualisation d'Information, nous considérons des représentations relativement complexes. Dans notre cas, le produit cartésien de toutes les valeurs de l'espace de paramètres considéré aurait mené à des millions d'objets expérimentaux. Or, il n'est pas possible de traiter de tels volumes dans une évaluation utilisateur. Nous avons donc mis en œuvre une évaluation automatique dont la nouveauté est d'être basée sur un réseau de neurones profond pour estimer la difficulté de la tâche et réduire le nombre de cas à tester dans l'évaluation utilisateur. L'avantage d'utiliser ce type de modèle pour l'évaluation automatique est qu'elle s'adapte à n'importe quelle tâche tant que celle-ci peut être exprimée programmatiquement, c'est-à-dire qu'il existe une fonction dérivable capable de quantifier l'exactitude de la résolution de la tâche proposée. Le modèle d'apprentissage travaillant directement à partir de l'image de la représentation, il apprend lui-même à extraire l'information visuelle nécessaire à la résolution de la tâche.

Cette approche se base sur l'hypothèse que les performances d'un modèle d'apprentissage profond peuvent être corrélées avec celles d'humains sur ce type de tâche de recherche visuelle. Nous avons étudié cette hypothèse au cours d'une évaluation de la corrélation entre les performances du réseau de neurones profond avec celles des participants humains. Les résultats de cette étude ont montré qu'il était possible de finement identifier les cas dans lesquels le modèle d'apprentissage propose une bonne estimation de la difficulté de la tâche. Plus spécifiquement, nous avons trouvé que le modèle était fortement corrélé aux performances de nos participants dans tous les cas ($R \geq 0.841$) sauf lorsque l'intrus était encodé par sa forme.

L'évaluation utilisateur a permis de mesurer les performances de 21 participants sur la tâche de détection d'intrus, d'étudier des hypothèses et de conclure sur la capacité limite d'attention des attributs visuels *Forme* et *Couleur*. Il est communément admis que cette limite se trouve autour du nombre magique de Miller, 7 ± 2. Premièrement, notre étude a montré que cette limite d'hétérogénéité était fortement dépendante de la manière dont était encodée l'intrus. En effet, les performances de participants mesurées pendant notre évaluation varient selon le nombre de dimensions rendant l'intrus unique (*i.e.*, un seul attribut, encodage redondant ou conjonction de dimensions) et selon les dimensions elles-mêmes. Nous avons pu confirmer que l'encodage redondant était de loin le plus efficace pour encoder l'intrus. Encoder l'intrus avec une seule dimension demeure également plus efficace que l'encodage avec une conjonction de dimensions. Nous avons également observé que la difficulté de la tâche était moins sensible à l'hétérogénéité des *couleurs* qu'à celle des *formes*. En effet, la capacité limite d'attention sur l'attribut *Couleur* est plus haute que celle de *Forme* dans les différents types d'intrus considérés. Ces limites sont également plus basses que le nombre magique de Miller, sauf dans le cas d'un encodage redondant de l'intrus où la difficulté de la tâche ne semble pas affectée par l'hétérogénéité de la représentation. Enfin, nous avons vérifié que l'hétérogénéité dans une dimension qui ne servait pas à encoder l'intrus (*e.g.*, la couleur lorsque l'intrus est unique par sa forme) n'avait pas d'impact sur la difficulté de la tâche, même si l'intrus est inconnu.

Pour concevoir des visualisations efficaces, nous recommandons donc de porter une attention particulière au nombre de couleurs et formes utilisées dans les visualisations. Si l'encodage redondant de l'information est possible, il faut le privilégier. Sinon, il faut limiter autant que possible l'encodage par conjonction de formes et couleurs, bien que nous soyons conscients que le choix du type d'encodage est souvent guidé par la nature des données. Dans les cas où la conjonction de ces attributs est inévitable, les résultats de notre étude montrent qu'il faut utiliser moins de 4 couleurs, et minimiser autant que possible le nombre de formes. Ces recommandations

sont bâties sur les résultats de notre expérience et leur application doit évidemment tenir compte des différences qui peuvent exister entre le cadre que nous avons défini, et le contexte des données pour lesquelles une visualisation est conçue.

De nombreuses pistes d'extension ont émergé au cours de ces travaux. En ce qui concerne l'évaluation de visualisations, une première idée consisterait à évaluer la difficulté de recherche d'un *ensemble d'intrus* plutôt que d'un intrus unique dans une visualisation. Par exemple, trouver un groupement d'intrus dans un ensemble de données aléatoires. Ce problème est également commun dans le domaine de la Visualisation d'Information et fait intervenir d'autres processus de traitement de l'information tels que la ségrégation de textures [178, 258]. Une autre piste serait d'étendre notre étude à d'autres attributs visuels largement utilisés tels que la taille des stimuli ou leur position. Bien que la conjonction d'attributs visuels pour encoder l'information rende la tâche difficile, il n'est parfois pas possible de l'éviter pour encoder de l'information volumineuse et hétérogène (*e.g.*, des données multi-dimensionnelles). Il serait alors important de savoir quelles sont les combinaisons d'attributs visuels qui sont les moins difficiles à traiter par notre système perceptif. Cela nécessiterait la comparaison de différentes conjonctions (*e.g.*, forme-couleur, forme-taille, voire des triplets forme-couleur-taille).

En ce qui concerne l'évaluation automatique via des modèles d'apprentissages profonds, il serait intéressant d'observer comment s'adapte l'estimation de notre modèle à des conditions inconnues (*e.g.*, nouvelles couleurs/formes, ou plus de 7 couleurs ou 5 formes). Nous pourrions aussi utiliser des modèles pour lesquels des biais de la perception humaine sont injectés dans l'entraînement. Par exemple, De San Roman *et al.* [259] ont entraîné des réseaux de neurones profonds à partir d'images et de cartes de saillance capturées d'une évaluation utilisateur pour biaiser volontairement le modèle en lui apprenant à porter son attention sur les mêmes zones qu'un humain. Ce type d'approche pourrait renforcer les corrélations entre les comportements de ces modèles et ceux des humains, et donc renforcer la confiance avec laquelle nous les utilisons pour estimer les performances humaines. Une autre extension consisterait à définir une taxonomie de modèles d'apprentissages profonds pour l'évaluation automatique de visualisations. Dans notre évaluation, nous avons dû mettre en œuvre les évaluations automatiques et utilisateurs avant de pouvoir calculer les corrélations entre les deux, puisque cela nécessitait de collecter leurs performances respectives. Dans un cas idéal, nous devrions connaître ces corrélations avant même de conduire l'évaluation automatique afin de savoir quelle architecture de réseau de neurones utiliser et comment interpréter ses résultats efficacement. Puisqu'il n'est pas possible de calculer ces corrélations sans les données d'utilisateurs, nous pensons qu'il serait essentiel de conduire une étude sur une variété de modèles, de visualisations et de tâches de manière à pouvoir guider la conception des futures évaluations automatiques avec cette taxonomie.

Chapitre VI

Synthèse, Conclusion et Perspectives

Les travaux présentés dans cette thèse ont contribué à différentes recherches en Visualisation d'Information au travers de la thématique IA4VIS ; en proposant des méthodes de *génération*, *traitement* et *évaluation* automatique de visualisations avec des modèles d'apprentissage.

Basé sur un réseau de neurones profond non supervisé, le Chapitre III a présenté notre nouvel algorithme de dessin de graphes. La méthode consiste à utiliser des architectures de réseaux de neurones profonds éprouvées dans la littérature, et à les adapter aux graphes en modifiant leur opération principale. L'implémentation que nous avons proposée, *(DNN)*^{2}, se base sur un réseau de neurones convolutif où les convolutions standards ont été remplacées par des convolutions de graphes. À travers plusieurs expérimentations, nous avons étudié l'impact de certains hyper-paramètres et certaines structures de graphes sur l'apprentissage du modèle. Cette étude a permis de montrer que même sans optimisation particulière de la conception du modèle d'apprentissage, il était possible d'atteindre un niveau de performances du même ordre que celui des algorithmes de l'état de l'art. Ces travaux montrent les bonnes capacités des approches par réseaux de neurones profonds pour le dessin de graphes, et ouvrent plusieurs pistes de travaux futurs dans le domaine DL4GD (pour « Deep Learning for Graph Drawing »).

Organisé en deux parties, le Chapitre IV a présenté deux algorithmes de suppression de chevauchements dans des nuages de points. Le premier, FORBID, modélise la suppression de chevauchements comme un problème conjoint de recherche de facteur d'échelle et de mouvements de nœuds dans l'espace géométrique. L'objectif de cet algorithme est de proposer un plongement solution sans chevauchement, et avec un compromis entre la compacité et la préservation du dessin initial. Pour cela, le facteur d'échelle idéal est trouvé par recherche dichotomique qui éloigne ou rapproche uniformément tous les points du nuage. À chaque facteur d'échelle, l'algorithme calcule des mouvements optimisant une fonction de Stress pour supprimer les chevauchements tout en préservant le plongement initial. L'optimisation est réalisée par un algorithme mimant une descente de gradient stochastique pour un nombre fixe d'itérations équivalent à un *budget* de mouvements. Il converge alors vers le facteur d'échelle idéal permettant de supprimer tous les chevauchements avec ce *budget* borné de mouvements. L'évaluation de FORBID démontre ses capacités à proposer un compromis unique dans la qualité du plongement produit.

Usant des mêmes composants que FORBID, GIST est le second algorithme présenté dans le Chapitre IV. Son objectif est de traiter les chevauchements dans l'espace visuel. L'idée est qu'il n'est pas nécessaire de *strictement* supprimer les chevauchements dans le plongement initial, tant qu'il est possible de garantir la visibilité de tous les éléments. Pour cela, GIST prend en considération une résolution d'image cible et optimise la taille des points dans le nuage de telle sorte qu'ils aient tous au moins 1 pixel de diamètre, puis essaie de maximiser ce diamètre.

Pour atteindre une meilleure scalabilité, l'algorithme emploie également une *tolérance* aux chevauchements qui permet d'alléger les contraintes à satisfaire et facilite grandement l'optimisation. La comparaison de GIST avec les principales techniques de la littérature capables de traiter des nuages allant jusqu'à 150000 points démontre ses capacités à produire des visualisations de qualité où l'information est visible et les structures préservées.

Reposant aussi sur les réseaux de neurones profonds, le Chapitre V a présenté une méthodologie d'évaluation automatique de difficulté de tâche de perception. L'idée de cette méthodologie est d'entraîner un DNN à résoudre une tâche sur une visualisation en faisant varier les paramètres de celle-ci. L'utilisation d'un DNN permet de traiter des grands volumes de données, et donc d'explorer largement l'espace de paramètres de la visualisation considérée. En multipliant les exemples donnés au réseau, il est alors possible d'agréger ses performances pour chaque paramètre (ou combinaison de paramètres) et ainsi d'identifier les cas complexes de manière précise. Cette approche d'évaluation automatique est rendue possible par les capacités des DNN à apprendre par eux-mêmes les caractéristiques importantes de l'image en entrée pour résoudre la tâche. Dans le Chapitre V, nous mettons en pratique cette méthode pour évaluer automatiquement la difficulté d'une tâche de recherche d'intrus dans une grille régulière de stimuli encodés par une forme et une couleur. L'objectif est d'identifier le seuil d'hétérogénéité (*i.e.*, nombre de couleur et/ou formes) à partir duquel la tâche de recherche d'intrus devient trop difficile.

Quantifier automatiquement la difficulté de la tâche nous permet d'obtenir un critère objectif pour réduire l'espace de paramètres. Ainsi, nous avons également conduit une évaluation utilisateur de la difficulté de la tâche sur un ensemble de visualisations sous-échantillonné sur la base des résultats de l'évaluation automatique. Cette évaluation nous a permis d'étudier nos hypothèses basées sur la connaissance de la littérature de recherche visuelle d'intrus, raffinées d'après les résultats de l'évaluation automatique. Elle a permis de montrer que la difficulté de la tâche dépend grandement de la manière dont l'intrus est encodé dans la visualisation. Ses résultats montrent également que le seuil d'hétérogénéité auquel la tâche de recherche d'intrus devient trop difficile est plus bas que ce qui est communément supposé (*i.e.*, le nombre magique de Miller [241] 7 ± 2).

Utilisant les résultats des deux évaluations précédentes, la dernière partie du Chapitre V est consacrée à l'étude des corrélations des performances du DNN (provenant de évaluation automatique) et des participants (provenant de l'évaluation utilisateur) ; les deux ayant réalisé la même tâche sur le même type de visualisation. Cette étude de corrélation était nécessaire, car le processus d'évaluation automatique reposant sur DNN n'est fiable que si les performances de celui-ci peuvent être mises en relation avec celles de participants humains. L'étude a confirmé l'existence de cette corrélation, avec des coefficients allant de 0.676 à 0.916. Nous avons également entraîné des modèles d'apprentissage automatique à prédire les performances qu'obtiendrait un participant humain à partir de métriques issues des performances du DNN. Nous supposons que si tel est le cas, c'est qu'il existe bien une relation entre les performances des humains et du DNN. Cette étude a conclu qu'il était possible d'estimer très finement les performances des participants à partir de celles du DNN, les corrélations entre les performances estimées et les performances réelles allant de 0.841 à 0.988.

In fine, les travaux abordés au cours de cette thèse auront permis d'apporter des contributions concrètes à la thématique IA4VIS, et plus largement à la Visualisation d'Information, en proposant de nouvelles méthodes de génération et d'évaluation de visualisations. Toutefois, il n'a évidemment pas été possible de traiter toutes les questions de recherche soulevées au cours de nos travaux. Ainsi, plusieurs perspectives d'amélioration et d'extension de nos contributions

restent ouvertes.

Perspectives

Les résultats présentés dans cette thèse ouvrent de nombreuses perspectives, notamment dans les travaux sur la génération et l'évaluation de visualisations avec des techniques d'apprentissage.

Dessin de Graphe

Si nos travaux se sont concentrés sur l'adaptation de réseaux de neurones convolutifs pour le dessin de graphe, il semble intéressant de mettre en œuvre d'autres types d'architectures. Les réseaux de neurones auto-attentifs [6] semblent être une piste efficace pour traiter des graphes : une couche d'attention étant relativement analogue à une matrice d'adjacence. La stratégie d'apprentissage elle-même pourrait évoluer. En effet, nous n'entraînons qu'un seul réseau à la fois pour résoudre la tâche. SmartGD [142] propose par exemple l'utilisation d'une boucle d'entraînement reprenant le système des réseaux antagonistes génératifs (GAN pour « Generative Adversarial Networks ») pour entraîner le modèle à dessiner des graphes. Étant donné les résultats de notre étude, nous pensons qu'une approche par meta-modèle pourrait permettre d'atteindre de bonnes performances, à la manière de TopoLayout [104]. Plusieurs modèles *dessinateurs* pourraient être entraînés sur des familles de graphes différentes, de manière à atteindre un haut niveau de performance sur celles-ci. Pour chaque graphe à dessiner, un modèle *agrégateur* serait ensuite entraîné à choisir (voire agréger) les plongements produits par les *dessinateurs* afin de façonner le plongement final. Cette solution permettrait à la fois de tirer parti d'un apprentissage efficace sur des familles de graphes spécifiques tout en préservant la capacité du méta-réseau à produire un dessin de tout type de graphe.

D'autres perspectives telles que l'utilisation d'un réseau esthète [43], ou l'introduction volontaire de biais dans l'apprentissage du modèle lui permettant de produire des visualisations optimales pour la perception humaine, constituent également des pistes intéressantes.

Suppression de Chevauchements

Dans le Chapitre IV, nous avons présenté deux algorithmes de suppression de chevauchements dans des plongements 2D, qui traitent respectivement de l'espace géométrique et de l'espace visuel. Là où l'espace visuel est une étape intermédiaire entre l'espace géométrique et l'interprétation de l'information par l'humain, nous pourrions également positionner un *espace de perception* entre l'espace visuel et l'humain. En effet, s'assurer qu'un élément est *visible* ne garantit pas nécessairement qu'il soit *perceptible*. Considérer cet *espace de perception* permettrait de se rapprocher du but original des méthodes de visualisation : produire une image à partir de laquelle un utilisateur peut comprendre la structure des données hautement dimensionnelles.

La plupart des algorithmes de suppression de chevauchements mentionnés dans ce document emploient du mouvement (ou dispersion) de points pour ajuster la visualisation. Si cette pratique est courante, elle n'est toutefois pas adaptée à de nombreuses visualisations. Par exemple, lorsque les axes X et Y d'un graphique ont un sens concret et interprétable (*e.g.*, des années, des tailles), déplacer les points pour supprimer les chevauchements peut créer des inversions dans l'ordre orthogonal. Ces inversions ne rendent pas simplement la visualisation plus difficile à lire, mais la rendent également *fausse*. Un autre exemple est la carte symbolique où le déplacement de points (positionnés sur des villes ou des pays) n'est, par conception, pas envisageable. Une

autre perspective de nos recherches est donc de trouver des solutions pour améliorer la qualité de ces types de visualisation, sans déplacement de points. Les chevauchements ne peuvent alors pas être supprimés, mais nous pensons que la perception de l'information peut tout de même être optimisée. Une première perspective est l'étude de l'ordre d'affichage des points dans l'espace visuel, de manière à maximiser le nombre d'éléments visibles. Cet ordre d'affichage n'est aujourd'hui pas utilisé et est généralement géré arbitrairement par les bibliothèques de rendu qui (i) ne calculent pas d'ordre (*i.e.*, affichage dans l'ordre d'arrivée des données) ou (ii) affichent les données dans l'ordre décroissant de leur taille (*i.e.*, le plus grand à l'arrière, le plus petit à l'avant). Cette dernière stratégie n'est pas complètement satisfaisante car elle ne permet pas de garantir la visibilité des éléments, et elle n'est pas applicable lorsque tous les points ont la même taille.

Évaluation automatique de visualisations

Les travaux présentés dans cette thèse ont montré qu'il était possible de comparer et d'évaluer automatiquement des visualisations de manière fiable avec des réseaux de neurones profonds. Cependant, plusieurs pistes d'amélioration restent ouvertes autour de ces travaux. Il est nécessaire d'améliorer nos connaissances des modèles d'apprentissage pour comprendre dans quelle mesure il est possible de se fier à leurs performances pour estimer la qualité de visualisations. Cependant, cela nécessite de comparer les performances du réseau à celles de participants humains. Or, il n'est pas envisageable que chaque expert conçoive sa propre étude de corrélation. Nous pensons donc qu'il serait intéressant de conduire une expérience générale ayant vocation à comparer différentes architectures de réseaux de neurones profonds pour résoudre les tâches principales rencontrées dans le domaine de la Visualisation d'Information, et de mettre en lien leurs performances avec celles de participants humains. Une telle étude pourrait proposer des *recommandations* aux experts pour que ceux-ci puissent mettre en œuvre ces modèles évaluateurs sur leurs visualisations sans avoir à mesurer par avance le niveau de confiance avec lequel ils peuvent interpréter les résultats de ces évaluateurs.

Jusqu'à présent, les tâches considérées dans la littérature IA4VIS étaient relativement simples (*e.g.*, comptage, localisation). Il serait intéressant de mesurer la capacité des réseaux de neurones profonds à évaluer la qualité de visualisations sur des tâches de plus haut niveau. Il conviendrait alors de s'assurer des corrélations entre les comportements de ces modèles et celles de participants humains. En effet, les réseaux de neurones profonds sont *a priori* efficaces pour résoudre des tâches très simples un grand nombre de fois, là où la perception humaine est plus efficace à traiter une information de haut niveau, nécessitant la mise en relation de plusieurs concepts et du raisonnement. Néanmoins, la démocratisation récente des *grands modèles de langage* (LLM pour « Large Language Model ») rend envisageable la mise en œuvre de l'apprentissage profond pour résoudre des tâches nécessitant des prérequis, du contexte et de l'abstraction.

Bibliographie

[1] C. Ware, *Information Visualization : Perception for Design*, 3rd ed. Morgan Kaufmann, 2012.

[2] S. Dos Santos and K. Brodlie, "Gaining understanding of multivariate and multidimensional data through visualization," *Computers & Graphics*, vol. 28, no. 3, pp. 311–325, 2004.

[3] L. Breiman, "Random Forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," In *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012, pp. 1106–1114.

[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[7] C. C. Aggarwal, *Neural Networks and Deep Learning - A Textbook*. Springer, 2018.

[8] X. Wang, Z. Wu, W. Huang, Y. Wei, Z. Huang, M. Xu, and W. Chen, "VIS+AI : integrating visualization with artificial intelligence for efficient data analysis," *Frontiers of Computer Science*, vol. 17, no. 6, pp. 1–12, 2023.

[9] P. Villalobos, J. Sevilla, T. Besiroglu, L. Heim, A. Ho, and M. Hobbhahn, "Machine Learning Model Sizes and the Parameter Gap," *arXiv preprint arXiv :2207.02852*, 2022.

[10] L.-E. Pommé, R. Bourqui, R. Giot, J. Vallet, and D. Auber, "NetPrune : A sparklines visualization for network pruning," *Visual Informatics*, 2023.

[11] J. Yuan, C. Chen, W. Yang, M. Liu, J. Xia, and S. Liu, "A survey of visual analytics techniques for machine learning," *Computational Visual Media*, pp. 1–34, 2020.

[12] V. Buhrmester, D. Münch, and M. Arens, "Analysis of explainers of black box deep neural networks for computer vision : A survey," *Machine Learning and Knowledge Extraction*, vol. 3, no. 4, pp. 966–989, 2021.

[13] B. L. Rosa, G. Blasilli, R. Bourqui, D. Auber, G. Santucci, R. Capobianco, E. Bertini, R. Giot, and M. Angelini, "State of the Art of Visual Analytics for eXplainable Deep Learning," *Computer Graphics Forum*, vol. 42, no. 1, pp. 319–355, 2023.

[14] J. Benois-Pineau, R. Bourqui, D. Petkovic, and G. Quenot, *Explainable Deep Learning AI : Methods and Challenges*. Elsevier, 2023.

[15] A. Wu, Y. Wang, X. Shu, D. Moritz, W. Cui, H. Zhang, D. Zhang, and H. Qu, "AI4VIS : Survey on Artificial Intelligence Approaches for Data Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 12, pp. 5049–5070, 2022.

[16] Q. Wang, Z. Chen, Y. Wang, and H. Qu, "A Survey on ML4VIS : Applying Machine Learning Advances to Data Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 12, pp. 5134–5153, 2022.

[17] J. Martin, "Premier accident mortel pour le pilote automatique de Tesla," Le Monde, www.lemonde.fr/pixels/article/2016/07/01/premier-accident-mortel-pour-le-pilote-automatique-de-tesla_6003284_4408996.html, 2016.

[18] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks : Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[19] A. Halnaut, R. Giot, R. Bourqui, and D. Auber, "Deep dive into deep neural networks with flows," In *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2020) : IVAPP*, vol. 3, 2020, pp. 231–239.

[20] L.-E. Pommé, R. Bourqui, and R. Giot, "H^2O : Heatmap by Hierarchical Occlusion," 2023.

[21] R. Garcia, A. C. Telea, B. C. da Silva, J. Tørresen, and J. L. D. Comba, "A task-and-technique centered survey on visual analytics for deep learning model engineering," *Computers & Graphics*, vol. 77, pp. 30–49, 2018.

[22] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM : Visual Explanations From Deep Networks via Gradient-Based Localization," In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 618–626.

[23] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You ?" : Explaining the Predictions of Any Classifier," In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2016, pp. 1135–1144.

[24] L. Bourroux, J. Benois-Pineau, R. Bourqui, and R. Giot, "Multi Layered Feature Explanation Method for Convolutional Neural Networks," In *International Conference on Pattern Recognition and Artificial Intelligence*. Springer, 2022, pp. 603–614.

[25] L. Giovannangeli, F. Lalanne, D. Auber, R. Giot, and R. Bourqui, "Deep Neural Network for DrawiNg Networks, $(DNN)^2$," In *Graph Drawing and Network Visualization*. Springer International Publishing, 2021, pp. 375–390.

[26] X. Wang, K. Yen, Y. Hu, and H.-W. Shen, "DeepGD : A Deep Learning Framework for Graph Drawing Using GNN," *IEEE Computer Graphics and Applications*, vol. 41, no. 5, pp. 32–44, 2021.

[27] L. Giovannangeli, F. Lalanne, R. Giot, and R. Bourqui, "FORBID : Fast Overlap Removal By stochastic gradIent Descent for Graph Drawing," In *International Symposium on Graph Drawing and Network Visualization*. Springer International Publishing, 2022, pp. 61–76.

[28] D. Haehn, J. Tompkin, and H. Pfister, "Evaluating 'graphical perception'with CNNs," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 1, pp. 641–650, 2018.

[29] H. Haleem, Y. Wang, A. Puri, S. Wadhwa, and H. Qu, "Evaluating the Readability of Force Directed Graph Layouts : A Deep Learning Approach," *IEEE Computer Graphics and Applications*, vol. 39, no. 4, pp. 40–53, 2019.

[30] L. Giovannangeli, R. Bourqui, R. Giot, and D. Auber, "Color and Shape efficiency for outlier detection from automated to user evaluation," *Visual Informatics*, vol. 6, no. 2, pp. 25–40, 2022.

[31] D. Deng, A. Wu, H. Qu, and Y. Wu, "DashBot : Insight-Driven Dashboard Generation Based on Deep Reinforcement Learning," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 1, pp. 690–700, 2023.

[32] S. Song, C. Li, Y. Sun, and C. Wang, "VividGraph : Learning to Extract and Redesign Network Graphs From Visualization Images," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 7, pp. 3169–3181, 2023.

[33] S. Song, C. Li, D. Li, J. Chen, and C. Wang, "GraphDecoder : Recovering Diverse Network Graphs from Visualization Images via Attention-Aware Learning," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–17, 2023.

[34] M. Espadoto, N. S. T. Hirata, and A. C. Telea, "Deep learning multidimensional projections," *Information Visualization*, vol. 19, no. 3, pp. 247–269, 2020.

[35] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[36] L. McInnes, J. Healy, and J. Melville, "Umap : Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv :1802.03426*, 2018.

[37] J. F. Kruiger, P. E. Rauber, R. M. Martins, A. Kerren, S. Kobourov, and A. C. Telea, "Graph Layouts by t-SNE," In *Computer Graphics Forum*, vol. 36. Wiley Online Library, 2017, pp. 283–294.

[38] O.-H. Kwon, T. Crnovrsanin, and K.-L. Ma, "What would a graph look like in this layout ? a machine learning approach to large graph visualization," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 478–488, 2017.

[39] J. X. Zheng, S. Pawar, and D. F. Goodman, "Graph drawing by stochastic gradient descent," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 9, pp. 2738–2748, 2018.

[40] R. Ahmed, F. D. Luca, S. Devkota, S. Kobourov, and M. Li, "Graph Drawing via Gradient Descent, $(GD)^2$," In *International Symposium on Graph Drawing and Network Visualization*. Springer, 2020, pp. 3–17.

[41] R. Ahmed, F. De Luca, S. Devkota, S. Kobourov, and M. Li, "Multicriteria Scalable Graph Drawing via Stochastic Gradient Descent, $(SGD)^2$," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 6, pp. 2388–2399, 2022.

[42] Y. Wang, Z. Jin, Q. Wang, W. Cui, T. Ma, and H. Qu, "DeepDrawing : A deep learning approach to graph drawing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 676–686, 2019.

[43] M. Tiezzi, G. Ciravegna, and M. Gori, "Graph Neural Networks for Graph Drawing," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[44] Y. Y. Leow, T. Laurent, and X. Bresson, "GraphTSNE : a visualization technique for graph-structured data," *arXiv preprint arXiv :1904.06915*, 2019.

[45] K. Yan, T. Zhao, and M. Yang, "GRAPHULY : GRAPH U-Nets-Based Multi-Level Graph LaYout," *IEICE TRANSACTIONS on Information and Systems*, vol. 105, no. 12, pp. 2135–2138, 2022.

[46] J. Choi, S. Jung, D. G. Park, J. Choo, and N. Elmqvist, "Visualizing for the Non-Visual : Enabling the Visually Impaired to Use Visualization," *Computer Graphics Forum*, vol. 38, no. 3, pp. 249–260, 2019.

[47] J. Fu, B. Zhu, W. Cui, S. Ge, Y. Wang, H. Zhang, H. Huang, Y. Tang, D. Zhang, and X. Ma, "Chartem : Reviving Chart Images with Data Embedding," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 337–346, 2021.

[48] P. Zhang, C. Li, and C. Wang, "VisCode : Embedding Information in Visualization Images using Encoder-Decoder Network," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 326–336, 2021.

[49] F. Chen, L. Piccinini, P. Poncelet, and A. Sallaberry, "Node Overlap Removal Algorithms : an Extended Comparative Study," *Journal of Graph Algorithms and Applications*, 2020.

[50] R. Nakano, "Neural painters : A learned differentiable constraint for generating brushstroke paintings," *arXiv preprint arXiv :1904.08410*, 2019.

[51] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv :1710.10196*, 2017.

[52] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[53] C. Ware, H. Purchase, L. Colpoys, and M. McGill, "Cognitive measurements of graph aesthetics," *Information visualization*, vol. 1, no. 2, pp. 103–110, 2002.

[54] M. Ghoniem, J.-D. Fekete, and P. Castagliola, "On the readability of graphs using node-link and matrix-based representations : a controlled experiment and statistical analysis," *Information Visualization*, vol. 4, no. 2, pp. 114–135, 2005.

[55] M. Okoe, R. Jianu, and S. G. Kobourov, "Node-link or Adjacency Matrices : Old Question, New Insights," *IEEE Transactions on Visualization and Computer Graphics*, 2018.

[56] H. C. Purchase, "Metrics for graph drawing aesthetics," *Journal of Visual Languages & Computing*, vol. 13, no. 5, pp. 501–516, 2002.

[57] H. Purchase, "Which aesthetic has the greatest effect on human understanding ?" In *International Symposium on Graph Drawing*. Springer, 1997, pp. 248–261.

[58] H. C. Purchase, R. F. Cohen, and M. James, "Validating graph drawing aesthetics," In *International Symposium on Graph Drawing*. Springer, 1995, pp. 435–446.

[59] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information processing letters*, vol. 31, no. 1, pp. 7–15, 1989.

[60] U. Brandes and C. Pich, "Eigensolver methods for progressive multidimensional scaling of large data," In *International Symposium on Graph Drawing*. Springer, 2006, pp. 42–53.

[61] D. Haehn, J. Tompkin, and H. Pfister, "Evaluating "Graphical Perception" with CNNs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 641–650, 2019.

[62] M. Handford, *Where's Wally ? the Wonder Book*. Walker Books, 1997.

[63] L. Giovannangeli, F. Lalanne, D. Auber, R. Giot, and R. Bourqui, "Toward Efficient Deep Learning for Graph Drawing (DL4GD)," *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[64] ——, "Source code of : Deep Neural Network for DrawiNg Networks, $(DNN)^2$," https://github.com/LoannGio/DNN2, 2023.

[65] ——, "Guaranteed Visibility in Scatterplots with Tolerance," *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[66] ——, "Source code of : FORBID : Fast Overlap Removal By stochastic gradIent Descent for Graph Drawing," https://github.com/LoannGio/FORBID, 2023.

[67] L. Giovannangeli, F. Lalanne, R. Giot, and R. Bourqui, "Online Demo of : Guaranteed Visibility in Scatterplots with Tolerance," https://labribkb.github.io/gist/, 2023.

[68] L. Giovannangeli, R. Bourqui, R. Giot, and D. Auber, "Toward automatic comparison of visualization techniques : Application to graph visualization," *Visual Informatics*, vol. 4, no. 2, pp. 86–98, 2020.

[69] L. Giovannangeli, R. Giot, D. Auber, J. Benois-Pineau, and R. Bourqui, "Analysis of Deep Neural Networks Correlations with Human Subjects on a Perception Task," In *25th International Conference Information Visualisation (IV)*, 2021, pp. 129–136.

[70] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2.

[71] J. L. Gross and T. W. Tucker, *Topological graph theory*. Courier Corporation, 2001.

[72] A. Lhuillier, C. Hurter, and A. Telea, "State of the art in edge and trail bundling techniques," In *Computer Graphics Forum*, vol. 36, no. 3. Wiley Online Library, 2017, pp. 619–645.

[73] O. Ersoy, C. Hurter, F. Paulovich, G. Cantareiro, and A. Telea, "Skeleton-based edge bundling for graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2364–2373, 2011.

[74] M. Wallinger, D. Archambault, D. Auber, M. Nöllenburg, and J. Peltonen, "Edge-path bundling : A less ambiguous edge bundling approach," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 313–323, 2021.

[75] B. Delaunay, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[76] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.

[77] D. P. Kingma and J. Ba, "Adam : A method for stochastic optimization," *arXiv preprint arXiv :1412.6980*, 2014.

[78] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, no. 7, pp. 2121–2159, 2011.

[79] G. Hinton, "Neural Networks for Machine Learning, Lecture 6c," Coursera online course, 2012.

[80] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[81] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2007.

[82] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[83] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[84] S. Leijnen and F. v. Veen, "The Neural Network Zoo," *Proceedings*, vol. 47, no. 1, p. 9, 2020.

[85] R. Bourqui, F. Gilbert, P. Simonetto, F. Zaidi, U. Sharan, and F. Jourdan, "Detecting structural changes and command hierarchies in dynamic social networks," In *2009 International conference on advances in social network analysis and mining*. IEEE, 2009, pp. 83–88.

[86] F. Gilbert, P. Simonetto, F. Zaidi, F. Jourdan, and R. Bourqui, "Communities and hierarchical structures in dynamic social networks : analysis and visualization," *Social Network Analysis and Mining*, vol. 1, no. 2, pp. 83–95, 2011.

[87] A. Lambert, J. Dubois, and R. Bourqui, "Pathway preserving representation of metabolic networks," *Computer Graphics Forum*, vol. 30, no. 3, pp. 1021–1030, 2011.

[88] T. Baumgartl, M. Petzold, M. Wunderlich, M. Hohn, D. Archambault, M. Lieser, A. Dalpke, S. Scheithauer, M. Marschollek, V. Eichel *et al.*, "In search of patient zero : Visual analytics of pathogen transmission pathways in hospitals," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 711–721, 2020.

[89] C. Rozenblat, G. Melançon, R. Bourqui, and D. Auber, "Comparing multilevel clustering methods on weighted graphs : The case of worldwide air passenger traffic 2000–2004," In *Methods for Multilevel Analysis and Visualisation of Geographical Networks*. Springer, 2013, pp. 141–154.

[90] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics : theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[91] J. J. Miller, "Graph Database Applications and Concepts with Neo4j," In *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, vol. 2324, 2013, pp. 141–147.

[92] M. Reif, B. Pinaud, T. Souvignet, G. Melançon, and Q. Rossy, "Visualizing Mobile Phone Communication Data in Criminal Investigations : the Case of Media Multiplexity," In *French Regional Conference on Complex Systems FRCCS 2023*, 2023.

[93] G. Brière, É. Darbo, P. Thébault, and R. Uricaru, "Consensus clustering applied to multi-omics disease subtyping," *BMC bioinformatics*, vol. 22, pp. 1–29, 2021.

[94] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2009.

[95] M. Hlawatsch, M. Burch, and D. Weiskopf, "Visual Adjacency Lists for Dynamic Graphs," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 11, pp. 1590–1603, 2014.

[96] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J.-D. Fekete, "ZAME : Interactive Large-Scale Graph Visualization," In *2008 IEEE Pacific Visualization Symposium*, 2008, pp. 215–222.

[97] S. Zhang, R. Xu, and Y. Quan, "Large graph layout optimization based on vision and computational efficiency : a survey," *Visual Intelligence*, vol. 1, no. 1, p. 14, 2023.

[98] S. Di Bartolomeo, T. Crnovrsanin, D. Saffo, and C. Dunne, "Designing Computational Evaluations for Graph Layout Algorithms : the State of the Art," 2023, OSF Preprints.

[99] W. T. Tutte, "How to draw a graph," *Proceedings of the London Mathematical Society*, vol. 3, no. 1, pp. 743–767, 1963.

[100] C. Gutwenger and P. Mutzel, "Planar Polyline Drawings with Good Angular Resolution," In *International Symposium on Graph Drawing*. Springer, 1998, pp. 167–182.

[101] S. Grivet, D. Auber, J.-P. Domenger, and G. Melançon, "Bubble tree drawing algorithm," In *Computer Vision and Graphics : International Conference, ICCVG 2004, Warsaw, Poland, September 2004, Proceedings*. Springer, 2006, pp. 633–641.

[102] E. M. Reingold and J. S. Tilford, "Tidier Drawings of Trees," *IEEE Transactions on software Engineering*, vol. SE-7, no. 2, pp. 223–228, 1981.

[103] T. Jankun-Kelly and K.-L. Ma, "MoireGraphs : Radial focus context visualization and interaction for graphs with visual nodes," In *IEEE Symposium on Information Visualization 2003 (IEEE Cat. No. 03TH8714)*. IEEE, 2003, pp. 59–66.

[104] D. Archambault, T. Munzner, and D. Auber, "Topolayout : Multilevel graph layout by topological features," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 2, pp. 305–317, 2007.

[105] R. Davidson and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing," *ACM Transactions on Graphics*, vol. 15, no. 4, pp. 301–331, 1996.

[106] A. Frick, A. Ludwig, and H. Mehldau, "A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration)," In *International Symposium on Graph Drawing*. Springer, 1994, pp. 388–403.

[107] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software : Practice and experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[108] A. Meidiana, S.-H. Hong, S. Cai, M. Torkel, and P. Eades, "Sublinear-Time Attraction Force Computation for Large Complex Graph Drawing," In *2021 IEEE 14th Pacific Visualization Symposium (PacificVis)*. IEEE, 2021, pp. 146–150.

[109] A. Meidiana, S.-H. Hong, M. Torkel, S. Cai, and P. Eades, "Sublinear Time Force Computation for Big Complex Network Visualization," *Computer Graphics Forum*, vol. 39, no. 3, pp. 579–591, 2020.

[110] S. Hachul and M. Jünger, "Drawing Large Graphs with a Potential-Field-Based Multilevel Algorithm," In *International Symposium on Graph Drawing 2004*. Springer, 2005, pp. 285–295.

[111] M. Klimenta and U. Brandes, "Graph drawing by classical multidimensional scaling : new perspectives," In *International Symposium on Graph Drawing*. Springer, 2012, pp. 55–66.

[112] D. Harel and Y. Koren, "Graph drawing by high-dimensional embedding," In *International symposium on graph drawing*. Springer, 2002, pp. 207–219.

[113] S. C. North, "Drawing graphs with NEATO," *NEATO User manual*, vol. 11, no. 1, 2004.

[114] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[115] T. Dwyer, "Scalable, Versatile and Simple Constrained Graph Layout," *Computer Graphics Forum*, vol. 28, no. 3, pp. 991–998, 2009.

[116] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk : Online learning of social representations," In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[117] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," In *1st International Conference on Learning Representations, ICLR 2013, Workshop Track Proceedings*, 2013.

[118] A. Grover and J. Leskovec, "node2vec : Scalable feature learning for networks," In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[119] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line : Large-scale information network embedding," In *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[120] J. Firth, "A synopsis of linguistic theory, 1930-1955," *Studies in linguistic analysis*, pp. 10–32, 1957.

[121] M. S. Granovetter, "The strength of weak ties," *American journal of sociology*, vol. 78, no. 6, pp. 1360–1380, 1973.

[122] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[123] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv :1609.02907*, 2016.

[124] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," In *International Conference on Machine Learning*. PMLR, 2019, pp. 7134–7143.

[125] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks ?" In *International Conference on Learning Representations*, 2019.

[126] J. Chen, T. Ma, and C. Xiao, "FastGCN : Fast Learning with Graph Convolutional Networks via Importance Sampling," In *International Conference on Learning Representations*, 2018.

[127] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[128] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," In *International Conference on Learning Representations*, 2018.

[129] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural : Higher-order graph neural networks," In *Proceedings of the AAAI conference on artificial intelligence*, 2019.

[130] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and Equivariant Graph Networks," In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[131] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal Neighbourhood Aggregation for Graph Nets," In *Advances in Neural Information Processing Systems 33 : Annual Conference on Neural Information Processing Systems 2020*, 2020.

[132] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," In *International conference on machine learning*. PMLR, 2017, pp. 1263–1272.

[133] Y. Xie, C. Yao, M. Gong, C. Chen, and A. K. Qin, "Graph convolutional networks with multi-level coarsening for graph classification," *Knowledge-Based Systems*, vol. 194, p. 105578, 2020.

[134] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee, "N-gcn : Multi-scale graph convolution for semi-supervised node classification," In *Uncertainty in artificial intelligence*. PMLR, 2020, pp. 841–851.

[135] J. Kim, T. Kim, S. Kim, and C. D. Yoo, "Edge-labeling graph neural network for few-shot learning," In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11–20.

[136] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[137] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv :1409.1556*, 2014.

[138] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[139] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[140] O. Ronneberger, P. Fischer, and T. Brox, "U-Net : Convolutional Networks for Biomedical Image Segmentation," In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, 2015.

[141] H. Gao and S. Ji, "Graph U-Nets," In *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 2083–2092.

[142] X. Wang, K. Yen, Y. Hu, and H.-W. Shen, "SmartGD : A GAN-Based Graph Drawing Framework for Diverse Aesthetic Goals," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–12, 2023.

[143] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," In *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014.

[144] E. Gansner and Y. Hu, "Efficient, proximity-preserving node overlap removal." *Journal of Graph Algorithms and Applications*, vol. 14, no. 1, pp. 53–74, 2010.

[145] R. Cutura, C. Morariu, Z. Cheng, Y. Wang, D. Weiskopf, and M. Sedlmair, "Hagrid—Gridify scatterplots with hilbert and gosper curves," In *Proceedings of the 14th International Symposium on Visual Information Communication and Interaction*, 2021, pp. 1–8.

[146] Z. Li, R. Shi, Y. Liu, S. Long, Z. Guo, S. Jia, and J. Zhang, "Dual Space Coupling Model Guided Overlap-Free Scatterplot," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 1, pp. 657–667, 2022.

[147] E. Bertini and G. Santucci, "By chance is not enough : preserving relative density through nonuniform sampling," In *Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004.* IEEE, 2004, pp. 622–629.

[148] X. Chen, J. Zhang, C.-W. Fu, J.-D. Fekete, and Y. Wang, "Pyramid-based Scatterplots Sampling for Progressive and Streaming Data Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 593–603, 2022.

[149] X. Chen, T. Ge, J. Zhang, B. Chen, C.-W. Fu, O. Deussen, and Y. Wang, "A recursive subdivision technique for sampling multi-class scatterplots," *IEEE transactions on visualization and computer graphics*, vol. 26, no. 1, pp. 729–738, 2019.

[150] L.-Y. Wei, "Multi-class blue noise sampling," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, pp. 1–8, 2010.

[151] T. Kamps, J. Kleinz, and J. Read, "Constraint-based spring-model algorithm for graph layout," In *International Symposium on Graph Drawing*. Springer, 1995, pp. 349–360.

[152] D. Harel and Y. Koren, "Drawing graphs with non-uniform vertices," In *Proceedings of the Working Conference on Advanced Visual Interfaces*, 2002, pp. 157–166.

[153] J. Yuan, S. Xiang, J. Xia, L. Yu, and S. Liu, "Evaluation of Sampling Methods for Scatterplots," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1720–1730, 2021.

[154] T. Dwyer, K. Marriott, and P. J. Stuckey, "Fast node overlap removal," In *International Symposium on Graph Drawing*. Springer, 2005, pp. 153–164.

[155] F. Chen, L. Piccinini, P. Poncelet, and A. Sallaberry, "Node overlap removal algorithms : A comparative study," In *International Symposium on Graph Drawing and Network Visualization*. Springer, 2019, pp. 179–192.

[156] K. Misue, P. Eades, W. Lai, and K. Sugiyama, "Layout adjustment and the mental map," *Journal of Visual Languages & Computing*, vol. 6, no. 2, pp. 183–210, 1995.

[157] K. Hayashi, M. Inoue, T. Masuzawa, and H. Fujiwara, "A layout adjustment problem for disjoint rectangles preserving orthogonal order," In *International Symposium on Graph Drawing*. Springer, 1998, pp. 183–197.

[158] X. Huang, W. Lai, A. Sajeev, and J. Gao, "A new algorithm for removing node overlapping in graph visualization," *Information Sciences*, vol. 177, no. 14, pp. 2821–2844, 2007.

[159] H. Strobelt, M. Spicker, A. Stoffel, D. Keim, and O. Deussen, "Rolled-out wordles : A heuristic method for overlap removal of 2d data representatives," In *Computer Graphics Forum*, vol. 31. Wiley Online Library, 2012, pp. 1135–1144.

[160] T. Dwyer, K. Marriott, and P. J. Stuckey, "Fast node overlap removal—correction," In *International Symposium on Graph Drawing*. Springer, 2006, pp. 446–447.

[161] W. Meulemans, “Efficient optimal overlap removal : Algorithms and experiments,” In *Computer Graphics Forum*, vol. 38. Wiley Online Library, 2019, pp. 713–723.

[162] L. Nachmanson, A. Nocaj, S. Bereg, L. Zhang, and A. Holroyd, “Node overlap removal by growing a tree,” In *International Symposium on Graph Drawing and Network Visualization*. Springer, 2016, pp. 33–43.

[163] G. M. Hilasaca, W. E. Marcílio-Jr, D. M. Eler, R. M. Martins, and F. V. Paulovich, “Overlap Removal of Dimensionality Reduction Scatterplot Layouts,” *arXiv preprint arXiv :1903.06262*, 2019.

[164] G. Strong and M. Gong, “Self-sorting map : An efficient algorithm for presenting multimedia data in structured layouts,” *IEEE Transactions on Multimedia*, vol. 16, no. 4, pp. 1045–1058, 2014.

[165] A. Halnaut, R. Giot, R. Bourqui, and D. Auber, “VRGrid : Efficient Transformation of 2D Data into Pixel Grid Layout,” In *26th International Conference Information Visualisation*, 2022.

[166] W. Wang, H. Wang, G. Dai, and H. Wang, “Visualization of Large Hierarchical Data by Circle Packing,” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’06. Association for Computing Machinery, 2006, p. 517–520.

[167] L. Micallef, G. Palmas, A. Oulasvirta, and T. Weinkauf, “Towards Perceptual Optimization of the Visual Design of Scatterplots,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 6, pp. 1588–1599, 2017.

[168] S. Hartwig, C. van Onzenoodt, P. Hermosilla, and T. Ropinski, “ClusterNet : A Perception-Based Clustering Model for Scattered Data,” *arXiv preprint arXiv :2304.14185*, 2023.

[169] A. Treisman and G. Gelade, “A feature-integration theory of attention,” *Cognitive Psychology*, vol. 12, no. 1, pp. 97–136, 1980.

[170] J. M. Wolfe and T. S. Horowitz, “Five factors that guide attention in visual search,” *Nature Human Behaviour*, vol. 1, no. 3, pp. 1–8, 2017.

[171] J. M. Wolfe, K. R. Cave, and S. L. Franzel, “Guided search : an alternative to the feature integration model for visual search.” *Journal of Experimental Psychology : Human perception and performance*, vol. 15, no. 3, p. 419, 1989.

[172] J. M. Wolfe and W. Gray, “Guided Search 4.0,” *Integrated models of cognitive systems*, pp. 99–119, 2007.

[173] J. M. Wolfe, “Guided Search 6.0 : An updated model of visual search,” *Psychonomic Bulletin & Review*, vol. 28, no. 4, pp. 1060–1092, 2021.

[174] ——, “Forty years after feature integration theory : An introduction to the special issue in honor of the contributions of Anne Treisman,” *Attention, Perception, & Psychophysics*, vol. 82, no. 1, pp. 1–6, 2020.

[175] ——, “Major issues in the study of visual search : Part 2 of “40 Years of Feature Integration : Special Issue in Memory of Anne Treisman”,” *Attention, Perception, & Psychophysics*, vol. 82, no. 2, pp. 383–393, 2020.

[176] J. Duncan and G. W. Humphreys, "Visual search and stimulus similarity," *Psychological Review*, vol. 96, no. 3, pp. 433–458, 1989.

[177] A. Treisman, "Focused attention in the perception and retrieval of multidimensional stimuli," *Perception & Psychophysics*, vol. 22, no. 1, pp. 1–11, 1977.

[178] H. Pashler, "Cross-dimensional interaction and texture segregation," *Perception & Psychophysics*, vol. 43, no. 4, pp. 307–318, 1988.

[179] P. T. Quinlan and G. W. Humphreys, "Visual search for targets defined by combinations of color, shape, and size : An examination of the task constraints on feature and conjunction searches," *Perception & Psychophysics*, vol. 41, no. 5, pp. 455–472, 1987.

[180] C. Healey and J. Enns, "Attention and Visual Memory in Visualization and Computer Graphics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 7, pp. 1170–1188, 2012.

[181] S. Haroz and D. Whitney, "How capacity limits of attention influence information visualization effectiveness," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2402–2410, 2012.

[182] C. Gramazio, K. Schloss, and D. Laidlaw, "The relation between visualization size, grouping, and user performance," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 1953–1962, 2014.

[183] Ç. Demiralp, M. S. Bernstein, and J. Heer, "Learning perceptual kernels for visualization design," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 1933–1942, 2014.

[184] H. Purchase, *Experimental Human-Computer Interaction : A Practical Guide with Visual Examples*. Cambridge University Press, 2012.

[185] M. Behrisch, M. Blumenschein, N. W. Kim, L. Shao, M. El-Assady, J. Fuchs, D. Seebacher, A. Diehl, U. Brandes, H. Pfister *et al.*, "Quality metrics for information visualization," In *Computer Graphics Forum*, vol. 37, no. 3. Wiley Online Library, 2018, pp. 625–662.

[186] J. Bethge, S. Hahn, and J. Döllner, "Improving Layout Quality by Mixing Treemap-Layouts Based on Data-Change Characteristics," In *Proceedings of the conference on vision, modeling and visualization*. Eurographics Association, 2017, pp. 69–76.

[187] W. S. Cleveland and R. McGill, "Graphical perception : Theory, experimentation, and application to the development of graphical methods," *Journal of the American statistical association*, vol. 79, no. 387, pp. 531–554, 1984.

[188] S. Cai, S.-H. Hong, X. Xia, T. Liu, and W. Huang, "A machine learning approach for predicting human shortest path task performance," *Visual Informatics*, vol. 6, no. 2, pp. 50–61, 2022.

[189] J. Lee Rodgers and W. A. Nicewander, "Thirteen ways to look at the correlation coefficient," *The American Statistician*, vol. 42, no. 1, pp. 59–66, 1988.

[190] R. Likert, "A technique for the measurement of attitudes." *Archives of psychology*, 1932.

[191] S. Shin, S. Hong, and N. Elmqvist, "Perceptual Pat : A Virtual Human Visual System for Iterative Visualization Design," In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2023.

[192] F. Yang, Y. Ma, L. Harrison, J. Tompkin, and D. H. Laidlaw, "How Can Deep Neural Networks Aid Visualization Perception Research ? Three Studies on Correlation Judgments in Scatterplots," In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2023.

[193] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[194] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[195] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[196] L. Tunstall, L. Von Werra, and T. Wolf, *Natural language processing with transformers*. " O'Reilly Media, Inc.", 2022.

[197] D. Auber, D. Archambault, R. Bourqui, M. Delest, J. Dubois, A. Lambert, P. Mary, M. Mathiaut, G. Melançon, B. Pinaud *et al.*, "TULIP 5," 2017.

[198] U. Brandes, M. Gaertler, and D. Wagner, "Experiments on graph clustering algorithms," In *European Symposium on Algorithms*. Springer, 2003, pp. 568–579.

[199] Y. Wang, Q. Shen, D. Archambault, Z. Zhou, M. Zhu, S. Yang, and H. Qu, "Ambiguityvis : Visualization of ambiguity in graph layouts," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 359–368, 2015.

[200] S. V. Dongen, "Graph clustering by flow simulation," Ph.D. dissertation, University of Utrecht, 2000.

[201] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[202] P. B. Nemenyi, *Distribution-free multiple comparisons.* Princeton University, 1963.

[203] S. van Wageningen and T. Mchedlidze, "Can an NN model plainly learn planar layouts ?" *arXiv preprint arXiv :2209.01075*, 2022.

[204] W. J. Conover, *Practical nonparametric statistics*. john wiley & sons, 1999, vol. 350.

[205] P. W. Angel, N. Rajab, Y. Deng, C. M. Pacheco, T. Chen, K. L. Cao, J. Choi, and C. A. Wells, "A simple, scalable approach to building a cross-platform transcriptome atlas," *PLoS Comput. Biol.*, vol. 16, no. 9, 2020.

[206] M. D. Luecken and F. J. Theis, “Current best practices in single-cell RNA-seq analysis : a tutorial,” *Molecular systems biology*, vol. 15, no. 6, p. e8746, 2019.

[207] J. De Leeuw, “Convergence of the majorization method for multidimensional scaling,” *Journal of classification*, vol. 5, no. 2, pp. 163–180, 1988.

[208] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau, “ActiVis : Visual exploration of industry-scale deep neural network models,” *IEEE transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 88–97, 2017.

[209] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, and H. Qu, “Understanding hidden memories of recurrent neural networks,” In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 2017, pp. 13–24. [Online]. Available : https://arxiv.org/pdf/1710.10777.pdf

[210] J. Wang, L. Gou, H. Yang, and H.-W. Shen, “Ganviz : A visual analytics approach to understand the adversarial game,” *IEEE transactions on visualization and computer graphics*, vol. 24, no. 6, pp. 1905–1917, 2018.

[211] M. Kahng, N. Thorat, D. H. P. Chau, F. B. Viégas, and M. Wattenberg, “Gan lab : Understanding complex deep generative models using interactive visual experimentation,” *IEEE transactions on visualization and computer graphics*, vol. 25, no. 1, pp. 1–11, 2018.

[212] H. Sanftmann and D. Weiskopf, “3D Scatterplot Navigation,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 11, pp. 1969–1978, 2012.

[213] K. Lynch, *The Image of the City*. MIT Press, 1960.

[214] E. C. Tolman, “Cognitive maps in rats and men.” *Psychological review*, vol. 55, no. 4, p. 189, 1948.

[215] U. Brandes and C. Pich, “An experimental study on distance-based graph drawing,” In *International Symposium on Graph Drawing*. Springer, 2008, pp. 218–229.

[216] M. Ortmann, M. Klimenta, and U. Brandes, “A sparse stress model,” In *International Symposium on Graph Drawing and Network Visualization*. Springer, 2016, pp. 18–32.

[217] M. Bostock, V. Ogievetsky, and J. Heer, “D^3 data-driven documents,” *IEEE transactions on visualization and computer graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.

[218] T. Jakobsen, “Advanced character physics,” In *Game developers conference*, vol. 3. IO Interactive, Copenhagen Denmark, 2001, pp. 383–401.

[219] E. R. Gansner and S. C. North, “An open graph visualization system and its applications to software engineering,” *Software : practice and experience*, vol. 30, no. 11, pp. 1203–1233, 2000.

[220] Y. Hu, “Efficient, high-quality force-directed graph drawing,” *Mathematica journal*, vol. 10, no. 1, pp. 37–71, 2005.

[221] L. Van Der Maaten, E. Postma, J. Van den Herik *et al.*, “Dimensionality reduction : a comparative,” *J Mach Learn Res*, vol. 10, no. 66-71, p. 13, 2009.

[222] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[223] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist : a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv :1708.07747*, 2017.

[224] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[225] L. Nachmanson, S. Pupyrev, T. Dwyer, and T. Hart, "Microsoft Automatic Graph Layout," https://www.microsoft.com/en-us/research/project/microsoft-automatic-graph-layout/, 2007.

[226] T. Gomez, T. Fréour, and H. Mouchère, "Metrics for Saliency Map Evaluation of Deep Learning Explanation Methods," In *Pattern Recognition and Artificial Intelligence*. Springer International Publishing, 2022, pp. 84–95.

[227] J. D. Hunter, "Matplotlib : A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[228] M. Shemanarev, "C++ Anti-Grained Geometry," https://github.com/ghaerr/agg-2.6/, 2006.

[229] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012.

[230] J. Bertin, *Semiology of Graphics : Diagrams, Networks, Maps*. University of Wisconsin Press, 1983.

[231] D. Huber and C. Healey, "Visualizing data with motion," In *VIS 05. IEEE Visualization, 2005.* IEEE, 2005, pp. 527–534.

[232] T. Itoh, Y. Yamaguchi, Y. Ikehata, and Y. Kajinaga, "Hierarchical data visualization using a fast rectangle-packing algorithm," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 3, pp. 302–313, 2004.

[233] J. Mackinlay, "Automating the Design of Graphical Presentations of Relational Information," *ACM Trans. Graph.*, vol. 5, no. 2, pp. 110—141, Apr. 1986.

[234] C. Ware and J. Beatty, "Using color dimensions to display data dimensions," *Human factors*, vol. 30, no. 2, pp. 127–142, 1988.

[235] C. Healey, "Choosing effective colours for data visualization," In *Proceedings of Seventh Annual IEEE Visualization'96*. IEEE, 1996, pp. 263–270.

[236] H. Chernoff, "The use of faces to represent points in k-dimensional space graphically," *Journal of the American statistical Association*, vol. 68, no. 342, pp. 361–368, 1973.

[237] F. Post, T. van Walsum, F. Post, and D. Silver, "Iconic techniques for feature visualization," In *Proceedings Visualization'95*. IEEE, 1995, pp. 288–295.

[238] M. Gleicher, M. Correll, C. Nothelfer, and S. Franconeri, "Perception of average value in multiclass scatterplots," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2316–2325, 2013.

[239] D. Altunbay, C. Cigir, C. Sokmensuer, and C. Gunduz-Demir, "Color graphs for automated cancer diagnosis and grading," *IEEE Transactions on Biomedical Engineering*, vol. 57, no. 3, pp. 665–674, 2009.

[240] H. Zhou, X. Yuan, H. Qu, W. Cui, and B. Chen, "Visual clustering in parallel coordinates," In *Computer Graphics Forum*, vol. 27, no. 3. Wiley Online Library, 2008, pp. 1047–1054.

[241] G. A. Miller, "The magical number seven, plus or minus two : Some limits on our capacity for processing information." *Psychological review*, vol. 63, no. 2, p. 81, 1956.

[242] B. W. Tatler, "The central fixation bias in scene viewing : Selecting an optimal viewing position independently of motor biases and image feature distributions," *Journal of vision*, vol. 7, no. 14, pp. 4–4, 2007.

[243] C. Nothelfer, M. Gleicher, and S. Franconeri, "Redundant encoding strengthens segmentation and grouping in visual displays of data." *Journal of Experimental Psychology : Human Perception and Performance*, vol. 43, no. 9, p. 1667, 2017.

[244] B. Bauer, P. Jolicoeur, and W. Cowan, "Visual search for colour targets that are or are not linearly separable from distractors," *Vision research*, vol. 36, no. 10, pp. 1439–1466, 1996.

[245] C. C. Gramazio, D. H. Laidlaw, and K. B. Schloss, "Colorgorical : Creating discriminable and preferable color palettes for information visualization," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 521–530, 2016.

[246] M. Harrower and C. A. Brewer, "ColorBrewer. org : an online tool for selecting colour schemes for maps," *The Cartographic Journal*, vol. 40, no. 1, pp. 27–37, 2003.

[247] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[248] P. Baldi, S. Brunak, Y. Chauvin, C. Andersen, and H. Nielsen, "Assessing the accuracy of prediction algorithms for classification : an overview," *Bioinformatics*, vol. 16, no. 5, pp. 412–424, 2000.

[249] W. Kruskal and W. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[250] G. H. Joblove and D. Greenberg, "Color Spaces for Computer Graphics," In *SIGGRAPH : Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*. Association for Computing Machinery, 1978, pp. 20–25.

[251] N. Camgöz, C. Yener, and D. Güvenç, "Effects of hue, saturation, and brightness on preference," *Color Research and Application*, vol. 27, no. 3, pp. 199–207, 2002.

[252] ——, "Effects of hue, saturation, and brightness : Part 2 : Attention," *Color Research & Application*, vol. 29, no. 1, pp. 20–28, 2004.

[253] D. Zwillinger and S. Kokoska, *CRC standard probability and statistics tables and formulae*. Crc Press, 1999.

[254] C. J. Kowalski, "On the effects of non-normality on the distribution of the sample product-moment correlation coefficient," *Journal of the Royal Statistical Society : Series C (Applied Statistics)*, vol. 21, no. 1, pp. 1–12, 1972.

[255] D. Zwillinger and S. Kokoska, *CRC standard probability and statistics tables and formulae*. Crc Press, 1999.

[256] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn : Machine learning in Python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[257] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[258] T. Callaghan, M. Lasaga, and W. Garner, "Visual texture segregation based on orientation and hue," *Perception & Psychophysics*, vol. 39, no. 1, pp. 32–38, 1986.

[259] P. P. De San Roman, J. Benois-Pineau, J.-P. Domenger, F. Paclet, D. Cataert, and A. De Rugy, "Saliency Driven Object recognition in egocentric videos with deep CNN : toward application in assistance to Neuroprostheses," *Computer Vision and Image Understanding*, vol. 164, pp. 82–91, 2017.