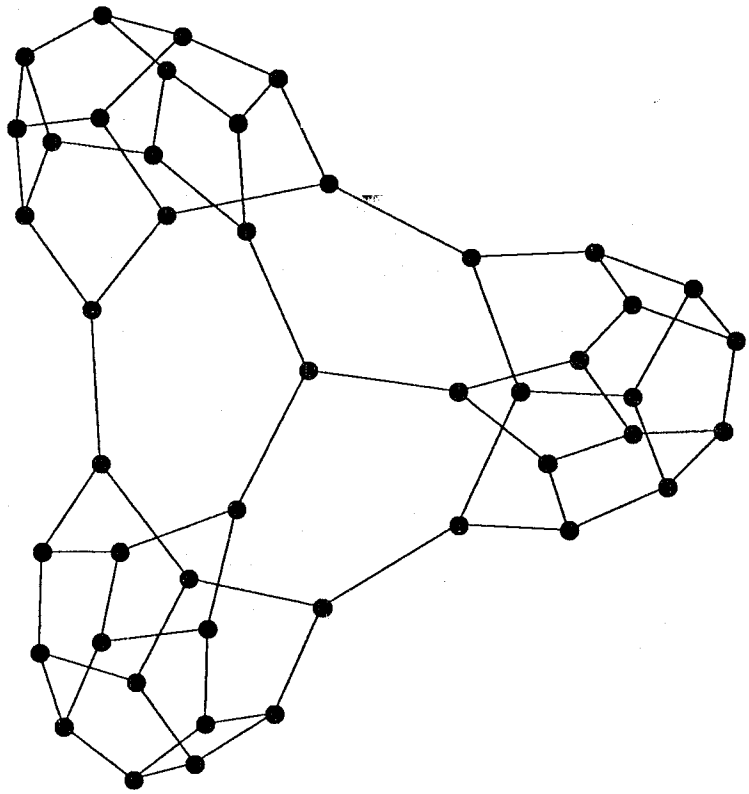
Cabri-graphes : un cahier de brouillon interactif pour la théorie des graphes

Olivier Baudon

Thèse

résentée par **Olivier Baudon**
our obtenir le titre de
octeur de l'Université Joseph Fourier - Grenoble I
rêté ministériel du 5 Juillet 1984
pécialité Mathématiques Appliquées



Cabri-graphes, un Cahier de Brouillon Interactif pour la Théorie des Graphes

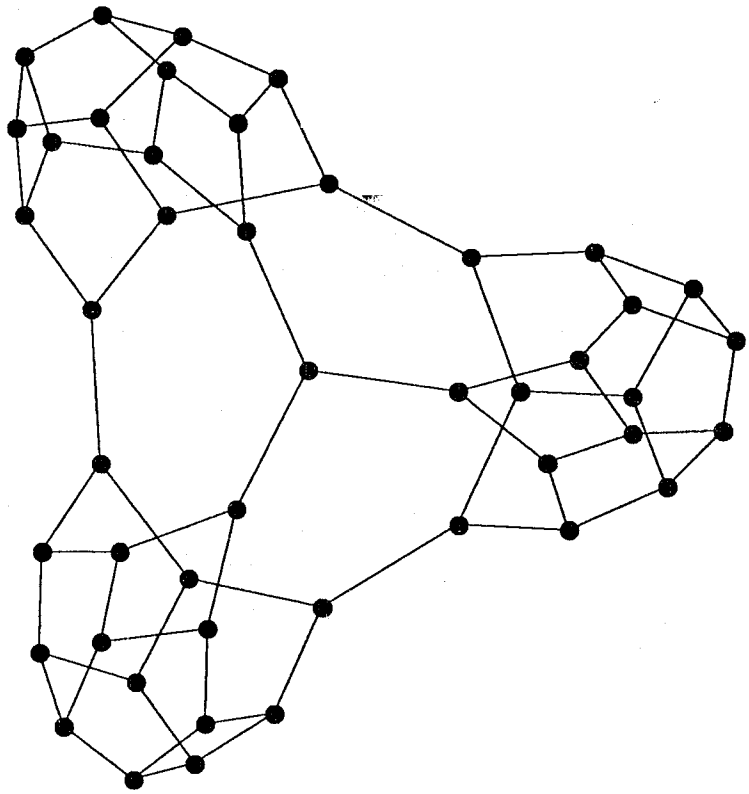
Thèse soutenue le 7 Février 1990

Composition du Jury :
Claude Benzaken (Président)
Michel Habib
Jean-Marie Laborde
Xavier Rousset
Dominique Sotteau
Nguyen Huy Xuong

Thèse préparée au Laboratoire de Structures Discrètes et de Didactique

Thèse

résentée par **Olivier Baudon**
our obtenir le titre de
octeur de l'Université Joseph Fourier - Grenoble I
rêté ministériel du 5 Juillet 1984
pécialité Mathématiques Appliquées



Cabri-graphes, un Cahier de Brouillon Interactif pour la Théorie des Graphes

Thèse soutenue le 7 Février 1990

Composition du Jury :
Claude Benzaken (Président)
Michel Habib
Jean-Marie Laborde
Xavier Rousset
Dominique Sotteau
Nguyen Huy Xuong

Thèse préparée au Laboratoire de Structures Discrètes et de Didactique

Cette thèse n'aurait pas existé sans les idées et le dévouement de Jean-Marie Laborde. Que sa générosité reçoive ici tous les remerciements qui lui sont dus, tant pour son travail de directeur de thèse que pour celui de directeur d'un laboratoire où il fait bon travailler.

Merci à Michel Habib et à Nguyen Huy Xuong d'avoir manifesté leur interêt pour mon travail en acceptant de rapporter sur ce document.

Je remercie également Claude Benzaken, Xavier Rousset et Dominique Sotteau pour la confiance accordée en participant au jury de cette thèse.

Que mes amis du Laboratoire de Structures Discrètes et de Didactique sachent ma gratitude pour la chaleur de leur accueil.

PLAN

TABLE DES FIGURES

INTRODUCTION

Cette thèse a pour objet l'étude et la réalisation d'un environnement logiciel destiné aux chercheurs, étudiants et enseignants en théorie des graphes. Nous avons appelé cet environnement Cabri-graphes, le mot Cabri étant l'acronyme de CAhier de Brouillon Interactif.

Le résultat présenté bénéficie d'un travail collectif où se sont impliqués des chercheurs du LSDD[1], des chercheurs étrangers lors de leur séjour à Grenoble[2], ainsi que des étudiants de l'Institut Grenoblois d'Études Informatiques, dans le cadre de projets que j'ai encadrés[3].

La réalisation de ce projet a été pendant toutes ces années ma principale occupation de recherche, ce qui m'a conduit à en aborder tous les aspects, parfois en collaboration avec d'autres chercheurs ou en reprenant un travail effectué par d'autres. Parmi les réalisations dont je suis plus particulièrement responsable, on peut citer le générateur de graphes aléatoires (Chapitre 2) et l'implémentation de l'algorithme de test de planarité basé sur les PQ-trees (Chapitre 3), ainsi que le calcul des invariants polynômiaux (Chapitre 1).

Le projet Cabri-graphes s'inscrit dans le projet national Cabri, dont il sera question au début du Chapitre 1, dirigé par Michel Habib et Jean-Marie Laborde [HL]. Il a inspiré au sein du LSDD le projet Cabri-géomètre [Baul] qui a en commun avec Cabri-graphes d'avoir bénéficié de la direction et du dynamisme

---

[1] Olivier Baudon, Claude Benzaken, Jérôme Bordier, Jean-Marie Laborde, Gérard Lejeune et Michel Mollard.
[2] Peter Eades et Ivan Havel.
[3] Hervé Frydlender et Annick Rimbod Pethiod, François Crocombette, Hassan Abdallah et Bernard Belorgey, Jean-Maurice Clochard.

de Jean-Marie Laborde. Cabri-géomètre reprend le concept de cahier de brouillon interactif en l'appliquant au domaine de la géométrie.

Cette thèse est à la jonction de deux disciplines : l'informatique et la combinatoire. C'est dire la variété des problèmes qui ont pu se poser pendant sa réalisation : théorie des graphes, algorithmique, génie logiciel ... Cette variété m'a conduit à développer certains passages afin de rendre la lecture accessible, quelque soit la discipline d'origine du lecteur.

Le chapitre 1 décrit notre travail sous ses différents aspects : objets manipulés, fonctionnalités, interface, implémentation. Une description d'autres éditeurs de graphes est également proposée, afin de montrer les points communs et les points de dissemblance qu'ils peuvent avoir avec notre travail.

Le chapitre 2 présente le générateur de graphes aléatoires, et donne une bibliographie commentée des travaux sur ce domaine publiés à ce jour.

Le chapitre 3 donne deux exemples de travaux motivés par le projet Cabri-graphes et dont la réalisation a été grandement facilitée par son existence. La première étude porte sur les tests de planarité d'un graphe. La seconde traite du rapport entre le diamètre d'un graphe et un arbre BFS donné de ce graphe.

Il est toujours difficile de décider du moment opportun pour rédiger un mémoire sur un tel sujet. L'existence de cette thèse ne signifie pas l'arrêt des recherches sur Cabri-graphes. Au contraire, je souhaite qu'elle marque un nouveau départ vers des objectifs plus ambitieux et qu'elle motive des coopérations toujours plus nombreuses et fructueuses.

CHAPITRE 1

LES ÉDITEURS DE GRAPHES

1.1 Pourquoi un éditeur de graphes ?

Les motivations avancées pour la réalisation d'un éditeur de graphes interactif sont généralement proches de celles exprimées dans le document de constitution du groupe Cabri dans le cadre du PRC "Mathématique et Informatique" [HL] :

« Dans les domaines de la théorie des graphes ou des ensembles ordonnés comme dans celles qui en font usage, il arrive très fréquemment que l'on ait recours à la réalisation de "petits dessins", formés essentiellement de sommets et d'arêtes; cela dans le but de les manipuler, de leur appliquer concrètement certaines transformations, ne serait-ce que pour vérifier telle ou telle propriété, conforter ou infirmer une idée, une conjecture.

Malheureusement cette pratique menée sur un brouillon ordinaire souffre de limitations, dûes par exemple à la taille relativement limitée des exemples que l'on peut traiter à la main ou encore au fait que la vérification d'une propriété peut être fort longue (comme c'est le cas, par exemple, lors de l'examen d'une "criticalité"); enfin la vérification, souvent, n'est au mieux que faiblement garantie. L'idée est donc naturelle de vouloir élargir les possibilités d'exploration par la mise à disposition des chercheurs du domaine (et d'étudiants) d'un outil graphique où le traditionnel cahier de brouillon serait remplacé par l'écran interactif d'un terminal associé à toute la puissance de calcul et de mémorisation de l'ordinateur.

C'est la réalisation d'un tel outil, sensé être d'accès aussi facile que possible pour être utilisé, pour partie, par des non informaticiens, qui constitue l'essence du projet CABRI ...

Vu l'ampleur du projet et si l'on prend en considération le nombre relativement faible de "programmeurs", il apparait comme évident que nous ne nous fixons pas pour objectif la réalisation d'un véritable produit mais plutôt celle d'un prototype évolutif, lié à un grand nombre de problèmes de recherche. »

Mais l'expérience a montré que l'intérêt du développement d'un tel cahier de brouillon informatique ne résidait pas uniquement dans sa mise à disposition des chercheurs et des enseignants.

Les concepteurs d'un tel logiciel disposent en effet de structures de représentation de graphes et des primitives les manipulant, ainsi que d'outils de visualisation, facilitant considérablement l'étude ou la conception d'algorithmes. Cet avantage peut être étendu à toute personne possédant les fichiers sources du logiciel et investissant dans la compréhension d'une partie de ces fichiers.

L'utilisation par les étudiants d'un éditeur de graphes, pour l'expérimentation d'outils et de concepts informatiques ou de théorie algorithmique des graphes, apporte généralement une motivation supplémentaire. De plus, il est possible d'offrir dans le cadre du développement du logiciel une très grande variété de projets.

Un éditeur de graphes peut enfin se révéler très utile à toute personne utilisant les graphes comme outil de modélisation : Recherche Opérationnelle, Informatique, Chimie, Réseaux de transports, Réseaux électriques, etc ...

1.2 Quelques réalisations

Didier Tallot dans sa thèse [Tal] fait remonter le premier éditeur de graphes à 1969 [Wol] ! Ce sujet n'est donc pas récent (le lecteur intéressé par une historique du problème pourra se référer à l'article de Dao et al. [DHRT]). A l'évidence, les progrès dans ce domaine sont très fortement liés aux avancées constatées en génie logiciel, en particulier dans le domaine de l'interactivité. L'arrivée des interfaces graphiques-souris, issues entre autres des travaux du Palo-Alto Research Center [SIKV] et popularisées par le Macintosh et les stations de travail, a ainsi ouvert de nouvelles perspectives. Les applications dont nous nous ferons l'écho dans la suite de cette thèse ont toutes une interface graphique-souris, avec toutefois des niveaux sensiblement différents d'interactivité.

Le groupe Cabri ...

Nous avons cité dans le paragraphe précédent le document de constitution du groupe Cabri. Ce groupe, issu d'un travail préliminaire effectué au Centre de Recherche Informatique de Montpellier [Gui], regroupait les équipes de recherche en combinatoire des organismes suivantes :

- Centre de Recherche en Informatique, Le Mans;
- Département d'Informatique de l'École des Mines de Saint-Étienne;
- Division Architecture et Trafic dans les Réseaux, CNET Paris A;
- Laboratoire d'Algèbre Ordinale, Lyon;
- Laboratoire de Recherche en Informatique, Orsay;
- Laboratoire de Structures Discrètes et de Didactique, Grenoble.

Outre le travail effectué à Montpellier, le document de constitution du groupe Cabri fait également référence à des travaux menés dans le cadre des projets LANGRAF [Zeg, Zin] à Paris VI (M. Chein) et LEG [Ver] à Bordeaux (R. Cori).

Trois réalisations dues à des membres du groupe Cabri seront étudiées ici.

La première, que nous appellerons Kabri, a été réalisée à l'Ecole des Mines de Saint-Étienne, dans l'équipe de Michel Habib. Cette expérience est rapportée en particulier dans la thèse de Didier Tallot [Tal, DHRT]. Elle a été poursuivie au sein du CNET dans le cadre des projets Ingres, puis Cassis [HTDR, CNET].

La seconde, Unicorn, provient de l'Université d'Orsay (Laboratoires LRI et ISEM). Elle fait l'objet de la seconde partie de la thèse de Jean-Michel Fourneau [Fou1, Fou2].

Enfin, la troisième, Cabri-graphes, autour de laquelle est bâtie cette thèse, a été élaborée par des chercheurs et enseignants du Laboratoire de Structures Discrètes et de Didactique de Grenoble [Ben, Lab1, Bau2].

... et les autres

En dehors des logiciels issus du groupe Cabri, nous avons pu étudier deux autres éditeurs : Grafo et VGMP (Visual Graph Manipulation Package for the Macintosh). Le premier est du à Augusto Arevalo Montero, de l'Universidad Metropolitana Caracas (Venezuela). Nous ne disposons malheureusement que d'une version ancienne (Février 1987). Le second logiciel a été écrit par Gordon Royle, à partir de programmes dus à Ronald C. Read [Rea2].

L'environnement matériel et logiciel

Parmi les cinq logiciels cités ci-dessus, seul le premier, Kabri, a été écrit pour une autre machine que le Macintosh : la SM90.

Il se distingue également par l'utilisation d'un langage interprété, CEYX, (langage orienté-objet extension de Le_Lisp), dans un soucis d'ouverture. Les deux produits du CNET, Ingres et Cassis, inspirés par les travaux sur Kabri,

utilisent l'environnement Y3, également basé sur Le_Lisp et qui comprend un langage orienté-objet, un moteur d'inférences et des interfaces graphiques. Cette ouverture du logiciel s'accompagne malheureusement d'une perte en portabilité par rapport à un produit équivalent écrit en C ou en Pascal.

Les autres éditeurs sont écrits en C (c'est le cas pour Cabri-graphes) ou en Pascal. Cabri-graphes comprend actuellement 67000 lignes de C et est développé dans l'environnement Think C. Son portage sur une station Unix, utilisant l'environnement graphique OSF-motifs, est d'ores et déjà planifié, afin d'augmenter la diffusion du logiciel. Ce portage devrait s'accompagner d'une réécriture dans un langage orienté-objet basé sur C, afin de bénéficier des avantages d'un tel langage en matière de lisibilité et de fiabilité. Notons qu'il existe déjà un éditeur de graphes sous Unix (Sun 3) écrit dans un langage orienté-objet (C++) [Him].

1.3 Définitions

Avant de poursuivre la description des différents éditeurs, nous donnons ici les principales définitions de théorie des graphes dont le lecteur aura besoin, notamment pour le paragraphe suivant. Ces définitions sont pour la plupart tirées du cours de Nguyen Huy Xuong [Xuo] ou du livre de Claude Berge [Ber].

Graphe

Un **graphe** G est un couple (V, E) formé de deux ensembles disjoints $V = \{v_1, v_2, \ldots v_n\}$ $(n \geq 1)$ et $E = \{e_1, e_2, \ldots e_m\}$ $(m \geq 0)$, tels que, pour tout $i \in [1..m]$, e_i est une paire d'éléments de V. V est appelé ensemble des **sommets** et E ensemble des **arêtes.**

Exceptionnellement, nous admettrons le cas où n est nul i.e. $G = (\varnothing, \varnothing)$. Nous dirons alors que G est le graphe vide, noté K_0.

Une arête $e = \{v, w\}$ est dite **incidente** à v et w, v et w étant les **extrémités** de e. v et w sont dits **adjacents** ou encore **voisins.** L'ensemble des sommets adjacents à v forme son **voisinage**, noté $\Gamma(v)$. Le nombre d'arêtes incidentes à v est appelé le **degré** de v.

Un graphe est dit **simple** lorsque deux arêtes différentes ont au moins une extrémité distincte (pas d'**arête multiple**) et qu'il n'existe pas d'arête dont les extrémités sont confondues (pas de **boucle**).

Transformations d'un graphe simple

Les quatres transformations de base pour modifier un graphe simple sont les suivantes :

$(V, E), v \notin V \rightarrow_{\mathbf{R1}} (V \cup \{v\}, E)$ *ajouter un sommet*

$(V, E), e \notin E \rightarrow_{R2} (V \cup e, E \cup \{e\})$ *ajouter une arête*

$(V, E), v \in V \rightarrow_{R3} (V \setminus \{v\}, E \setminus \{e, v \in e\})$ *supprimer un sommet*

$(V, E), e \in E \rightarrow_{R4} (V, E \setminus \{e\})$ *supprimer une arête*

Isomorphisme et étiquetage

Deux graphes simples $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ sont **isomorphes** si et seulement si il existe une bijection f de V_1 dans V_2 telle que :
$\forall v \forall w (\{v, w\} \in E_1 \Leftrightarrow \{f(v), f(w)\} \in E_2)$

Par extension, le concept de graphe pourra désigner, soit l'objet mathématique défini au début de ce paragraphe (un ensemble de sommets et un ensemble d'arêtes), soit la classe d'isomorphisme d'un tel objet. Dans le premier cas, on parlera de graphes **étiquetés**, la différenciation des sommets se faisant par leur **étiquette**, dans le second de graphes **non étiquetés**.

Orientation

Un graphe G = (V, E) est dit **orienté** lorsque pour toute arête e de E, on a ordonné l'ensemble de ses extrémités. E n'est plus alors un ensemble de paires, mais un ensemble de couples de sommets et ses éléments sont appelés des **arcs** (et non plus arête). Soit un arc e = (v, w). v est appelé l'**extrémité initiale** de e et w son **extrémité terminale**. De plus, v est dit **prédécesseur** de w et w **successeur** de v. On notera par $\Gamma^+(v)$ l'ensemble des successeurs de v et par $\Gamma^-(v)$ l'ensemble de ses prédécesseurs.

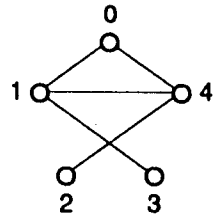
Dessin d'un graphe

Afin de mieux appréhender un graphe, on utilise généralement un dessin dans le plan, matérialisé par une feuille de papier ou l'écran d'un ordinateur. A chaque sommet est associé un point du plan et à chaque arête un arc de Jordan

reliant ses extrémités. Fréquemment, les arcs de Jordan utilisés sont des segments de droite. Il suffit alors de connaître les coordonnées dans le plan de chaque sommet pour que le dessin soit entièrement défini (figure 1).

Dans le cas d'un graphe orienté, les arcs de Jordan seront munis d'une flèche précisant l'orientation de chaque arc.

Un graphe simple étiqueté :
G = ({0, 1, 2, 3 ,4}, {{0, 1}, {0, 4}, {1, 3}, {1, 4}, {2, 4}})



- figure 1 -

Sous-graphes

Soit G = (V, E) un graphe et S un sous-ensemble non vide de V. Le **sous-graphe** de G **induit** par S est le graphe $G_S = (S, E_S)$ où E_S désigne le sous-ensemble de E des arêtes ayant leurs deux extrémités dans S. Un **graphe partiel** de G est un graphe G' = (V, E') où E' est un sous-ensemble de E. Un **sous-graphe** de G sera un graphe H = (W, F) où F est un sous-ensemble de E, W un sous-ensemble de V et où tout sommet extrémité d'au moins une arête de F appartient à W. Un sous-graphe H = (W, F) ne contenant pas de sommet de degré 0 sera appelé **arête-induit** et noté $G_{[F]}$. Dans ce cas, W sera égal au sous-ensemble de V des sommets extrémité d'au moins une arête de F. Par conséquent, H sera entièrement déterminé par F, ce qui justifie la notation.

Transformations d'un graphe simple (suite)

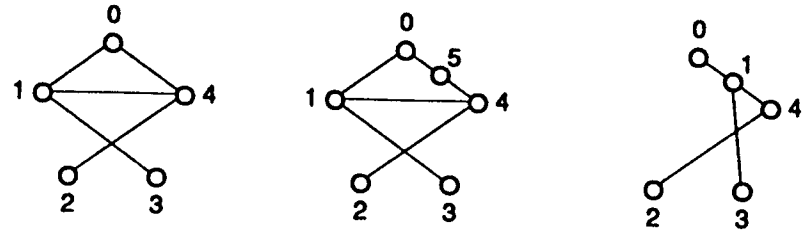
Il existe un très grand nombre de transformations usuellement utilisées par les chercheurs en théorie des graphes, définies à l'aide des opérations de base et

des notions de sous-graphes. Les plus courantes dans les éditeurs de graphes sont les suivantes :

- subdivision d'une arête :

$(V, E), e \in E, v \notin e \rightarrow_{R5} (V \cup \{v\}, (E \cup \{\{v, w\}, w \in e\}) \setminus \{e\})$

Subdivisions de l'arête {0, 4} par un nouveau sommet (5) ou par un sommet existant (1) :

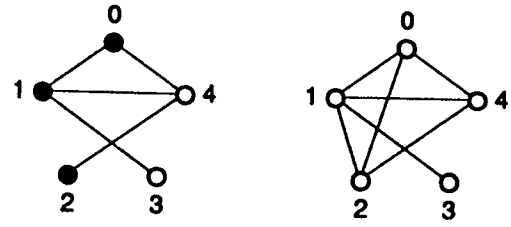


- figure 2 -

- compléter le sous-graphe induit par $S \subseteq V$:

$(V, E), S \subseteq V \rightarrow_{R6} (V, E \cup \mathcal{P}_2(S))$

Compléter le sous-graphe induit par S = {0, 1, 2} :

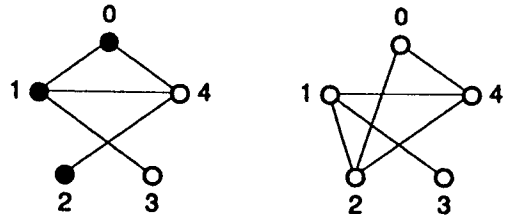


- figure 3 -

- complémenter le sous-graphe induit par $S \subseteq V$:

$(V, E), S \subseteq V \rightarrow_{R7} (V, E \Delta \mathcal{P}_2(S))$ où Δ désigne la différence symétrique.

Complémenter le sous-graphe induit par S = {0, 1, 2} :



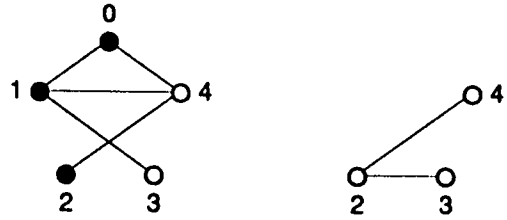
- figure 4 -

- fusionner les sommets de $S \subseteq V$:

$(V, E), S \subseteq V, s \in S \rightarrow_{R8} ((V \setminus S) \cup \{s\}, E')$

avec $E' = (E \setminus \{\{v, w\}, v \in S \text{ et } w \in V\}) \cup \{\{s, w\}, w \in \Gamma(S)\}$ où $\Gamma(S)$ désigne l'ensemble des sommets de $V \setminus S$ voisins d'au moins un sommet de S. Un cas particulier de fusion est celui où S est une arête. On parle alors de contraction d'une arête.

Fusionner les sommets de S = {0, 1, 2} (avec s = 2) :

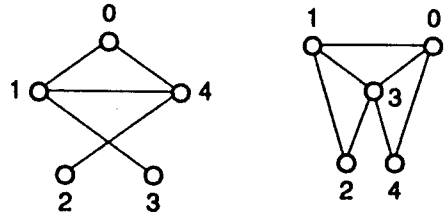


- figure 5 -

- graphe représentatif des arêtes (line graph) :

$(V, E) \rightarrow_{R9} (E, \{\{e_1, e_2\} \in \mathcal{P}_2(E), e_1 \cap e_2 \neq \varnothing\})$

Graphe représentatif des arêtes :



- figure 6 -

1.4 Les fonctionnalités d'un Cahier de Brouillon pour la théorie des graphes

Quels types de graphes ?

Ainsi que nous l'avons vu dans les définitions, il existe plusieurs types de graphes : orientés ou non, simples ou non.

Cabri-graphes traite des graphes non orientés simples, ainsi qu'Unicorn et VGMP. Kabri manipule des graphes orientés sans boucles, Grafo des graphes orientés ou non, simples (dans le cas de graphes orientés, il est possible d'avoir simultanément un arc et son opposé).

Le problème du traitement des graphes possédant des arêtes multiples et des boucles provient essentiellement de la visualisation. En effet, il est alors nécessaire de pouvoir représenter des arêtes autrement que par des segments de droites, ce qui rend l'interface beaucoup plus difficile à gérer : temps d'affichage plus long, difficulté de détecter la présence d'une arête passant par la position du curseur, possibilité de déformer une arête autrement qu'en modifiant la position d'une de ses extrémités ... Une première étude en vue de représenter des arêtes par des courbes de Bezier d'ordre 3 dans Cabri-graphes a été menée dans le cadre d'un mémoire de maîtrise [AB]. Un investissement plus important doit être envisagé avant d'espérer disposer d'une telle possibilité.

Les objets intrinsèques

D. Tallot [Tal] sépare les objets manipulés par l'utilisateur de Kabri en deux catégories : les objets **intrinsèques** (graphes, sommets, ...) et les objets de **contrôle** (menus, fenêtres, ...). Nous utiliserons ici la même distinction. Par contre, sauf nécessité, nous ne considérerons pas la représentation d'un objet comme un objet elle-même, mais comme un attribut de l'objet représenté.

Dans tous les logiciels auquels nous nous intéressons, le **graphe** et les **sommets** sont des objets intrinsèques. On retrouve également comme objet dans trois d'entre eux (Cabri-graphes, Kabri et Unicorn) un sous-ensemble de sommets, appelé **sélection de sommets** (ou plus simplement sélection lorsqu'il n'y a pas d'ambiguïté). Ce sous-ensemble permet l'utilisation de la notion de sous-graphe induit.

Dans Cabri-graphes et Kabri, les **arêtes** (les arcs pour Kabri) peuvent être également considérées comme des objets et non pas seulement comme des relations entre deux sommets. Comme pour les sommets, il existe dans ces deux logiciels un sous-ensemble d'arêtes, appelé **sélection d'arêtes**. Ce sous-ensemble permet l'utilisation de la notion de sous-graphe arête-induit et nous le considérerons également comme un objet intrinsèque.

Les objets de contrôle

Les éditeurs dont il est question ici ont tous une interface graphique-souris. C'est pourquoi on retrouve pour chacun d'entre eux les mêmes objets de contrôle : fenêtre, menu, curseur, ... objets désormais familier à la plupart des concepteurs ou utilisateurs de logiciels. Un lecteur non averti pourra se référer à l'ouvrage décrivant l'interface sur le Macintosh : "The Macintosh User Interface Guidelines" [HEA].

Opérations sur les objets intrinsèques

Les opérations sur les objets intrinsèques peuvent se classifier en fonction de leurs arguments, de leur complexité, et de la nature de leur résultat.

Les plus importantes sont les opérations que nous qualifierons d'**élémentaires**, utilisées pour la création ou la modification d'un objet intrinsèque. Elles se caractérisent par un usage bref, répété, et simple. Un soin

tout particulier doit être apporté à leur implémentation, tant pour l'interface que pour la rapidité d'exécution.

Les autre opérations, dites **secondaires**, se distingueront des opérations élémentaires par leur plus faible utilisation. Elles seront souvent plus sophistiquées et pourront nécessiter l'utilisation de dialogues lors de leur appel.

Remarquons que la séparation opérations élémentaires/opérations secondaires est subjective. Elle dépend en fait des objectifs que se sont donnés les concepteurs de l'éditeur. A titre d'exemple, nous avons choisi de considérer le calcul du graphe représentatif des arêtes comme une opération secondaire. Si cette opération est relativement simple, sans interface sophistiquée, et est décrite dans tous les manuels de théorie des graphes, elle ne nous apparait toutefois pas comme essentielle, comparativement au calcul d'un graphe ou sous-graphe complet par exemple.

Nous présentons les opérations élémentaires en fonction de leur argument (ou de l'un de leurs arguments).

Opérations élémentaires sur les sommets

Il s'agit naturellement de la classe d'opérations la plus commune à tous les éditeurs. On y trouve les opérations suivantes :

- créer un sommet;
- détruire un sommet;
- modifier les attributs d'un sommet (voisinage, position, dessin, étiquette, poids ...);
- fusionner deux sommets;

L'opération "détruire un sommet", que l'on rencontre dans Kabri, VGMP et Grafo, n'existe pas directement dans Cabri-graphes et dans Unicorn, mais est remplacée par "détruire le sous-graphe induit".

Opérations élémentaires sur les arêtes/arcs

Seuls Cabri-graphes et Kabri considèrent réellement les arêtes comme des objets. Les opérations suivantes apparaissent uniquement dans ces deux applications :

- créer une arête ex nihilo, i.e. en créant simultanément ses deux extrémités ;

- créer une arête incidente à un sommet existant, l'autre extrémité étant crée simultanément;

- créer une arête incidente à deux sommets existants. Cette opération n'est pas sémantiquement différente de "ajouter un sommet w au voisinage du sommet v", mais s'en distingue par son interface;

- subdiviser une arête. Cette fonctionnalité est plus développée dans Kabri que dans Cabri-graphes. En effet, il est possible dans le premier de subdiviser une arête à l'aide d'un sommet déjà existant, alors que le second ne permet la subdivision que lors de la création d'un nouveau sommet.

La subdivision des arêtes existe aussi dans Unicorn et VGMP, mais passe par la désignation de deux sommets adjacents. Grafo permet la valuation d'un arc en désignant son sommet initial, puis son sommet terminal.

Opérations élémentaires sur la sélection de sommets et le sous-graphe induit par la sélection

Dans Cabri-graphes et Unicorn, les opérations élémentaires sur la sélection de sommets se devaient d'imiter par leur contenu et leur interface les opérations élémentaires sur la sélection de fichiers et dossiers du bureau électronique du Macintosh.

Si les opérations de Kabri sont différentes quand à l'interface, elles suivent le même découpage fonctionnel que Cabri-graphes et Unicorn :

- initialiser la sélection à un sommet;
- initialiser la sélection à un ensemble de sommets inscrits dans un rectangle ;
- différence symétrique entre la sélection et un singleton;
- différence symétrique entre la sélection et un ensemble de sommets inscrits dans un rectangle;
- complémenter la sélection par rapport à l'ensemble des sommets;
- sélectionner tous les sommets;
- vider la sélection;
- détruire le sous-graphe induit;
- déplacer le sous-graphe induit;
- compléter le sous-graphe induit;
- supprimer les arêtes du sous-graphe induit;
- complémenter le sous-graphe-induit;
- fusionner les sommets de la sélection.

Opérations élémentaires sur la sélection d'arêtes et le sous-graphe arête-induit par la sélection

La sélection d'arêtes n'existe que dans Cabri-graphes et Kabri. De plus, elle n'a été rajoutée dans Cabri-graphes que depuis peu de temps. C'est pourquoi un certain nombre de fonctionnalités prévues ne sont pas encore implémentées. Par exemple, il n'existe aucune opération secondaire sur cette sélection, contrairement à la sélection de sommets. On devrait en fait pouvoir retrouver sur le sous-graphe arête-induit les mêmes fonctionnalités que sur le sous-graphe induit, de même qu'il devrait être possible de transformer un sous-graphe arête-induit en sous-graphe induit sans sommet isolé et inversement. Ce n'est malheureusement pas encore le cas, et l'utilisateur doit se contenter des fonctions suivantes :

- initialiser la sélection à une arête;

- initialiser la sélection à un ensemble d'arêtes inscrites dans un rectangle;
- différence symétrique entre la sélection et un singleton;
- différence symétrique entre la sélection et un ensemble d'arêtes inscrites dans un rectangle;
- vider la sélection;
- détruire le sous-graphe arête-induit;
- déplacer le sous-graphe arête-induit.

Notons qu'il est prévu dans Cabri-graphes, lors de la création d'un sommet ou d'une arête, d'ajouter la possibilité d'insérer ce sommet ou cette arête dans la sélection les concernant. Ceci permettra en particulier de préserver le contenu des sélections lors d'adjonction de sommets ou d'arêtes. Il est en effet impossible actuellement d'ajouter un sommet sans vider la sélection, avant ou après la création.

Opérations élémentaires sur le graphe

On retrouve sur le graphe les opérations élémentaires sur le sous-graphe induit à l'exception de la fusion :

- déplacer le graphe;
- détruire le graphe;
- compléter le graphe;
- complémenter le graphe;
- supprimer les arêtes du graphe.

Opérations secondaires

Ce sont souvent les opération secondaires qui feront qu'un chercheur choisira un éditeur de graphes plutôt qu'un autre.

Malheureusement, chaque chercheur en théorie des graphes a des besoins très spécifiques et il semble rigoureusement impossible de les satisfaire tous.

Aussi, il est indispensable que tout chercheur ayant un minimum de compétence en informatique puisse utiliser conjointement et facilement ses propres fonctions et l'éditeur.

Kabri est le seul logiciel a avoir répondu à ce problème par l'utilisation d'un langage orienté-objet interprété, sous environnement LISP, entraînant malheureusement par là une perte de portabilité par rapport aux éditeurs écrits en Pascal ou en C. L'utilisation de fichiers texte dans Cabri-graphes et Unicorn permet d'apporter un début de réponse, que devrait compléter la réécriture de Cabri-graphes à l'aide d'un langage orienté-objet et avec des objectifs de réutilisation du code.

La description des opérations secondaires offertes dans les logiciels Cabri-graphes, Grafo, Unicorn et VGMP sera faite dans les paragraphes : "Les opérations secondaires de Cabri-graphes" et "Les opérations secondaires de Grafo, Unicorn et VGMP". Il n'existe pas de description d'opérations secondaires de Kabri dans la thèse de D. Tallot.

Opérations sur les objets de contrôle

La plupart des opérations sur les objets de contrôle existant sur le Macintosh, en particulier dans les logiciels graphiques (MacPaint II, MacDraw II ...), ont été implémentées dans Cabri-graphes, auquel s'ajoutent quelques fonctionnalités originales (plein écran, possibilité d'éditer le presse-papier lorsqu'il contient un graphe, possibilité de redéfinir la taille des fenêtres au lancement de l'application ...). Il serait fastidieux d'en faire ici le détail, bien qu'il s'agisse là d'un travail important et souvent difficile[1].

[1] L'auteur tient à préciser que le mérite de ce travail revient pour sa plus grande part à J.M. Laborde.

Les autres éditeurs ont des niveaux de développement très divers en ce qui concerne les opérations sur les objets de contrôle. Il faut cependant noter qu'aucun d'entre eux n'est aussi évolué que Cabri-graphes, faute sans doute de moyens et de temps.

Les entrées-sorties

Pour clore ce paragraphe, nous donnerons quelques indications sur les entrées-sorties.

Le fait de pouvoir stocker des graphes sur un support magnétique représente un avantage non négligeable par rapport à l'archivage de feuilles de brouillon. Il serait fort intéressant de pouvoir bénéficier d'un stockage "intelligent" sous la forme d'une base de données. Cela nécessite à notre avis un travail de réflexion important si l'on veut arriver à un résultat satisfaisant.

Le stockage sous la forme de fichiers texte nous parait tout à fait indispensable pour les raisons exposées plus haut. Il faut d'autre part que l'automate de lecture d'un tel fichier laisse le maximum de liberté à un utilisateur pour décrire son graphe. L'automate actuel est présenté dans l'Annexe 2.

Cabri-graphes et Unicorn disposent tous les deux d'un presse-papier. Les deux applications permettent ainsi le transfert du dessin du graphe dans les différents logiciels graphiques ou d'édition de texte, ce que nous avons fait pour la plupart des illustrations présentes dans cette thèse. Le presse-papier de Cabri-graphes permet également le transfert de texte. Ce texte provient d'une fenêtre spéciale, appelée "Journal", éditable et dans laquelle s'affiche le résultat de certaines opérations : graphe sous la forme de liste d'adjacence, nombre et polynôme chromatique, spectre du graphe.

1.5 Définitions complémentaires

Ce paragraphe dresse la liste des notions traitées par l'un au moins des éditeurs que nous avons étudiés. Sauf précision contraire, on considérera dans l'ensemble du paragraphe un graphe G = (V, E), simple, avec |V| = n et |E| = m. De même que pour le paragraphe 1.3, les définitions les plus usuelles sont tirées des ouvrages de N.H. Xuong [Xuo] et C. Berge [Ber], les autres provenant d'articles.

Chaînes et cycles

Une **chaîne** C de **longueur k** est une suite de sommets $v_0, v_1, \ldots, v_k$ telle que pour tout $i \leq k-1$, $\{v_i, v_{i+1}\} \in E$. Si tous les v_i sont distincts, la chaîne C sera dite **élémentaire**.

Un **cycle** est une chaîne dont les extrémités sont confondues. Un cycle **élémentaire** est un cycle dont tous les sommets, à l'exception des extrémités, sont distincts.

Un cycle **hamiltonien** est un cycle élémentaire contenant tous les sommets du graphe. Un graphe hamiltonien est un graphe possédant un tel cycle. Un cycle **eulérien** C est un cycle passant par toutes les arêtes du graphe une et une seule fois, i.e. $C = (v_i)_{i \leq m+1}$ avec $E = \{\{v_i, v_{i+1}\}, i \leq m\}$. Un graphe eulérien est un graphe possédant un tel cycle.

La **maille** est la longueur du plus petit cycle élémentaire. Un graphe possédant un cycle élémentaire pour toute longueur comprise entre 3 et n sera dit **pancyclique**.

La **subdivision d'ordre k** d'un graphe G est obtenue en subdivisant chaque arête de G par k sommets, i.e. en remplaçant chaque arête par une chaîne de longueur k+1.

Connexité

Soit ~ la relation d'équivalence sur V définie par x ~ y si et seulement si il existe une chaîne entre x et y. Une **classe connexe** est une classe d'équivalence de ~. Une **composante connexe** est un sous-graphe induit par une classe connexe. Un **graphe connexe** est un graphe possédant une seule composante connexe.

Un **ensemble d'articulation** est un ensemble de sommets dont la suppression disconnecte le graphe et minimal pour cette propriété. Un **point d'articulation** est un sommet v tel que {v} est un ensemble d'articulation. Soit k un entier naturel. G sera dit **k-connexe** si tous ses ensembles d'articulation sont de cardinal supérieur ou égal à k. La **connectivité** d'un graphe est le cardinal minimum de ses ensembles d'articulation.

Soit S un sous-ensemble de V. Le **cocyle** de S est l'ensemble des arêtes ayant exactement une extrémité dans S. G sera dit **k-arête-connexe** si tous ses cocycles sont de cardinal supérieur ou égal à k. L'**arête-connectivité** d'un graphe est le cardinal minimum de ses cocycles.

Voisinage

Le **voisinage** d'un sous-ensemble S de V, noté $\Gamma(S)$, est l'ensemble des sommets de $V \setminus S$ voisins d'au moins un sommet de S.

Le **voisinage étendu** d'un sous-ensemble S de V, noté $\Gamma^*(S)$, est l'union des sommets de S et des sommets de $\Gamma(S)$.

Un graphe **régulier** est un graphe dont tous les sommets ont le même degré.

Un **0-2 graphe** est un graphe où deux sommets distincts ont soit deux soit aucun voisins en commun.

La **matrice d'adjacence** d'un graphe G est la matrice $A = (a_{ij})_{i \in V, j \in V}$ où a_{ij} est le nombre d'arêtes ayant i et j comme extrémités. Le **spectre** d'un graphe est la liste des valeurs propres de sa matrice d'adjacence.

Distance

La **distance** entre deux sommets v et w, notée d(v, w), est la longueur d'une plus courte chaîne d'extrémités v et w.

L'**excentricité** d'un sommet v est définie par $ex(v) = \max_{w \in V} d(v, w)$.

Le **diamètre** et le **rayon** de G, notés respectivement D et r, sont les valeurs définies par $D = \max_{v \in V} ex(v)$ et $r = \min_{v \in V} ex(v)$.

Les **sommets périphériques** et les **centres** d'un graphe sont respectivement les sommets réalisant le diamètre et les sommets réalisant le rayon. Un graphe dont tous les sommets sont périphériques est dit **sphérique** (le terme d'équiexcentrique est également utilisé).

Soit L une liste d'entiers entre 0 et n-1. **Changer les distances d'un graphe** G = (V, E) à l'aide de la liste L consiste à remplacer G par le graphe $G' = (V, \{\{v, w\} \in \mathcal{P}_2(V), d(v, w) \in L\})$.

Soit G-v le graphe obtenu en supprimant le sommet v de G, et G-e le graphe obtenu en supprimant l'arête e. La **vulnérabilité** d'un graphe G est l'accroissement maximum du diamètre lorsque l'on supprime un sommet de G : $\text{vulnérabilité}(G) = \max_{v \in V} (D(G - v) - D(G))$. L'**arête-vulnérabilité** de G est

l'accroissement maximum du diamètre lorsque l'on supprime une arête de G :
arête-vulnérabilité(G) = $\max_{e \in E} (D(G - e) - D(G))$.

Intervalle

L'**intervalle** induit par une paire de sommets {v, w}, noté I(v, w), est l'ensemble des sommets appartenant à une plus courte chaîne entre v et w. Autrement dit, $I(v, w) = \{z \in V, d(v, z) + d(z, w) = d(v, w)\}$

Soit v un sommet. La **fonction de Möbius** de base v est la fonction M définie par :

$M(v) = 1$

$\forall w \neq v, M(w) = - \sum_{y \in I(v,w) \setminus w} M(y)$

Les **valeurs de Möbius** d'un graphe seront les valeurs que peuvent prendre les différentes fonctions de Möbius sur ce graphe.

Un graphe **médian** est un graphe où pour toute 3-partie {x, y, z} de sommets, $|I(x, y) \cap I(y, z) \cap I(x, z)| = 1$.

Un graphe **intervalle-monotone** est un graphe tel que
$\forall x, y \in V (\{a, b\} \subseteq I(x, y) \Rightarrow I(a, b) \subseteq I(x, y))$

Un graphe **distance-monotone** est un graphe tel que
$\forall x, y, z \in V, (z \notin I(x, y) \Rightarrow (\exists z' \in I(x, y), d(z, z') > d(x, y)))$

Produits de graphes

Soit $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$. Le **produit cartésien** de G_1 et G_2, noté $G_1 \square G_2$ est le graphe $(V_1 \times V_2, E')$ avec

$\forall \{(v_1, v_2), (w_1, w_2)\} \in \mathcal{P}_2(V_1 \times V_2)$,

$$\{(v_1, v_2), (w_1, w_2)\} \in E' \Leftrightarrow \begin{cases} v_1 = w_1 \text{ et } \{v_2, w_2\} \in E_2 \\ \text{ou} \\ \{v_1, w_1\} \in E_1 \text{ et } v_2 = w_2 \end{cases}$$

Le **produit croisé** de G_1 et G_2, noté $G_1 \times G_2$ est le graphe $(V_1 \times V_2, E')$ avec $\forall \{(v_1, v_2), (w_1, w_2)\} \in \mathcal{P}_2(V_1 \times V_2)$,
$\{(v_1, v_2), (w_1, w_2)\} \in E' \Leftrightarrow \{v_1, w_1\} \in E_1$ et $\{v_2, w_2\} \in E_2$

Soit $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$. L'union de G_1 et G_2, noté $G_1 \cup G_2$ est le graphe $(V_1 \cup V_2, E_1 \cup E_2)$.

Le **produit complet** de G_1 et G_2, noté $G_1 \boxtimes G_2$ est le graphe $(G_1 \square G_2) \cup (G_1 \times G_2)$ (les produits cartésien, croisé et complet sont respectivement appelés somme cartésienne, produit cartésien et produit normal dans [Ber]).

L'**hypercube** de dimension n, noté Q_n, est le graphe dont l'ensemble des sommets est $\{0, 1\}^n$ et où deux sommets sont adjacents si et seulement si ils ne diffèrent que d'une composante. Soit K_i le graphe complet à i sommets. Q_n peut être obtenu par la formule récursive suivante : Q_n = si n = 1 alors K_1 sinon $K_2 \square Q_{n-1}$.

Planarité

Un graphe **planaire** est un graphe représentable sur le plan avec les contraintes suivantes :

(1) les sommets sont représentés par des points;

(2) les arêtes sont représentées par des courbes simples reliant les points représentant leurs extrémités;

(3) deux arêtes ne s'intersectent qu'en leurs éventuelles extrémités communes.

Soit (2') la contrainte (2) où l'on remplace "courbes simples" par "segments de droite". Tout graphe simple planaire peut être représenté dans le plan en respectant les contraintes (1), (2') et (3) [Far, Wag]. D'autre part, il est connu que tout graphe planaire est représentable sur la sphère en respectant les contraintes (1), (2) et (3).

Un **graphe plan** (resp. **sphérique**) est une représentation d'un graphe planaire dans le plan (resp. dans la sphère), respectant les contraintes (1), (2) et (3). On ne considérera pas comme distincts deux graphes plans ou sphériques que l'on peut amener à faire coïncider par déformation élastique de la surface sur laquelle ils sont représentés.

On appelle **face** d'un graphe plan (ou sphérique) une région du plan (ou de la sphère) limitée par des arêtes et telle que deux points arbitraires dans cette région peuvent être reliés par une courbe simple ne rencontrant ni sommet ni arête.

Il est possible de passer d'un graphe sphérique à un graphe plan par projection stéréographique. Tout graphe plan possède exactement une face infinie, alors qu'un graphe sphérique n'en possède aucune. Pour passer d'un graphe sphérique à un graphe plan, il faut donc marquer dans le premier une face quelconque dans laquelle sera choisi le centre de la projection stéréographique. Cette face sera la face infinie sur le plan.

On appelle **plongement topologique** d'un graphe planaire une suite de listes circulaires $(r)_{v \in V}$, telle qu'il existe un graphe plan où pour tout v, r_v est la liste des voisins de v considérés dans l'ordre trigonométrique indirect (sens des aiguilles d'une montre). Pour la classe des graphes simples, il y a équivalence

entre la notion de graphe sphérique et celle de plongement topologique, i.e. à un graphe sphérique, on peut associer un et un seul plongement topologique [Edm]. Les listes circulaires d'un plongement topologique sont appelés des **rotations**.

Sous-graphes particuliers

Un **stable** est un sous-ensemble de V induisant un sous-graphe sans arête. Le cardinal maximum d'un tel sous-ensemble sera noté par $\alpha(G)$.

Un graphe **biparti** est un graphe dont les sommets sont partitionnables en deux stables.

Une **clique** est un sous-ensemble de V induisant un sous-graphe complet. Le cardinal maximum d'un tel sous-ensemble sera noté par $\omega(G)$.

Une **forêt** est un graphe sans cycle. Un **arbre** est une forêt connexe. Un arbre **couvrant** d'un graphe G est un arbre graphe partiel de G. Une **arborescence** est un arbre dont on a distingué un sommet, appelé **racine**. Cette racine induit une orientation naturelle, où chaque arc est orienté du sommet le plus proche vers le sommet le plus éloigné de la racine.

Un **couplage parfait** est un ensemble d'arêtes disjointes de G telles que tout sommet est incident à une (et une seule) d'entre elles.

Coloration

Une **coloration** d'un graphe G est une application c définie sur V telle que $\{v, w\} \in E \Rightarrow c(v) \neq c(w)$. Une **k-coloration** est une coloration c telle que $|c(V)| = k$. On note par $\chi(G)$ le plus petit entier pour lequel il existe une $\chi(G)$-coloration. $\chi(G)$ est appelé le **nombre chromatique** de G.

Soit v et w deux sommets non adjacents de G. On note par $G_{v,w}^{\sim}$ le graphe obtenu en fusionnant v et w et par $G_{v,w}^{+}$ le graphe obtenu en rajoutant l'arête

$\{v, w\}$. $P(G, \mu)$ est la fonction qui exprime le nombre de μ-colorations de G. On démontre que cette fonction est un polynôme, appelé **polynôme chromatique**, qui peut être obtenu par la définition récursive suivante :

$$P(G, \mu) = \begin{cases} \mu(\mu-1) \dots (\mu - n + 1) \text{ si G est le graphe complet à n sommets} \\ P(G^{+}_{v,w}, \mu) + P(G^{\sim}_{v,w}, \mu) \text{ sinon} \end{cases}$$

où $\{v, w\}$ désigne une paire de sommets non adjacents dans G.

Sommet-transitivité

G est dit **sommet-transitif** si et seulement si pour toute paire de sommets $\{v, w\}$, il existe un automorphisme π de G tel que $\pi(v) = w$.

G est dit **fortement sommet-transitif** si et seulement si pour toute paire de sommets $\{v, w\}$ et pour toute bijection β de $\Gamma(v)$ dans $\Gamma(w)$, il existe un automorphisme π de G tel que $\pi(v) = w$ et pour tout x voisin de v, $\pi(x) = \beta(x)$.

Plongement

Un **plongement** d'un graphe $G_1 = (V_1, E_1)$ dans un graphe $G_2 = (V_2, E_2)$ est une application p de V_1 dans V_2 telle que
$\{v_1, v_2\} \in E_1 \Leftrightarrow \{p(v_1), p(v_2)\} \in E_2$. Si une telle application existe, G_1 sera dit plongeable dans G_2.

Densité

La **densité** d'un graphe G = (V, E) est la valeur $\frac{2m}{n(n-1)}$ comprise entre 0 (pour un stable) et 1 (pour un graphe complet).

Complexité

Nous ne donnons ici que le minimum de notions sur la théorie de la complexité en optimisation combinatoire, nécessaires à la compréhension du

texte. Le problème de la complexité des algorithmes est évidemment crucial pour tout personne s'intéressant à l'aspect algorithmique des graphes. Pour plus d'informations, nous renvoyons le lecteur à l'article de vulgarisation de H.R. Lewis et C.H. Papadimitriou [LP] ou à l'ouvrage de référence qu'est le livre de M.R. Garey et D.S. Johnson [GJ].

Un algorithme est dit **polynômial** lorsque le nombre total d'opérations élémentaires (opérations mathématiques, comparaisons) effectuées pour aboutir à la solution est borné par un polynôme en la taille du problème en entrée. Un problème est dit polynômial s'il existe un algorithme polynômial pour le résoudre.

Un **problème NP** (pour Non déterministe Polynômial) est un problème de décision (réponse en oui ou non) tel que si la réponse est oui, il existe une preuve du oui vérifiable à l'aide d'un algorithme polynômial. Par exemple, tout problème de décision polynômial est NP car le oui peut être vérifié en temps polynômial par l'algorithme qui a permis de l'obtenir ! On démontre qu'il existe des problèmes, dits **NP-complets**, tels que la connaissance d'un algorithme polynômial pour résoudre l'un d'entre eux permettrait la résolution en temps polynômial de tous les problèmes NP. Dans l'état actuel des connaissances, on ne sait s'il existe ou non un tel algorithme.

Par extension, on appelera NP-complet tout problème d'optimisation pouvant se ramener en temps polynômial à un problème de décision NP-complet. Par exemple, le calcul du cardinal d'un stable maximum dans un graphe à n sommets peut se ramener à n problèmes "Existe-t-il un stable de cardinal supérieur ou égal à k ?" avec $1 \leq k \leq n$.

Un **invariant** est une fonction définie sur un graphe stable par isomorphisme, i.e. indépendante de l'étiquetage du graphe. Un invariant sera dit polynômial ou non polynômial selon qu'il existe ou non dans l'état actuel des

connaissances un algorithme polynômial permettant son calcul. Si le problème de son calcul est NP-complet, l'invariant sera également appelé NP-complet.

1.6 Les opérations secondaires de Cabri-graphes

Classification

Ces opérations sont présentées en fonction du type de résultat fourni : aide à la construction du graphe, modification de la sélection de sommets, calcul d'invariants, modification du dessin du graphe, modification de l'étiquetage.

Dans ce paragraphe, G désignera le graphe présent dans la fenêtre active, V et E ses ensembles de sommets et d'arêtes, S l'ensemble des sommets sélectionnés, n le nombre de sommets de G et m son nombre d'arêtes.

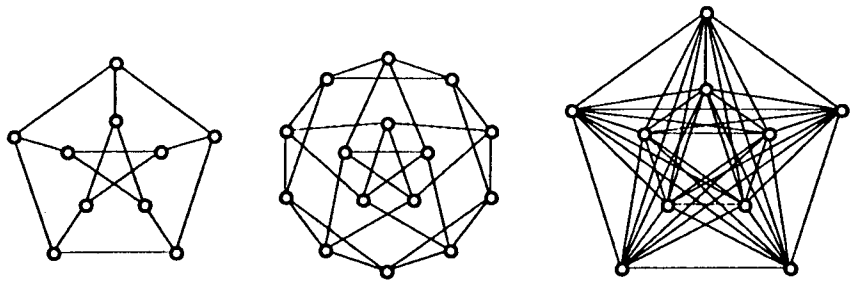
Aide à la construction du graphe

Nous proposons les constructions suivantes qui portent sur le graphe ou sur son sous-graphe induit par la sélection :

- graphe représentatif des arêtes (figure 7.1);
- produits cartésien, croisé et complet du graphe de la fenêtre active et du graphe contenu dans le presse-papier (figure 7.2);
- subdivision d'ordre k, le facteur k étant fixé par l'utilisateur (figure 7.3);
- suppression des sommets de degré 2 dont les voisins sont non adjacents, ces voisins étant ensuite reliés;
- changement des distances du graphe à l'aide d'une liste donnée par l'utilisateur (figure 7.1).

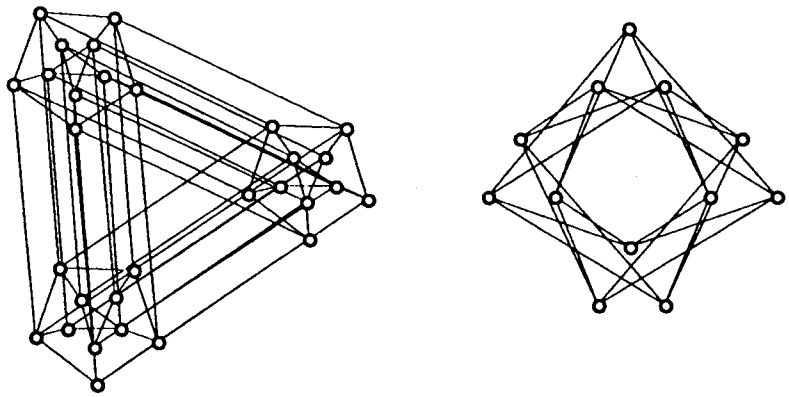
La fonction "Coller" permet de rajouter au graphe de la fenêtre active un graphe précédemment copié ou construit dans le presse-papier. Notons que l'action "Copier" portera sur le sous-graphe induit par la sélection si cette dernière est non vide. D'autre part, si lors du "Coller", la sélection est non vide, Cabri-graphes rajoutera l'ensemble des arêtes entre le graphe collé et les sommets de la sélection (figure 7.3).

Le graphe de Petersen, son graphe représentatif des arêtes et le graphe obtenu par changement de distances à l'aide de la liste (1 2) :



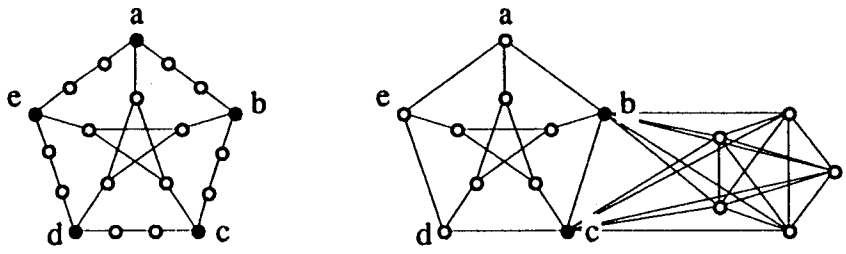
- figure 7.1 -

Produit cartésien du graphe de Petersen et du cycle de longueur 3, et produit croisé des cycles de longueur 3 et 4 :



- figure 7.2 -

Subdivision d'ordre 2 du sous-graphe de Petersen induit par S = {a, b, c, d, e} , et collage d'un graphe complet à 5 sommets au graphe de Petersen, les sommets b et c étant sélectionnés :



- figure 7.3 -

Il existe également dans Cabri-graphes un générateur paramétrable de graphes aléatoires qui fera l'objet du chapitre 2.

Modification d'une sélection de sommets

Il existe deux catégories d'opérations secondaires sur la sélection : la première catégorie est celles des fonctions opérant sur le graphe ou sur le sous-graphe induit par la sélection, mais qui dans le cas d'un sous-graphe induit ne tiennent pas compte du reste du graphe. La seconde catégorie calcule une nouvelle sélection en prenant en compte la sélection initiale et l'ensemble du graphe.

La première catégorie regroupe les fonctions suivantes :

- sommets de degré minimum;
- sommets de degré maximum;
- sommets d'étiquette numérique minimum;
- sommets d'étiquette numérique maximum;
- sommets d'étiquette numérique donnée;
- sommets de forme ou de taille donnée;
- centres du graphes;
- sommets périphériques;
- intervalle de deux sommets;
- stable maximal;
- clique maximale;
- points d'articulation.

Dans le cas des sommets d'étiquette numérique, de forme ou de taille donnée, l'utilisateur doit préciser la valeur, la forme ou la taille désirée, de même qu'il doit choisir deux sommets pour le calcul d'un intervalle.

La seconde catégorie est plus restreinte et comprend les fonctions suivantes :

- voisinage des sommets contenus dans la sélection;

- voisinage étendu des sommets de la sélection;
- union des classes connexes des sommets de la sélection.

Calcul d'invariants

Différentes approches ont été mises en oeuvre concernant le calcul d'invariants non polynômiaux. La première est de ne donner une réponse qu'avec une certaine probabilité, dépendant du graphe considéré, mais qui soit exacte (et non approximée) si elle est obtenue. La seconde est de fournir une approximation de la solution et la troisième de proposer un algorithme donnant une solution exacte, mais ne pouvant être exécuté que sur des graphes de taille limitée.

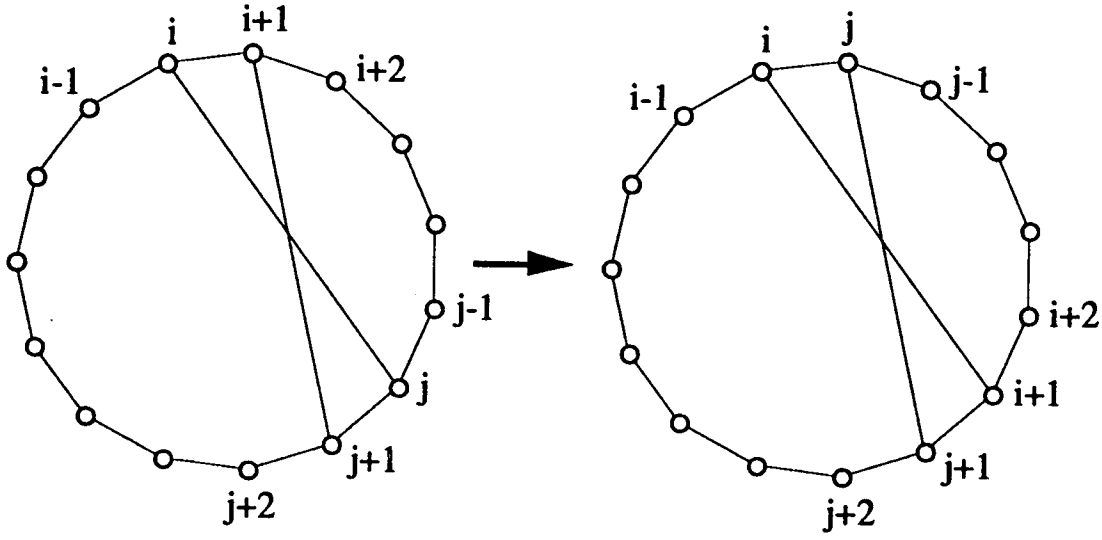
Pour le problème de l'existence d'un cycle hamiltonien, nous utilisons un algorithme, basé sur l'heuristique du voyageur de commerce appelée 2-opt , qui fournit un cycle hamiltonien avec une certaine probabilité. Cette probabilité est malheureusement inconnue, mais dépend trivialement de la densité du graphe.

Considérons un graphe complet à n sommets K_n, où les arêtes de G sont munies d'un poids nul et les arêtes de son complémentaire d'un poids égal à 1. L'opération de croisement d'arêtes pour un cycle hamiltonien de K_n est l'opération consistant à échanger dans ce cycle une paire d'arêtes non consécutives ({u, u'} ; {v, v'}) par la paire ({u, v}; {v', u'}) (figure 8). L'algorithme 2-opt consiste à rechercher dans K_n un cycle hamiltonien (i.e. une permutation sur les sommets) de poids minimal pour l'opération de croisement d'arêtes. Ce cycle est de poids nul si et seulement si c'est un cycle hamiltonien de G. Le cycle de départ est obtenu par un tirage aléatoire sur l'ensemble des permutations de [1..n].

Une étude statistique sur la qualité de l'algorithme 2-opt, appliquée au problème plus général du voyageur de commerce, est présentée dans l'article de

Golden et Stewart [GS]. Cette étude porte également sur d'autres heuristiques du problème du voyageur de commerce, qui peuvent être utilisées dans le cadre du problème de la recherche d'un cycle hamiltonien.

Opération de croisement d'arêtes dans un cycle hamiltonien de K_n :

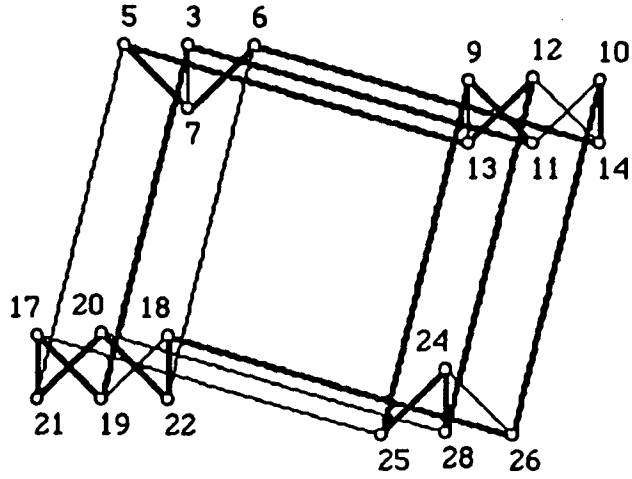


- figure 8 -

Dans certain cas, il est impossible de définir un algorithme autre que non polynômial. Il peut être toutefois intéressant de disposer d'un tel algorithme, qui s'il n'est applicable qu'à des graphes de petites tailles, sera toutefois nettement plus performant que ce que l'on peut faire manuellement. Le cas le plus exemplaire est celui du calcul du polynôme chromatique (figure 10).

Cabri-graphes teste également s'il existe un plongement du graphe de la fenêtre active dans celui du presse-papier. Ce test est toutefois facilité si l'utilisateur précise que le graphe du presse-papier est sommet-transitif, ou mieux encore, fortement sommet-transitif (figure 9).

Plongement d'un cycle à 20 sommets dans le sous-graphe de Q_5 (hypercube de dimension 5) induit par les sommets ayant 3 ou 4 composantes à 1. Ce plongement permet de vérifier et visualiser l'existence d'un cycle hamiltonien sur ce graphe :



- figure 9 -

Pour les invariants non polynômiaux numériques, l'approche la plus classique est de fournir une approximation de la réponse sous la forme d'un encadrement. C'est ce qui est fait dans Cabri-graphes pour le cardinal maximum d'un stable $\alpha(G)$, le cardinal maximum d'une clique $\omega(G)$ et le nombre chromatique $\chi(G)$ (figure 11). Dans le cas du nombre chromatique, un algorithme donnant une réponse précise est également disponible, qui peut être utilisé si l'encadrement ne donne pas satisfaction et si le graphe est suffisamment petit (figure 10).

Notons par $\alpha_{min}(G)$ une borne inférieure de $\alpha(G)$, par $\chi_{max}(G)$ une borne supérieure de $\chi(G)$ et par $\overline{G}$ le complémentaire du graphe G. Les bornes de $\alpha(G)$, $\omega(G)$ et $\chi(G)$ affichées par Cabri-graphes sont données à l'aide des inégalités suivantes :

$\alpha_{min}(G) \leq \alpha(G) \leq \chi_{max}(\overline{G})$

$\alpha_{min}(\overline{G}) \leq \omega(G) \leq \chi_{max}(G)$

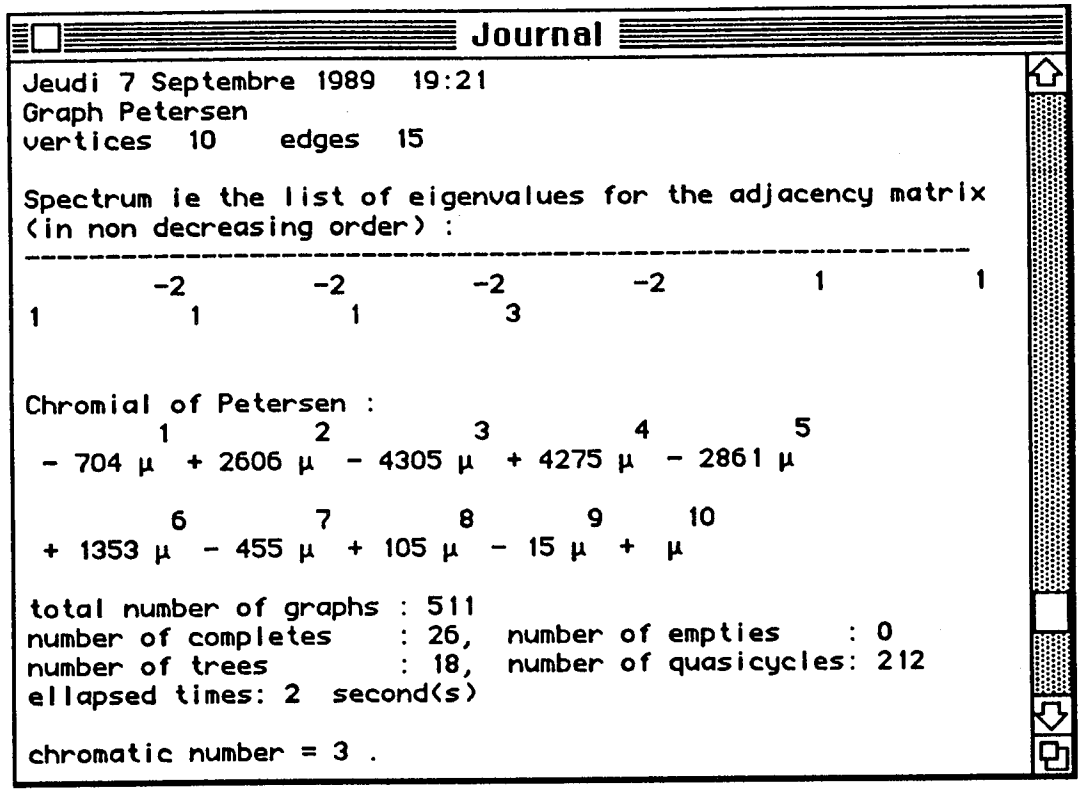
$\alpha_{min}(\overline{G}) \leq \chi(G) \leq \chi_{max}(G)$

La borne inférieure de $\alpha(G)$ utilisée par Cabri-graphes est obtenue par le calcul d'un stable maximal à l'aide d'un algorithme "glouton" (un sommet introduit dans le stable n'est jamais retiré). Cet algorithme glouton parcourt l'ensemble des sommets en respectant l'ordre sur les degrés. Un sommet est introduit dans le stable si et seulement si il n'est adjacent à aucun sommet déjà présent.

La borne supérieure de $\chi(G)$ utilisée par Cabri-graphes est obtenue par le calcul d'une coloration à l'aide d'un nombre minimal de couleurs. Cette coloration se fait également par un algorithme "glouton" (un sommet est colorié une fois pour toutes) considérant l'ensemble des sommets en respectant l'ordre sur les degrés. Un sommet est toujours colorié à l'aide de la plus petite couleur non utilisée par ses voisins déjà coloriés. Il existe des méthodes plus efficaces (obtenant en moyenne un encadrement plus précis) [Tur], mais elles n'ont pas été pour l'instant implémentées. Le lecteur trouvera en annexe, une étude sur les performances des algorithmes utilisés pour l'encadrement de $\alpha(G)$ et $\omega(G)$ par Cabri-graphes.

Le spectre d'un graphe n'est pas calculé par une algorithme exact, mais par une méthode convergent vers la solution (Méthode de Gauss). L'étude de la complexité d'une telle méthode relève de l'analyse numérique (figure 10).

Spectre, polynôme chromatique et nombre chromatique du graphe de Petersen :



- figure 10 -

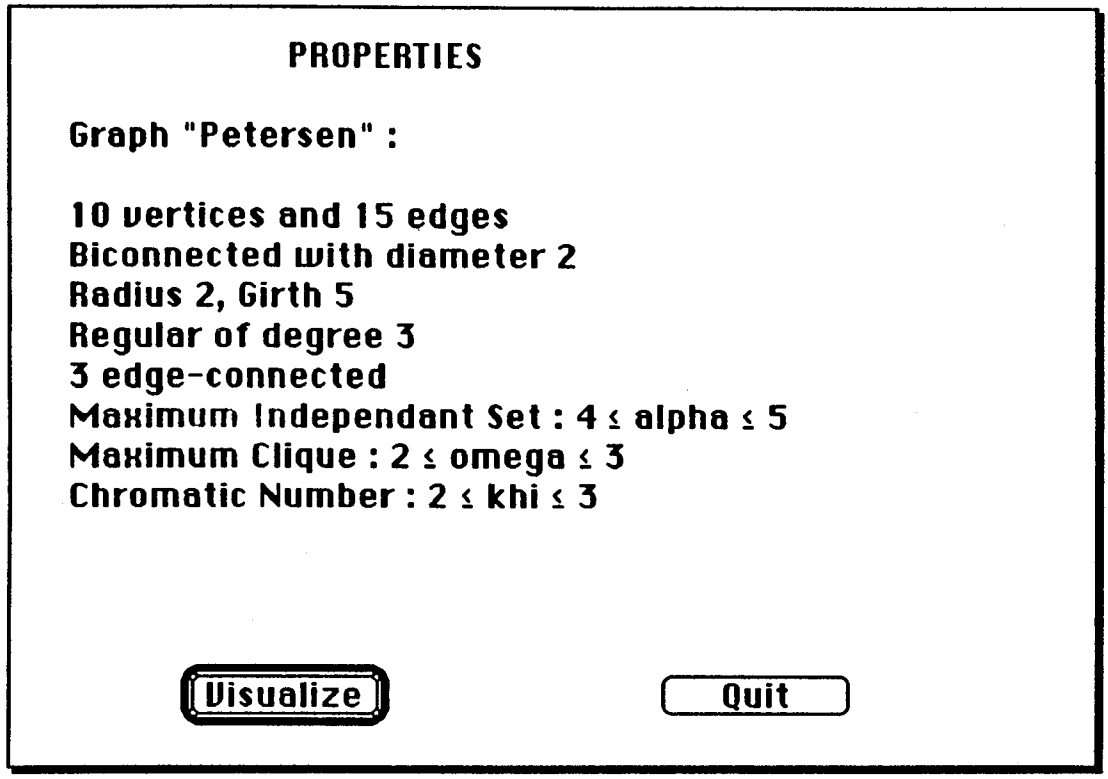
Parmi les invariants calculables à l'aide d'un algorithme polynômial, Cabri-graphes calcule ou vérifie les valeurs et propriétés suivantes (figure 11) :

- nombre de sommets, d'arêtes et de composantes connexes;
- degrés maximum et minimum;
- existence d'un cycle eulérien;
- existence d'un cycle;
- maille, diamètre et rayon;
- nombre de points d'articulation (ou 2-connexité);
- bipartition;
- arête-connectivité;
- planarité;
- 0-2 graphe;
- graphe median;

- intervalle-monotonie;
- distance-monotonie;
- sphéricité du graphe;
- valeurs de Möbius.

Les algorithmes utilisés dans Cabri-graphes pour calculer ces valeurs ou vérifier ces propriétés sont bien connus. Le lecteur trouvera aisément leur description, par exemple dans [Eve], [GM] ou [RND].

Quelques propriétés du graphe de Petersen :



- figure 11 -

Modification du dessin du graphe

Il faut distinguer deux types de dessins automatiques de graphes : les modifications purement géométriques du dessin, indépendantes de la structure du graphe, mais prenant en compte uniquement les coordonnées des sommets, et les modifications liées à la structure du graphe et à une ou plusieurs de ses propriétés.

Les modifications géométriques proposées sont :

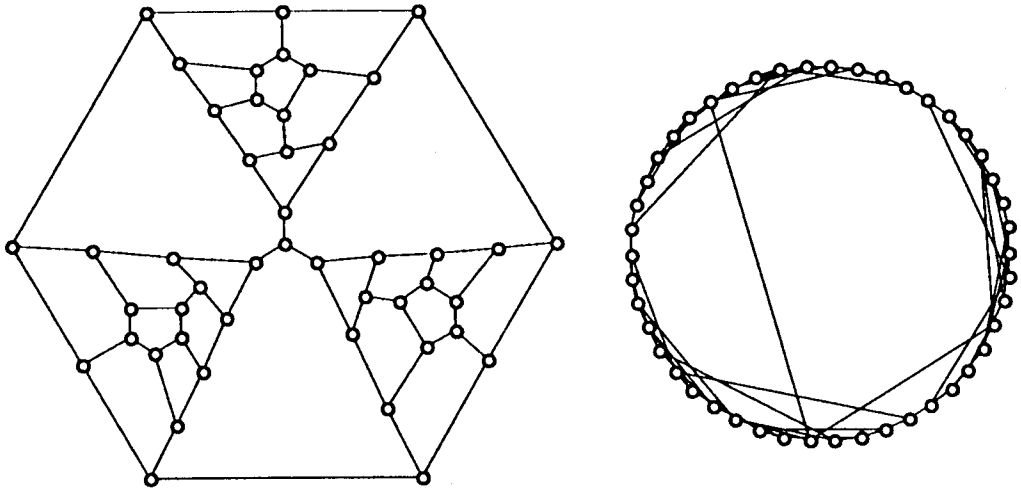
- symétries axiales par rapport aux droites verticale et horizontale passant par le barycentre des sommets;

- rotation par rapport au barycentre des sommets;

- homothétie par rapport au barycentre des sommets;

- projection sur le cercle centré au barycentre des sommets et minimisant la somme des distances entre la position initiale des sommets et leur position finale (figure 12).

Représentation usuelle du graphe de Tutte et projection sur un cercle de cette représentation :



- figure 12 -

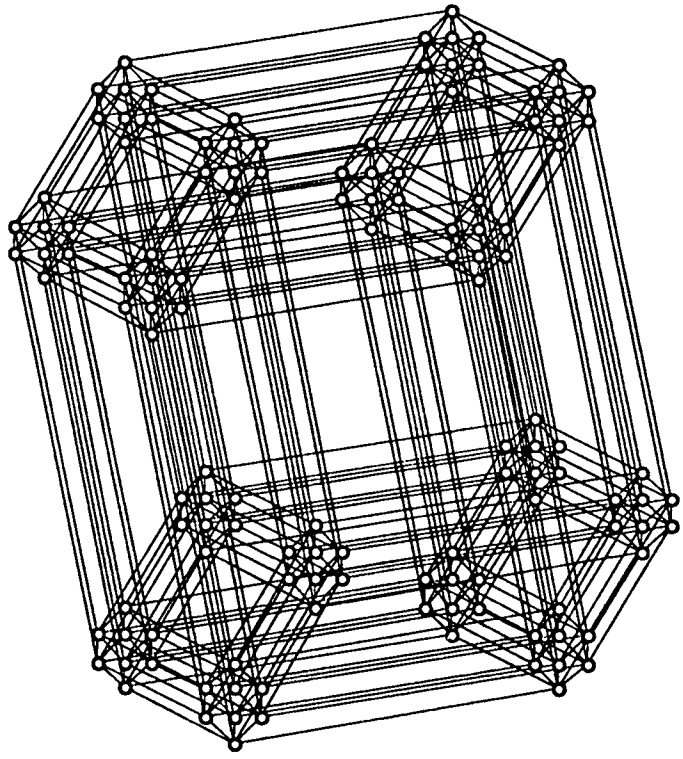
Cabri-graphes permet également l'utilisation d'une grille magnétique, l'écartement entre deux points étant paramétrable.

Les algorithmes de dessin automatique prenant en compte la structure sont généralement plus intéressants à utiliser. Malheureusement, beaucoup d'invariants liés au dessin du graphe ne sont pas polynômiaux. C'est le cas en particulier du nombre minimum de croisements d'arêtes dans une représentation plane du graphe.

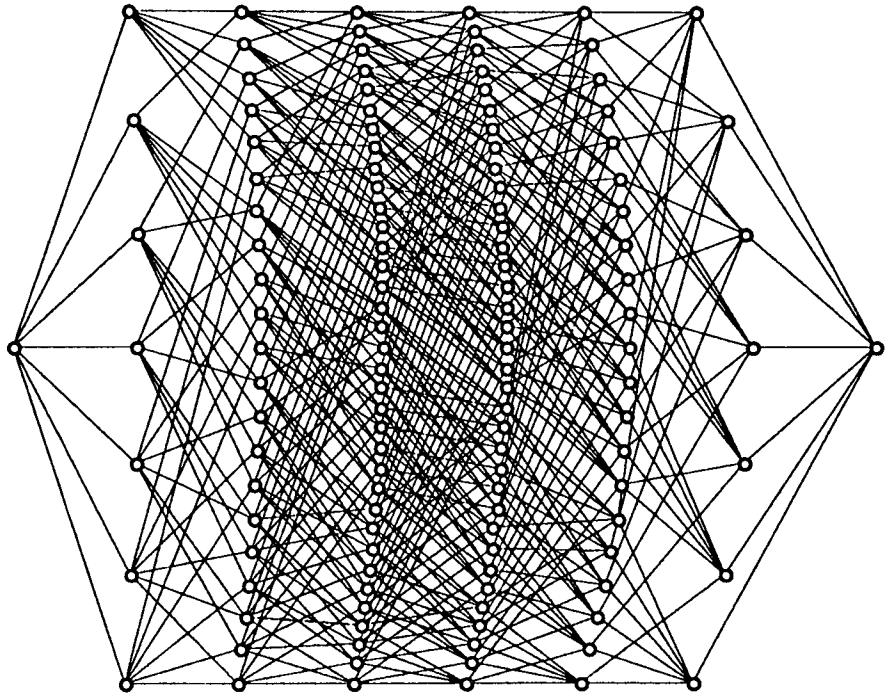
Nous proposons dans Cabri-graphes les dessins suivant :

- mise en couche par rapport à un sommet (figure 13);

Q_7, hypercube de dimension 7 :



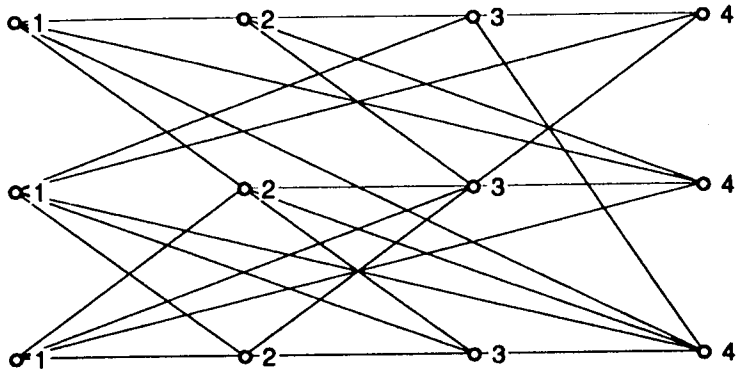
Mise en couche de Q_7 :



- figure 13 -

- décomposition en stables (figure 14);

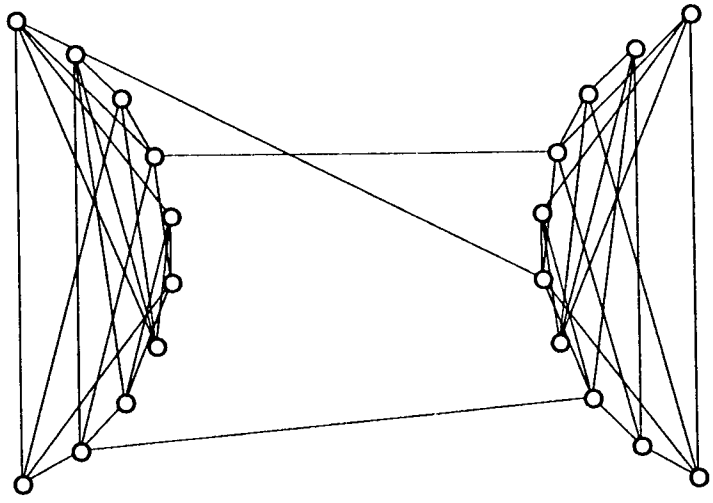
Décomposition en stables de l'icosaèdre :



- figure 14 -

- partition liée à un cocycle de cardinal minimum (figure 15);

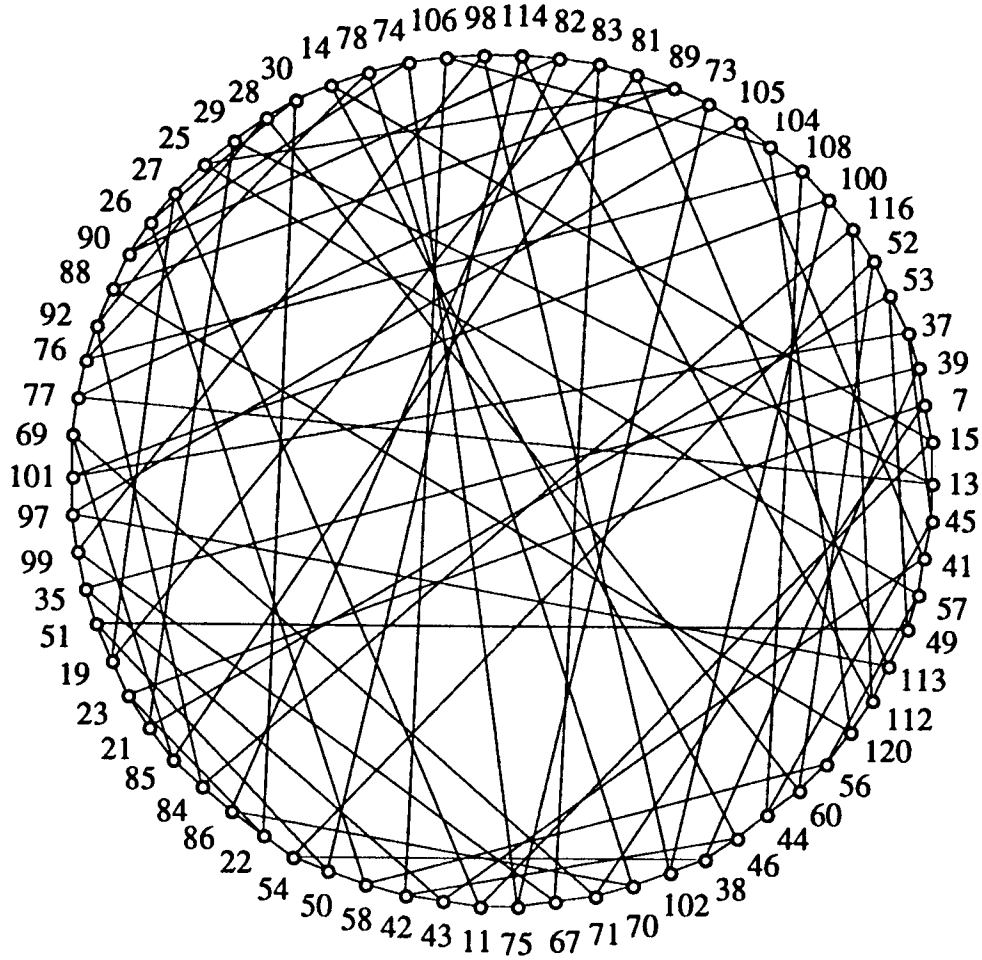
Partition d'un graphe liée à un cocycle de cardinal minimum :



- figure 15 -

- dessin lié à l'existence d'un cycle hamiltonien et à son obtention par l'heuristique 2-opt (figure 16);

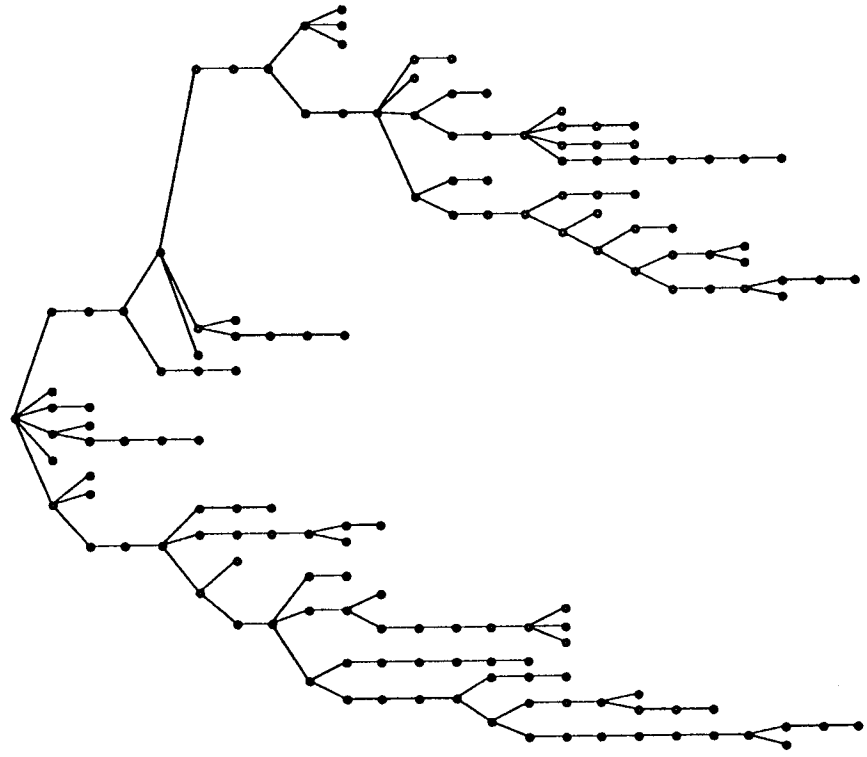
Cycle hamiltonien du graphe formé des couches centrales de Q_7, sous-graphe de Q_7 induit par les sommets dont le 7-uplet a 3 ou 4 composantes à 1. Les sommets sont étiquetés par la valeur en base 10 de leur mot binaire dans Q_7 :



- figure 16 -

- dessin spécifique d'une arborescence (figure 17);

Arborescence à 150 sommets créée par le générateur aléatoire et dessinée à l'aide de l'algorithme de dessin spécifique des arborescences :

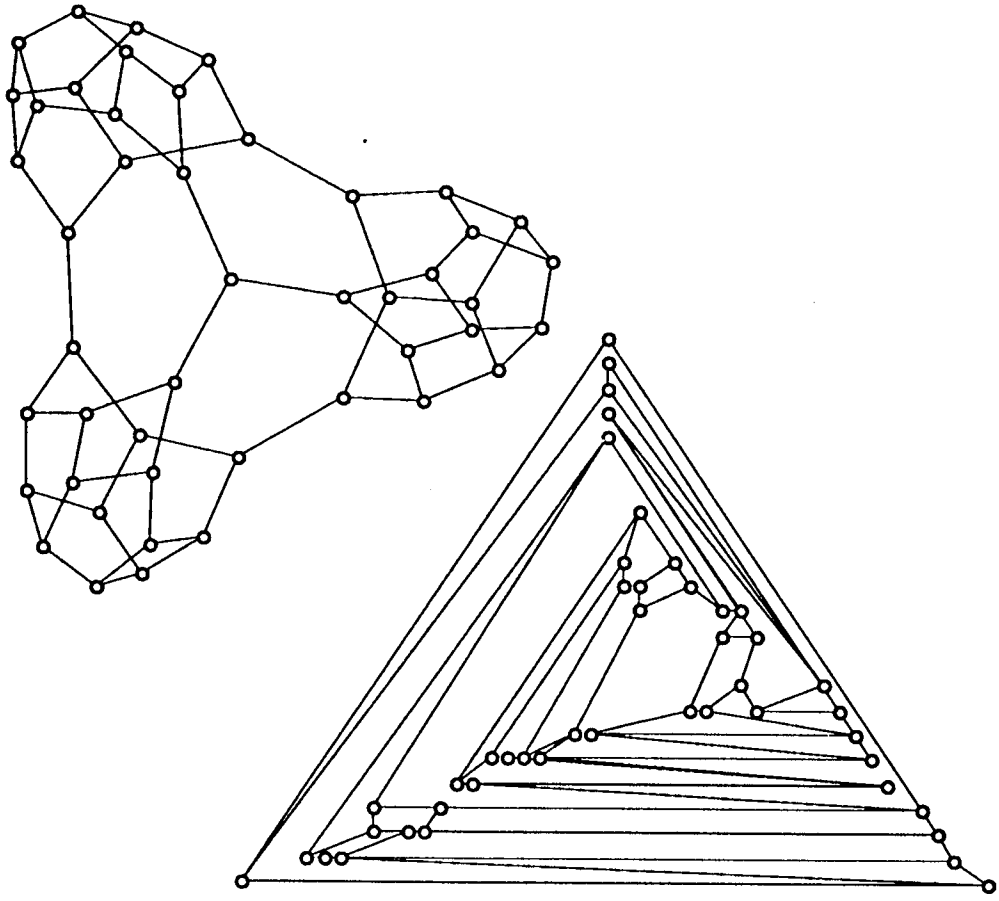


- figure 17 -

- dessin obtenu par l'heuristique d'Eades [Ead] (figures 18, 19 et 20);

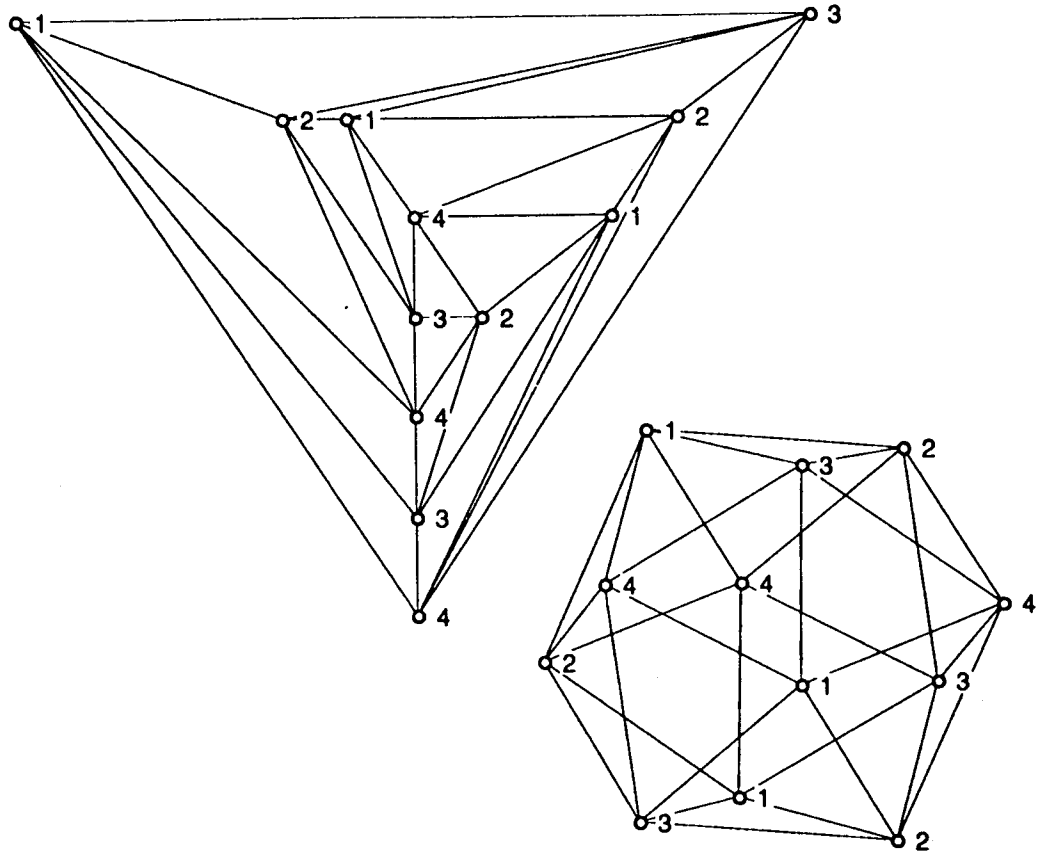
- dessin d'un graphe planaire sans croisement d'arêtes, obtenu par l'algorithme de de Fraysseix, Pach et Pollack [FPP] (figures 18 et 19).

Heuristique de Eades et algorithme de de Fraysseix et al. appliqués à la projection sur un cercle du graphe de Tutte (figure 12) :



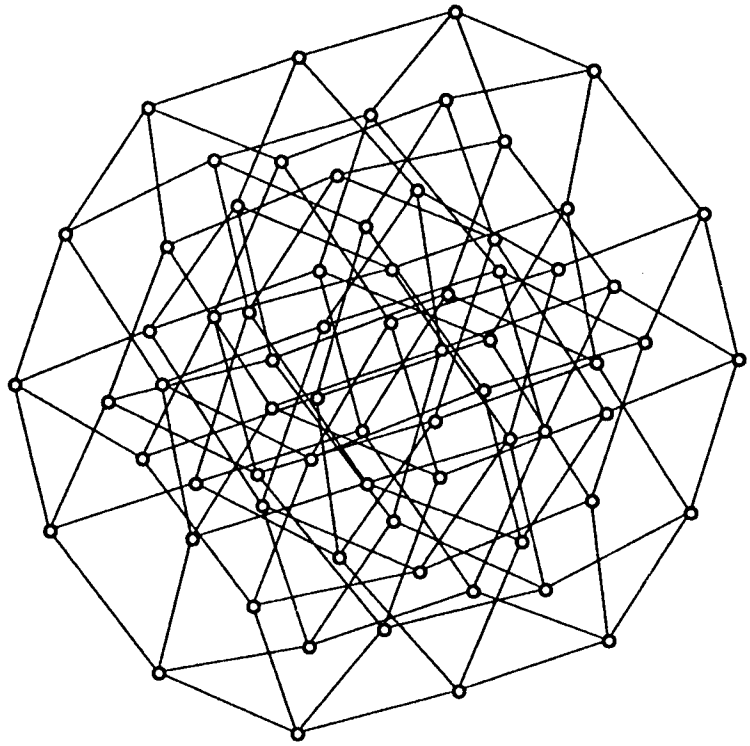
- figure 18 -

Algorithme de de Fraysseix et al. et heuristique de Eades appliqués à l'icosaèdre (figure 14) :



- figure 19 -

Application de l'heuristique de Eades aux couches centrales de Q_7, sous-graphe de Q_7 induit par les sommets dont le 7-uplet a 3 ou 4 composantes à 1 :



- figure 20 -

L'heuristique de Eades a pour effet de mettre en évidence les symétries d'un graphe. Elle utilise le principe suivant : on place entre toute paire de sommets un ressort exerçant une force d'attraction si la paire appartient à E et une force de répulsion sinon. L'heuristique consiste à chercher par itérations successives un état d'équilibre.

Il existe une conjecture, formulée par Eades à propos de cette heuristique : "si le graphe est planaire et qu'un état d'équilibre est trouvé, alors cet état correspond à une représentation sur la sphère sans croisement d'arêtes, projetée dans le plan". (figures 18 et 19).

L'algorithme de de Fraysseix et al. permet, à partir d'un plongement topologique, d'obtenir un dessin sans croisement d'arêtes d'un graphe planaire, où les arêtes sont représentées par des segments de droite et où les sommets sont

placés sur une grille de taille inférieure ou égale à (2n - 4) x (n - 2). Contrairement à l'heuristique de Eades, ce dessin est indépendant de la représentation initiale, mais dépend du choix d'une arête placée au sommet et sur toute la largeur de la grille.

Le dessin spécifique des arborescences est un dessin sans croisement d'arêtes. Les sommets sont répartis régulièrement sur les coordonnées horizontales en fonction de leur distance à la racine. Les feuilles sont réparties régulièrement sur les coordonnées verticales dans un ordre compatible avec la contrainte de non croisement des arêtes. La coordonnée verticale d'un sommet intérieur est la moyenne des coordonnées verticales de ses fils.

Les distances des sommets à la racine sont obtenues par une exploration en largeur (cf. [RND] et chapitre 3), l'ordre des feuilles compatible avec le non croisement d'arêtes par une exploration en profondeur (cf. [RND]). Le calcul des coordonnées verticales se fait à l'aide d'une file, initialisée à l'ensemble des feuilles dans l'ordre calculé par l'exploration en profondeur. Un nouveau sommet est introduit dans la file lorsque tous ses successeurs en sont sortis. L'algorithme de dessin spécifique des arborescences est donc linéaire.

Il est également possible de modifier le dessin d'un graphe en modifiant le mode de représentation d'un ou plusieurs sommets. Trois figures géométriques sont disponibles : le cercle, le carré et le triangle. La dimension d'un sommet est également paramétrable. Enfin, la dernière modification possible est celle de l'étiquetage (figure 21).

Modification de l'étiquetage

Il nous faut tout d'abord redéfinir la notion d'étiquetage. En effet, dans la définition donnée au paragraphe 1.3, l'étiquetage doit être différent en chaque sommet. Le sens donné à l'étiquetage dans Cabri-graphes est plus large puisque

cette propriété n'est généralement pas vérifiée, sauf dans certains cas qui seront précisés.

Il existe dans Cabri-graphes deux types d'étiquetages : numérique et alpha-numérique. Ces deux étiquetages ne peuvent être visualisés en même temps. Par contre, l'étiquetage numérique est conservé lorsque l'on passe en mode alpha-numérique et inversement. Enfin, il est possible de visualiser le graphe sans aucun étiquetage.

L'étiquetage de départ est numérique, chronologique, le premier sommet créé étant numéroté 0. Sauf cas particuliers, un sommet nouvellement créé sera numéroté par le plus petit entier non encore utilisé. Il existe donc des "trous" dans la numérotation si l'on a supprimé des sommets au cours de la création du graphe.

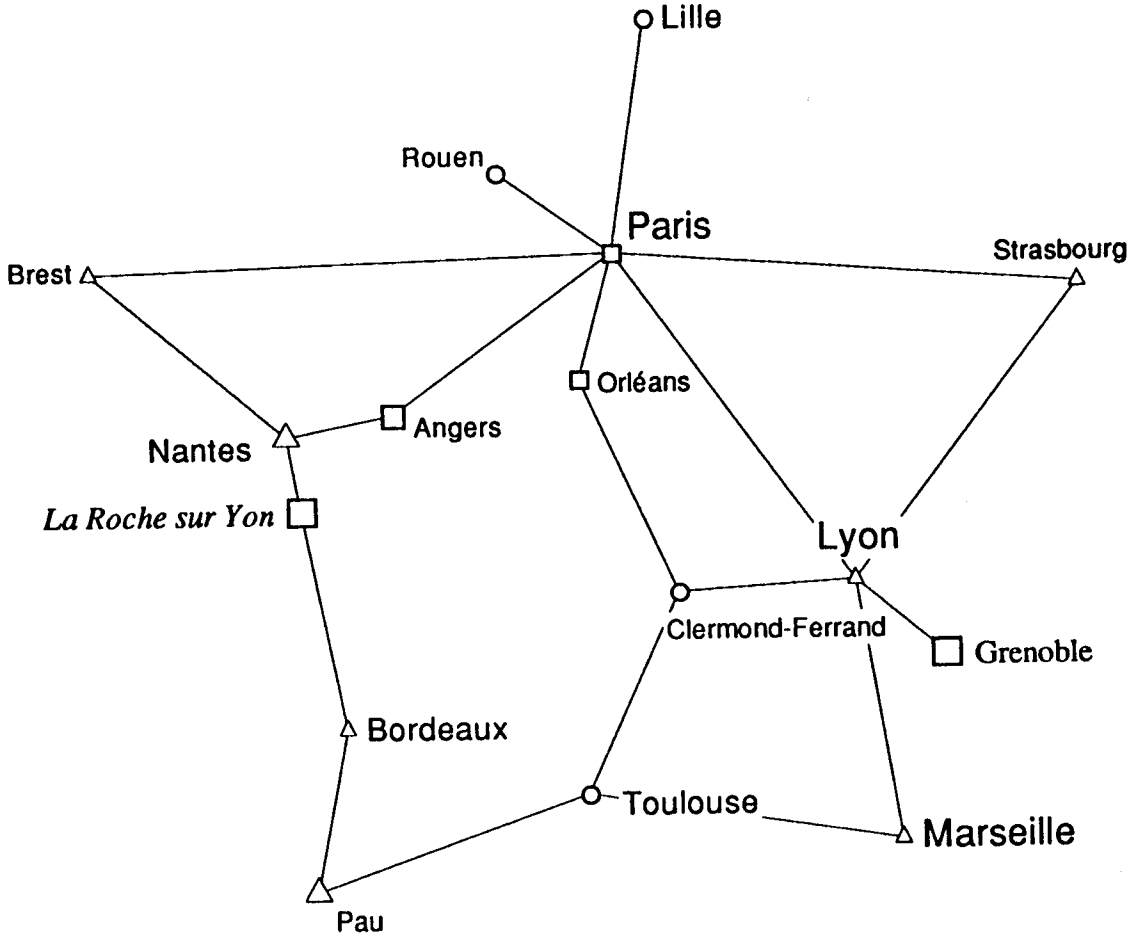
La fonction **numérotation chronologique** renumérote les sommets entre 0 et n-1 en leur affectant leur ordre de création parmi les sommets existant. Les sommets créés après l'appel à cette fonction seront numérotés à partir de n. La fonction **degré ascendant** numérote les sommets entre 0 et n-1 en respectant l'ordre sur les degrés des sommets. La fonction **degré descendant** numérote les sommets entre 0 et n-1, l'ordre sur les degrés étant inverse à cette numérotation. La fonction "numérotation chronologique" ne porte que sur les sommets de la sélection si celle-ci est non vide, "degré ascendant" et "degré descendant" portant sur l'ensemble du graphe quelque soit l'état de la sélection. Ces trois fonctions garantissent une étiquette différente en chaque sommet du graphe ou du sous-graphe induit auquel elles sont appliquées.

Les autres étiquetages numériques disponibles sont des numérotations à l'aide des fonctions degré, excentricité, de Möbius, distance par rapport à un sommet et uniforme. Dans les cas des fonctions de Möbius et distance par rapport

à un sommet, l'utilisateur devra préciser le sommet de base et pour la numérotation uniforme la valeur de la constante.

Quelque soit le mode d'étiquetage, il est possible de modifier la police, la taille et le style des caractères utilisés, ainsi que la position de l'étiquette par rapport au sommet (figure 21).

Exemple d'utilisation des différentes possibilités de représentation de sommets et d'étiquettes :



- figure 21 -

1.7 Les opérations secondaires de Grafo, Unicorn et VGMP

Le choix des opérations secondaires

Ainsi que nous l'avons dit au paragraphe 1.4, ce choix est lié aux objectifs des concepteurs du logiciel. On peut cependant dans tous les éditeurs classer ces opérations dans l'une des catégories suivantes : génération (aléatoire ou non) de graphes, aide à la construction de graphes, calcul de propriétés, aide au dessin. Après la présentation détaillée des opérations secondaires de Cabri-graphes, voici en quelques mots résumées les options des autres éditeurs étudiés :

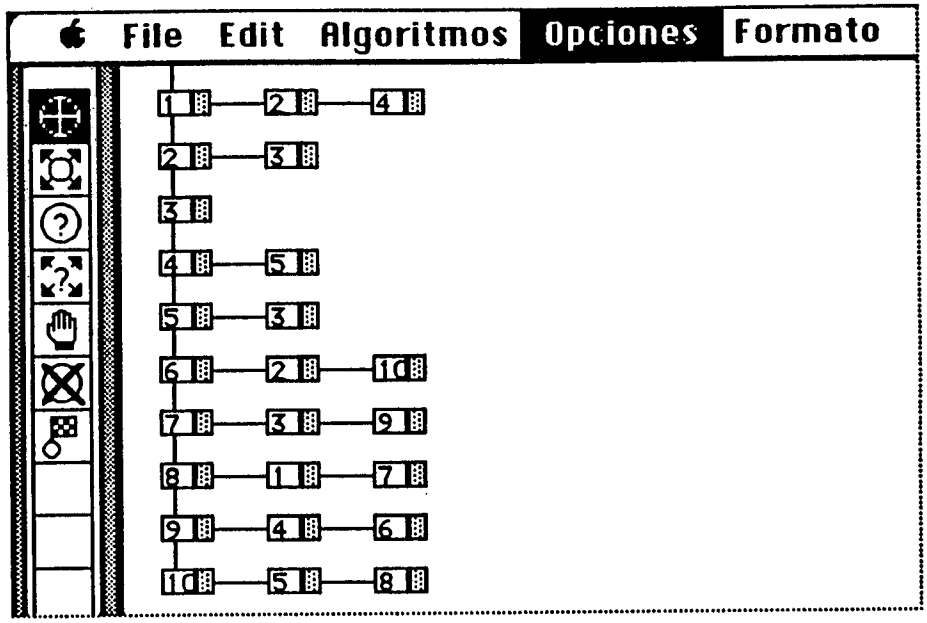
Les opérations secondaires de Grafo

La version de Grafo dont nous disposons est relativement ancienne. Il n'y donc que peu de fonctions implémentées.

Notons cependant que les fonctions présentes sont surtout axées sur l'étude algorithmique des graphes (pour les définitions, voir [RND]) : parcours en profondeur et tri topologique pour les graphes orientés, composantes connexes, arbre couvrant de poids minimum (Prim), coloration gloutonne pour les graphes non orientés. Le déroulement des algorithmes est visualisé et leur vitesse d'exécution paramétrable. Le premier sommet considéré par l'algorithme est à préciser par l'utilisateur.

Pour le dessin, la seule aide est donnée par la présence d'un grille, non magnétique. Enfin, il est possible de visualiser le graphe comme un ensemble de listes chaînées donnant la liste des successeurs en chaque sommet (figure 22).

Représentation d'un graphe par les listes des successeurs dans Grafo :



- figure 22 -

Les opérations secondaires d'Unicorn

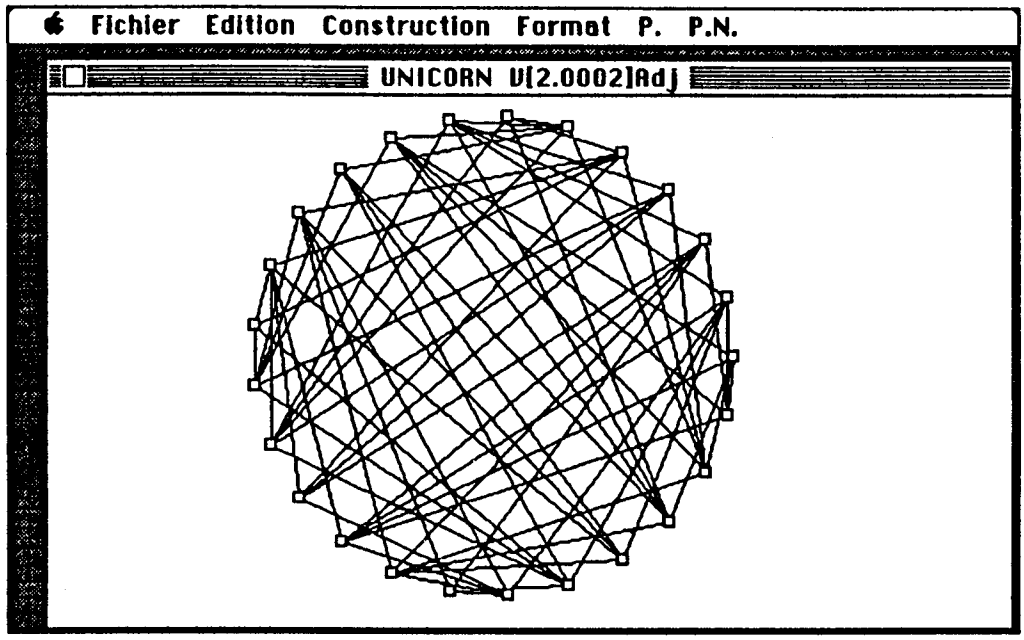
Unicorn se caractérise par ses nombreuses fonctionnalités liées à l'étude de réseaux d'interconnexion.

En particulier, il propose des générateurs de graphes servant traditionnellement de modèles de réseaux d'interconnexion (pour les définitions, voir [Foul]) : hypercube généralisé, graphes de Kautz et de De Bruijn généralisés (figure 23), graphes de cordes généralisés, produits de graphes. Les autres constructions possibles sont les graphes de Petersen généralisés et les graphes représentatifs des arêtes.

Unicorn permet le calcul des principaux invariants intervenant dans les caractéristiques d'un réseau : nombre de sommets et d'arêtes, degrés maximum et minimum, diamètre, rayon, moyenne et variance des distances, vulnérabilité, vulnérabilité moyenne, arête-vulnérabilité. Sont également calculables les

propriétés suivantes : bipartition, existence d'un cycle de longueur k (k étant fixé par l'utilisateur), pancyclisme, hamiltonicité, cardinal maximum d'un stable.

Graphe de Kautz généralisé dans Unicorn :



- figure 23 -

Les calculs sont effectués sur le sous-graphe induit par la sélection. Dans certains cas, l'affichage des résultats s'accompagnent d'une visualisation (arête-vulnérabilité, cycle de longueur donné) ou d'une modification de la sélection (centres, stable maximum, sommets de degré maximum et minimum, vulnérabilité) justifiant la réponse. Il est également possible de visualiser le plus court chemin entre deux sommets et d'affecter à la sélection le voisinage ou le voisinage étendu d'une sélection initiale.

Unicorn propose des fonctions d'aide au dessin, opérant sur le sous-graphe induit : mise en couche, mise en cercle, heuristique d'Eades [Ead], alignement sur une grille, une droite horizontale ou verticale, symétrie axiale par rapport à une droite horizontale ou verticale, rotation et homothétie par rapport au barycentre des sommets du graphe avec un facteur fixé par l'utilisateur.

Les opérations secondaires de VGMP

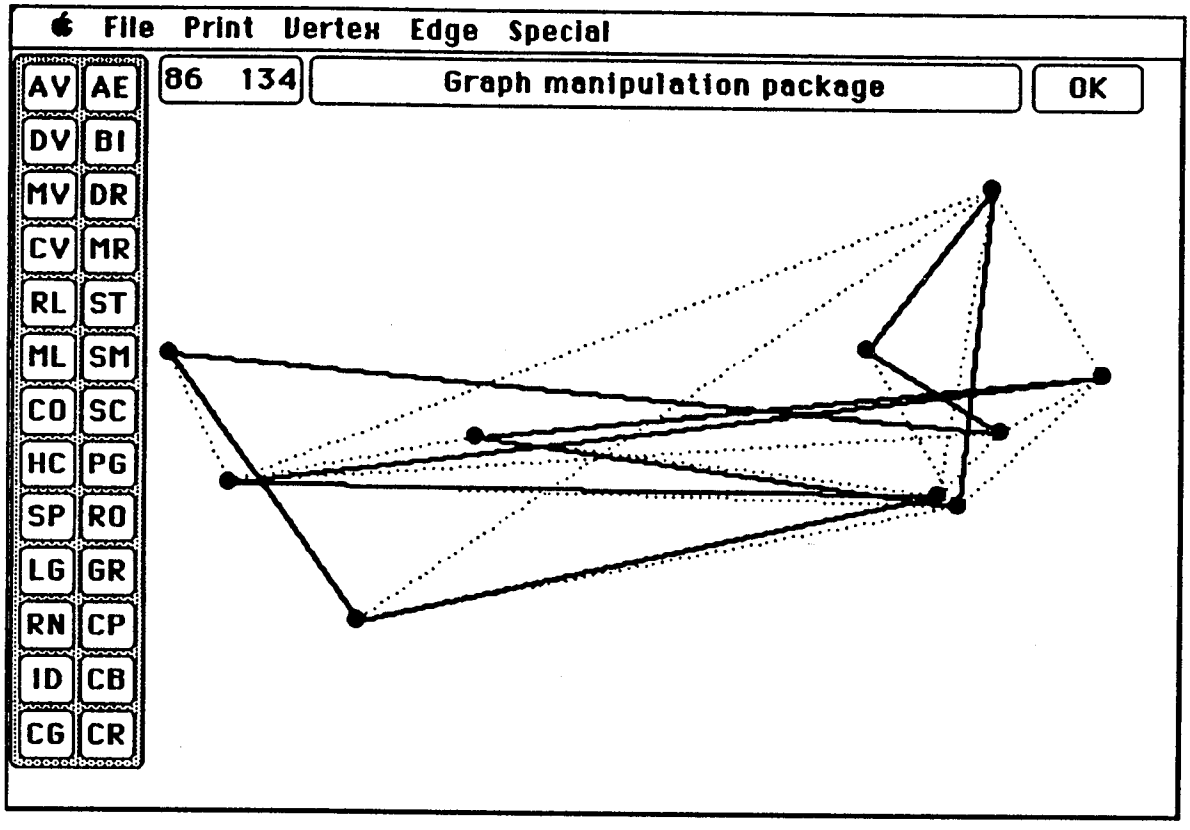
Il existe dans VGMP quelques fonctions de transformation d'un graphe : calcul du graphe représentatif des arêtes, destruction répétitive des sommets de degré 0 et 1, destruction répétitive des sommets de degré 0, 1 et 2, les sommets de degré 2 étant remplacés par une arête si leurs voisins n'étaient pas déjà adjacents. On peut également rajouter toutes les arêtes possibles entre deux ensembles de sommets, chacun de ces ensembles étant inscrit dans un rectangle.

Il est possible de générer un graphe aléatoirement, le nombre de sommets et la probabilité d'exister de chaque arête étant fixés par l'utilisateur par l'intermédiaire d'un dialogue.

VGMP propose plusieurs représentations de sommets : cercle, carré, invisible, rond, ... et d'arêtes : pleine ou pointillée, fine ou épaisse, ... (figure 24). Le graphe peut être étiqueté, les sommets étant alors numérotés dans l'ordre de leur création. La position de l'étiquetage par rapport à un sommet peut être modifiée.

VGMP permet de tester l'existence d'un cycle hamiltonien, et sa visualisation s'il existe par changement du mode de représentation de ses arêtes (figure 24). On peut par la même méthode visualiser un arbre couvrant dans chaque composante connexe.

Visualisation d'un cycle hamiltonien dans VGMP. Le graphe a été généré aléatoirement, ses arêtes étant représentées par des lignes pointillées :



- figure 24 -

1.8 Quelques aspects de l'interface dans Cabri-graphes

Pour une interface de qualité

Maints ouvrages ont été écrits pour définir ce que doit être une interface de qualité. Rappelons simplement quelques points particuliers :

1) le principe essentiel est bien entendu celui résumé par les lettres WYSIWYG (What you see is what you get). A cette règle de base, nous en ajouterons deux qui ont particulièrement guidées notre travail :

2) il faut respecter les a priori de l'utilisateur. Celui-ci doit pouvoir deviner comment exécuter une fonction, à partir de sa connaissance des logiciels fonctionnant sur le même environnement matériel/système d'exploitation d'une part, et des autres fonctions du logiciel d'autre part. Rappelons à ce propos que la gamme Macintosh a pour particularité d'avoir été la première à disposer d'un guide de programmation de l'interface [HEA]. Les logiciels graphiques MacPaint et MacDraw nous ont été également souvent servis de références.

3) Enfin, il faut une ergonomie adaptée à la fréquence d'utilisation de chaque fonctionnalité. Ainsi, nous avons particulièrement soigné celle des opérations élémentaires.

L'expérimentation nous semble absolument nécessaire pour permettre de définir correctement une interface. Le respect des a priori impose d'autre part que cette expérience est propre à chaque machine et système d'exploitation utilisés. C'est pourquoi il faut replacer les solutions proposées ici dans le cadre particulier du Macintosh.

Quelques aspects de l'interface sur le Macintosh

Rappelons que le Macintosh dispose d'une **souris munie d'un bouton unique** et d'un clavier comportant des touches **modificatrices.** Ces touches, au nombre de trois ou quatre selon les modèles de clavier, ne créent pas d'événement, mais leur état (enfoncée ou relâchée) peut être testé à tout moment. Cabri-graphes n'utilise que les touches modificatrices présentes sur tous les claviers : la touche **Commande**, représentée par les symboles ⌘ ou ♁, la touche **Option** représentée par le symbole ⌥ et la touche **Majuscule** représentée par le symbole ⇧.

Les logiciels fonctionnant sur le Macintosh et respectant les conventions d'interfaçage de cette machine utilisent tous des **menus déroulants.** Afin d'améliorer l'ergonomie des logiciels, il est possible d'utiliser des **raccourcis clavier**, c'est à dire d'appeler une fonction présente dans un menu en appuyant conjointement sur la touche Commande et une touche caractère. Le caractère à utiliser est précisé dans le menu correspondant (figure 32). Si un raccourci souhaité n'est pas prévu par le concepteur du logiciel, il est très aisé de le rajouter, cette opération ne nécessitant pas d'intervention sur les fichiers sources du logiciel.

Interface des opérations élémentaires

La plupart des opérations élémentaires ont une interface par **manipulation directe.** Cela signifie que leur appel se fait uniquement à l'aide de la souris et éventuellement d'une ou plusieurs touches modificatrices, sans passer par l'intermédiaire d'une palette, d'un dialogue ou d'un menu.

Parmi les autres fonctions élémentaires, l'appel des fonctions ci-dessous se fait par l'intermédiaire des menus déroulants, avec éventuellement des raccourcis claviers :

- complémenter la sélection par rapport à l'ensemble des sommets;
- sélectionner tous les sommets (⌘A);
- compléter le sous-graphe induit / le graphe (⌘K);
- supprimer les arêtes du sous-graphe induit / du graphe (⌘D);
- complémenter le sous-graphe-induit / le graphe;
- fusionner les sommets de la sélection.

Les opérations détruire le sous-graphe induit / le graphe / le sous-graphe arête-induit se font à l'aide de la touche **effacer** ou de la touche **retour-arrière**, combinées avec la touche Option pour le sous-graphe arête-induit.

Les curseurs

Le choix de l'opération exécutée par manipulation directe dépend à la fois de la position du curseur par rapport aux objets intrinsèques de la fenêtre active, de l'état des sélections, et de l'état des touches modificatrices. Afin de signaler à l'utilisateur, au moins partiellement, quelle va-t-être l'opération exécutée, Cabri-graphes modifie l'icône du curseur :

+ : création d'un sommet;

 : création d'une arête;

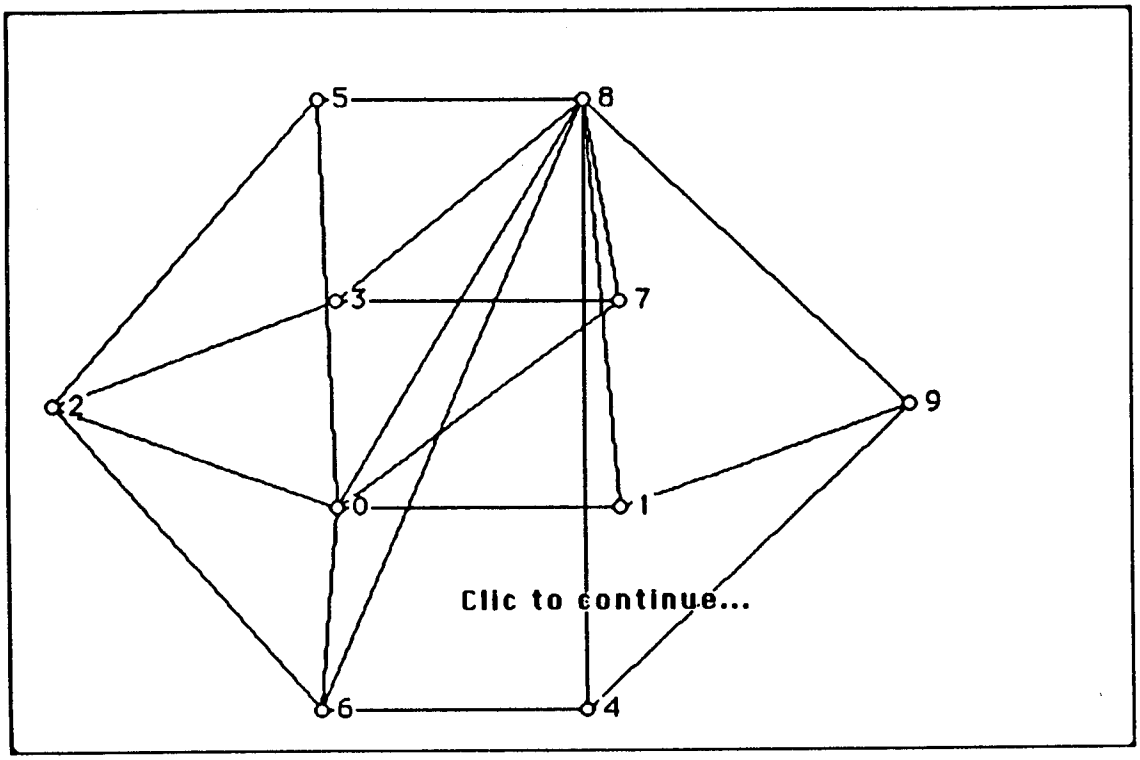
 : action sur une sélection (sommets ou arêtes);

 : insertion d'un caractère dans une étiquette alpha-numérique;

 : déplacement d'une étiquette ou de la lucarne;

Lorsque le curseur a la forme d'une montre  , cela signifie que l'utilisateur doit patienter. Le curseur peut également se présenter sous la forme d'une chaîne de caractère précisant à l'utilisateur ce qu'il doit faire (figure 25).

Exemple de curseur chaîne :



- figure 25 -

Avant de présenter une analyse par cas de l'ensemble de l'interface par manipulation directe, nous donnons quelques exemples d'opérations élémentaires effectuées par manipulation directe en spécifiant leur interface.

Création d'un sommet ou d'une sélection de sommets par manipulation directe

Le curseur, sous forme de croix, doit se trouver éloigné de tout autre sommet et les touches modificatrices toutes relevées (figure 26.1). De plus aucun sommet ne doit être sélectionné. Dans ce cas, si l'on appuie sur le bouton de la souris, un cercle représentant le nouveau sommet apparaît, ainsi qu'une mire (figure 26.2).

Si l'on déplace le curseur autour du point où l'on a appuyé, le nouveau sommet est également déplacé et la mire permet de le positionner plus facilement

par rapport aux autres sommets déjà existants. Le sommet sera définitivement créé lors du relâchement du bouton.

Si le curseur est déplacé trop fortement avant le relâchement du bouton, Cabri-graphe considère que l'on veut sélectionner des sommets et non effectuer une création (figure 26.3). Cela se matérialise par l'apparition d'un rectangle en pointillé. Les sommets compris à l'intérieur de ce rectangle lors du relâchement de la souris formeront la nouvelle sélection et apparaîtront alors en noir.

Lorsque le curseur se trouve à proximité d'un sommet, il se transforme en flèche, indiquant par là que le fait d'appuyer sur le bouton de la souris aura pour effet de sélectionner ou d'amorcer un déplacement de ce sommet (figure 26.4). Si l'on relâche le bouton de la souris sans avoir entre temps déplacé celle-ci, le sommet est sélectionné et apparaît alors en noir. Si la touche Majuscule n'était pas enfoncée lors de l'enfoncement du bouton, les autres sommets qui pouvaient appartenir à la sélection sont désélectionnés. Sinon le nouveau sommet sélectionné est simplement rajouté à la sélection.

Déplacement d'un sommet

Si après avoir appuyé sur le bouton de la souris, on déplace celle-ci, une mire apparaît, signifiant que l'on déplace le sommet et permettant un nouveau positionnement de celui-ci plus facile. Tout au long du déplacement, le sommet est constamment redessiné, ainsi que ses arêtes incidentes, ce qui constitue l'un des aspects les plus intéressants du logiciel. Cette idée a été reprise dans le cadre d'un logiciel de géométrie élémentaire, Cabri-Géomètre [Baul], où son intérêt apparaît de façon encore plus spectaculaire.

La manipulation de la sélection de sommets dans Cabri-graphes suit en tout point celle de la manipulation de fichiers et dossiers du bureau électronique du

Macintosh, sauf sur un point : un sommet déplacé ne reste pas sélectionné, sauf s'il l'était déjà avant le déplacement.

Création d'une arête

Si la touche Option est enfoncée, Cabri-graphes considère que l'on cherche à créer ou manipuler des arêtes.

Si le curseur est éloigné de toute arête déjà existante, il se présente alors sous la forme d'un crayon.

Si de plus il se trouve éloigné de tout sommet, le fait d'appuyer sur le bouton de la souris a pour effet de créer la première extrémité d'une nouvelle arête. Si un sommet se trouve à proximité du curseur, appuyer sur le bouton de la souris aura pour effet de considérer ce sommet comme première extrémité d'une nouvelle arête.

Lors du déplacement de la souris, le bouton restant appuyé, la nouvelle arête apparaît et sa deuxième extrémité suit le déplacement du curseur (figure 26.6), sauf dans le cas ou celui-ci se rapproche d'un sommet existant (figure 26.7). La deuxième extrémité de l'arête est alors "attirée" par ce sommet. Cela signifie que si l'utilisateur relâche la souris dans cette situation, ce sommet déjà existant formera la deuxième extrémité de la nouvelle arête. Sinon un nouveau sommet sera créé.

Si la première et la deuxième extrémité de l'arête sont confondues et si la première extrémité était un sommet non encore existant, cette opération n'a pour effet que de créer un nouveau sommet. Si la première et la deuxième extrémité de l'arête sont confondues et que la première extrémité était un sommet déjà existant, cette opération n'a aucun effet.

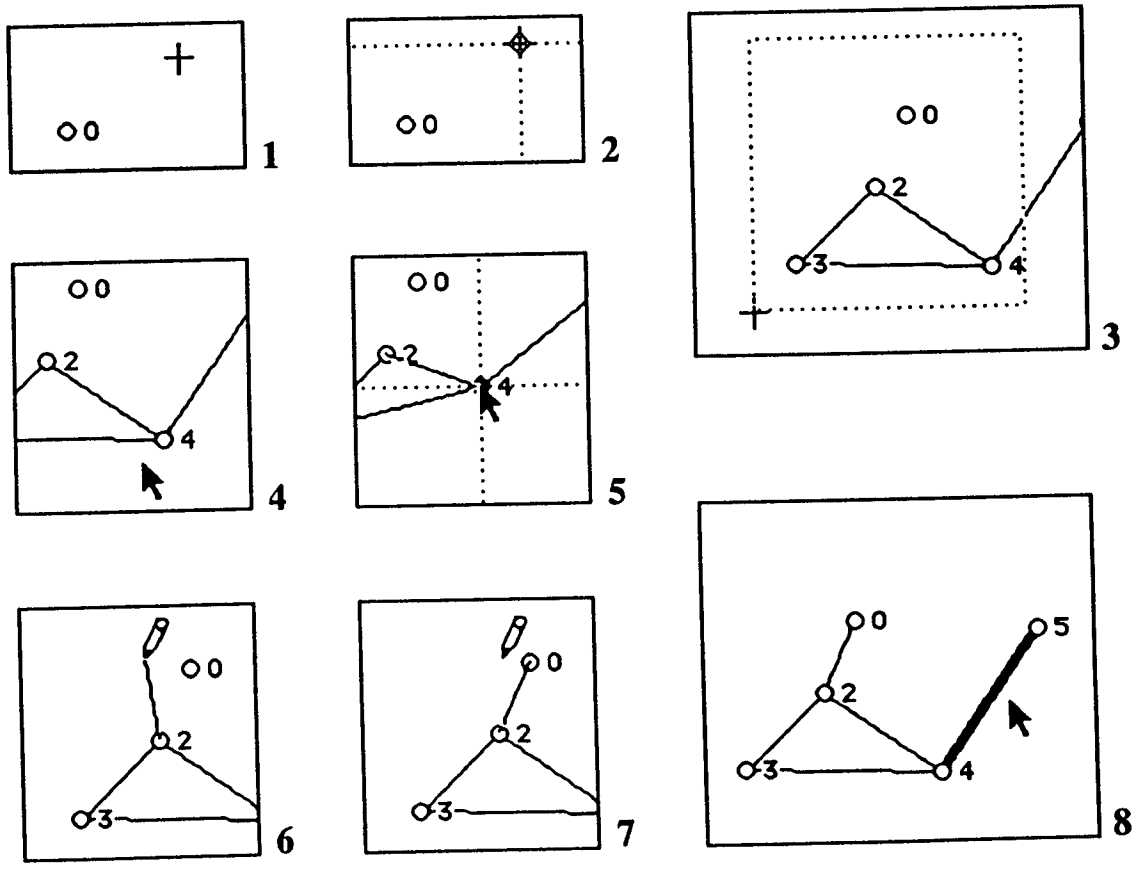
Sélection d'une arête

Si la touche Option est enfoncée et le curseur près d'une arête, ce dernier apparaît sous la forme d'une flèche, signifiant par là que l'enfoncement du bouton de la souris aura dans ce cas une incidence sur l'appartenance de l'arête à la sélection d'arêtes. Deux cas se présentent :

Si la touche Majuscule n'est pas enfoncée, l'arête devient la seule arête sélectionnée, quelque soit l'état précédent de la sélection d'arêtes.

Si la touche Majuscule est enfoncée, l'arête devient sélectionnée si elle ne l'était pas encore, ou au contraire désélectionnée si elle appartenait à la sélection. Le reste de la sélection n'est pas modifié.

Exemples de manipulation directe :



- figure 26 -

Analyse par cas de l'interface par manipulation directe

Les tableaux suivants présentent une analyse par cas de l'interface par manipulation directe. Cette analyse ne correspond pas exactement à la version actuelle du logiciel, mais à une version à venir, où l'interface devrait être à la fois plus homogène et plus complète.

Cinq paramètres décident de l'opération à effectuer :

1) l'action effectuée sur la souris (manipulation du bouton et déplacement). Cette action est décrite dans la colonne **Action sur la souris;**

2) la position du curseur par rapport aux objets intrinsèques de la fenêtre active (le curseur est toujours considéré comme étant dans la fenêtre active). Cette position est décrite dans la colonne **Objet;**

3) l'état de la sélection de sommets. Cet état est décrit par la colonne **S;**

4) l'état de la sélection d'arêtes. Cet état est décrit par la colonne **A;**

5) l'état des touches modificatrices. Cet état est décrit dans la colonne **Touches;**

La colonne **Curseur** donne les différents états du curseur pendant la durée de l'opération.

Les effets de l'opération exécutée sont décrits dans la colonne **Effet.**

Exemple : Action sur la souris = Enfoncer et relâcher au point p;
Objet = aucun;
S = ∅;
A = non précisée;
Touches = Aucune;
Curseur = + ;
Effet = Créer un sommet au point p;

Interprétation : "L'action sur la souris consiste à enfoncer, puis relâcher le bouton de la souris sans la déplacer, alors que le curseur se situe au point p de la fenêtre active. Aucun sommet n'est situé au point p et aucune arête n'y passe. La sélection de sommets est vide, celle des arêtes dans un état quelconque. Aucune touche modificatrice n'est enfoncée. Le curseur doit pendant l'ensemble de l'action avoir la forme d'une croix, et l'effet de cette opération est la création d'un sommet au point p."

Rect(p, p') désigne l'ensemble des sommets ou des arêtes, selon le contexte, compris à l'intérieur du plus petit rectangle contenant p et p'.

Afin de ne pas alourdir le tableau, déjà fort long, nous nous sommes contentés de référencer les actions possibles sur un graphe valué (étiquetage numérique) et de rajouter en fin de tableau les principales actions spécifiques à l'étiquetage alpha-numérique.

Action sur la souris	Objet	S	A	Touches	Curseur	Effet
Enfoncer et relâcher au point p	Aucun	$\varnothing$		Aucune	+	Crée un sommet au point p
Enfoncer et relâcher au point p	Aucun			⇧	+	Crée un sommet v au point p $S \leftarrow S \cup \{v\}$
Enfoncer et relâcher au point p	Pas de sommet	$\neq \varnothing$		Aucune	↖	$S \leftarrow \varnothing$
Enfoncer et relâcher au point p	Sommet v	$v \notin S$		Aucune	↖	$S \leftarrow \{v\}$
Enfoncer et relâcher au point p	Sommet v			⇧	↖	$S \leftarrow S \Delta \{v\}$
Enfoncer et relâcher rapidement au point p	Arête a	$\varnothing$		Aucune	+	Crée un sommet au point p
Enfoncer et relâcher rapidement au point p	Arête a			⇧	+	Crée un sommet v au point p $S \leftarrow S \cup \{v\}$
Enfoncer et relâcher au point p après un bip sonore	Arête a	$\varnothing$		Aucune	+	Subdivise l'arête a au point p

Action sur la souris	Objet	S	A	Touches	Curseur	Effet
Enfoncer et relâcher au point p	Arête a			⇧ ⌥		$A \leftarrow A \Delta \{a\}$
Enfoncer au point p, déplacer et relâcher au point p'	Pas de sommet			Aucune Pas ⌥	+	$S \leftarrow Rect(p,p')$
Enfoncer au point p, déplacer et relâcher au point p'	Pas de sommet			⇧ Pas ⌥	+	$S \leftarrow S \Delta Rect(p,p')$
Enfoncer au point p, déplacer et relâcher au point p'	Sommet v Pas de sommet	$v \notin S$		Aucune		$S \leftarrow \varnothing$ Déplace le sommet v de p en p'
Enfoncer au point p, déplacer et relâcher au point p'	Sommet v Pas de sommet	$v \in S$		Aucune		Effectue sur les éléments de S une translation de vecteur $\overrightarrow{pp'}$
Enfoncer au point p, déplacer et relâcher au point p'	Sommet v Pas de sommet			⇧		Déplace le sommet v de p en p'

Action sur la souris	Objet	S	A	Touches	Curseur	Effet
Enfoncer au point p, déplacer et relâcher au point p'	Sommet v Sommet v'	$v \notin S$		Aucune Pas⌥		$S \leftarrow \varnothing$ Fusionne v et v'. La fusion est étiquetée comme v
Enfoncer au point p, déplacer et relâcher au point p'	Sommet v Sommet v'	$v \notin S$		Aucune ⌥		$S \leftarrow \varnothing$ Fusionne v et v'. La fusion est étiquetée comme v'
Enfoncer au point p, déplacer et relâcher au point p'	Sommet v Sommet v'			⇧ Pas⌥		Fusionne v et v'. La fusion est étiquetée comme v
Enfoncer au point p, déplacer et relâcher au point p'	Sommet v Sommet v'			⇧ ⌥		Fusionne v et v'. La fusion est étiquetée comme v'
Enfoncer au point p, déplacer et relâcher au point p'	Pas d'arête		$\varnothing$	⌥		Crée une arête d'extrémités en p et p' S'il n'existait pas de sommets en p ou p', ces sommets sont créés
Enfoncer au point p, déplacer et relâcher au point p'	Pas de sommet			Aucune ⌥	+	$A \leftarrow Rect(p,p')$

Action sur la souris	Objet	S	A	Touches	Curseur	Effet
Enfoncer au point p, déplacer et relâcher au point p'	Pas de sommet			⇧ ⌥	+ ↖	$A \leftarrow A \Delta Rect(p,p')$
Enfoncer au point p, déplacer et relâcher au point p'	Pas d'arête			⇧⌥	✎	Crée une arête a d'extrémités en p et p' S'il n'existait pas de sommets en p ou p', ces sommets sont créés $A \leftarrow A \cup \{a\}$
Enfoncer au point p, déplacer et relâcher au point p'	Arête a		$a \notin A$	⌥	↖	$A \leftarrow \varnothing$ Effectue une translation sur a de vecteur $\overrightarrow{pp'}$
Enfoncer au point p, déplacer et relâcher au point p'	Arête a		$a \in A$	⌥	↖	Effectue sur les éléments de A une translation de vecteur $\overrightarrow{pp'}$
Enfoncer au point p, déplacer et relâcher au point p'	Arête a			⇧⌥	↖	Effectue une translation sur a de vecteur $\overrightarrow{pp'}$

Action sur la souris	Objet	S	A	Touches	Curseur	Effet
Enfoncer au point p, déplacer et relâcher au point p'				⌘		Effectue une translation de vecteur $\overrightarrow{pp'}$ de la feuille
Étiquettes alphanumériques						
Enfoncer et relâcher au point p	Étiquette					Rend l'étiquette active
Enfoncer au point p ...	Pas d'étiquette					Si une étiquette est active, la rend inactive, sinon même effet que pour un graphe valué
Enfoncer au point p, déplacer et relâcher au point p'	Étiquette			⌘		Effectue une translation de vecteur $\overrightarrow{pp'}$ de l'étiquette

Interface des opérations secondaires

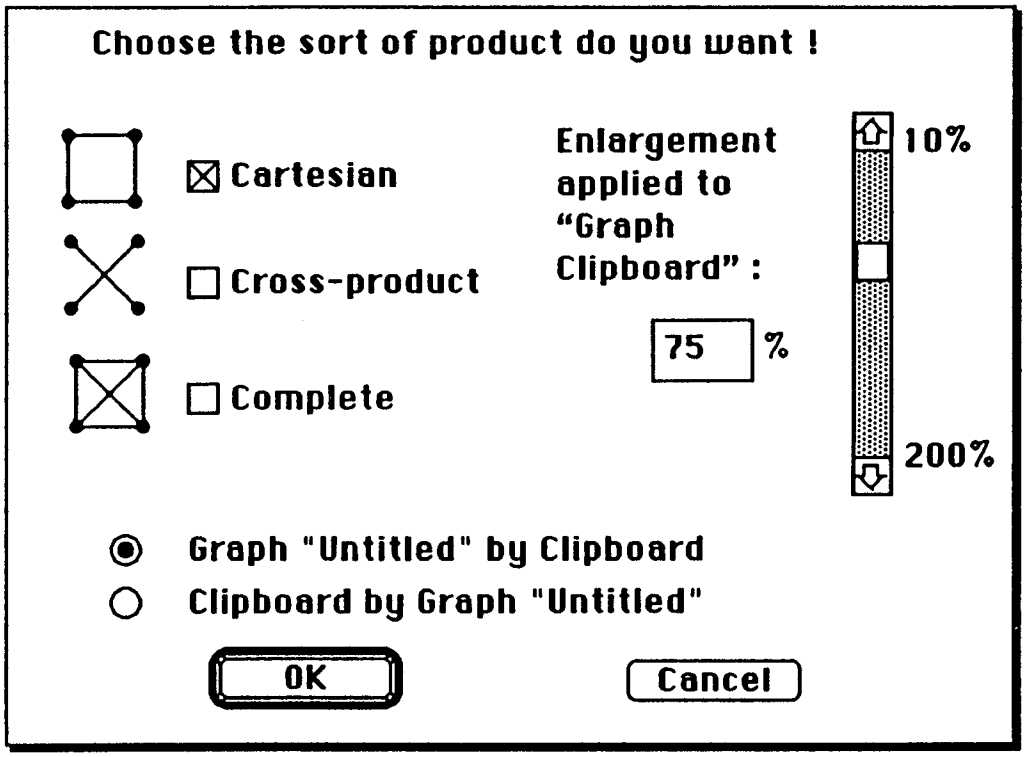
Les opérations secondaires sont toutes appelées par l'intermédiaires des **menus déroulants**. Certaines nécessitent l'utilisation de **dialogues intermédiaires**. Nous ne présenterons ici que quelques uns de ces dialogues qui nous paraissent significatifs de notre travail. Le lecteur trouvera également au Chapitre 2 une présentation détaillée de la gestion du dialogue utilisé pour la génération de graphes aléatoires.

Multiplicité des interfaces

Certaines opérations secondaires peuvent être exécutées de plusieurs façons différentes. C'est le cas par exemple du produit cartésien d'un graphe G quelconque et du graphe K_2, produit de graphes le plus couramment utilisé, en particulier pour la création d'hypercubes.

Il est possible d'utiliser directement ce produit en appelant la fonction Product by K2 du menu Transform (figure 32) ou de copier K2 dans le presse-papier et d'appeler la fonction Product by ClipBoard du même menu Transform (figure 32). Dans le premier cas, la suite de l'opération s'apparentera à celle du collage du graphe du presse-papier, décrite ci-dessous. Dans le second, une fenêtre de dialogue apparaîtra, permettant de spécifier qu'il s'agit d'un produit cartésien (et non croisé ou complet) et de paramètrer en partie le dessin du graphe produit obtenu (figure 27).

La fenêtre de dialogue de la fonction Product by ClipBoard :



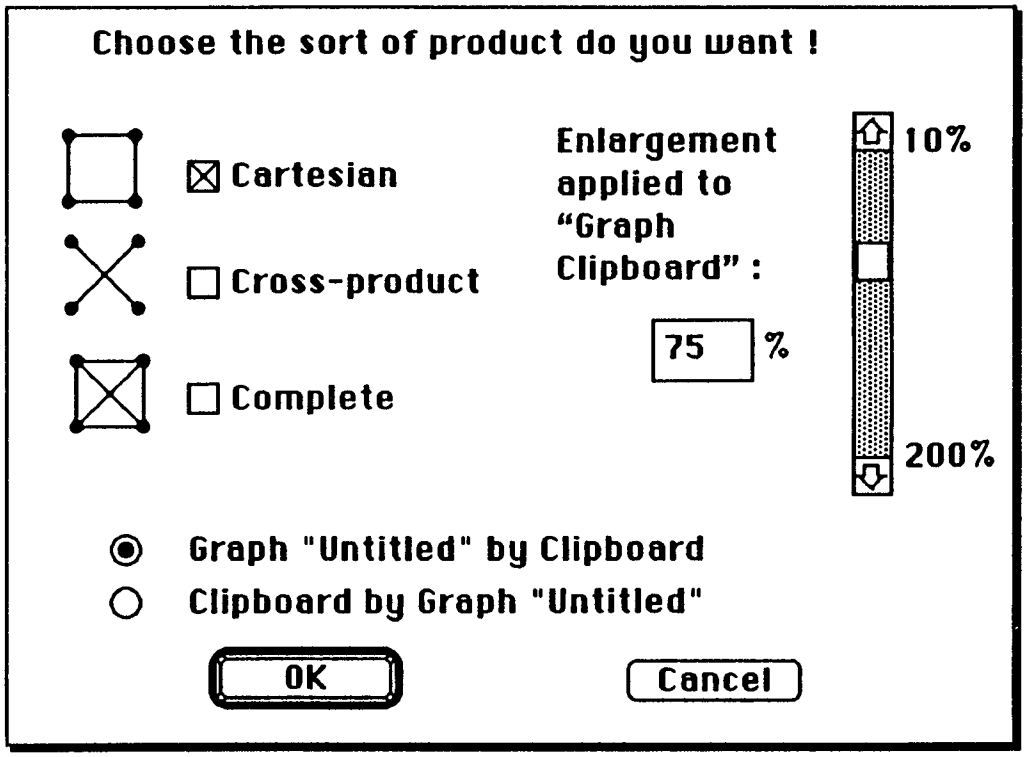
- figure 27 -

Spécification d'un rectangle de destination

Pour coller le graphe du presse-papier dans la fenêtre active, fonction appelée par l'item Paste du menu Edit, il est nécessaire de spécifier un rectangle de destination du graphe collé. Un message apparaît pour signaler cette attente, muni d'un bouton permettant l'annulation de l'opération (figure 28). De plus, le curseur est représenté par deux petites flèches orthogonales. La position de ce curseur lorsque l'utilisateur appuie sur le bouton de la souris, détermine un premier point du rectangle de destination v; le curseur prend alors la forme d'une main. Sa position lors du relâchement du bouton détermine un deuxième point v'. Soit R(v, v') le plus petit rectangle contenant v et v'.

Si aucune touche modificatrice n'est enfoncée lors du relâchement du bouton, R(v, v') sera le rectangle de destination. Si la touche Majuscule est enfoncée, le graphe collé subira uniquement un déformation par similitude (le

La fenêtre de dialogue de la fonction Product by ClipBoard :



- figure 27 -

Spécification d'un rectangle de destination

Pour coller le graphe du presse-papier dans la fenêtre active, fonction appelée par l'item Paste du menu Edit, il est nécessaire de spécifier un rectangle de destination du graphe collé. Un message apparaît pour signaler cette attente, muni d'un bouton permettant l'annulation de l'opération (figure 28). De plus, le curseur est représenté par deux petites flèches orthogonales. La position de ce curseur lorsque l'utilisateur appuie sur le bouton de la souris, détermine un premier point du rectangle de destination v; le curseur prend alors la forme d'une main. Sa position lors du relâchement du bouton détermine un deuxième point v'. Soit R(v, v') le plus petit rectangle contenant v et v'.

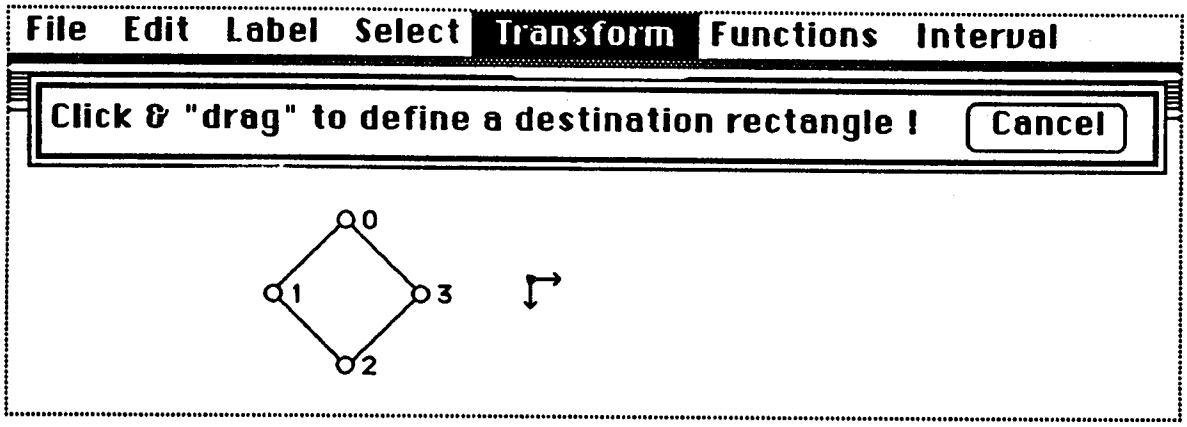
Si aucune touche modificatrice n'est enfoncée lors du relâchement du bouton, R(v, v') sera le rectangle de destination. Si la touche Majuscule est enfoncée, le graphe collé subira uniquement un déformation par similitude (le

rapport hauteur/largeur reste constant). Le rectangle de destination sera alors le plus petit rectangle respectant cette contrainte et contenant R(v, v'). Si la touche Commande est enfoncée, le graphe collé ne subira aucune déformation. Le rectangle de destination aura un coin en v, l'autre point de la diagonale passant par v étant le plus proche possible de v'.

Tant que le bouton de la souris n'est pas relâché, un rectangle grisé apparaît, correspondant au rectangle de définition obtenu si l'on relâche la souris (figure 29).

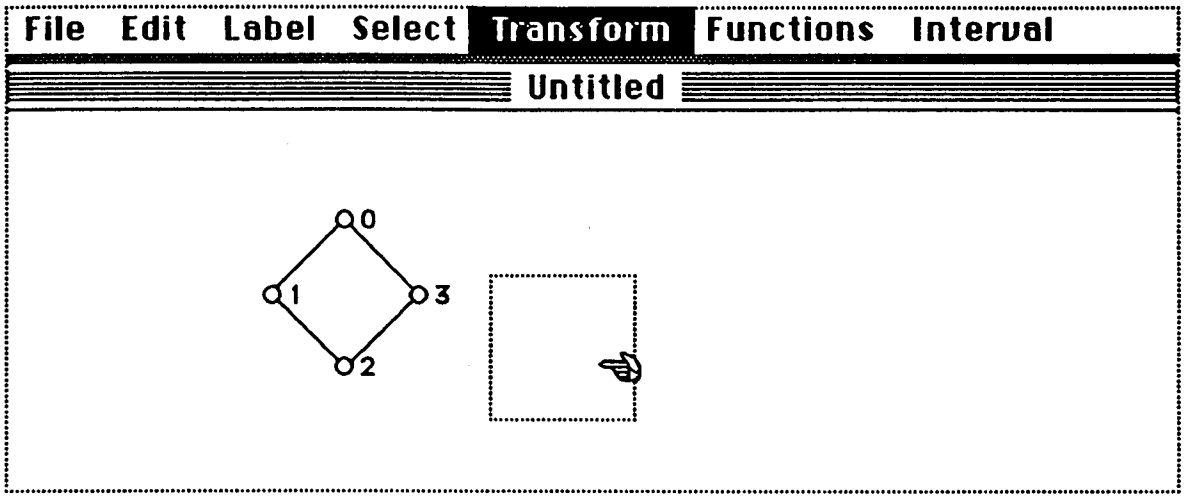
Si le rectangle de destination est jugé trop petit, la procédure de spécification du rectangle est réinitialisée, avec un message demandant un rectangle plus grand.

Message d'attente d'un rectangle de destination pour le collage du graphe du presse-papier :



- figure 28 -

Spécification d'un rectangle de destination. La touche Commande est enfoncée, ce qui explique qu'aucun coin du rectangle grisé ne soit à la position du curseur :



- figure 29 -

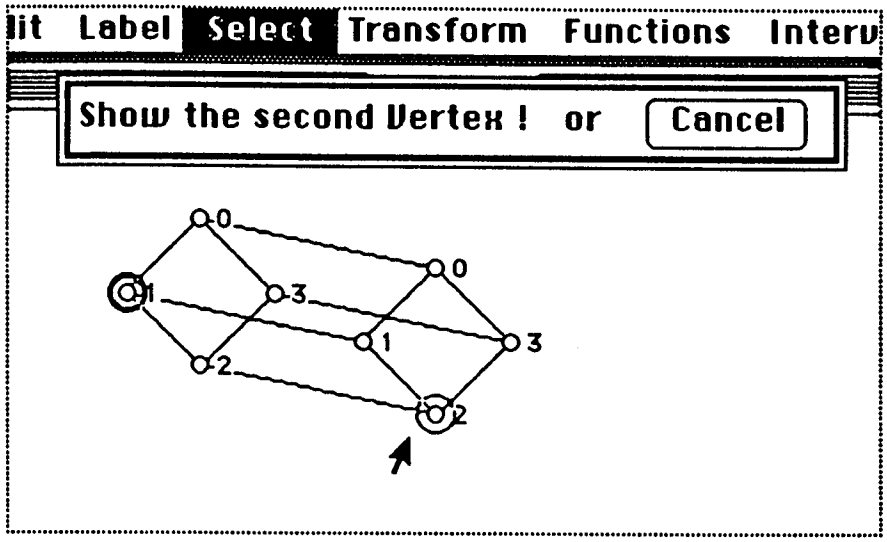
Désignation d'un ou plusieurs sommets

Pour plusieurs fonctions, il est nécessaire de désigner un ou plusieurs sommets. C'est le cas par exemple des sommets v et v' pour la sélection des sommets de l'intervalle I(v, v'). Cette fonction est appelée par l'item Interval... du menu Select.

Après l'appel de cette fonction, un message apparaît, demandant la désignation d'un premier sommet v, alors que le curseur prend la forme d'une flèche. Le message est de plus accompagné d'un bouton permettant l'annulation de l'opération. Lorsque l'on déplace le curseur et que celui-ci s'approche d'un sommet, ce sommet est entouré par un petit cercle. Le sommet v sera le sommet entouré au moment où l'utilisateur appuiera, puis relâchera le bouton de la souris. Si aucun sommet n'est entouré, le clic n'est pas pris en compte. Après la désignation de v, un deuxième dialogue apparaît, demandant la sélection du deuxième sommet v'. Le sommet v est quand à lui entouré par un petit cercle gras (figure 30). La désignation de v' est identique à celle de v.

Si la sélection de sommets S n'est pas vide au moment de l'appel de la fonction Interval..., seuls les sommets sélectionnés doivent être pris en compte car la fonction est alors appliquée au sous-graphe induit par S. C'est pourquoi seuls les sommets de la sélection seront entourés au moment du passage du curseur à leur proximité.

Désignation du second sommet pour la sélection d'un intervalle :



- figure 30 -

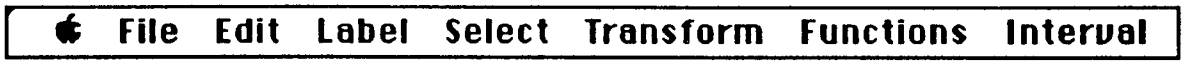
Les menus

Les figures ci-dessous présentent la barre de menus, puis chacun des menus déroulants de Cabri-graphes. Les items suivis de "..." sont ceux d'opérations nécessitant l'utilisation d'un dialogue.

Certains menus ou items peuvent changer selon l'état du système. Un menu ou un item en grisé signifiera qu'aucune action du menu ou que l'action associée à l'item ne sont pas utilisables. Le texte de l'item **Undo**, permettant l'annulation de la dernière opération effectuée, est généralement remplacé par un texte plus significatif exprimant quelle action peut être annulée. Les noms des différentes fenêtres présentes apparaissent au bas du menu Edit.

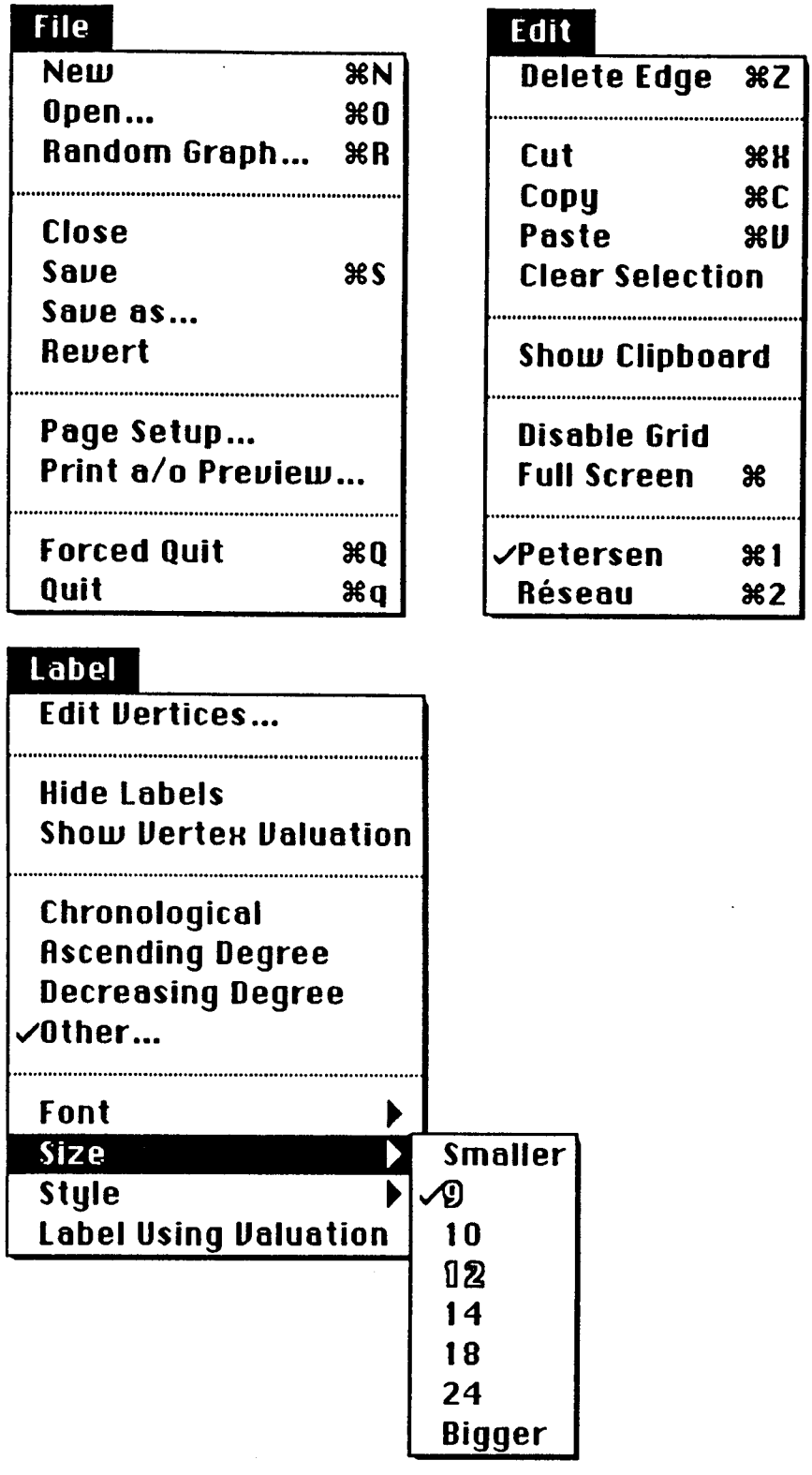
L'enfoncement de la touche Option a également pour effet de modifier certains items des menus Edit et Transform. Si la touche Option est enfoncée, l'item **Undo** est remplacé par **Disable Undo**. L'opération Disable Undo permet de désactiver la fonction Undo, et ainsi de diminuer les temps de réponses, en particulier si l'on travaille sur un "gros" graphe. L'item **Enable Grid** est remplacée par **Set Grid...**, qui permet de reparamètrer la grille. L'item **Subdivide...** est remplacé par l'item **Erase Vertices of Degree 2.**

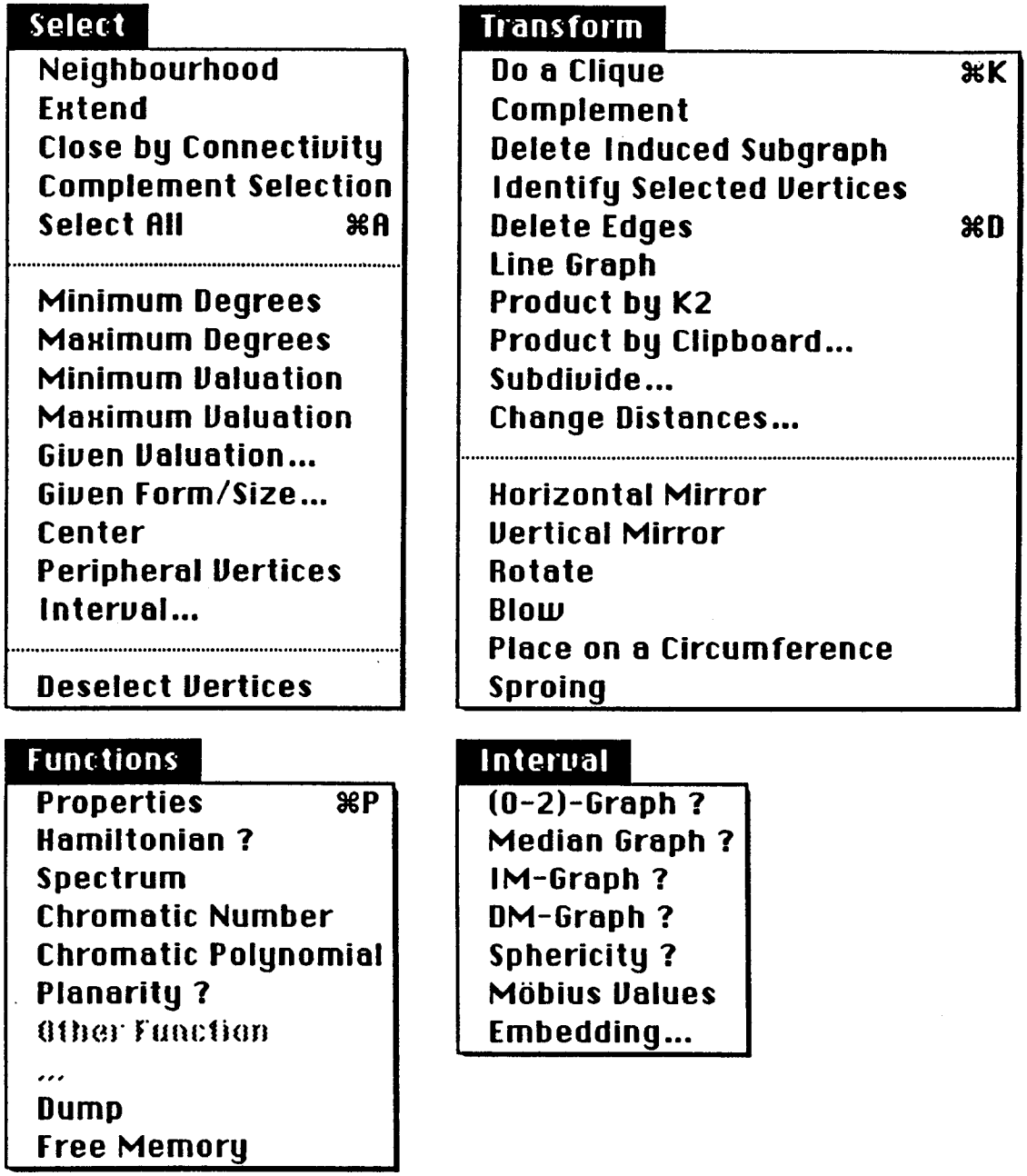
La barre de menus :



- figure 31 -

Les menus :


File
New ⌘N
Open... ⌘O
Random Graph... ⌘R
Close
Save ⌘S
Save as...
Revert
Page Setup...
Print a/o Preview...
Forced Quit ⌘Q
Quit ⌘q
Edit
Delete Edge ⌘Z
Cut ⌘X
Copy ⌘C
Paste ⌘V
Clear Selection
Show Clipboard
Disable Grid
Full Screen ⌘
✓Petersen ⌘1
Réseau ⌘2
Label
Edit Vertices...
Hide Labels
Show Vertex Valuation
Chronological
Ascending Degree
Decreasing Degree
✓Other...
Font
Size
Style
Label Using Valuation
Smaller
✓9
10
12
14
18
24
Bigger


Select
Neighbourhood
Extend
Close by Connectivity
Complement Selection
Select All ⌘A
Minimum Degrees
Maximum Degrees
Minimum Valuation
Maximum Valuation
Given Valuation...
Given Form/Size...
Center
Peripheral Vertices
Interval...
Deselect Vertices
Transform
Do a Clique ⌘K
Complement
Delete Induced Subgraph
Identify Selected Vertices
Delete Edges ⌘D
Line Graph
Product by K2
Product by Clipboard...
Subdivide...
Change Distances...
Horizontal Mirror
Vertical Mirror
Rotate
Blow
Place on a Circumference
Sproing
Functions
Properties ⌘P
Hamiltonian ?
Spectrum
Chromatic Number
Chromatic Polynomial
Planarity ?
Other Function
...
Dump
Free Memory
Interval
(0-2)-Graph ?
Median Graph ?
IM-Graph ?
DM-Graph ?
Sphericity ?
Möbius Values
Embedding...

- figure 32 -

1.9 Implémentation

Représentation d'un graphe en machine

Il n'est pas très judicieux de représenter en machine un graphe par la liste de ses sommets et la liste de ses arêtes. Les représentations utilisées, que ce soit dans le cadre d'un éditeur de graphes ou plus simplement en vue de la programmation d'un algorithme, s'inspirent généralement de l'une des représentations mathématiques suivantes :

dans un graphe simple, l'application Γ de V dans $\mathcal{P}(V)$ qui à tout sommet v associe son voisinage définit entièrement le graphe G, qui pourra alors être décrit par le couple (V, Γ) i.e. par une suite de sous-ensembles de V indexée par V.

Un graphe (simple ou non) peut être également défini à l'aide d'une matrice. Il existe principalement deux définitions matricielles :

- par sa matrice d'adjacence $A(G) = (a_{ij})_{i\in V, j\in V}$ ou a_{ij} est le nombre d'arêtes ayant i et j comme extrémités;

- par sa matrice d'incidence $I(G) = (i_{ve})_{v\in V, e\in E}$ ou i_{ve} est le nombre de fois où e est incidente à v.

Notons que dans un graphe simple, le vecteur $(a_{ij})_{j\in V}$ est le vecteur caractéristique du sous-ensemble de V : $\Gamma(i)$.

Lorsque G est représenté par (V, Γ), les règles de transformations R1 (ajouter un sommet), R2 (ajouter une arête), R3 (supprimer un sommet) et R4 (supprimer un arête) décrites au paragraphe 1.3 s'écrivent :

$$(V, \Gamma),\ v \notin V \rightarrow_{R1} (V \cup \{v\}, \Gamma') \text{ avec } \begin{cases} \Gamma'(x) = \Gamma(x),\ \forall\, x \in V \\ \Gamma'(v) = \varnothing \end{cases}$$

(V, Γ), v, w tels que $v \notin V$ ou $w \notin \Gamma(v) \rightarrow_{\mathbf{R2}} (V', \Gamma')$

$$\text{avec} \begin{cases} V' = V \cup \{v, w\} \\ \Gamma'(x) = \Gamma(x), \forall\, x \in V \setminus \{v,w\} \\ \Gamma'(v) = \begin{cases} \Gamma(v) \cup \{w\} \text{ si } v \in V \\ \{w\} \text{ sinon} \end{cases} \\ \Gamma'(w) = \begin{cases} \Gamma'(w) \cup \{v\} \text{ si } w \in V \\ \{v\} \text{ sinon} \end{cases} \end{cases}$$

$(V, \Gamma), v \in V \rightarrow_{\mathbf{R3}} (V \setminus \{v\}, \Gamma')$ avec $\Gamma'(x) = \Gamma(x) \setminus \{v\}, \forall\, x \in V \setminus \{v\}$

$$(V, \Gamma), v \in V, w \in \Gamma(v) \rightarrow_{\mathbf{R4}} (V, \Gamma') \text{ avec } \begin{cases} \Gamma'(x) = \Gamma(x), \forall\, x \in V \setminus \{v,w\} \\ \Gamma'(v) = \Gamma(v) \setminus \{w\} \\ \Gamma'(w) = \Gamma(w) \setminus \{v\} \end{cases}$$

Quelques spécifications du module de gestion des objets intrinsèques

Nous présentons ci-dessous l'ensemble des spécifications des types représentant les objets intrinsèques. Sont également données les spécifications des fonctions et actions sur les structures représentant des ensembles de sommets ou d'arêtes, ainsi que quelques fonctions et actions sur le graphe courant et la sélection courante. Donner la totalité de ces dernières serait trop long pour pouvoir être inclus dans ce texte. Pour les mêmes raisons, la description des algorithmes a été omise. Les fonctions et actions sur un graphe g quelconque ont des spécifications semblables à celles sur le graphe courant, g étant alors passé en paramètre. Le formalisme utilisé est celui du livre de P.C. Scholl et J.C. Peyrin [SP].

Le type **descriptif** et les constantes **n_long** et **n_etiq** peuvent être modifiés en vue d'une personnalisation du logiciel.

Les types

gr : le type <g : un graphe, suivant : un entier ≥ 0, valuation : un étiq_num,
alphanum : un booléen, sel : un ens, sel_a : un ens_a, d : un dessin,
fact : un entier > 0, dec_h : un entier, dec_v : un entier,
modif : une modification>

{gr exprime le contenu mathématique d'une fenêtre.
g, sel et sel_a désignent respectivement le graphe, la sélection de sommets et la sélection d'arêtes du graphe de la fenêtre;
suivant contient la valeur de l'étiquette numérique affectée au prochain sommet créé;
valuation précise le mode d'étiquette numérique utilisé : chronologique, inconnu ou autre ...
alphanum est vrai si et seulement si le graphe est étiqueté à l'aide de chaînes alphanumériques;
d et fact sont respectivement le forme et la taille du dessin par défaut du prochain sommet créé;
dec_h et dec_v sont respectivement le décalage vertical et le décalage horizontal par défaut de l'étiquette du prochain sommet créé par rapport à sa position (en pixel);
modif précise la teneur de la dernière modification de g, sel ou sel_a de gr}

étiq_num : le type {chronologique, inconnu, ...}

dessin : le type {cercle, carré, triangle}

modification : le type {aucune, ajout_arête, supprime_arête,
ajout_sommet_simple, ajout_sommet, stable_selection,
complement_selection, clique_selection, ...}

graphe : le type <n : un entier ≥ 0, m : un entier ≥ 0, tabcell : tab>
{n de g est le nombre de sommets du graphe g, m son nombre d'arêtes et tabcell contient les renseignements pour l'ensemble des sommets de g, dont les voisinages}

tab : le type séquence non vide de table
{les renseignements sur les sommets sont regroupés par tables, chaque table contenant les renseignements sur n_bloc sommets consécutifs. Les renseignements sur le $s^{ème}$ sommet de g seront dans le (s mod n_bloc)^{ème} élément de la (s div n_bloc + 1)^{ème} table de la séquence tabcell de g}

table : le type tableau sur int_bloc de cell

int_bloc : l'intervalle [0..n_bloc-1]

n_bloc : l'entier n_long*taille_long

taille_long : un entier
{nombre de bits d'un long : 32 sur un 68000}

n_long : un entier ≥ 0
{à spécifier par le programmeur}

cell : le type <étiq : une étiquette, p : un point, vois : un ens, val : un entier,
d : un dessin, fact : un entier > 0, dec_h : un entier,
dec_v : un entier>
{décrit un sommet s. étiq représente l'étiquette alphanumérique de s, p sa position dans la fenêtre, vois son voisinage, val son étiquette numérique, d et f respectivement sa forme et sa taille, dec_h et dec_v les décalages horizontaux et verticaux de son étiquette par rapport à sa position}

étiquette : le type <texte : tableau sur [0..n_etiq-1] de caractères, p : une police, t : une taille, s : un style>
{décrit l'étiquette d'un sommet. texte est le texte de l'étiquette, terminée par le caractère NUL, p, s et t la police, la taille et le style utilisés}

police : le type {applfont, athens, chicago, cairo, courier, geneva, helvetica, london, losangeles, newyork, monaco, sanfransisco, symbol, times, toronto, taliesin, venice}

taille : le type entier > 0

style : le type {bold, italic, underline, outline, shadow, condense, extend, normal}

n_etiq : un entier ≥ 0
{à spécifier par le programmeur}

point : le type <h : un entier ≥ 0, v : un entier ≥ 0>
{h et v sont les coordonnées horizontales et verticale d'un point dans la fenêtre}

ens : le type <nb : un entier ≥ 0, v : un vecteur, l : une liste>
{représente un ensemble de sommets e, chaque sommet étant codé par un entier positif. nb de e est le cardinal de e, v son vecteur caractéristique et l la liste de ses éléments. Le poids de v de e (nombre de composantes de v non nulles) et le nombre d'éléments de la liste l de e devront toujours être égaux à nb de e. De plus, le sommet s sera dans la liste l si et seulement si la $s^{ème}$ composante de v est non nulle}

vecteur : le type séquence non vide de super_long
{un vecteur v est découpé en super_longs de n_bloc bits. La $s^{ème}$ composante de v est non nulle si et seulement si le (s mod n_bloc)^{ème} élément du (s div n_bloc + 1)^{ème} super_long de v est à 1}

bit : le type entier sur {0, 1}

super_long : le type tableau sur int_bloc de bit

liste : le type séquence de sommet

sommet : le type <index : un entier ≥ 0, desc : un descriptif>
{un sommet s dans un ensemble est représenté par l'entier index de s, desc de s décrivant l'appartenance de s à l'ensemble. Par exemple, si cet ensemble est celui des voisins d'un sommet t, desc permettra d'exprimer les caractéristiques de l'arc (t, s)}

descriptif : un type
{à spécifier par le programmeur}

ens_a : le type <nb : un entier ≥ 0, l : une liste_a>
{décrit un ensemble d'arêtes e, sous la forme de couples d'extrémités, chaque extrémité étant codée par un entier positif ou nul. Le nombre d'éléments de la liste l de e devra toujours être égal à nb de e}

liste_a : le type séquence d'arête

arête : le type <extr1, extr2 : des entier ≥ 0>
{une arête a dans un ensemble est représenté par ses extrémités extr1 de a et extr2 de a, codées par des entiers}

Les globaux

gr_actif : un gr
{la variable de type gr associée à la fenêtre active}

le_graphe : le g de gr_actif

insere_vect : une action (la donnée i : un entier ≥ 0,
la donnée-résultat v : un vecteur)
ei : $\forall k \in [1, K], \forall j \in$ int_bloc, $s\_sl(k, v)_j \in \{0, 1\}$ et $\forall k > K$, $s\_sl(k, v) = nil$
ef : $s\_sl(i \text{ div } n\_bloc + 1, v)_{i \bmod n\_bloc} = 1$ et
$\forall k \in [K+1, i \text{ div } n\_bloc + 1], \forall j \in$ int_bloc,
$(k-1)*n\_bloc + j \neq i \Rightarrow sl(k, v)_j = 0$

in_vect : une fonction (i : un entier ≥ 0, v : un vecteur) $\rightarrow$ un booléen
{in_vect(i, v) $\leftarrow$ $s\_sl(i \text{ div } n\_bloc + 1, v) \neq nil$ et
$s\_sl(i \text{ div } n\_bloc + 1, v)_{i \bmod n\_bloc} = 1$}

Actions et fonctions sur le type ens

dans : une fonction (i : un entier ≥ 0, e : un ens) $\rightarrow$ un booléen
{dans(i, e) $\leftarrow$ nb de e $\neq 0$ et in_vect(i, v de e) et l de e = L & [<i, ?>] & L'}

insere : une action (la donnée i : un entier ≥ 0, la donnée-résultat e : un ens)
ei : nb de e = nbi et $\neg$dans(i, e)
ef : nb de e = nbi + 1 et dans(i, e)

enleve : une action (la donnée i : un entier ≥ 0, la donnée-résultat e : un ens)
ei : nb de e = nbi et dans(i, e)
ef : nb de e = nbi - 1 et $\neg$dans(i, e)

vide_ens : une action (la donnée-résultat e : un ens)
ei : $\forall$
ef : e = <0, [sl], []> et $\forall$ i $\in$ int_bloc, $sl_i = 0$

unir : une action (la donnée e1, la donnée-résultat e2 : des ens)
ei : I = {i, dans(i, e1)} et J = {i, dans(i, e2)}
ef : nb de e2 = |I $\cup$ J| et $\forall$ i, dans(i, e2) $\leftrightarrow$ (i $\in$ I ou i $\in$ J)

base : une matrice sur int_long x int_long de bit

{utilisée pour modifier une composante d'un super_long}

Initialisation de la matrice base

prepa : un action ()

ei : $\forall$

ef : $\forall$ i, j $\in$ [0..taille_long-1], $base_{i,j} = 1 \leftrightarrow i = j$

Actions et fonctions sur le type super_long

insere_sl : une action (i : un entier sur int_bloc

la donnée-résultat sl : un super_long)

ei : $\forall$

ef : $sl_i = 1$

enleve_sl : une action (la donnée i : un entier sur int_bloc

la donnée-résultat sl : un super_long)

ei : $\forall$

ef : $sl_i = 0$

present_sl : une fonction (i : un entier sur int_bloc,

sl : un super_long) $\rightarrow$ un booléen

{present_sl(i, sl) $\leftarrow$ $sl_i = 1$}

Actions et fonctions sur le type vecteur

s_sl : une fonction (i : un entier > 0, v : un vecteur) $\rightarrow$ un super_long

{s_sl(i, [sl]) $\leftarrow$ si i = 1 alors sl sinon nil

s_sl(i, sl_0v) $\leftarrow$ si i = 1 alors sl sinon s_sl(i-1, v)}

complem : une action (la donnée-résultat e : un ens)

ei : I = {i, dans(i, e)} et ∀ i, dans(i, e) → i < n de le_graphe

ef : nb de e = n de le_graphe - |I| et dans(i, e) ↔ (i < n de le_graphe et i ∉ I)

remplir : une action (la donnée-résultat e : un ens)

ei : ∀ i, dans(i, e) → i < n de le_graphe

ef : nb de e = n de le_graphe et dans(i, e) ↔ (i < n de le_graphe)

Actions et fonctions sur le type ens_a

dans_a : une fonction (i,j : des entiers ≥ 0, e : un ens_a) → un booléen

{dans_a(i, j, <0, l>) ← faux

dans_a(i, j, <n, []>) ← faux

dans_a(i, j, <n + 1, a_0e>) ← (extr1 de a = i et extr2 de a = j) ou dans_a(n, e)}

insere_a : une action (la donnée i,j : des entier ≥ 0,
la donnée-résultat e : un ens_a)

ei : nb de e = nbi et ¬dans_a(i, j, e)

ef : nb de e = nbi+1 et dans_a(i, j, e)

enleve_a : une action (la donnée i,j : des entier ≥ 0,
la donnée-résultat e : un ens_a)

ei : nb de e = nbi et dans_a(i, j, e)

ef : nb de e = nbi-1 et ¬dans_a(i, j, e)

vide_ens_a : une action (la donnée-résultat e : un ens_a)

ei : ∀

ef : e = <0, []>

unir_a : une action (la donnée e1, la donnée-résultat e2 : des ens_a)

ei : I = {(i, j), dans_a(i, j, e1)} et J = {(i, j), dans_a(i, j, e2)}

ef : nb de e2 = |I ∪ J| et ∀ i, j, dans(i, j, e2) ↔ ((i, j) ∈ I ou (i, j) ∈ J)

Quelques actions et fonctions sur le_graphe

s_table : une fonction (i : un entier > 0) → une table

{s_table(i) ← si i = 1 et tabcell de le_graphe = [t] alors t sinon nil

s_table(i) ← si i = 1 et tabcell de le_graphe = t_0T alors t sinon s_table(i-1)}

s_cell : une fonction (i : un entier ≥ 0) → une cell

{s_cell(i) ← si s_table(i div n_bloc +1) = nil alors nil

sinon s_table(i div n_bloc +1)$_{i \bmod n\_bloc}$}

FabCell : une action (la donnée i : un entier ≥ 0, les résultats c : une cell,

t : une table)

ei : ∀

ef : s_table(i div n_bloc + 1) ≠ nil et

t = s_table(i div n_bloc + 1) et

c = s_cell(i)

voisinage : une fonction (i : un entier ≥ 0) → un ens

{voisinage(i) = si s_cell(i) ≠ nil alors vois de s_cell(i)

sinon nil}

adja : une fonction (i, j : des entiers ≥ 0) → un booléen

{adja(i, j) = (voisinage(i) ≠ nil) et (voisinage(j) ≠ nil)

et puis dans(i, voisinage(j)) et dans(j, voisinage(i))}

plus_arc : une action (les données i, j : des entiers ≥ 0)

ei : i ≠ j et voisinage(i) ≠ nil et puis ¬dans(j, voisinage(i))

et nb de voisinage(i) = d_i

ef : dans(j, voisinage(i)) et nb de voisinage(i) = d_i + 1

plus_arête : une action (les données i, j : des entiers ≥ 0)

ei : m de le_graphe = mi et i ≠ j et ¬adja(i, j) et voisinage(i) ≠ nil et
voisinage(j) ≠ nil et puis nb de voisinage(i) = d_i et nb de voisinage(j) = d_j

ef : m de le_graphe = mi + 1 et adja(i, j) et
nb de voisinage(i) = d_i + 1 et nb de voisinage(j) = d_j + 1
et modif de gr_actif = ajout_arête

moins_arc : une action (les données i, j : des entiers ≥ 0)

ei : voisinage(i) ≠ nil et puis dans(j, voisinage(i))
et nb de voisinage(i) = d_i

ef : ¬dans(j, voisinage(i)) et nb de voisinage(i) = d_i - 1

moins_arête : une action (les données i, j : des entiers ≥ 0)

ei : m de le_graphe = mi et mi > 0 et adja(i, j) et
et nb de voisinage(i) = d_i
et nb de voisinage(j) = d_j

ef : m de le_graphe = mi - 1 et ¬adja(i, j) et
nb de voisinage(i) = d_i - 1 et nb de voisinage(j) = d_j - 1
et modif de gr_actif = supprime_arête

ajout_simple : une action (la donnée p : un point)

ei : n de le_graphe = ni et suivant de gr_actif = suivi et
valuation de gr_actif = vali

ef : n de le_graphe = ni + 1 et
s_cell(ni) = <<texte, applfont, 7, normal>, p, <0, [sl], []>, suivi,
d de gr_actif, fact de gr_actif, dec_h de gr_actif, dec_v de gr_actif>
et $texte_0$ = NUL
et ∀ i ∈ int_bloc, sl_i = 0 et suivant de gr_actif = suivi + 1 et
vali ≠ chrono ⇒ valuation de gr_actif = inconnue et
modif de gr_actif = ajout_sommet_simple

ajout : une action (la donnée p : un point)

ei : n de le_graphe = ni et m de le_graphe = mi et suivant de gr_actif = suivi et

valuation de gr_actif = vali et

$\forall$ i, dans(i, sel de gr_actif) $\Rightarrow$ (voisinage(i) $\neq$ nil et d_i = nb de voisinage(i))

ef : n de le_graphe = ni + 1 et m de le_graphe = mi + nb de sel de gr_actif et

s_cell(ni) = <<texte, applfont, 7, normal>, p, sel de gr_actif , suivi,

d de gr_actif, fact de gr_actif, dec_h de gr_actif, dec_v de gr_actif>

et $texte_0$ = NUL et

$\forall$ i, dans(i, sel de gr_actif) $\Rightarrow$ (adja(ni, i) et nb de voisinage(i) = d_i + 1)

et suivant de gr_actif = suivi + 1 et

vali $\neq$ chrono $\Rightarrow$ valuation de gr_actif = inconnue et

modif de gr_actif = ajout_sommet

modifier : une action (la donnée variation : un entier sur {0,1,2})

ei : n de le_graphe = ni et m de le_graphe = mi et

$\forall$ i, dans(i, sel de gr_actif) $\Rightarrow$ (voisinage(i) $\neq$ nil et d_i = nb de voisinage(i) et

V_i = {j, dans(j, sel de gr_actif) et adja(i,j)}) et

$\overline{V_i}$ = {j, dans(j, sel de gr_actif) et $\neg$adja(i,j)})

ef : varie = 0 ou 1 $\Rightarrow$

($\forall$ i, dans(i, sel de gr_actif) $\Rightarrow \forall$ j $\in$ V_i, $\neg$adja(i,j))

varie = 1 ou 2 $\Rightarrow$

($\forall$ i, dans(i, sel de gr_actif) $\Rightarrow \forall$ j $\in$ $\overline{V_i}$, adja(i,j))

varie = 0 $\Rightarrow$ (nb de voisinage(i) = d_i - $|V_i|$ et

m de le_graphe = mi - {$\Sigma|V_i|$, i tel que dans(i, sel de gr_actif)}/2

et modif de gr_actif = stable_selection)

varie = 1 $\Rightarrow$ (nb de voisinage(i) = d_i - $|V_i|$ + $|\overline{V_i}|$ et

m de le_graphe = mi - {$\Sigma|V_i|$, i tel que dans(i, sel de gr_actif)}/2 +

{$\Sigma|\overline{V_i}|$, i tel que dans(i, sel de gr_actif)}/2

et modif de gr_actif = complement_selection)

varie = 2 $\Rightarrow$ (nb de voisinage(i) = $d_i + |\overline{V_i}|$ et

m de le_graphe = mi + $\{\Sigma|\overline{V_i}|$, i tel que dans(i, sel de gr_actif)$\}/2$

et modif de gr_actif = clique_selection)

ANNEXE 1

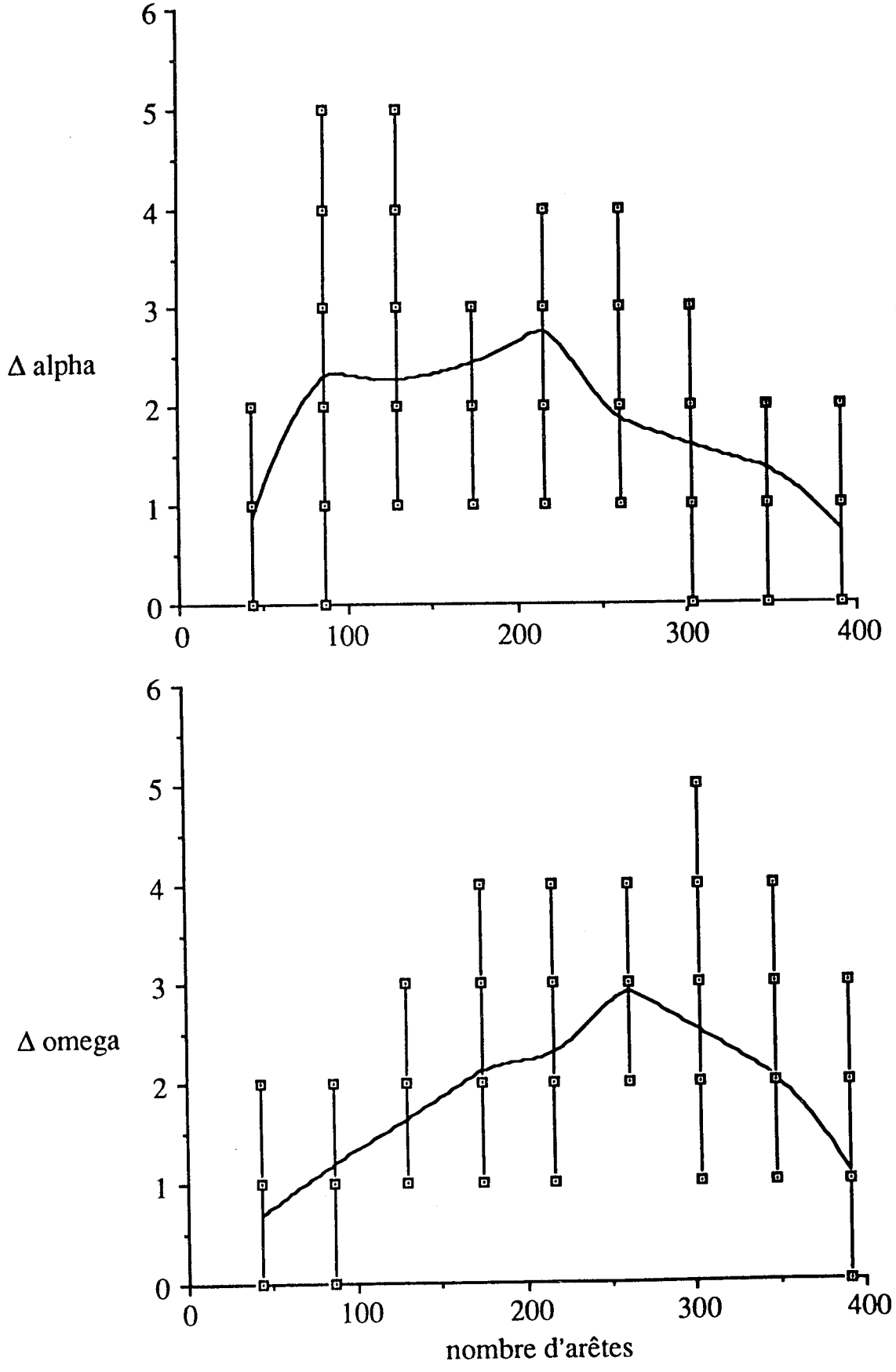
PRÉCISION DE L'ENCADREMENT DE α ET ω DANS CABRI-GRAPHES

Les valeurs de l'encadrement de α (cardinal maximum d'un stable) et ω (cardinal maximum d'une clique) ont été calculés sur des graphes à 30, puis 50 sommets, de densité allant de 0,1 à 0,9 par pas de 0,1. Pour chaque densité, 25 graphes ont été générés aléatoirement, selon une loi de probabilité uniforme.

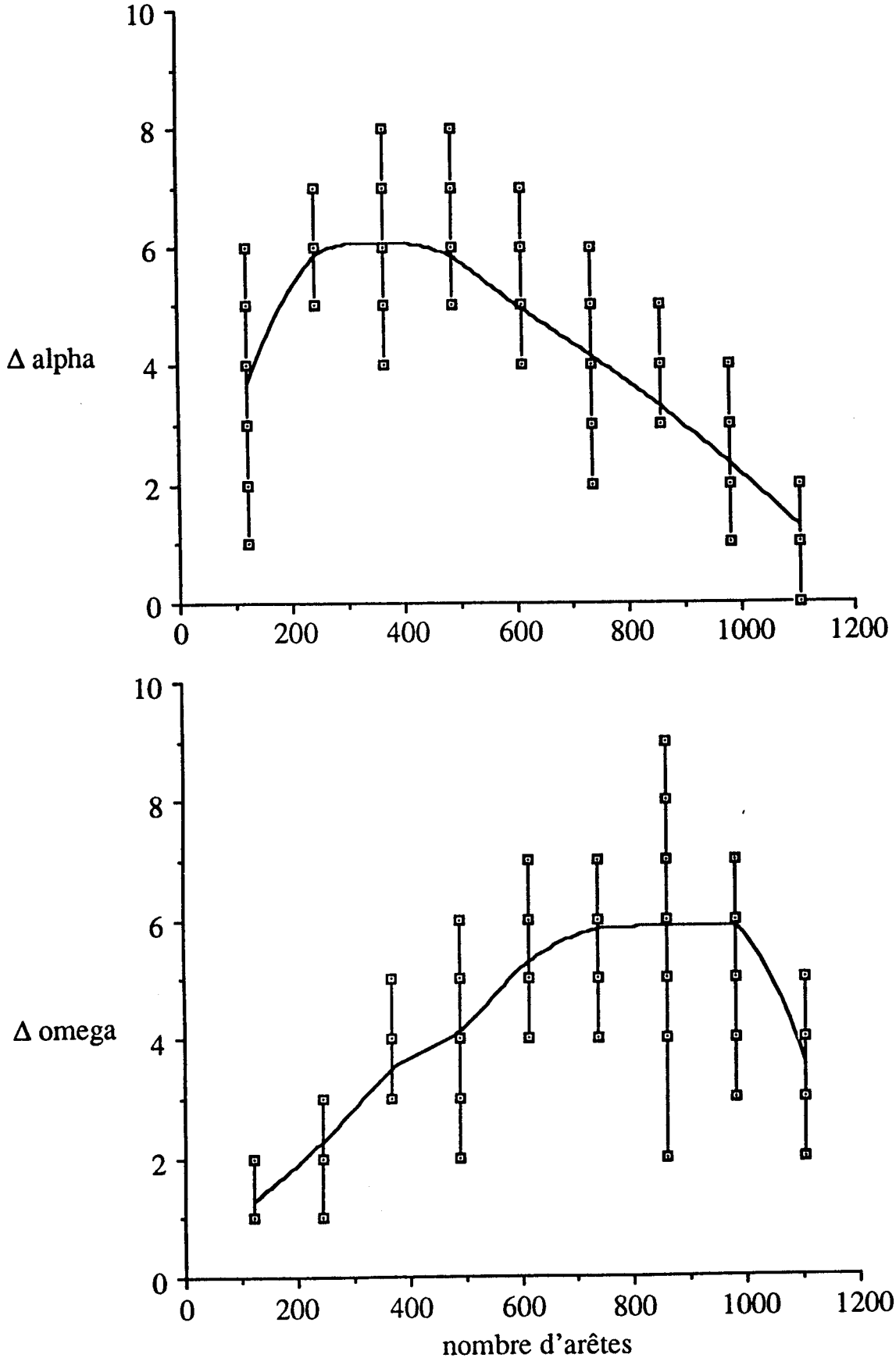
Les 4 graphiques de cette annexe présentent, en fonction du nombre d'arêtes, les différents écarts obtenus entre les bornes maximum et minimum, respectivement pour α et ω. Une courbe permet d'approximer l'écart moyen entre ces bornes pour toutes valeurs de m.

On remarquera la symétrie entre les courbes concernant α et celles de ω. Cette symétrie provient du fait que les bornes minimum et maximum pour $\alpha(G)$ sont respectivement les bornes maximum et minimum pour $\omega(\overline{G})$.

Encadrement de α et ω pour les graphes à 30 sommets



Encadrement de α et ω pour les graphes à 50 sommets



ANNEXE 2

REPRÉSENTATION D'UN GRAPHE DANS UN FICHIER TEXTE

Syntaxe

L'alphabet de cette grammaire est composé de tous les caractères reconnus par la norme ASCII. CR désigne le caractère Carriage Return (code ASCII 13), SPC le caractère espace (code ASCII 32), ε le mot vide.

<graphe> ::= <entête><options><cardinaux><suite_sommets>

<entête> ::= <commentaire1><numéro_de_version><commentaire2>CR

<commentaire1> ::= <commentaire1><car1> | ε

<car1> ::= tout caractère sauf 1 2 3 4 5 6 7 8 9 . ,

<numéro_de_version> ::= <entier_positif> |
<entier_positif>.<entier_positif> |
<entier_positif>,<entier_positif> |
.<entier_positif> |
,<entier_positif>

<entier_positif> ::= <chiffre> | <entier_positif><chiffre>

<chiffre> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<commentaire2> ::= <commentaire2><car2> | ε

<car2> ::= tout caractère sauf CR

<options> ::= <commentaire3><une_option><options> | <commentaire3>

<commentaire3> ::= <commentaire3><car3> | ε

<car3> ::= tout caractère sauf n a w g s #

<une_option> ::= <non_étiqueté> |
<alpha_num> |
<sans_annuler> |
<grille> |
<sélections>

<non_étiqueté> ::= n<commentaire4>SPC

<commentaire4> ::= <commentaire4><car4> | ε

<car4> ::= tout caractère sauf SPC

<alpha_num> ::= a<commentaire4>SPC

<sans_annuler> ::= w<commentaire4>SPC

<grille> ::= g<commentaire5><maille_grille>

<maille_grille> ::= <entier>

<commentaire5> ::= <commentaire5><car5> | <commentaire5b><car5> | ε

<commentaire5b> ::= <commentaire5>- | <commentaire5b>-

<commentaire5c> ::= <commentaire5><car5> |
<commentaire5b><car5><commentaire5>

<car5> ::= tout caractère sauf 1 2 3 4 5 6 7 8 9 -

<entier> ::= <entier_positif> | <entier_négatif>

<entier_négatif> ::= -<entier_positif>

<sélections> ::= <sélection_sommets> | <sélection_arêtes> | ε

<sélection_sommets> ::=
s<commentaire4>SPC<commentaire5><sommet_sélectionné><espaces>
<reste_S>

<sommet_sélectionné> ::= <entier_positif>

<espaces> ::= SPC<espaces> | ε

<reste_S> ::=
<car6><commentaire5><sommet_sélectionné><espaces>
<reste_encore_S> |
<sommet_sélectionné><espaces><reste_encore_S> |
CR<car6b><commentaire5><sommet_sélectionné><espaces>
<reste_encore_S> |
CR-<commentaire5c><sommet_sélectionné><espaces>
<reste_encore_S> |
CR<sommet_sélectionné><espaces><reste_encore_S> |
ε

<car6> ::= tout caractère sauf CR 1 2 3 4 5 6 7 8 9 -

<car6b> ::= tout caractère sauf # s 1 2 3 4 5 6 7 8 9 -

<reste_encore_S> ::=
-<commentaire5c><sommet_sélectionné><espaces><reste_encore_S> |
<reste_S>

<sélection_arêtes> ::=
s<commentaire4>SPC<commentaire5><extrémité1><espaces>-
<commentaire5><extrémité2><reste_A>

<extrémité1> ::= <entier_positif>

<extrémité2> ::= <entier>

<reste_A> ::=
<car6><commentaire5><arête_sélectionnée><espaces><reste_A> |
<arête_sélectionnée><espaces><reste_A> |
CR<car6b><commentaire5><arête_sélectionnée><espaces><reste_A> |
CR-<commentaire5c><arête_sélectionnée><espaces><reste_A> |
CR<arête_sélectionnée><espaces><reste_A> |
-<commentaire5c><arête_sélectionnée><espaces><reste_A> |
ε

<arête_sélectionnée> ::= <extrémité1><commentaire5><extrémité2>

<cardinaux> ::=
#<commentaire4>SPC<commentaire5><n><suite_cardinaux>CR

<n> ::= <entier_positif>

<suite_cardinaux> ::=
<commentaire7> |
<commentaire7>#<commentaire4>SPC<commentaire5><cardinal>
<suite_cardinaux>

<commentaire7> ::= <commentaire7><car7> | ε

<car7> ::= tout caractère sauf # CR

<cardinal> ::= <entier>

<suite_sommets> ::= <sommet><suite_sommets> | ε

<sommet> ::= <lignes_vides><commentaire8><caractéristiques>CR

<lignes_vides> ::= CR<lignes_vides> | ε

<commentaire8> ::= <commentaire8><car8> | ε

<car8> ::= tout caractère sauf d c v s o l CR

<caractéristiques> ::= <caractéristiques2><voisinage><caractéristiques2>

<caractéristiques2> ::= <coordonnées><caractéristiques2> |
<valuation><caractéristiques2> |
<forme><caractéristiques2> |
<offset><caractéristiques2> |
<étiquetage><caractéristiques2> |
ε

<voisinage> ::= d<commentaire4>SPC<commentaire5><degré><voisins>

<degré> ::= <entier_positif>

<voisins> ::= <commentaire5><un_voisin><voisins> | ε

<un_voisin> ::= <entier_positif>

<coordonnées> :: =
c<commentaire4>SPC<commentaire5><c_horizontale><commentaire5>
<c_verticale> |
ε

<c_horizontale> ::= <entier>

<c_verticale> ::= <entier>

<valuation> :: = v<commentaire4>SPC<commentaire5><valeur> | ε

<valeur> ::= <entier>

<taille> ::= s<commentaire9><une_taille>

<une_taille> ::= <entier>

<commentaire9> ::= <car9><commentaire4>SPC<commentaire5> | SPC | ε

<car9> ::= tout caractère sauf h

<forme> ::=
sh<commentaire4>SPC<espaces><une_forme><commentaire10> |
sSPC<espaces><une_forme><commentaire10>

<une_forme> ::= r | s | t

<commentaire10> ::= <car10><commentaire10> | ε

<car10> ::= tout caractère sauf SPC CR

<offset> ::=
o<commentaire4>SPC<commentaire5><d_horizontal><commentaire5>
<d_vertical>

<d_horizontal> ::= <entier>

<d_vertical> ::= <entier>

<étiquetage> ::=
l<commentaire11><car11b><espaces><étiquette>SPC<font>
<commentaire2>

<commentaire11> ::= <commentaire11><car11> | ε

<car11> ::= tout caractère sauf SPC :

<car11b> ::= SPC | :

<étiquette> ::= <car4><commentaire4>

<font> ::=
<commentaire12>f<commentaire11><car11b><espaces><police><taille_p>
<style>

<commentaire12> ::= <commentaire12><car12> | ε

<car12> ::= tout caractère sauf f

<police> ::= <entier_positif> | applfont | Applfont | athens | Athens |
chicago | Chicago | cairo | Cairo | courier | Courier |
geneva | Geneva | helvetica | Helvetica | london |
London | losAngeles | Losangeles | newyork | Newyork |
newYork | NewYork | new york | New york | new York |
New York | monaco | Monaco | sanFransisco |
SanFransisco | sanFran | SanFran | symbol | Symbol |
times | Times | toronto | Toronto | taliesin | Taliesin |
Venice

<taille_p> ::=
<commentaire13><car13b><espaces><commentaire5><une_taille_p>

<une_taille_p> ::= <entier_positif>

<commentaire13> ::= <commentaire13><car13> | ε

<car13> ::= tout caractère sauf `SPC s`

<car13b> ::= `SPC` | `s`

<style> ::=
<commentaire12>`f`<commentaire11><car11b><espaces><un_style>

<un_style> ::= <entier_positif> | `(`<commentaire14><nom_de_style>`)`

<commentaire14> ::= <commentaire14><car14> | ε

<car14> ::= tout caractère sauf `n b i u o s c e`

<nom_de_style> ::= `bold` | `italic` | `underline` | `outline` | `shadow` |
`condense` | `extend` | `normal`

Sémantique

graphe : le graphe

numéro_de_version : contenu dans l'entête; précise le numéro de la version du logiciel ayant créé le graphe; doit être compris entre 3 et 3,5 pour que Cabri-graphes accepte de lire le graphe.

non_étiqueté : si non vide, alors le graphe apparaîtra non étiqueté.

alpha_num : si non vide, alors l'étiquetage du graphe sera alpha-numérique.

sans_annuler : si non vide, alors la fonction Undo sera désactivée.

grille : si non vide, alors la maille de la grille doit être précisée; sinon, la grille sera inactive et de maille 20.

maille_grille : maille de la grille; si négative, la grille sera inactive.

sélection_sommets : si non vide, alors la sélection de sommets sera non vide et égale à l'union des sélections de sommets lues.

sommet_sélectionné : un élément de la sélection de sommets.

sélection_arêtes : si non vide, alors la sélection d'arêtes sera non vide et égale à l'union des sélections d'arêtes lues.

extrémité1, extrémité2 : extrémités d'une arête sélectionnée. Si extrémité2 est négatif, la seconde extrémité sera sa valeur absolu.

n : nombre de sommets du graphes.

cardinal : si non vide, alors le dernier cardinal lu sera le nombre d'arêtes du graphe. En fin de lecture, un test sera fait pour savoir si l'automate a effectivement trouvé le nombre d'arêtes précisé.

sommet : un sommet; le nombre d'éléments de ce langage auxiliaire doit être égal à n.

degré : degré du sommet

un_voisin : un voisin du sommet; pour chaque sommet, le nombre d'éléments de ce langage auxiliaire doit être égal à degré.

c_horizontale, c_verticale : coordonnées horizontales et verticales du sommet; ne peuvent être omises dans l'état actuel du logiciel, mais prévues comme optionnelles.

valeur : valeur affectée au sommet en cas d'étiquetage numérique; si vide, la valeur par défaut sera le numéro du sommet dans l'ordre de lecture.

une_taille : taille du dessin du sommet; si vide, cette taille sera égale à 7.

une_forme : forme du dessin du sommet; si vide ou r, cette forme sera ronde, si s carrée et si t triangulée.

d_horizontal, d_vertical : décalages horizontal et vertical (en pixel) par rapport au centre du dessin du sommet; si vide, ce décalage sera initialisé à (7, 4).

étiquetage : étiquette alphanumérique du sommet; si vide, cette étiquette sera initialisée à une chaîne vide.

étiquette : chaîne de caractères de l'étiquette alphanumérique.

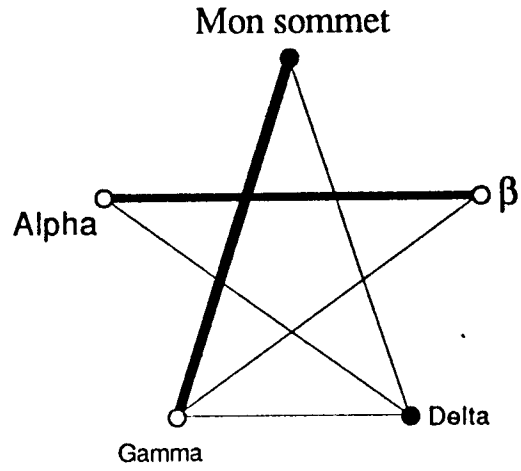
police : police utilisée pour l'étiquette (en clair ou numéro seulement).

une_taille_p : taille de la police utilisée.

un_style : style utilisé pour l'étiquette (en clair entre parenthèse ou numéro seulement).

Exemple

Exemple d'enregistrement d'un graphe sous forme de fichier texte. Les sommets ont pour étiquetage numérique, non visible ici, leur distance par rapport au sommet "Mon sommet" :




```
CABRI Version 3.0, apri 1989
alpha_num grid : -20
selection: 4 0
selection of edges: 1 - 2 0 - 3
#vertices: 5 #edges: 6

0 degree: 2    3 4
    coord.: 226 75 val.: 0 shape:  round  size: 3 offsets: -35 -9
    label: Mon sommet  font: 20 (Times) size: 14 face: 0 (normal)
1 degree: 2    2 4
    coord.: 156 127 val.: 2 shape:  round  size: 3 offsets: -34 14
    label: Alpha  font: 21 (Helvetica) size: 12 face: 0 (normal)
2 degree: 2    3 1
    coord.: 297 127 val.: 2 shape:  round  size: 3 offsets: 7 4
    label: b  font: 23 (Symbol) size: 14 face: 0 (normal)
3 degree: 3    0 2 4
    coord.: 183 210 val.: 1 shape:  round  size: 3 offsets: -22 17
    label: Gamma  font: 1 (applFont) size: 9 face: 0 (normal)
4 degree: 3    1 0 3
    coord.: 270 210 val.: 1 shape:  round  size: 3 offsets: 7 4
    label: Delta  font: 1 (applFont) size: 9 face: 0 (normal)
```

- figure 33 -

CHAPITRE 2

LA GÉNÉRATION DE GRAPHES ALÉATOIRES

2.1 Le problème

Énoncé

Soit $\mathscr{P}$ un ensemble de propriétés et $\mathscr{G}_n(\mathscr{P})$ l'ensemble des graphes à n sommets possédant ces propriétés. Le problème traité dans ce chapitre consiste à trouver un algorithme permettant de générer aléatoirement des graphes de $\mathscr{G}_n(\mathscr{P})$ selon une loi de probabilité uniforme ou plus simplement garantissant à tout graphe de $\mathscr{G}_n(\mathscr{P})$ une probabilité non nulle d'être généré.

Motivation

L'objectif d'un générateur aléatoire de graphe est de permettre le test de conjectures du type "tout graphe possédant les propriétés $\mathscr{P}$ possède P" ou P désigne une propriété n'appartenant pas à $\mathscr{P}$.

Il paraît toutefois indispensable pour répondre pleinement à cet objectif de disposer conjointement d'une base de donnée de graphes, afin de soumettre la conjecture à des graphes très particuliers ou encore à l'ensemble des graphes de $\mathscr{G}_n(\mathscr{P})$ lorsque n est suffisamment petit [Cve, CK, CKS]. Cabri-graphes ne dispose malheureusement pas à l'heure actuelle d'un tel outil, dont la réalisation est sans aucun doute très complexe.

Dans le cadre de Cabri-graphes, nous avons favorisé le nombre de possibilités pour choisir $\mathscr{P}$, et l'utilisation d'un dialogue adéquat pour définir cet ensemble. C'est pourquoi dans la majorité des cas, nous n'avons pas cherché à obtenir des graphes selon une loi de probabilité uniforme, mais seulement à garantir à tout graphe vérifiant l'ensemble des propriétés demandées une probabilité non nulle d'apparaître.

2.2 Étude bibliographique

Une étude sur les articles traitant des algorithmes de génération aléatoire de graphes a été soumise pour publication [Bau2] et est rapportée en Annexe 2. Depuis 1980, de nombreux articles sur ce sujet ont été publiés, présentant surtout des algorithmes tels que la probabilité d'être généré soit la même pour tous les graphes de $\mathcal{G}_n(\mathcal{P})$. Toutefois aucun de ces algorithmes n'a été mis en oeuvre dans le cadre de Cabri-graphes. En effet, ils sont très spécialisés, c'est à dire qu'ils ne concernent qu'un ensemble $\mathcal{P}$ donné et peuvent difficilement être adaptés pour d'autres ensembles de propriétés.

De plus, certaines propriétés très importantes en théorie des graphes n'ont jamais été traitées dans le cadre d'une génération uniforme. C'est le cas par exemple de l'hamiltonicité. Rappelons que l'on ne connaît pas d'algorithme polynômial pour trouver un cycle hamiltonien dans un graphe, ni à fortiori pour en connaître le nombre. Il semble que pour ce type de problèmes, on s'oriente vers l'étude à posteriori des propriétés probabilistes d'algorithmes de génération, en utilisant des méthodes inspirées par la théorie des graphes aléatoires.

2.3 Le choix des propriétés

Propriétés traitées par Cabri-graphes

Cabri-graphes permet de générer aléatoirement des graphes à n sommets, m arêtes et possédant éventuellement les propriétés suivantes :

- connexe
- de diamètre D
- hamiltonien
- possédant un couplage parfait
- biparti
- régulier de degré r

L'ensemble des graphes répondant aux propriétés désirées par l'utilisateur sera noté par $\mathcal{G}_{n,m}(\mathcal{P})$.

Utilisation des connaissances sur $\mathcal{G}_{n,m}(\mathcal{P})$

Ces connaissances sont utilisées dans trois buts différents :

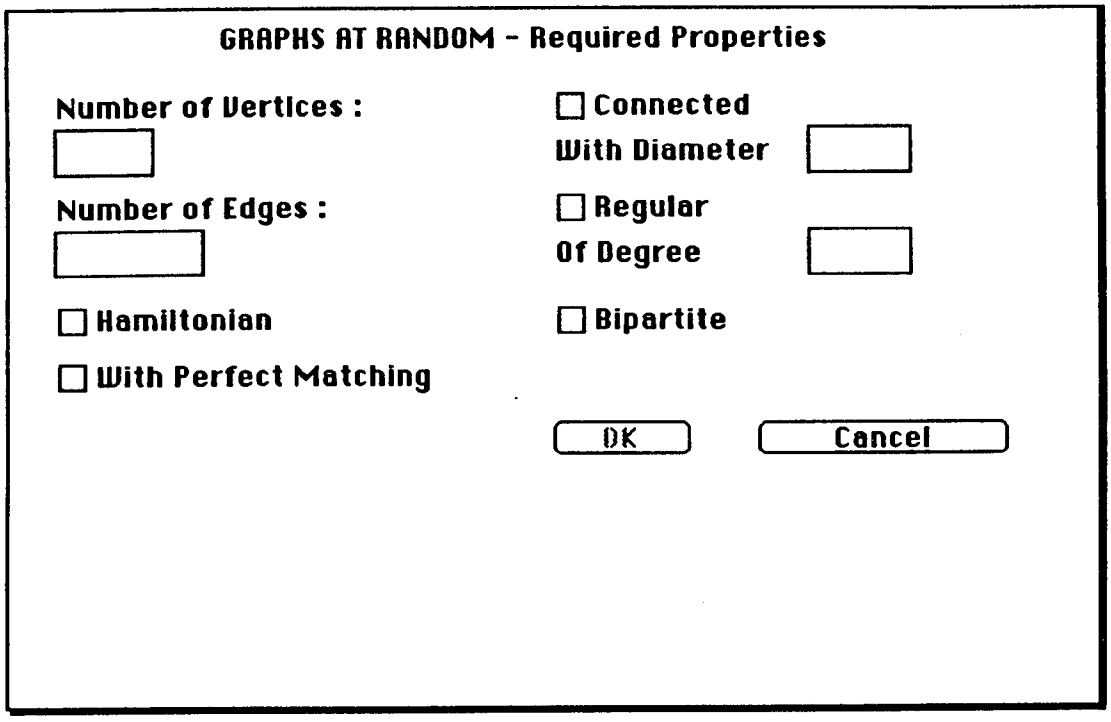
1) savoir si l'ensemble $\mathcal{G}_{n,m}(\mathcal{P})$ est vide. Si oui, ce fait est signalé à l'utilisateur ainsi que le théorème qui a permis d'aboutir à cette conclusion (figure 34).

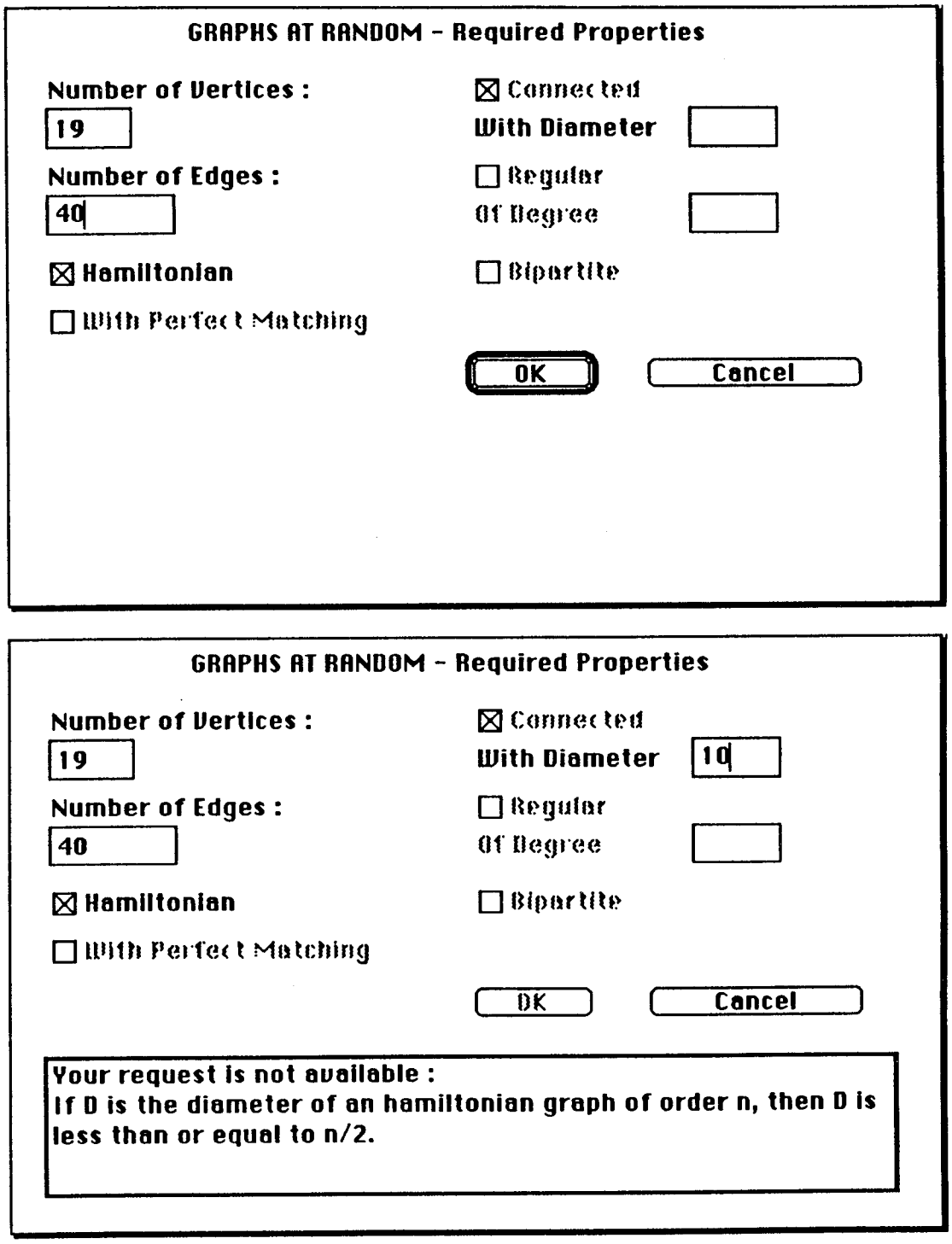
2) Si les graphes de $\mathcal{G}_{n,m}(\mathcal{P})$ possèdent une propriété non spécifiée par l'utilisateur, et que cette propriété intervient dans le dialogue du choix de $\mathcal{P}$, ce fait est signalé à l'utilisateur (figure 34).

3) Il est parfois préférable de redéfinir $\mathcal{G}_{n,m}(\mathcal{P})$ à l'aide d'un ensemble de propriétés $\mathcal{P}'$ équivalent à $\mathcal{P}$, ceci afin d'avoir un temps d'exécution de l'algorithme plus court.

L'ensemble des théorèmes utilisés dans le cadre du générateur de graphes aléatoires de Cabri-graphes est donné en Annexe 1. Cette liste a été définie par expérimentation. Les théorèmes sont représentés par des expressions booléennes.

Fonctionnement du dialogue de choix des propriétés. La première figure représente le dialogue tel qu'il apparaît pour la première fois. Les propriétés choisies seront mémorisées d'un appel à l'autre. Dans la deuxième figure, l'utilisateur veut générer un graphe hamiltonien à 19 sommets et 40 arêtes. Le dialogue montre qu'un tel graphe est forcément connexe, ni biparti ni régulier et ne possède pas de couplage parfait. Dans le troisième dialogue, l'utilisateur veut de plus que son graphe ait un diamètre égal à 10. Le dialogue lui signale que cela est impossible :




GRAPHS AT RANDOM - Required Properties
Number of Vertices :
19
Number of Edges :
40
Hamiltonian
With Perfect Matching
Connected
With Diameter
Regular
Of Degree
Bipartite
OK
Cancel
GRAPHS AT RANDOM - Required Properties
Number of Vertices :
19
Number of Edges :
40
Hamiltonian
With Perfect Matching
Connected
With Diameter
10
Regular
Of Degree
Bipartite
OK
Cancel
Your request is not available :
If D is the diameter of an hamiltonian graph of order n, then D is less than or equal to n/2.

- figure 34 -

2.4 L'algorithme de génération

Caractéristiques des propriétés

Le générateur utilise les connaissances suivantes sur les propriétés traitées :

- Complexité d'un algorithme permettant de tester la propriété.

Exemple : il est par exemple très rapide de vérifier si le graphe généré est bien connexe, mais pas s'il possède un cycle hamiltonien.

- Existence d'un fonction à seuil sur le nombre d'arêtes [Web, Bol].

Exemple : si un graphe G est généré avec un probabilité uniforme sur $\mathcal{G}_{n,g(n)}(\varnothing)$, et si $g(n) = \frac{n}{2}\log(n) + \varphi(n)$, avec $\varphi(n)$ tendant vers l'infini (aussi lentement que l'on veut) quand n tend vers l'infini, alors la probabilité que G soit connexe tend vers 1 quand n tend vers l'infini [Erd].

- Propriétés des graphes partiels.

Exemple : le graphe partiel d'un graphe régulier de degré r est de degré maximum au plus r.

- Existence d'un graphe partiel particulier.

Exemple : un graphe est connexe si et seulement si il possède un arbre couvrant.

Exemple d'application : Si l'utilisateur désire un graphe connexe dont le nombre d'arêtes est supérieur ou égal à $\frac{n}{2}\log(n) + \varphi(n)$ (φ étant nulle dans la version actuelle), Cabri-graphes générera un graphe de $\mathcal{G}_{n,m}(\varnothing)$ avec une probabilité uniforme sur cet ensemble et testera à la fin de la construction si le graphe généré est effectivement connexe. Si ce n'est pas le cas, il procédera à un nouveau tirage et ce jusqu'à l'obtention d'un graphe connexe. Par contre, si le nombre d'arêtes est inférieur à $\frac{n}{2}\log(n) + \varphi(n)$, l'algorithme générera un arbre à

n sommets avec une probabilité uniforme, puis rajoutera le nombre d'arêtes nécessaire. Ainsi, si m est supérieur à $\frac{n}{2}\log(n) + \varphi(n)$ ou égal à n-1, les graphes de $\mathcal{G}_{n,m}(\mathcal{P})$, où $\mathcal{P}$ se limite à la connexité, sont générés avec une probabilité uniforme.

Fonctions intermédiaires

Le générateur utilise les fonctions suivantes :

Rand(n) (non décrite) qui renvoie un entier compris entre 0 et n selon une loi de probabilité uniforme,

Permutation(n) qui génère aléatoirement avec équiprobabilité un élément de l'ensemble des permutations sur [0..n-1],

Prüfer(G) qui génère aléatoirement et selon une loi de probabilité uniforme un arbre à n sommets étiquetés entre 0 et n-1.

Arête(n, m) qui à toute paire {n, m} d'entiers associe un entier tel que :

$$\text{Arête}(n, m) > \text{Arête}(n', m') \Leftrightarrow \begin{cases} \max(n, m) > \max(n', m') \\ \text{ou} \\ \max(n, m) = \max(n', m') \text{ et} \\ \min(n, m) > \min(n', m'). \end{cases}$$

Sommets(a) qui à tout entier a associe le couple d'entiers (n, m) tel que Arête(n, m) = Sommets(a) et n < m.

Arête et Sommets permettent de coder les paires d'entiers sur [0..n-1] par des entiers sur $[0..\frac{n(n-1)}{2} - 1]$ sans connaître la valeur de n.

Permutation : une fonction(n : entier) → un tableau sur [0..n-1] d'entiers sur [0..n-1];
{Tire une permutation de [0..n-1] selon une loi de probabilité uniforme [RND pp 171]*}*

lexique
permuter : l'action(la donnée-résultat T : un tableau sur [0..n-1] d'entiers)
lexique
cpt : un entier sur [0..n-1]
algorithme
{ei : le tableau T est initialisé}
cpt parcourant [0..n-1] en sens inverse
$T_{Rand(cpt)} \leftrightarrow T_{cpt}$ *{échange de* $T_{Rand(cpt)}$ *et* T_{cpt}*}*
{ef : le tableau T est une permutation du tableau initial}

réalisation
Permutation(n) : permuter([0,1,2,...,n-1])

La fonction Prüfer est basée sur le codage, dit de Prüfer, des arbres à n sommets étiquetés entre 0 et n-1 par des (n-2)-uplets d'entiers sur [0..n-1] [Prü, HP pp 21-22]. Un algorithme, de complexité en O(n) pour le décodage d'un (n-2)-uplet est présentée dans [NW pp270-273]. Celui présenté ici est quelque peu différent, mais de même complexité.

Prüfer : une fonction(n : un entier > 0) → un graphe
{Tire un arbre à n sommets étiqueté entre 0 et n-1 selon une loi de probabilité uniforme}

lexique
nouveau_stable : une fonction(n : une entier) → un graphe
{construit un stable à n sommets étiquetés entre 0 et n-1}

plus_arête_g : une action(les données i, j : des entiers ≥ 0,
la donnée-résultat g : un graphe)
{ajoute l'arête {i, j} au graphe g. spécifications semblables à plus_arête (cf Chapitre 1 §1.9), mais agit sur g au lieu de le_graphe}

produire_code : l'action(le résultat code : un tableau sur [0..n-3] d'entiers sur [0..n-1])
{produit le code de Prüfer d'un arbre à n sommets}
lexique
cpt : un entier sur [0..n-1]
algorithme
{ei : ∀}
cpt parcourant [0..n-3]
$code_{cpt}$ ← Rand(n-1)
{ef : code est initialisé}

produire_décode : l'action(
la donnée code : un tableau sur [0..n-3] d'entiers sur [0..n-1],
le résultat décode : un tableau sur [0..n-1] d'entiers sur [0..n-1])
{calcule le décode d'un code de Prüfer}
lexique
cpt, place : un entier sur [0..n-1]
ordre : un tableau sur [0..n-1] d'entiers sur [-1..n-1]
algorithme
{ei : code est initialisé}
cpt parcourant [0..n-1]
$ordre_{cpt}$ ← -1
$décode_{cpt}$ ← -1
cpt parcourant [0..n-3]
$ordre_{code_{cpt}}$ ← cpt
place ← 0

```
        cpt parcourant [0..n-1]
          selon ordre_cpt
            = -1 :
              tant que décode_place ≠ -1 place ← place + 1
              décode_place ← cpt
            ≠ -1 :
              selon décode_ordre_cpt+1
                = -1 :
                  décode_ordre_cpt+1 ← cpt
                ≠ -1 :
                  tant que décode_place ≠ -1 place ← place + 1
                  décode_place ← cpt
      {ef : code et décode sont initialisés}

  ajouter_arbre : une action(la donnée-résultat g : un graphe)
    lexique
      c : un tableau sur [0..n de g -3] d'entiers sur [0..n de g -1]
      d : un tableau sur [0..n de g -1] d'entiers sur [0..n de g -1]
      cpt : un entier sur [0..n de g -1]
      {n de g désigne le nombre de sommets de g (cf  Chapitre 1 §1.9)}
    algorithme
    {ei : g est initialisé}
      code(c)
      décode(c,d)
      cpt parcourant [0..n de g - 3]
        plus_arête_g(c_cpt, d_cpt, g)
      plus_arête_g(d_n de g -2, d_n de g -1, g)
    {ef : on a rajouté les arêtes d'un arbre couvrant à g}

réalisation de Prüfer
  Prüfer(n) : ajouter_arbre(nouveau_stable(n))


Arête : une fonction(n1, n2: des entiers ≥ 0) → un entier ≥ 0

réalisation
Arête(n1,n2) : si n1 < n2 alors (n2*(n2-1))/2 + n1
                sinon (n1*(n1-1))/2 + n2
```

Sommets : une fonction(a: un entier ≥ 0) $\rightarrow$ un tableau sur [0..1] d'entiers ≥ 0

lexique

n : l'entier $\left\lfloor \frac{1+\sqrt{1+8*a}}{2} \right\rfloor$

réalisation

Sommets(a) : [a - (n*(n-1))/2 , n]

L'algorithme

L'algorithme de génération aléatoire se divise en trois étapes :

- une étape d'initialisation d'un graphe G à n sommets, et de tableaux servant à l'adjonction des arêtes et au contrôle des propriétés,
- une boucle où l'on rajoute des arêtes à G jusqu'à ce qu'il possède m arêtes où qu'on ne puisse lui rajouter une arête sans que l'une des propriétés demandées ne soit violée,
- une étape de validation où l'on vérifie que G possède bien les propriétés demandées.

Étape d'initialisation

Si G doit être connexe, sans être obligatoirement hamiltonien, régulier, de diamètre D fixé, ou posséder un couplage parfait, et que $m < \frac{n}{2} \log(n)$, on initialise G à un arbre à n sommets étiquetés de 0 à n-1, à l'aide de la fonction Prüfer.

Sinon, le graphe G est initialisé à un stable à n sommets étiquetés de 0 à n-1, puis selon les propriétés désirées :

- s'il doit être hamiltonien, on ajoute à G un cycle de longueur n;
- s'il doit posséder un couplage parfait, sans être forcément hamiltonien, on ajoute à G un ensemble de $\frac{n}{2}$ arêtes disjointes.

Le cycle ou le couplage parfait sont fabriqués à l'aide d'une permutation Permutation(n).

Le tableau E1 est initialisé à Permutation($\frac{n(n-1)}{2}$). Un deuxième tableau E2, permutation réciproque de E1, permet de modifier en temps constant un élément quelconque de E1.

têteE1 désigne l'indice du premier élément de E1 $\neq -1$, cardE1 le nombre d'éléments de E1 $\neq -1$. Marquer une paire {v,w} signifie : s'il existe un élément de E1 égal à Arête(v,w), i.e. si l'élément $E1_{E2_{Arête(v, w)}}$ est $\neq -1$, alors le mettre à -1 et mettre à jour les variables têteE1 et cardE1.

Si G doit être biparti, ses composantes connexes sont bicoloriées, la $i^{ème}$ étant coloriée à l'aide des couleurs 2i-1 et 2i. Chaque sommet colorié par k est ajouté à une liste $couleur_k$. On marque toute paire {v, w} telle que v et w appartiennent à une même liste $couleur_k$.

Si G doit avoir un diamètre D donné, la matrice M des distances entre les sommets est calculée. Deux sommets p_0 et p_1 sont choisis avec équiprobabilité, p_0 parmi tous les sommets de G, p_1 parmi les sommets à distance au moins D de p_0. On marque toute paire {v, w} telle que
$d(v, p_0) + d(w, p_1) < D-1$ ou $d(v, p_1) + d(w, p_0) < D-1$.

Boucle d'adjonction d'arêtes

Le test de sortie de boucle est "|E| + cardE1 < m ou |E| = m" où E désigne l'ensemble des arêtes déjà présentes dans G.

La boucle d'adjonction d'arêtes commence par le tirage d'une arête, i.e. par ajouter l'arête $\{Sommets(E1_{têteE1})_0, Sommets(E1_{têteE1})_1\}$ à G. L'adjonction d'une arête ne sera jamais remise en cause. Lorsqu'une arête {v, w} est rajoutée, elle

est marquée ainsi que les arêtes de E1, dont l'ajout rendrait G tel, qu'il ne serait plus un graphe partiel d'un graphe de $\mathcal{G}_{n,m}(\mathcal{P})$:

Si G doit être biparti et si v et w n'appartenaient pas à la même composante connexe, il est nécessaire de recolorier la composante connexe de v à l'aide des couleurs ayant servies à colorier la composante connexe de w. Soit k la couleur de v et k' l'autre couleur utilisée pour bicolorier sa composante connexe (k' = k - 1 + 2*(k mod 2)). Soit j la couleur de w et j' l'autre couleur utilisée pour bicolorier sa composante connexe. Les sommets appartenant à kouleur_k sont recoloriés par la couleur j' et les sommets appartenant à $\text{kouleur}_{k'}$ par la couleur j. Toute paire {x, y} telle que x appartient à la liste kouleur_k et y à la liste $\text{kouleur}_{j'}$ est marquée, ainsi que toute paire {x', y'} telle que x' appartient à la liste $\text{kouleur}_{k'}$ et y à la liste kouleur_j. Enfin, les listes kouleur_k et $\text{kouleur}_{j'}$ d'une part, et les listes $\text{kouleur}_{k'}$ et $\text{kouleur}_{j'}$ d'autre part, sont concaténées.

Si G doit être de diamètre D fixé, il faut mettre à jour la matrice des distances M :

1) $M_{v,w} \leftarrow 1$.

2) pour tout sommet x différent de v et w, faire

$$M_{x,v} \leftarrow \min(M_{x,v}, M_{x,w} + 1)$$

$$M_{x,w} \leftarrow \min(M_{x,w}, M_{x,v} + 1)$$

3) pour toute paire de sommets {x, y} telle que $\{x, y\} \cap \{v, w\} = \varnothing$ faire

si $M_{x,v} + M_{y,w} < D\text{-}1$ ou $M_{x,w} + M_{y,v} < D\text{-}1$ alors marquer {x, y}.

Si G doit être régulier de degré r et si le degré de v, après adjonction de {v, w} est égal à r, alors on marque toutes les paires {v, x} où x désigne un sommet différent de v. De même, si le degré de w est égal à r, alors on marque toutes les paires {w, x} où x désigne un sommet différent de w.

Étape de validation

Il faut tester si G possède m arêtes et si ces propriétés étaient demandées, qu'il est bien connexe, régulier et de diamètre D.

Si ce n'est pas le cas, il faut recommencer à l'étape d'initialisation.

Remarques

Tout graphe appartenant à $\mathcal{G}_{n,m}(\mathcal{P})$ possède une probabilité non nulle d'être généré. Il suffit pour cela que ses arêtes soient les m premières de E1. Dans certains cas, les graphes sont de surcroît générés avec une probabilité uniforme, par exemple si $\mathcal{P} = \emptyset$, $\mathcal{P}$ = {connexe} avec m = n-1 ou $m \geq \frac{n}{2} \log(n)$, $\mathcal{P}$ = {régulier} etc.

La complexité de cet algorithme ne peut être évaluée qu'en termes de temps d'exécution moyen. En effet, le temps d'obtention d'un graphe, par exemple connexe tel que $m \geq \frac{n}{2} \log(n)$ peut être infini. La complexité moyenne dépendra de la probabilité qu'un graphe soit rejeté pendant la boucle d'adjonction d'arêtes ou lors de l'étape de validation. L'obtention d'un graphe, correct ou non, étant en $O(n^2)$, et si l'on note par $p(n, m, \mathcal{P})$ la probabilité qu'un graphe soit accepté, il est clair que la complexité moyenne pour l'obtention d'un graphe est en

$$O\left(\frac{n^2}{p(n, m, \mathcal{P})}\right).$$

Les deux tableaux ci-dessous donnent :

1) sur 25 essais, le nombre de graphes acceptés par l'étape de validation possédant 30 sommets, m arêtes et un diamètre D. La lettre R signifie que le dialogue a détecté une impossibilité :

m	44	87	131	174	218	261	305	348
D = 3	0	5	25	25	25	25	25	25
D = 6	0	14	22	17	14	16	2	R
D = 10	3	3	2	0	0	R	R	R

2) sur 25 essais, le nombre de graphes acceptés par l'étape de validation possédant n sommets et réguliers de degré r :

n	30	50	80
r = 3	15	16	17
r = 5	14	12	18
r = 10	6	4	3

ANNEXE 1

THÉORÈMES UTILISÉS PAR LE GÉNÉRATEUR DE GRAPHES ALÉATOIRES

Compatibilité des propriétés choisies

Ces théorèmes sont utilisés pour spécifier à l'utilisateur si l'ensemble $\mathcal{G}_{n,m}(\mathcal{P})$ est vide. n désigne le nombre de sommets désirés, m le nombre d'arêtes, d le degré si G doit être régulier, D le diamètre si sa valeur est fixée.

1) $m \leq n(n-1)/2$

2) connexe $\Rightarrow m \geq n-1$

3) connexe et régulier et $n \geq 3 \Rightarrow d \geq 2$

4) $D \leq n-1$

5) $D \leq m$

6) hamiltonien $\Rightarrow D \leq n/2$

7) régulier de degré 2 et connexe $\Rightarrow D = \left\lfloor \frac{n}{2} \right\rfloor$

8) régulier de degré $\geq 3 \Rightarrow n \leq \frac{d(d-1)^D - 2}{d - 2}$ (théorème de Moore, [BB]).

9) régulier de degré 2, biparti et connexe $\Rightarrow n = 2D$

10) régulier de degré ≥ 2 et biparti $\Rightarrow n \leq \frac{2(d-1)^D - 1}{d - 2}$ (théorème de Moore [BB])

11) $m \leq D + (n - D - 1)(n - D + 4)/2$ ([BB])

12) $n \geq 2 \Rightarrow D \geq 1$

13) $D = 1 \Rightarrow m = n(n-1)/2$

14) biparti et $m = \frac{n^2}{4} \Rightarrow D = 2$

15) biparti et $D = 2 \Rightarrow \exists\, k,\ m = k(n-k)$

16) couplage parfait $\Rightarrow m \geq n/2$

17) couplage parfait et régulier $\Rightarrow d \geq 1$

18) couplage parfait $\Rightarrow$ n pair

19) hamiltonien $\Rightarrow m \geq n$

20) hamiltonien $\Rightarrow n \geq 3$

21) hamiltonien et biparti $\Rightarrow$ n pair

22) hamiltonien et régulier $\Rightarrow d \geq 2$

23) régulier $\Rightarrow \exists\, k,\ 2m = kn$

24) régulier $\Rightarrow$ d ou n pair

25) $d < n$

26) $2m = dn$ (redondant avec d ou n pair si m connu)

27) régulier $\Rightarrow \exists\, k,\ 2m = kd$

28) régulier $\Rightarrow 2m \leq d(d+1)$

29) biparti et régulier $\Rightarrow$ n pair ou $d = 0$

30) biparti et régulier $\Rightarrow 2d \leq n$

31) biparti $\Rightarrow m \leq \left\lfloor \frac{n}{2} \right\rfloor \left(n - \left\lfloor \frac{n}{2} \right\rfloor \right)$

Invalidation des propriétés

Ces théorèmes sont utilisés pour rendre non paramétrable l'existence d'une propriété ou la valeur d'une variable, lorsque celles-ci sont déja déterminées par l'ensemble des propriétés déjà choisies. On remarquera le nombre de théorèmes spécifiques aux petits graphes ($n \leq 2$).

1) hamiltonien $\Rightarrow$ connexe

2) $D = 0 \Rightarrow$ $n = 1$, $m = 0$, régulier de degré 0, biparti, ¬couplage parfait, ¬hamiltonien

3) régulier et biparti $\Rightarrow$ couplage

4) régulier de degré 0 $\Rightarrow$ m = 0, biparti, $\neg$couplage parfait, $\neg$hamiltonien

5) régulier de degré 1 $\Rightarrow$ couplage parfait, biparti, $\neg$hamiltonien

6) n = 1 $\Rightarrow$ m = 0, régulier de degré 0, D = 0, connexe, $\neg$couplage parfait

7) $n \leq 2 \Rightarrow$ biparti, régulier, $\neg$hamiltonien

8) n = 2 et (m = 1 ou connexe) $\Rightarrow$ couplage parfait, D = 1 *à voir*

9) n impair $\Rightarrow$ $\neg$couplage parfait

10) n impair et hamiltonien $\Rightarrow$ $\neg$biparti

11) n pair et hamiltonien $\Rightarrow$ couplage parfait

12) $n > 2$ et régulier de degré $\geq n/2 \Rightarrow$ hamiltonien, connexe

13) $n > 2$ et n impair et régulier de degré $\geq n/2 \Rightarrow$ $\neg$couplage parfait

14) D = 1 $\Rightarrow$ $m = n(n-1)/2$, régulier de degré n-1

15) D = 1 et n pair => couplage parfait

16) D = 1 et $n \geq 3 \Rightarrow$ hamiltonien, $\neg$biparti

17) $2m \neq kn \Rightarrow$ $\neg$régulier

18) $m \leq n - 2 \Rightarrow$ $\neg$connexe

19) m = n - 1 et connexe $\Rightarrow$ biparti

20) $m \leq n-1 \Rightarrow$ $\neg$hamiltonien

21) $m > (n-1)(n-2)/2 \Rightarrow$ connexe

22) $m > (n-1)(n-2)/2 + 1 \Rightarrow$ hamiltonien

23) $m = n(n-1)/2 \Rightarrow$ régulier de degré n-1, D = 1, connexe

24) $m = n(n-1)/2$ et n pair $\Rightarrow$ couplage parfait

25) $m = n(n-1)/2$ et $n \geq 3 \Rightarrow$ $\neg$biparti

26) m = 0 $\Rightarrow$ régulier de degré 0, biparti

27) $m \leq 2 \Rightarrow$ biparti

28) $m > \frac{n^2}{4} \Rightarrow$ $\neg$biparti

29) $m = \frac{n^2}{4}$ et biparti $\Rightarrow$ hamiltonien, connexe, D = 2

30) $m = \frac{n^2}{4}$ et biparti et n pair $\Rightarrow$ régulier de degré n/2, couplage parfait

31) 2m = n et couplage parfait $\Rightarrow$ régulier de degré 1, biparti

32) $m > \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor$ et biparti $\Rightarrow$ connexe

33) m = n, connexe, régulier $\Rightarrow$ hamiltonien, $D = \left\lfloor \frac{n}{2} \right\rfloor$

34) m = n, connexe, régulier $\Rightarrow$ (n pair $\Leftrightarrow$ couplage parfait, biparti)

35) m = n et hamiltonien $\Rightarrow$ régulier de degré 2, $D = \left\lfloor \frac{n}{2} \right\rfloor$, (n pair $\Leftrightarrow$ biparti)

Traitement

Ces théorèmes sont utilisés pour donner une liste de propriétés équivalente à celle demandée par l'utilisateur, mais plus facile à utiliser par l'algorithme.

1) régulier de degré $\geq$ n/2 $\Rightarrow$ hamiltonien $\leftarrow$ vrai

2) (hamiltonien et m = n) ou (régulier de degré 2 et connexe) $\Rightarrow$
régulier $\leftarrow$ faux, biparti $\leftarrow$ faux, connexe $\leftarrow$ faux, hamiltonien $\leftarrow$ vrai, diamètre $\leftarrow$ *

3) biparti et régulier de degré non nul $\Rightarrow$ couplage parfait $\leftarrow$ vrai

4) $m \geq (n-1)(n-2)/2 + 1 \Rightarrow$ connexe $\leftarrow$ faux

5) hamiltonien $\Rightarrow$ connexe $\leftarrow$ faux, couplage parfait $\leftarrow$ faux

ANNEXE 2

ÉTUDE BIBLIOGRAPHIQUE SUR LA GÉNÉRATION DE GRAPHES ALÉATOIRES

Graphes simples non étiquetés

La plupart des algorithmes générant des graphes aléatoires produisent des graphes simples, étiquetés et éventuellement orientés. Néanmoins, les trois travaux suivant concernent les graphes simples non étiquetés et non orientés :

- Dixon et Wilf [DW] : tirage d'un graphe avec équiprobabilité parmi les graphes non étiquetés à n sommets.

Leur algorithme utilise un algorithme intermédiaire donnant une orbite avec équiprobabilité parmi l'ensemble des orbites d'un groupe G sur un ensemble E. Il est en $O(n^2)$ en moyenne si l'on suppose connu préalablement le nombre de graphes non étiquetés à n sommets ou si l'on accepte l'utilisation d'une approximation de ce nombre.

- Wormald [Wor1] : tirage d'un graphe avec équiprobabilité parmi

1) les graphes non étiquetés à n sommets,

2) les graphes non étiquetés à n sommets et m arêtes,

3) les graphes non étiquetés réguliers à n sommets.

Le premier algorithme est inspiré de celui de Dixon et Wilf, mais Wormald propose également un nouvel algorithme avec essais pour choisir l'orbite d'un

groupe. Il ne calcule pas le nombre de graphes non étiquetés mais utilise un approximation asymptotique. Cet algorithme est en $O(n^2)$ en moyenne et est de fait efficient pour $n \geq 50$.

Par des méthodes similaires, les deux autres algorithmes génèrent avec équiprobabilité des graphes à n sommets et m arêtes en $O(n^2 m^{1/2})$ pour presque toutes les valeurs de m et des graphes réguliers de faible degré en O(n).

- Nijenhuis et Wilf [NW1, pp 274-282] : tirage d'un graphe parmi les arborescences à n sommets non étiquetés

Cet algorithme est basé sur une formule de récurrence. Un test du χ^2 sur un échantillon de 450 arborescences montre que la distribution associée à cet échantillon ne peut être considérée comme non uniforme.

- Wilf [Wil] : tirage d'un graphe parmi les arbres à n sommets non étiquetés avec équiprobabilité.

Cet algorithme utilise le précédent, ainsi que le rapport entre le nombre d'arbres ayant 1 centre et celui des arbres en ayant 2.

Graphes simples étiquetés

Il est évidemment très facile de générer un graphe avec équiprobabilité si l'on connait un code de l'ensemble auquel il appartient similaire au codage de Prüfer pour les arbres étiquetés. Malheureusement, de tels codes sont rares. Une équipe de Münich, dirigée par G. Tinhofer, s'intéresse actuellement à une telle application du code de Cori [Cor] pour la classe des graphes planaires.

De nombreuses méthodes de génération de graphes aléatoires sont issues de travaux sur le dénombrement (exact ou asymptotique) de classes de graphes.

- Nijenhuis et Wilf [NW2] : tirage avec équiprobabilité parmi les graphes étiquetés connexes à n sommets. Cet algorithme est basé sur une formule récursive de dénombrement de ces graphes.

- Wormald [Wor2] : tirage avec équiprobabilité parmi

1) les graphes étiquetés cubiques à n sommets,

2) les graphes étiquetés à n sommets et ayant une séquence de degrés fixée à l'avance,

3) les graphes étiquetés bipartis à n sommets et ayant une séquence de degrés fixée à l'avance.

Le premier algorithme est basé sur une méthode de dénombrement exacte [Wor3]. Les deux autres utilisent des méthodes de dénombrement asymptotiques (par exemple [Bol2]) et un algorithme intermédiaire construisant une matrice en 0-1 avec des sommes sur les lignes et les colonnes fixées à l'avance. Les deux derniers algorithmes sont à essais et Wormald montre que la probabilité d'obtenir un graphe est de l'ordre de $e^{\frac{1-r^2}{4}}$ si le graphe généré doit être régulier de degré r.

- Jerrum et Sinclair [JS, Sin] : tirage avec équiprobabilité parmi les graphes étiquetés de séquence de degrés fixée à l'avance.

Leur algorithme est basé sur le concept des chaînes de Markov, et est utilisable en particulier pour des graphes réguliers de degré au plus $\sqrt{\frac{n}{2}}$. Il cite de plus des travaux non publiés sur le même problème, utilisables pour des graphes réguliers de degré au plus $n^{1/5}$ [Fri] et $n^{1/3}$ [MW, SJ].

- Guénoche [Gué2] : tirage avec équiprobabilité parmi les graphes étiquetés bipartis à n sommets et ayant une séquence de degrés fixée à l'avance.

Cet algorithme est de complexité élevée et est plutôt destiné à l'énumération de ces graphes.

Ainsi que nous l'avons fait pour Cabri-graphes, il est peut s'avérer intéressant d'utiliser pour la génération de graphes aléatoires des résultats de la théorie des graphes aléatoires. C'est le sujet d'un article de Tinhofer [Tin3].

Tinhofer a également utilisé une structure particulière de représentation des graphes pour générer aléatoirement des graphes simples étiquetés à n sommets [Tin1, Tin2] :

1) quelconques,

3) avec m arêtes,

2) connexes,

4) avec m arêtes et connexes,

5) connexes sans cycle (i.e. des arbres),

6) ayant une séquence des degrés fixée à l'avance,

7) connexes et ayant une séquence des degrés fixée à l'avance,

8) connexes sans cycle (i.e. des arbres) et ayant une séquence des degrés fixée à l'avance,

9) eulériens,

10) bipartis.

La plupart de ces algorithmes génèrent des graphes avec équiprobabilité (cas 1, 2, 3, 4, 5 et 9). Il est également possible de calculer en cours de génération la probabilité d'un graphe ou d'un graphe connexe ayant une séquence de degrés fixée. De plus, ces algorithmes peuvent facilement être modifiés pour obtenir des graphes orientés.

Guénoche [Gué1] et Läuchli [Läu] ont proposés chacun un algorithme de génération de graphes plans, permettant ainsi une génération aléatoire où chaque graphe plan a une probabilité non nulle d'être généré.

L'algorithme de Guénoche génère aléatoirement des graphes cubiques plans et utilise la dualité entre ces graphes et les graphes plans maximaux en terme de nombre d'arêtes.

L'algorithme de Läuchli est basé sur un système d'induction possédant deux règles de production et dont la base diffère selon la connectivité du graphe désirée (rappelons qu'un graphe planaire est au plus 3-connexe).

Les deux règles d'induction sont :

1) si deux sommets non adjacents appartiennent à une même face, relier ces deux sommets par une arête,

2) remplacer un sommet par une arête.

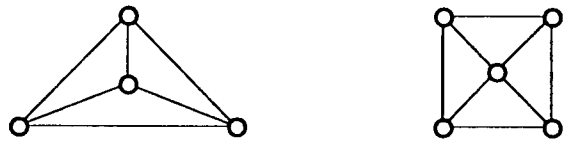
Les bases sont fonction de la connectivité désiré :

1) pour les graphes non 2-connexes à p composantes connexes, la base est formée d'un stable à p sommets,

2) pour les graphes 2-connexes, elle est formée d'un cycle de longueur 3,

3) pour les graphes 3-connexes, elle est formé de l'ensemble des roues (Wheels) à n sommets ($n \geq 4$). Une roue à n sommets est le graphe composé d'un cycle à n-1 sommets et d'un sommet relié à tous les sommets du cycle.

Roues à 4 et 5 sommets :



- figure 35 -

Pour terminer, citons un algorithme de génération de graphes k-coloriables, du à Turner [Tur], et utilisé pour tester les performances d'heuristiques de coloration de graphes. Turner montre que si les graphes ne sont pas générés avec équiprobabilité, la loi de distribution relative à cet algorithme n'est toutefois pas très éloignée de la loi uniforme.

CHAPITRE 3

AUTOUR DE CABRI-GRAPHES

Introduction

Ce chapitre présente deux études qui ont été à la fois motivées et facilitées par le projet Cabri-graphes. Il convient de mentionner dans cette introduction une troisième étude sur une conjecture d'I. Havel, effectuée par Jean-Marie Laborde et ayant fait l'objet d'un rapport de recherche [Lab2].

3.1 Planarité

Les algorithmes de test de planarité et plongement topologique

L'étude des tests de planarité de complexité linéaire, dans le cadre de Cabri-graphes, a fait l'objet d'un rapport de recherche, de deux mémoires de DEA et d'un mémoire de maîtrise MST-MLA que nous avons encadrés.

Nous avons implémenté dans Cabri-graphes trois algorithmes de test de planarité, tous trois linéaires : l'algorithme d'Hopcroft et Tarjan [HT, Deo], l'algorithme de Booth et Lueker [BL] et l'algorithme de de Fraysseix et Rosenstiehl [FR1]. Les deux derniers donnent également un plongement topologique du graphe testé si celui-ci est planaire.

Ces trois algorithmes ont été implémentés par des programmeurs différents (Hopcroft-Tarjan par H. Frydlender et A. Rimbod Pethiod [FRP] et corrigé par F. Crocombette [Cro], de Fraysseix-Rosenstiehl par J.-M. Clochard [Clo] à partir d'une implémentation sur PC de ses auteurs et Booth-Lueker par mes soins). Leurs performances, compte-tenu de la diversité des programmeurs, sont comparables.

Aussi, nous ne ferons ici qu'une comparaison des difficultés de mise en oeuvre que nous avons rencontrées, en signalant les articles et ouvrages que nous avons utilisés.

Hopcroft-Tarjan

L'algorithme d'Hopcroft et Tarjan, le plus étudié dans la littérature car le premier test de planarité linéaire existant, est celui dont l'implémentation dans Cabri-graphes est la plus courte et a également été très rapide, grâce notamment à une présentation de l'algorithme dans le livre de E. M. Reingold, J. Nievergelt

and N. Deo [RND, pp 364-385]. Cette implémentation (commentaires et décomposition en blocs compris) totalise 1300 lignes.

L'obtention d'un plongement topologique à partir de cet algorithme n'a pas été programmée. Le lecteur intéressé pourra se rapporter aux articles de D. Gries et J. Xue [GX] et J.-G. Penaud [Pen].

Si cet algorithme est le plus rapide à mettre en oeuvre, il est aussi le plus difficile à comprendre. Une présentation des concepts théoriques de cet algorithme, très appréciable pour sa compréhension, est donné dans le livre de S. Even [Eve, pp 172-180].

de Fraysseix-Rosenstiehl

L'implémentation de cet algorithme dans Cabri-graphes a été grandement facilitée par celle pour PC que nous ont communiquée ses auteurs, que je remercie ici. La longueur de l'implémentation se situe à mi-chemin entre celle d'Hopcroft-Tarjan et celle de Booth-Lueker (2500 lignes, commentaires, décomposition en blocs et obtention du plongement topologique inclus). Il en est de même pour sa compréhension. L'article présentant cet algorithme n'ayant pas été publié à ce jour, le lecteur pourra se rapporter à l'article présentant sa justification théorique [FR2] et au mémoire de J.-M. Clochard [Clo].

Booth-Lueker

Cet algorithme est une amélioration, grâce à l'utilisation d'une structure de données appelée PQ-trees, d'un algorithme quadratique, du à Lempel, Even et Cederbaum [LEC]. Il est certainement le plus facile à comprendre, car le plus naturel. Contrairement aux deux précédents, il ne déplace jamais des sommets et arêtes déjà placés les uns par rapport aux autres, mais construit de façon gloutonne un graphe plan en ajoutant les sommets un par un dans la face extérieure du graphe plan déjà construit, ceci dans un ordre appelé

st-numérotation [EvT]. Malheureusement, il est aussi le plus difficile et le plus long à programmer (3600 lignes, commentaires, décomposition en blocs et obtention du plongement topologique inclus), malgré une présentation remarquable par sa clarté. L'obtention du plongement topologique est décrite dans l'article de N. Chiba, T. Nishizeki, S. Abe and T. Ozawa [CNAO]. Pour une description plus théorique de l'algorithme, le lecteur pourra se référer au livre de S. Even [Eve] ou à notre rapport de recherche ([Bau1] et Annexe 1).

Citons pour terminer ce paragraphe la bibliographie commentée de P. Eades and R. Tamassia [EaT], dans laquelle le lecteur trouvera l'ensemble des références sur le problème du dessin automatique de graphes, planaires ou non, parus avant 1987, ainsi que le livre de N. Chiba and T. Nishizeki [NC] et l'article de T. Nishizeki [Nis] qui lui permettra de connaître l'ensemble des travaux sur cette classe de graphes importante d'un point de vue pratique que sont les graphes planaires. Nous avons également appris récemment l'existence d'un quatrième algorithme linéaire de test de planarité et de plongement, du à A.V. Karzanov [Kar].

3.2 Diamètre et excentricité des feuilles d'un arbre de plus courtes chaînes

L'algorithme BFS

Contrairement à l'algorithme de parcours en profondeur d'un graphe : DFS (Depth First Search [RND]), il n'existe pas de définition récursive de l'algorithme de parcours en largeur, appelé BFS (Breadth First Search). BFS est basé sur l'utilisation d'une file et non pas d'une pile comme DFS.

```
fonction BFS(G: graphe; v: sommet) : arborescence;
    var
        Q :       file;
        x, y :    sommet;
début
    pour tout sommet x de G faire visite(x) <- *;
    heure <- 1;
    visite(v) <- heure;
    BFS <- (V(G),∅);
    racine(BFS) <- v;
{le sommet v entre dans la file Q :}
    Q <= v;
    tant que Q non vide faire
        début
{le sommet x est initialisé au premier sommet de Q et retiré de
cette file :}
            x <= Q;
            pour tout y voisin de x faire    (B)
                si visite(y) = * alors
                    début
                        Q <= y;
                        heure <- heure + 1;
                        visite(y) <- heure;
{on ajoute l'arête xy à l'arborescence :}
                        BFS <- BFS + xy;
                    fin;
```

```
        fin;
fin;
```

On appellera arbre BFS de racine r une arborescence BFS(G, r). Cette arborescence dépend bien évidemment de l'ordre dans lequel sont considérés les voisins de x dans la boucle (B). L'ordre de visite par BFS des sommets de G sera donné par la fonction visite().

L'algorithme BFS est beaucoup moins traité dans la littérature que DFS. Les renseignements qu'il fournit concernent essentiellement des invariants liés à la notion de distance : diamètre, rayon, maille, ...

En programmant le calcul du diamètre d'un graphe, nous nous sommes intéressés aux informations que pourrait fournir BFS à partir de toutes les feuilles de l'arbre BFS d'un sommet donné. Ceci après avoir remarqué que pour de nombreux graphes, quelque soit l'arbre BFS considéré, celui-ci possédait au moins un sommet périphérique parmi ses feuilles.

Arbre de plus courtes chaînes

Soit T une arborescence couvrante d'un graphe G = (V, E) de racine r. T est un arbre de plus courtes chaînes de G si et seulement si
$\forall v \in V, d_T(v, r) = d_G(v, r)$. En d'autres termes, pour tout sommet v de G, la chaîne élémentaire de T reliant v à r est une chaîne de longueur minimum entre v et r dans G.

Soit G un graphe et T un arbre couvrant de G. On appelle coarbre de T le graphe constitué par les arêtes de G n'appartenant pas à T. Ce graphe est noté co(T).

Une caractérisation bien connue des arbres de plus courtes chaînes est la suivante : T est un arbre de plus courtes chaînes de racine r si et seulement si pour toute arête {v, v'} de co(T), $d(v', r) \in [d(v, r) - 1, d(v, r) + 1]$.

Caractérisation des arbres BFS

En particulier, un arbre BFS est un arbre de plus courtes chaînes. Il n'y a cependant pas équivalence entre les deux notions.

Voici une caractérisation connue des arbres BFS à partir de la notion d'arbre de plus courtes chaînes :

Soit T une arborescence de racine r et v un sommet de T. Un ancêtre de v est un sommet situé sur la chaîne élémentaire reliant v à r. Le père de v est le sommet à la fois ancêtre et voisin de v.

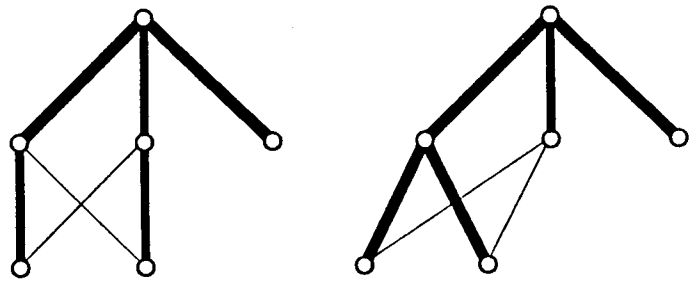
Soit T un arbre de plus courtes chaînes de G, de racine r.

T est un arbre BFS si et seulement si il existe une numérotation des sommets num_BFS telle que :

1) $d(v, r) < d(v', r) \Rightarrow num\_BFS(v) < num\_BFS(v')$

2) $(\{v, v'\} \in co(T)$ et $d(v, r) = d(v', r) + 1)$
$\Rightarrow num\_BFS(pv) < num\_BFS(v')$ où pv désigne le père de v.

Arbre de plus courtes chaînes non BFS et arbre BFS. Les arêtes des arbres sont épaisses, celles des coarbres fines :



- figure 36 -

Théorèmes

Soit T un arbre de plus courtes distances d'un graphe G. On note par D(G) le diamètre de G et par $\delta(G,T)$ la valeur $\max\{d(f, x), f \in F \text{ et } x \in V\}$ où F désigne l'ensemble des feuilles de T.

Théorème 1 : $1 \leq D(G) / \delta(G,T) \leq 4/3$

Preuve

Nous allons également montrer que les bornes de cette inégalité sont les meilleures possibles.

1) $1 \leq D(G) / \delta(G,T)$

Trivial car $F \subseteq V$. La borne est atteinte lorsque l'on considère un arbre de plus courtes chaînes ayant pour racine un sommet périphérique de G.

2) $D(G) / \delta(G,T) \leq 4/3$

Soit a et b deux sommets tels que $d(a, b) = D(G)$. Soit f_a (resp. f_b) une feuille de T ayant a (resp. b) pour ancêtre. On a :

$d(a, b) \leq d(a, f_a) + d(f_a, b)$

donc $d(a, b) \leq d(a, f_a) + \delta(G,T)$ (1)

de même, $d(a, b) \leq d(b, f_b) + \delta(G,T)$ (2)

1^{cr} cas : $d(a, f_a) \leq 1/3\ \delta(G,T)$ ou $d(b, f_b) \leq 1/3\ \delta(G,T)$

par (1) et (2) on a : $d(a, b) \leq 4/3\ \delta(G,T)$

$2^{ème}$ cas : $d(a, f_a) > 1/3\ \delta(G,T)$ et $d(b, f_b) > 1/3\ \delta(G,T)$

$d(f_a, r) = d(f_a, a) + d(a, r)$ car a est sur une chaîne de longueur minimum entre f_a et r.

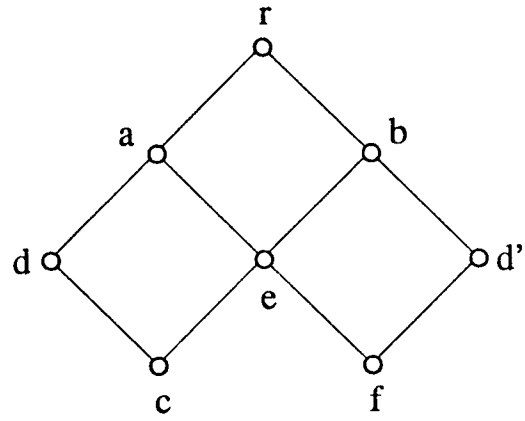
donc $d(a, r) \leq \delta(G,T) - d(f_a, a)$

comme $d(a, f_a) > 1/3\ \delta(G,T)$, $d(a, r) < 2/3\ \delta(G,T)$

Par le même raisonnement, on a $d(b, r) < 2/3\ \delta(G,T)$

Comme $d(a, b) \leq d(a, r) + d(b, r)$, on obtient $d(a, b) < 4/3\ \delta(G,T)$ ◊

Considérons G le graphe suivant :



et G^k le graphe obtenu en subdivisant k fois chaque arête de G. Notons par $C_{u,v}$ la chaîne de G^k provenant de la subdivision de l'arête (u, v) de G. Soit T^k l'arbre de plus courtes chaînes de G^k obtenu en supprimant de G^k l'arête de $C_{b,e}$ incidente à e, l'arête de $C_{e,c}$ incidente à c et l'arête de $C_{e,f}$ incidente à f.

$$\forall\ k,\ \delta(G^k,T^k) = \frac{4}{3} D(G^k).$$

En effet, les seuls sommets périphériques de G^k sont d et d', d'excentricité 4k. D'autre part l'excentricité des sommets c et f est égale à 3k, celle des voisins de c et f respectivement situés sur les chaînes $C_{e,c}$ et $C_{e,f}$ est égale à 3k-1 et celle du sommet voisin de e situé sur la chaîne $C_{b,e}$ est égale à 2k+1. Par conséquent, $\delta(G^k,T^k) = 3k$.

Remarque 1

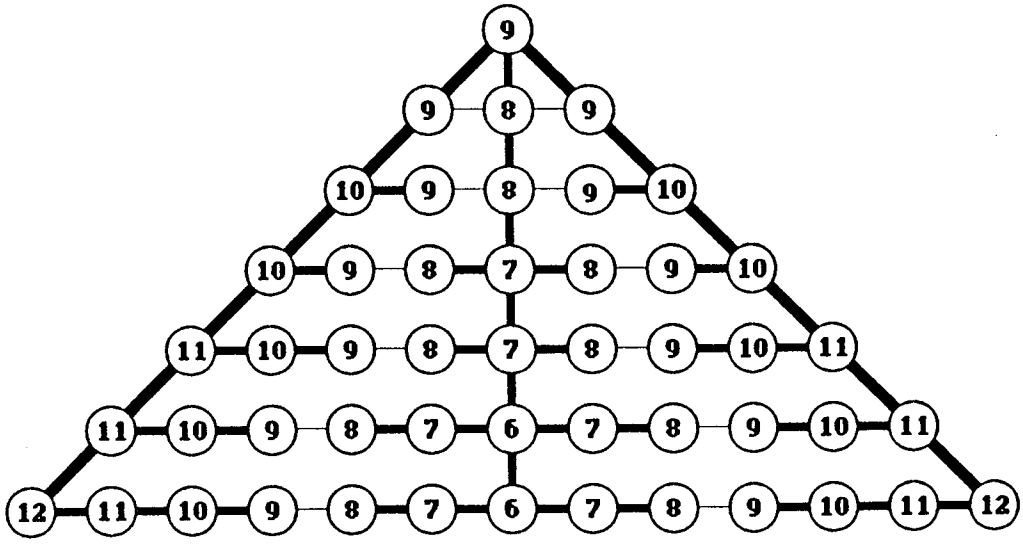
T^k n'est pas un arbre BFS de G^k et G^k ne possède pas d'arbre BFS B tel que $\delta(G^k,B) = \frac{4}{3} D(G^k)$. Il existe cependant des arbres BFS pour lesquels cette borne

est également atteinte. Le plus petit exemple, en terme de nombre de sommets, de famille de graphes que nous ayons trouvée est la suivante :

Le graphe G^k est composé de k+1 chaînes de longueur 2i, i variant de 0 à k (la chaîne de longueur 0 étant le graphe à un sommet K_1). De plus, on relie le premier sommet (resp. le sommet du milieu, le dernier sommet) de chaque chaîne au premier sommet (resp. au sommet du milieu, au dernier sommet) des chaînes de longueur immédiatement supérieure et inférieure. Les seuls sommets périphériques de G^k sont les sommets extrémaux de la chaîne de longueur 2k qui sont à distance 2k.

Soit T^k l'arbre BFS de G^k défini de la façon suivante :
la racine de T^k est le sommet composant la chaîne de longueur 0;
les arêtes de T^k sont d'une part les arêtes reliant les chaînes entre elles, d'autre part les $\lfloor i/2 \rfloor$ premières, $\lfloor i/2 \rfloor$ dernières arêtes et $2\lfloor (i-1)/2 \rfloor$ arêtes centrales de la chaîne de longueur 2i, pour i variant de 1 à k. On a ainsi $\forall$ k, $\delta(G^k,T^k) = \left\lfloor \frac{3}{2} k \right\rfloor$ et pour tout k pair non nul, $D(G^k) = \frac{4}{3} \delta(G^k,T^k)$.

Le graphe G^6. L'arbre T^6 est représenté par les arêtes en gras, les sommets étant étiquetés par leur excentricité dans G^6 :



- figure 37 -

Remarque 2

Nous nous sommes également intéressés aux rapports entre D(G) et β(G,T), entre r(G) et δ(G,T) et entre r(G) et β(G,T), où $\beta(G,T) = \min_{f \in F} \max_{x \in V} d(f, x)$, et r(G) désigne le rayon de G. Les encadrements trouvés sont beaucoup moins surprenant que la borne supérieure du théorème 1. On a :

Théorème 2 :

$1 \leq D(G) / \beta(G,T) \leq 2$ (1)

$1/2 \leq r(G) / \delta(G,T) \leq 1$ (2)

$1/2 \leq r(G) / \beta(G,T) \leq 1$ (3)

Preuve :

les bornes supérieures de (2) et (3) et la borne inférieure de (1) découlent des inégalités suivantes :

$r(G) \leq \beta(G,T) \leq \delta(G,T) \leq D(G)$. (a)

Les bornes inférieures de (2) et (3) ainsi que la borne supérieure de (1) sont obtenues en utilisant (a) et l'inégalité suivante :

$$\forall x, \forall y, \max_{z \in V} d(x,z) \geq \frac{1}{2} \max_{z \in V} d(y,z) \quad \text{(b)}$$

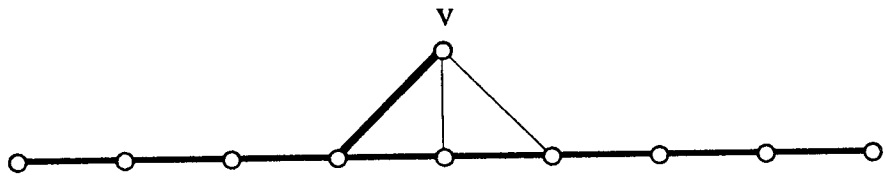
Les bornes de (1), (2) et (3) sont les meilleures possibles :

- considérons un cycle C et T un arbre BFS quelconque de C. On a : $r(C) = \beta(C,T) = \delta(C,T) = D(C)$, ce qui prouve que les bornes supérieures de (2) et (3) et la borne inférieure de (1) sont les meilleures possibles;

- considérons un chaîne de longueur paire C et T l'arbre BFS ayant pour racine le sommet milieu de C. On a : $2r(C) = \beta(C,T) = \delta(C,T)$, ce qui prouve que les bornes inférieures de (2) et (3) sont les meilleures possibles;

- considérons le graphe G formé d'un chaîne C de longueur paire et d'un sommet v relié au centre de la chaîne et à ses deux voisins. Soit T l'arbre BFS de ce graphe ayant pour racine l'une des extrémités de la chaîne, et dont les arêtes sont les arêtes de C, plus l'arête reliant v à son voisin le plus proche de la racine de T (figure 38). On a $D(G) = 2\,\beta(G,T)$, ce qui prouve que la borne supérieure de (1) est la meilleure possible.

Graphe G et arbre BFS T tel que $D(G) = 2\,\beta(G,T)$. L'arbre T est représenté par les arêtes épaisses :



- figure 38 -

ANNEXE 1

ALGORITHMES LINÉAIRES DE PLONGEMENT DE GRAPHES PLANAIRES

Cette annexe a été publiée sous la forme d'un rapport de recherche [Bau1].

Elle présente tout d'abord l'algorithme de test de planarité de Booth et Lueker sous une forme originale, ainsi que la modification apportée par Chiba, Nishizeki, Abe et Ozawa pour l'obtention d'un plongement topologique.

Elle propose également une modification de cet algorithme permettant de tester l'existence d'un graphe plan possédant certaines rotations fixées à l'avance. Ce problème peut être résolu de façon plus simple en rajoutant quelques arêtes au graphe avant le test, mais la modification proposée ici montre que l'on peut obtenir un résultat équivalent en testant si une permutation appartient à la frontière d'un PQ-arbre et en prenant en compte la direction d'indicateurs.

A LINEAR ALGORITHM FOR EMBEDDING PLANAR GRAPHS, WHERE CERTAIN ROTATIONS ARE ARBITRARILY FIXED

Olivier Baudon
LSD-IMAG
BP 53X, 38041 Grenoble Cedex
France

ABSTRACT

This paper deals with planarity testing and embedding planar graphs. In particular, we study an algorithm for embedding a planar graph with some arbitrarily fixed rotations.

INTRODUCTION

The first known linear time algorithm which tests planarity is due to Hopcroft and Tarjan [1,2]. Unfortunately, it is difficult, using this algorithm, to embed a graph in linear time. Rosenstiehl and de Fraysseix proposed a second linear time algorithm [3], with which it is easier to embed graphs.

There is a third linear time algorithm called "PQ-tree algorithm", due to Booth and Lueker [4] which is a linearization of the planarity testing algorithm proposed by Lempel, Even and Cederbaum [5]. Chiba and al. in [6] showed how to obtain one or all the embeddings of a graph, using PQ-trees. In this paper, we use this algorithm to compute an embedding where some rotations (see below) are fixed.

PART I : PLANAR GRAPHS AND PQ-TREES

In this section, we present the theorical background of the PQ-tree algorithm. For graph concepts, we use definitions similar to those found in any text on graph theory e.g. [8,9]. For terminology and more informations about the data structure "PQ-tree", we advise the reader to refer back to the original paper [4].

definition 1 : A graph G is planar iff there exists a mapping of the vertices and edges into the plane such that :
(1) each vertex is mapped into a distinct point ;
(2) each edge (v, w) is mapped onto a simple curve, with the vertices v and w mapped onto the endpoints of the curve ;
(3) mapping of distinct edges have only the mappings of their common endpoints in common.

theorem 1 [15] : An embedding of a graph G=(V,E) is uniquely determined by the family $\{r(v)\}_{v\in V}$ of rotations. Each r(v) is a circular list such that there exists a mapping where for each v, r(v) is the clockwise order of the neighbours of v. By embedding a planar graph, we mean to build such a family $\{r(v)\}_{v\in V}$.

As multiple edges, loops, or directed edges have no incidence on the planarity of a graph, in what follows, we consider only simple undirected graphs.

theorem 2 : A graph G is planar iff each of its 2-connected components is planar.

As a consequence, in planarity testing, one considers only 2-connected graphs. Linear time algorithms for determining 2-connected components of a graph are well known (see [10,11]).

definition 2 [5] : Let (s,t) be an edge of a graph G=(V,E). A 1-1 function g from V to $\{1, \ldots, |V|\}$ is called an st-numbering iff
(1) g(s)=1 and g(t)=|V| ;
(2) every vertex v different from s and t, has two neighbours w and w' such that $g(w) < g(v) < g(w')$.

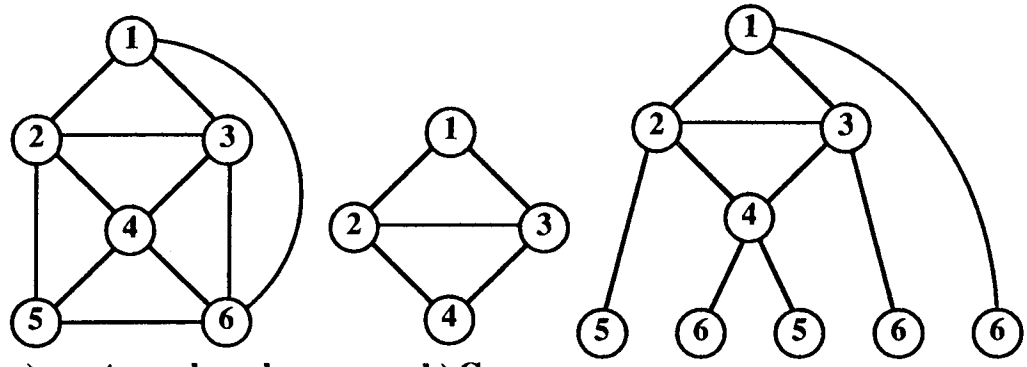
theorem 3 [5] : A graph G is 2-connected iff for every edge (s,t) of G, there exists an st-numbering.

Algorithms computing st-numbering are described in [11,12].

In the following, G=(V,E) denotes an st-numbered graph. It means that $V= \{1, \ldots, n\}$ and the identity on V is a 1n-numbering. V_k denotes $\{1, \ldots, k\}$, W_k the set of vertices of V_k having neighbours in $\overline{V_k} = \{k+1, \ldots, n\}$, and $G_k=(V_k, E_k)$ the subgraph of G induced by V_k.

theorem 4 [5] : G is planar iff for every $k \in \{1, \ldots, n-1\}$, there exists an embedding of G_k such that vertices of W_k are on the same face (this face will be considered afterwards as the external face).

definition 3 [6] : The bush form of G_k, denoted by B_k, is the graph obtained from G_k by adding pendent edges, each of which corresponding to an edge of G having exactly one endpoint in V_k. These pendant edges are called virtual, as their endpoints in $\overline{V_k}$.



a) an st-numbered graph G **b) G_4** **c) B_4**

- figure 1 -

Let $R(B_k)$ be the set of embedding of B_k such that the virtual vertices are on the external face. By convention, we draw the virtual vertices on an horizontal line, below G_k, as in figure 1c. If b is such an embedding, $\pi(b)$ denotes the permutation of virtual edges obtained by reading them from left to right. $R'(B_k)$ denotes the subset of $R(B_k)$ consisting of embeddings b such that the virtual vertices labeled k+1 appear consecutively in $\pi(b)$.

theorem 5 [5] : There is a 1-1 correspondence between the set of embeddings of G_{k+1} such that vertices of W_{k+1} are in the same face, and $R'(B_k)$. In fact, an embedding of G_{k+1}, where vertices of W_{k+1} are on the same face, is obtained from an embedding of $R'(B_k)$ by identify the virtual vertices labeled k+1.

definition 4 [4] : A PQ-tree is a rooted tree with three types of nodes : P-nodes, Q-nodes and leaves. The difference between a P and a Q-node is that the set of children of a Q-node is linearly ordered. These lists can be read in either forward or reversed order. Thus, respecting the order on the set of children of a Q-node means to consider this set in either forward or reversed order.

A frontier of a PQ-tree is a permutation on its leaves obtained by DFS (Depth-First-Search), respecting the orders on the children of Q-nodes. We denote by F(T) the set of the frontiers of the PQ-tree T (see figure 2).

representation of $R(B_k)$ using PQ-tree :

A PQ-tree T_k is said to represent $R(B_k)$ iff

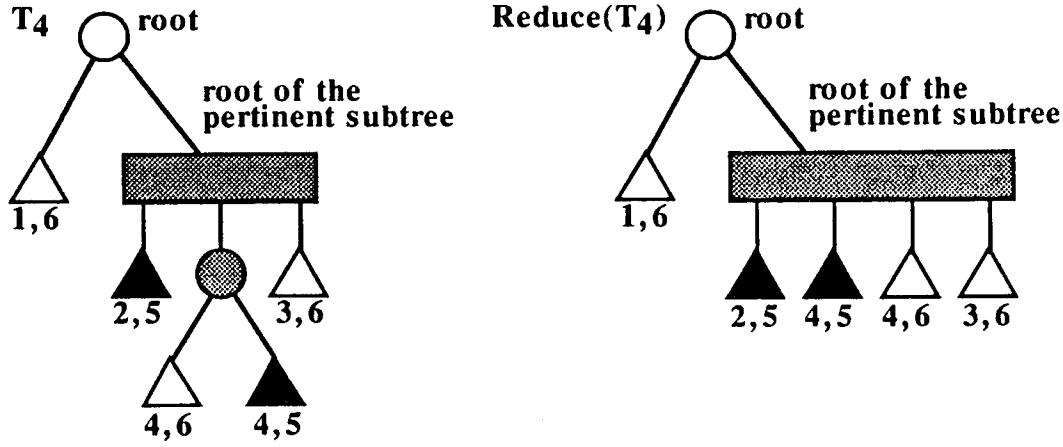
(1) its leaves represent the virtual edges of B_k ;

(2) P-nodes correspond to cut vertices v of B_k, which admit two virtual edges linked by disjoint chains whose intermediate vertices are greater than v ;

(3) Q-nodes correspond to nonseparable components of B_k, with second smallest vertex v, which admits two virtual edges linked by disjoint chains whose intermediate vertices are greater

than v ;
Consequently, one has $F(T_k) = \{\pi(b), b \in R(B_k)\}$

In figure 2, the PQ-tree T_4 represents $R(B_4)$, where B_4 is the bush form drawn in figure 1c. P-nodes are drawn as circles, Q-nodes as rectangles and leaves as triangles. The children of a node are drawn below it.



$$F(\text{Reduce}(T_4)) = \left\{ \begin{array}{l} ((1,6),(2,5),(4,5),(4,6),(3,6)) \\ ((1,6),(3,6),(4,6),(4,5),(2,5)) \\ ((2,5),(4,5),(4,6),(3,6),(1,6)) \\ ((3,6),(4,6),(4,5),(2,5),(1,6)) \end{array} \right\}$$

$$F(T_4) = F(\text{Reduce}(T_4)) \cup \left\{ \begin{array}{l} ((1,6),(2,5),(4,6),(4,5),(3,6)) \\ ((1,6),(3,6),(4,5),(4,6),(2,5)) \\ ((2,5),(4,6),(4,5),(3,6),(1,6)) \\ ((3,6),(4,5),(4,6),(2,5),(1,6)) \end{array} \right\}$$

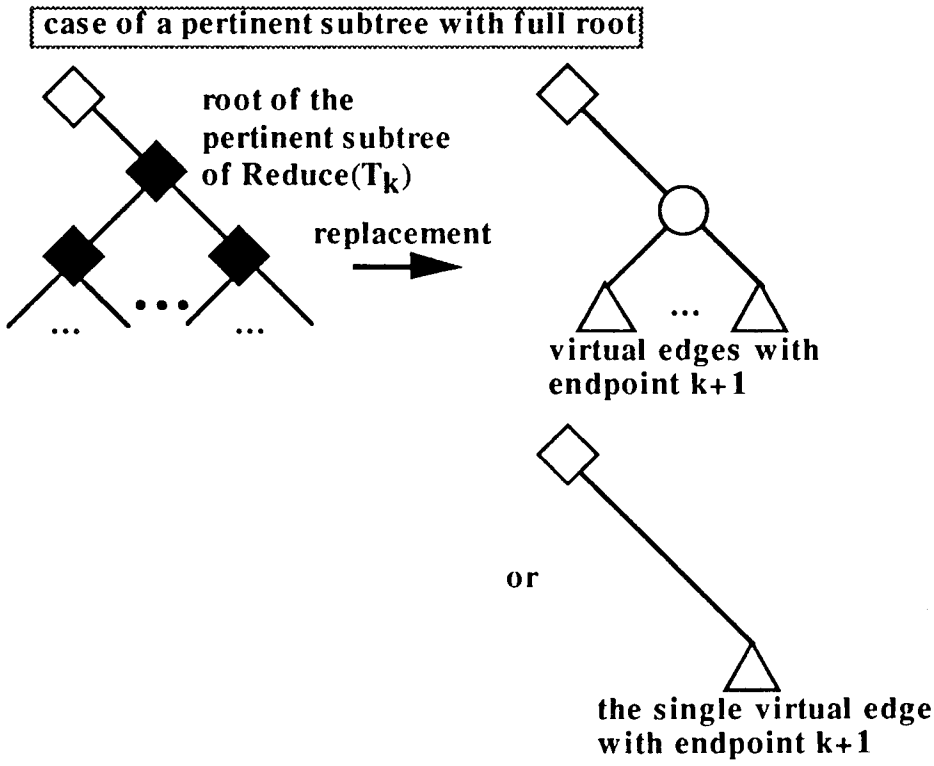
- figure 2 -

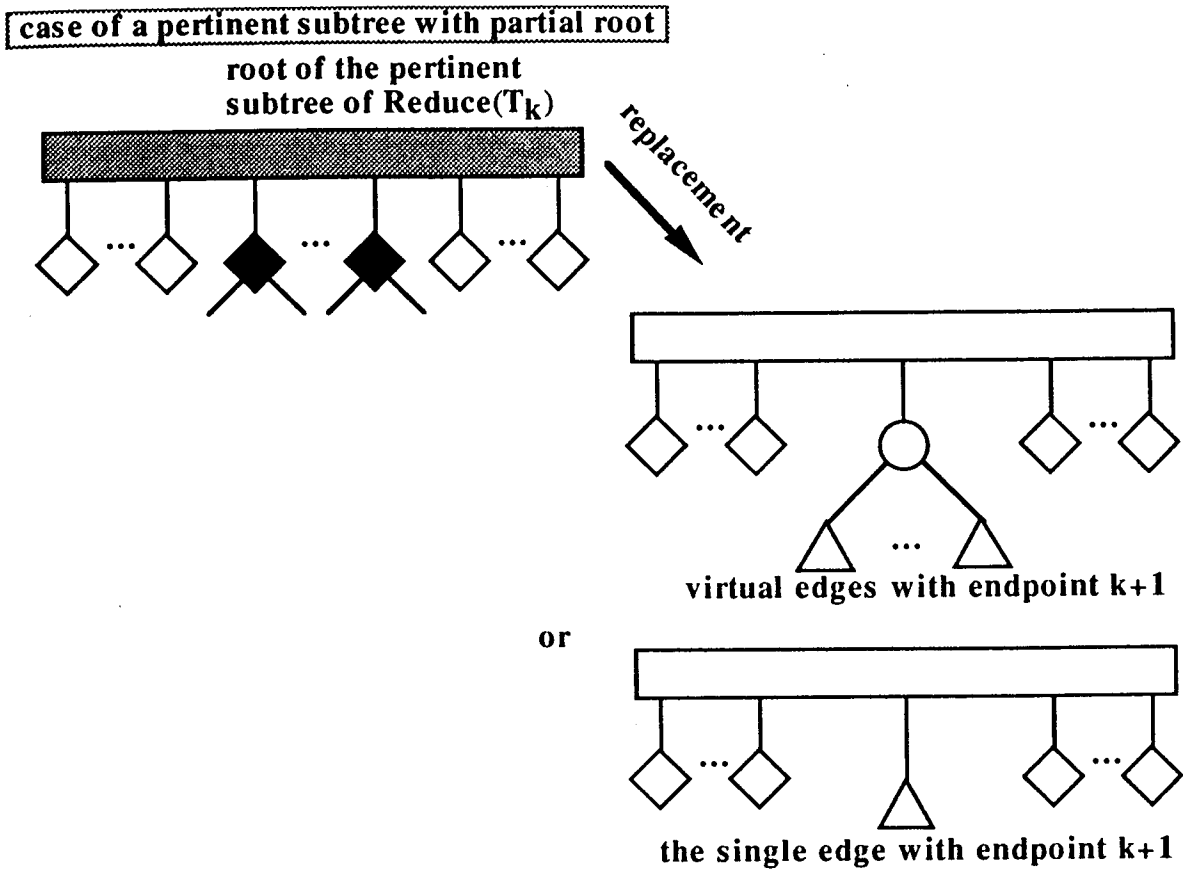
definition 5 : $\text{Reduce}(T_k)$ is a PQ-tree such that $F(\text{Reduce}(T_k))$ is the subset of $F(T_k)$ where virtual edges with endpoint k+1 appear consecutively. In effect, $F(\text{Reduce}(T_k)) = \{\pi(b), b \in R'(B_k)\}$.

For each k, the PQ-tree algorithm ensures a $\text{Reduce}(T_k)$. If $\text{Reduce}(T_k)$ is the empty tree, it means that the tested graph is not planar. It would be too long to explain here the reduction step in PQ-tree algorithm (see [4]). We will only study the "Replacement Step", i.e. how to obtain T_{k+1} from $\text{Reduce}(T_k)$.

definition 6 : A full (respectively empty) node in the PQ-tree T_k or Reduce(T_k) is
(1) a virtual edge with (resp. without) endpoint k+1 ;
(2) a node with only full (resp. empty) children.
The pertinent subtree of a PQ-tree is the subtree of minimum height containing all the full nodes. A node of the pertinent subtree with both empty and full children is called partial as a node with at least one partial children. In the figures, a full (respectively empty, partial) node is drawn in black (resp. white, grey). In Reduce(T_k), the pertinent subtree is formed by all the full nodes, and possibly a partial Q-node as root.

Let S be the set of virtual edges of B_{k+1} with endpoint k+1. Let T(S) be the PQ-tree formed by a
P-node and the elements of S as leaves, directly linked to it. If |S| = 1, T(S) is limited to a single leaf. T_{k+1} is obtained by replacing the full nodes of Reduce(T_k) by T(S), as shown in figure 3.





- figure 3 -

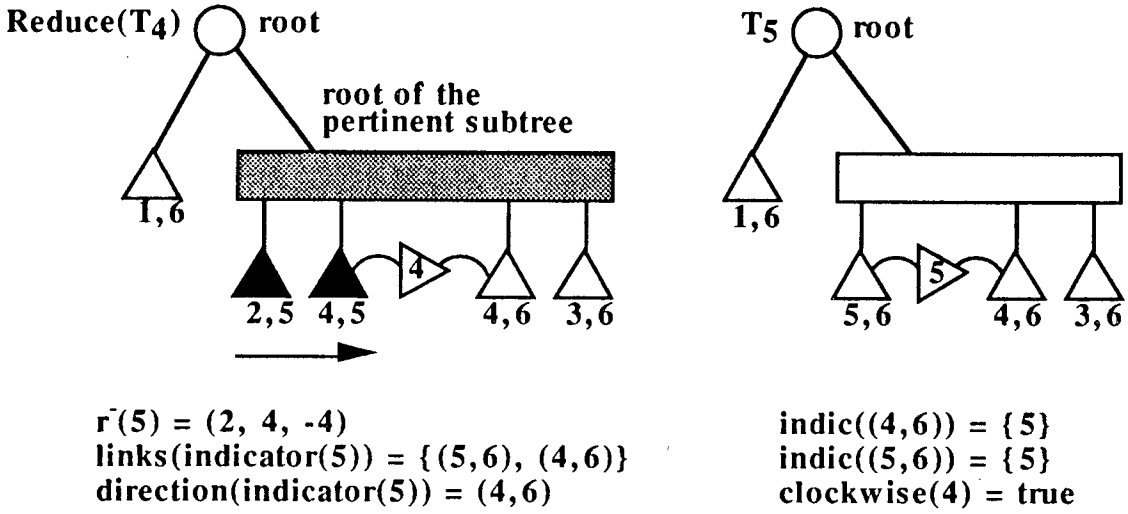
PART II : EMBEDDING PLANAR GRAPHS USING PQ-TREES

This section is a summary of [6].

theorem 6 : Given an embedding of a planar st-numbered graph G, every rotation r(v) can be written as a circular list $w_1, \ldots, w_i, w_{i+1}, \ldots, w_j$ such that $w_1, \ldots, w_i$ are less than v and $w_{i+1}, \ldots, w_j$ are greater than v. The sequence $w_1, \ldots, w_i$ (respectively $w_{i+1}, \ldots, w_j$) is called the upward (resp. downward) rotation of v and is denoted by $r^-(v)$ (resp. $r^+(v)$).

Given the upward rotation of each vertex, it is easy to recover the complete embedding of G, using the algorithm of Chiba and al. [6]. The set of admissible upward rotations for a vertex v is given by the set of the frontiers of the pertinent subtree in Reduce(T_{v-1}). But if the vertex v belongs to a nonseparable component of G_v, represented by a Q-node, only half of the admissible upward rotations of v will be consistent with the upward rotation of n chosen at the end of the algorithm. The other half will be consistent with the inverted upward rotation of n. To control this problem, we use "Indicators".

Example : consider the figures 1 and 2. The admissible upward rotations for the vertex 5 are (2,4) and (4,2). But if we choose (1,5,4,3) as upward rotation for t[illegible] then only (2,4) is possible.



- figure 4 -

An upward embedding of v is a permutation on the full leaves of Reduce(T_{v-1}). This permutation is obtained by a DFS on the pertinent subtree, respecting the orders of the children of Q-nodes. If the root of this pertinent subtree is a partial Q-node, then at the end of the replacement step, we must add an indicator labeled v below it (see figure 4). This indicator memorizes the direction in which the higher full nodes of the pertinent subtree have been encountered during the DFS.

Actually, an indicator is linked to a Q-node by two adjacent children of the Q-node, and its direction is given by one of those links. When an indicator v is linked to a full node of a PQ-tree T_w, it is removed with the full nodes when we build T_{w+1}, except if that node is the root of the pertinent subtree. We add a mark, say "-v", in the upward rotation of w+1, to indicate that the direction of $r^-(v)$ is linked to those of $r^-(w+1)$. A boolean variable, clockwise(v), is initialized true if the indicator of v has been encountered in its direction during the computation of $r^-(w+1)$, false otherwise (see figure 4).

Before recovering the complete embedding, we examine the upward rotations from $r^-(n)$ to $r^-(1)$. Each time we encounter a negative vertex -v in a rotation $r^-(w)$, we remove it. Furthermore, we assign true to clockwise(v) if clockwise(v) was equal to clockwise(w), false otherwise. When we consider $r^-(v)$, we reverse it if clockwise(v) equals false. Otherwise, $r^-(v)$ is kept unchanged.

Note that in [6], indicators are particular leaves of the PQ-tree. In our program, we have used an other data structure, more exogenous, which conserves the original definition of PQ-tree. Each node x has a pointer to a set "indic(x)" containing its linked indicators. If indic(x) is not empty, a field containing a reference to x is created (this reference can be the address of x). An indicator i has two pointers to the fields containing the reference to its linked nodes x and x'. Similarly, direction(i) is a pointer to the field containing the reference of the linked node given the direction of i. Then, when we exchange two nodes in the PQ-tree, updating indicators is a O(1)-time operation. Similarly, translating a set of indicators is a O(1)-time operation. Two others pointers give the addresses of i in indic(x) and indic(x'). Thus, removing i from the PQ-tree is also a O(1)-time operation. An indicator is manipulated only once during the reduction step iff it is linked with two full nodes or, if the root of the pertinent subtree is partial, with a higher full node and its empty immediate sibling. It is removed from the PQ-tree at the same time as the full nodes of the pertinent subtree. As there are at most n indicators created, the PQ-tree algorithm with indicators remains linear.

PART III : EMBEDDING PLANAR GRAPHS WHERE CERTAIN ROTATIONS ARE ARBITRARILY FIXED

For technological reasons, it may be necessary to verify that a planar graph can be embedded with given rotations for some vertices [14]. It is well known that a 3-connected graph has essentially only one embedding, that is, it has a unique embedding apart from reflection. However, a graph with connectivity less than 3 may have many different embeddings. The algorithm of Chiba and al. permits the computation of all the embeddings of a planar graph. But it would be too long to verify that there exists one with the fixed rotations. We propose here an algorithm, based on PQ-trees, which finds an embedding, if any, of a graph with some fixed rotations. If the algorithm fails, such an embedding does not exist. We call "Fixed" the set of vertices for which rotations are given, and "Forced" the set of vertices for which rotations are forced by some vertices of Fixed. Forced is initialized at Fixed.

The first step of our algorithm is to separate upward and downward rotations for each vertex of Fixed (see theorem 6). This is done by a function which returns false if such separation is impossible. The structures representing $r^-(v)$ and $r^+(v)$ are double linear lists [13].

The main difference between our algorithm and those of [6] lies in the replacement step (called "vertex addition step" in [6]) and it changes according to whether the vertex v belongs to Fixed or not.

If the vertex v belongs to Fixed, we need to verify that $r^-(v)$ is admissible. We process the nodes of the pertinent subtree of Reduce(T_{V-1}) in FIFO order, using a queue initialized to

the set of the full leaves of T_{v-1}. A node is place onto the queue when all its children have been treated. We use a sequence E representing a frontier which must have the pertinent subtree of Reduce(T_{v-1}) after deletion of nodes whose parents have been treated. Initially, E equals $r^-(v)$. To process a node means to see if it's children form a contiguous subsequence of E. In addition, for a Q-node, the contiguous subsequence must respect the order of the children.

We must also ensure that the continuation of the algorithm gives the wanted result for the fixed downward rotations. This can be done by adding the new leaves of T_v as children of a Q-node with an indicator given the direction of $r^+(v)$.

The replacement step for vertices which do not belong to Fixed is fairly similar to the "vertex addition step" presented in [6]. Nevertheless, we must ensure that a fixed upward rotation will not be reversed at the end of the algorithm. Let v be an element of Forced. It means that clockwise(v) must have the same value before the recovering of the complete rotation of v than when we have computed the upward rotation of v. Otherwise, the upward rotation of a fixed vertex would be reversed. When such an indicator v is encountered during the replacement step from Reduce(T_{w-1}) to T_w, we consider two cases :

- $w \notin$ Forced : If the indicator v has been encountered in the opposite direction, then change clockwise(v) and assign false to clockwise(w). Thus, clockwise(v) will be changed again if clockwise(w) is false before recovering the complete rotation of w. To ensure that, we add w to Forced. Similary, if the indicator v has been encountered in its direction, we assign true to clockwise(w) and add w to Forced, clockwise(v) being unchanged.
- $w \in$ Forced : Thus, clockwise(w) can not be changed. If the indicator v has been encountered in the opposite direction and clockwise(w) equals true or the indicator v has been encountered in its direction and clockwise(w) equals false, then the tested graph can not be embedded with the given fixed rotations. Otherwise, if clockwise(w) equals false, we change clockwise(v) because clockwise(v) will be changed again before recovering the complete rotation of v. If clockwise(w) equals true, clockwise(v) is unchanged.

PART IV : ALGORITHM

Most of the global variables of our algorithm are those used in [4]. We present them quickly before describing the replacement step, which marks out the difference between our algorithm and that of [4]. We use the same formalism as in [4].

Indicator(x) : Indicator labeled x, added during the replacement step of T_{x-1}. Indicator(x) is encountered in its direction if the node given the direction is encountered after the other link. For an indicator linked to a (higher) full and an empty node, we say that it is encountered in its direction if the full node gives the direction and is the first node encountered, or the full node does not give the direction and is the last higher full node encountered.

Clockwise(x) : Boolean used to know if $r^-(v)$ must be reversed before the recovering of the complete embedding.

Type(x) : Type of the node x (P-node, Q-node or leaf).

Queue : A FIFO list, used in the function Replace_Forced, to sequencing the order in which nodes are treated. This list is initialized to the list of full leaves and a node is added iff all its children have been treated.

E : Sequence used in Replace_Forced. E represents a frontier which must have the pertinent subtree of Reduce(T_{v-1}) after deletion of nodes whose parents have been treated.

Note that in a PQ-tree, every node has a set containing all its full children. A Q-node knows its two extremal children and a P-node the address of one of its children. Contrary to a P-node, a Q-node does not know the number of its children. A child of a Q-node knows its immediate siblings and the children of a P-node are linked as a circular list. A node knows its parent iff it is a child of a P-node, an extremal child of a Q-node or it is full. As we consider here nodes in the pertinent subtree of a reduced PQ-tree, every node, except the root if it is partial, knows its parent. For the same reason, every node, except the root if it is partial, knows the number of its children, which is equal to the number of its full children.

```
/* Replacement Step of Reduce(Tv-1) */
if v ∈ Fixed then                  /* r(v) is fixed.                          */
   return Replace_Fixed(v);        /* This function will be described afterwards */
else
   begin
      Let w1, ... , wj be the minimum endpoints of the virtual edges scanned, in this order, by
DFS on the      pertinent subtree in Reduce(Tv-1), respecting orders on the children of Q-
nodes;
      r⁻(v) := (w1, ... , wj);
      Clockwise(v) := true;
      for each indicator Indicator(x) encountered while the DFS, including those on right and
      left side of the higher full nodes if the root is partial do
         if x ∈ Forced then
            begin
               if v ∈ Forced then
                  begin
                     if (Indicator(x) has been encountered in the opposite direction and
                     Clockwise(v) = true) or (Indicator(x) has been encountered in its direction
                     and Clockwise(v) = false) then
                        return false   /* Embedding with fixed rotations is impossible */
                     else
                        begin
                           add -x to r⁻(v);
```

```
                    if Clockwise(v) = false then Clockwise(x) := not Clockwise(x);
                  end;
              end
            else /* v ∉ Forced    */
              begin
                add v to Forced;
                add -x to r⁻(v);
                if Indicator(x) has been encountered in the opposite direction then
                  begin
                    Clockwise(v) := false;
                    Clockwise(x) := not Clockwise(x);
                  end;
              end;
          end
        else /* x ∉ Forced    */
          begin
            add -x to r⁻(v);
            if Indicator(x) has been encountered in the opposite direction then
              Clockwise(x) := false;
          end;
      Let S be the set of edges with minimum endpoint v;
      if the root of the pertinent subtree is a partial Q-node then
        begin
          if |S| = 1 then
            replace the full nodes and their linked indicators by a leaf labeled as the single
            element of S;
          else
            begin
              replace all the full nodes and their linked indicators by a P-node;
              add to the PQ-tree the elements of S as leaves linked to the added P-node;
            end;
          add an indicator labeled v below the root of the pertinent subtree, with direction in
          which the higher full nodes had been encountered by the DFS if Clockwise(v) =
          true, with the opposite direction otherwise;
        end
      else /* the root of the pertinent subtree is full */
        if |S| = 1 then
          replace the pertinent subtree, and the indicators linked to its nodes, by a leaf labeled
          as the single element of S;
        else
          begin
            replace the pertinent subtree, and the indicators linked to its nodes, by a P-node;
            add to the PQ-tree the elements of S as leaves linked to the added P-node;
          end;
    end;

function Replace_Fixed ( v : vertex;
                         ) : boolean;
    begin
```

```
Initialize Queue to the set of the full leaves of T_V, and E to r⁻(v);
Clockwise(v) := true;
while |Queue| > 0 do                    /* Main loop */
  begin
    remove N from the front of Queue;
    If N is the last child not yet treated of a node P, then place P onto Queue;
    case of Type(N) =    /* Treatment of the children of N, according to its type, */
                         /* then replacement of these children by N in E          */
      Leaf : ;  /* Nothing to do */
      P-node :
        If the Children of N do not appear as a contiguous subsequence of E then
          return false
        else
          replace this contiguous subsequence by N;
      Q-node :
        if N is full then /* First case : the Q-node is full */
          begin
            if the subsequence of E between the endmost children of N is equal to
            the set of the children of N with same or reversed order then
              begin
                for each indicator "Indicator(x)" linked with children of N do
                  if the link given the direction of Indicator(x) is the predecessor
                  in E of the other link then
                    begin
                      if x ∈ Forced then
                        return false
                      else
                        begin
                          Clockwise(x) := false;
                          add -x to r⁻(v);
                        end;
                    end
                  else
                    add -x to r⁻(v);
                replace the subsequence of E, between the endmost children of N,
                by N;
              end
            else
              return false;
          end
        else /* Second case : the Q-node is partial (thus, necessary the root of  */
             /* the pertinent subtree)                                            */
          if E is equal to the set of the full children of N with same or reversed order
          then
            for each indicator "Indicator(x)" linked with at least a full children of N
            do
              if the link of Indicator(x) given its direction is the predecessor in E
              of the other link
```

```
                    or the link of Indicator(x) given its direction is an empty node and
                    the other link is the first element of E
                    or the link of Indicator(x) given its direction is the last element of E
                    and the other link is an empty node then
                      begin
                        if x ∈ Forced then
                          return false
                        else
                          begin
                            Clockwise(x) := false;
                            add -x to r⁻(v);
                          end;
                      end
                    else
                      add -x to r⁻(v);
                  else
                    return false;
              end_case_of;
          Let S be the sequence of edges with minimum endpoint v, such that the endpoints
          different than v appear with same order than in r⁺(v).
          if the root of the pertinent subtree is a partial Q-node then
            begin
              replace the pertinent subtree, and the indicators linked to its nodes, by leaves
              labeled as the element of S. The sequence S below the Q-node must follow
              the inverted order of the higher full nodes in E.
              add an indicator labeled v with direction following those of the higher full
              nodes in E.
            end
          else
            begin
              if |r⁺(v)| = 1 then
                replace the pertinent subtree, and the indicators linked to its nodes, by a
                leaf labeled as the single element of S;
              else
                begin
                  replace the pertinent subtree, and the indicators linked to its nodes, by a
                  Q-node;
                  add to the PQ-tree the elements of S as leaves linked to the added Q-
                  node;
                  add an indicator labeled v below the added Q-node with the opposite
                  direction than the elements of S;
                end;
            end;
        end;    /* End of the main loop    */
      return true;
    end.
```

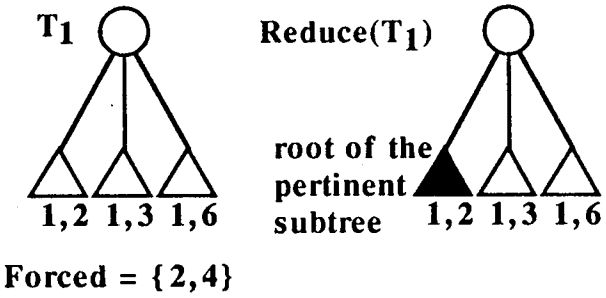
PART V : A COMPLETE EXAMPLE

We want to embed the graph of figure 1a with fixed rotations for vertices 2 and 4 :

r(2) = (1,5,4,3) and r(4) = (2,5,6,3)

Initialization : Fixed = {2,4}, Forced = {2,4}
$r^-(2) = (1)$, $r^+(2) = (5,4,3)$,
$r^-(4) = (3,2)$, $r^+(4) = (5,6)$,
$r^-(1) = ()$

Reduce and Replacement Steps :

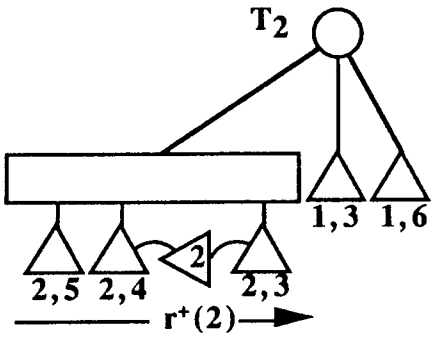


Forced = {2,4}

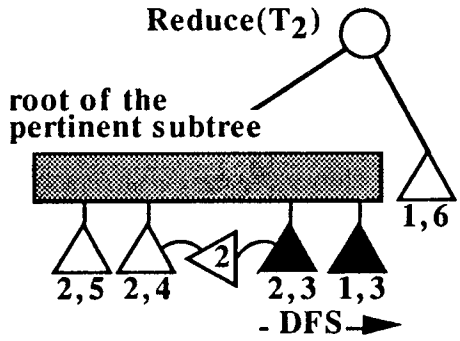
$r^-(2) = (1)$

Clockwise

1	true	4	true
2	true	5	true
3	true	6	true



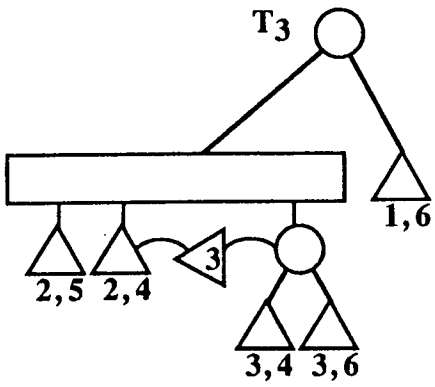
Forced = {2,4,3}



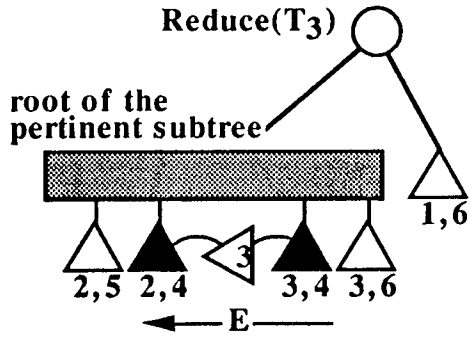
$r^-(3) = (-2,2,1)$

Clockwise

1	true
2	false
3	false
4	true
5	true
6	true



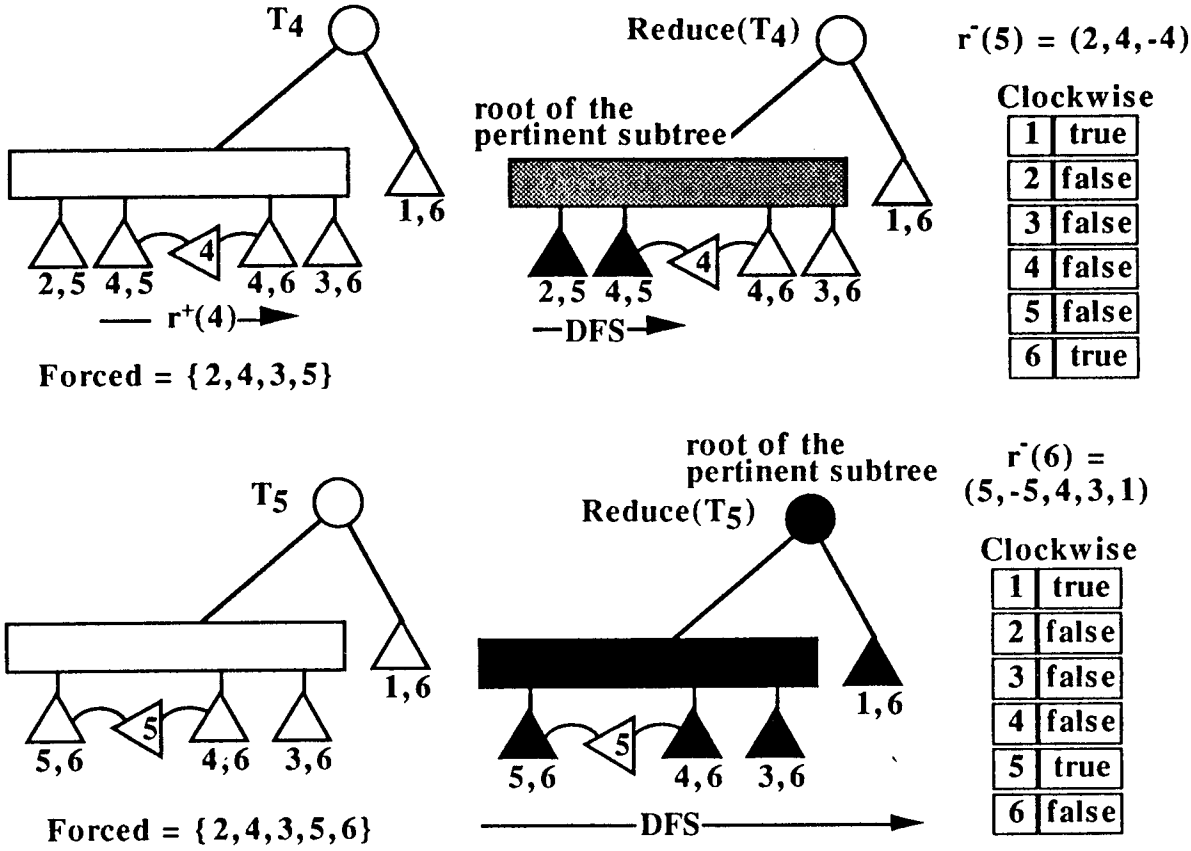
Forced = {2,4,3}



$r^-(4) = (3,-3,2)$

Clockwise

1	true
2	false
3	false
4	true
5	true
6	true



Removal of the marks "-v" in the upward embeddings :

$r^-(6) = (1,3,4,5)$ $r^-(5) = (4,2)$ $r^-(4) = (3,2)$

$r^-(3) = (1,2)$ $r^-(2) = (1)$ $r^-(1) = ()$

Complete Embedding (computed with the algorithm of Chiba and al. [4]) :

$r(6) = (1,3,4,5)$ $r(5) = (6,4,2)$ $r(4) = (5,6,3,2)$

$r(3) = (4,6,1,2)$ $r(2) = (5,4,3,1)$ $r(1) = (2,3,6)$

Bibliography

[1] J.E. Hopcroft and R.E. Tarjan, Efficient Planarity Testing, J. of ACM 21 (1974), 549-568.

[2] N. Deo, Note on Hopcroft and Tarjan's Planarity Algorithm, J. of ACM 23 (1976), 74-75.

[3] H. de Fraysseix and P. Rosenstiehl, A Depth-First-Search Characterization of Planarity, Annals of Discrete Math. 13 (1982), 75-80.

[4] K.S. Booth and G.S. Lueker, Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms, J. of Computing and System Sciences 13 (1976) 335-379.

[5] A. Lempel, S. Even and I. Cederbaum, An Algorithm for Planarity Testing of Graphs, Theory of Graphs - International Symposium, Rome July 1966, P. Rosenstiehl Ed. (1967, Gordon and Breach New-York) 215-232.

[6] N. Chiba, T. Nishizeki, S. Abe and T. Ozawa, A Linear Algorithm for Embedding Planar Graphs Using PQ-trees, J. of Computing and System Sciences 30 (1985) 54-76.

[7] P. Eades and R. Tamassia, Algorithms for Automatic Graph Drawing : An Annoted Bibliography, Technical Report 82, Dept. of Compter Science, University of Queensland - Australia (1987).

[8] J.A. Bondy and U.S.R. Murty, Graph Theory with Applications, MacMillan Press LTD, London and Basingstoke (1976).

[9] C. Berge, Graphes, Gauthier Villars (1983).

[10] E.M. Reingold, J. Nievergelt, N. Deo, Combinatorial Algorithms - Theory and Practice, Prentice-Hall (1977).

[11] S. Even, Graph Algorithms, Computer Science Press (1979).

[12] S. Even, R.E. Tarjan, Computing an st-Numbering, Theorical Computer Science 2 (1976)
339-334.

[13] R.E. Tarjan, Data Structures and Network Algorithms, CBMS-NSF Regional Conference Series in Applied Mathematics 44, SIAM (1983).

[14] J. Kuntzmann, Théorie des Réseaux - Graphes, Dunod, Paris (1972).

[15] J. Edmonds, A combinatorial representation for polyhedral surfaces, Notices AMS 7 (1960) 646.

BIBLIOGRAPHIE

[AB] **H. Abdallah et B. Belorgey**, Représentation des arêtes courbes dans Cabri. *Mémoire MLA2*, IGEI Université Joseph Fourier, Grenoble (1989).

[AHU] **A.V. Aho, J.E. Hopcroft and J.D. Ullman**, *Data Structures and Algorithms*. Addison-Wesley (1983).

[Bau1] **O. Baudon**, A linear algorithm for embedding planar graphs, where certain rotations are arbitrarily fixed. *Rapport de Recherche LSD-IMAG* **778-I**, Grenoble (1989).

[Bau2] **O. Baudon**, Generating random graphs, connected with the editor of graphs CABRI. Soumis pour publication.

[Baul] **Y. Baulac**, Un micromonde de géométrie, Cabri-géomètre. *Thèse de l'Université Joseph Fourier,* Grenoble (1990).

[BB] **J.-C. Bermond and B. Bollobas**, The diameter of graphs - a survey. *Congressus Numerantium* **32** (1981) 3-27.

[Ben] **C. Benzaken**, Un éditeur de graphe simple, d'aide à l'enseignement et à la recherche. *Rapport Technique LSD-IMAG* **1**, Grenoble (1986).

[Ber] **C. Berge**, *Graphes*, Gauthier Villars (1983).

[BL] **K.S. Booth and G.S. Lueker**, Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-Tree algorithms. *J. Comput. System Sci.* **13** (1976) 335-379.

[Bol1] **B. Bollobas**, *Random Graphs*. Academic Press (1985).

[Bol2] **B. Bollobas**, The asymptotic number of labelled graphs, with given degree sequence. *European J. of Combinatorics* **1** (1980) 311-316.

[CKS] **D.M. Cvetkovics, L.L. Kraus and S.K. Simic**, Discussing graph theory with computer I, Implementation of graph theoric algorithms. *Univ. Beograd Publ. Elektrotehn. Fak. Ser. Mat. Fiz.* **716-734**, (1981) 100-104.

[CK] **D.M. Cvetkovics and L.L. Kraus**, Graph, an expert system for the classification and the extension of the knowledge in the field of graph theory. *Research Report Faculty of electric engineering*, Belgrade University (1983).

[Clo] **J.-M. Clochard**, Algorithme de test de la planarité des graphes d'après le critère de Wu-Liu et dessin sur une grille. *Mémoire DEA Recherche Opérationnelle*, IGEI Université Joseph Fourier, Grenoble (1989).

[CM] **M. Capobianco and J.C. Molluzzo**, *Examples and Counterexamples in Graph Theory*. North-Holland, New-York (1978).

[CNAO] **N. Chiba, T. Nishizeki, S. Abe and T. Ozawa**, A linear algorithm for embedding planar graphs using PQ-trees. *J. Comput. System Sci.* **30** (1985) 54-76.

[CNET] **CNET et SEMA GROUP**, Cassis, environnement de conception de réseaux. *Rapport Technique CNET* (1989).

[Cor] **R. Cori**, *Un Code pour les Graphes Planaires et ses Applications*. Astérisque **27** SMF (1975).

[Cro] **F. Crocombette**, Étude de la planarité et du dessin de graphes planaires avec la méthode de Tarjan. *Mémoire DEA Recherche Opérationnelle*, IGEI Université Joseph Fourier, Grenoble (1988).

[Cve] **D.M. Cvetkovics**, Discussing graph theory with computer II, Theorems suggested by the computer. *Publication de l'Institut Mathématique* **33**, (1983) 29-33.

[Deo] **N. Deo**, Note on Hopcroft and Tarjan's planarity algorithm. *J. ACM* **23**, (1976) 74-75.

[DHRT] **M. Dao, M. Habib, J.-P. Richard and D. Tallot**, Cabri, an interactive system for graph manipulation. Graph Theoric Concepts in Computer Science, Proceeding of the International Workshop WG' 86, Bernried June 1986. *Lecture notes in Comput. Sci.* **246**, Springer, New-York (1986) 58-67.

[DW] **J.D. Dixon and H.S. Wilf**, The random selection of unlabeled graphs. *J. Algorithms* **4** (1983) 205-213.

[Ead] **P. Eades**, A heuristic for graph drawing. *Congressus Numerantium* **42** (1984) 149-160.

[EaT] **P. Eades and R. Tamassia**, Algorithms for automatic graph drawing, an annotated bibliography. *Technical Report Dpt of Computer Science*, University of Queensland (1987).

[Edm] **J. Edmonds**, A combinatorial representation for polyhedral surfaces. *Notices AMS* **7** (1960).

[ER] **P. Erdös et A. Renyi**, On random graphs I. *Publ. Math. Debrecen* **6** (1959) 290-297.

[Eve] **S. Even**, *Graph Algorithms*. Computer Science Press (1979).

[EvT] **S. Even and R.E. Tarjan**, Computing an st-numbering. *Theorical Computer Science* **2** (1976) 339-344.

[Far] **I. Fàry**, On straight line representation of planar graphs. *Acta Sci. Math. Szeged* **11** (1948) 229-233.

[Fou1] **J.-M. Fourneau**, Réseaux : graphes et évaluations de performances. *Thèse Université Paris-Sud* (1987).

[Fou2] **J.-M. Fourneau**, Unicorn, une maquette de Cabri pour les réseaux d'interconnexion. *Rapport de Recherche ISEM* **051**, Université Paris-Sud (1986).

[FPP] **H. de Fraysseix, J. Pach and R. Pollack**, Small sets supporting Fary embedding of planar graphs. *20th ACM Symp. on theory of computing* (1988) 426-433.

[FR1] **H. de Fraysseix et P. Rosenstiehl**, L'algorithme gauche-droite pour le plongement des graphes dans le plan. A paraître.

[FR2] **H. de Fraysseix and P. Rosenstiehl**, A Depth-First-Search characterization of planarity. *Annals of Discrete Math.* **13** (1982) 75-80.

[Fri] **A. Frieze**, On random regular graphs with non-constant degre. A paraître.

[FRP] **H. Frydlender et A. Rimbod Pethiod**, Algorithme de Tarjan de détermination de la planarité d'un graphe. *Mémoire MLA2*, IGEI Université Joseph Fourier, Grenoble (1987).

[GJ] **M.R. Garey and D.S. Johnson**, *Computers and Intractability : a guide to the theory of NP-completeness*. H. Freeman and sons, San Francisco (1979).

[GM] **M. Gondran et M. Minoux**, *Graphes et Algorithmes*. Eyrolles (1987).

[GS] **B.L. Golden and W.R. Stewart**, Empirical analysis of heuristics. *The Traveling Salesman Problem, a Guided Tour of Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics (1985).

[Gué1] **A. Guénoche**, Génération de graphes planaires maximum. *Rapport de Recherche GRTC-CNRS*, Marseille (1986).

[Gué2] **A. Guénoche**, Graphes bipartis de degrés fixés aléatoires. *Rapport de Recherche GRTC-CNRS*, Marseille (1986).

[Gui] **Y. Guido**, Cabri : un cahier de brouillon informatisé pour l'étude de la théorie des graphes. *Thèse 3° cycle Université de Montpellier* (1984).

[GX] **D. Gries and J. Xue**, The Hopcroft-Tarjan planarity algorithm, presentations and improvements. *Technical Report Dpt of Computer Science* **88-906**, Cornell University, Ithaca New-York (1988).

[HEA] **J. Hoffman, C. Espinosa and A. Averill**, The Macintosh user interface guidelines. *Inside Macintosh* **1**, Addison-Wesley (1985) 23-70.

[Him] **M. Himsolt**, Entwicklung eines grapheneditors. *Diplomarbeit Universität Passau* (1988).

[HL] **M. Habib et J.-M. Laborde**, document de constitution du groupe Cabri. *Rapport PRC Mathématique et Informatique* (1983).

[HP] **F. Harary and E.M. Palmer**, *Graphical Enumeration*. Academic Press (1973).

[HTDR] **M. Habib, D. Tallot, M. Dao et J.-P. Richard**, Cabri, un outil de manipulation de graphes. *SM 90 une architecture modulaire et ses applications*, Actes des journées SM90, Eyrolles (1986) 514-523.

[HT] **J. Hopcroft and R. Tarjan**, Efficient planarity testing. *J. ACM* **21**, (1974) 549-568.

[JS] **M. Jerrum and A. Sinclair**, Fast uniform generation of regular graphs. *Research Report* **CSR-281-88**, University of Edinburgh (1988).

[Kar] **A.V. Karzanov**, Recognizing planarity and embedding a graph in the plane in linear time. *Algorithms of Discrete Optimizations and their applications to Computer Systems*, Yaroslav University (1983) 58-80 (en Russe).

[KR] **B.W. Kernighan et D.M. Ritchie**, *Le langage C.* Masson (1986).

[Lab1] **J.-M. Laborde**, Cabri, a tool for research and teaching in graph theory. *Colloque La Combinatoire et l'Informatique*, Montreal (1987).

[Lab2] **J.-M. Laborde**, Le plongement dans l'hypercube des arbres d'au plus 16 sommets. *Rapport de Recherche LSD-IMAG* **755-I**, Grenoble (1988).

[Läu] **P. Läuchli**, Generating all planar 0-, 1-, 2-, 3-connected graphs. Graph Theoric Concepts in Computer Science, Proceeding of the International Workshop WG' 80, Bad Honnef, June 1980. *Lecture notes in Comput. Sci.* **100**, Springer, New-York (1981) 379-382.

[LEC] **A. Lempel, S. Even and I. Cederbaum**, An algorithm for planarity testing of graphs. *Theory of Graphs*, Internat. Sympos. Rome, July 1966, Gordon and Breach, New-York (1967) 215-232.

[LP] **H.R. Lewis et C.H. Papadimitriou**, L'efficacité des algorithmes. *Pour la Science* **5** (1978) 62-75.

[MW] **B. McKay and N. Wormald**, Uniform generation of random regular graphs of moderate degree. A paraître.

[NC] **T. Nishiseki and N. Chiba,** *Planar Graphs : Theory and Algorithms.* Annals of discretes mathematics **32** North-Holland (1988).

[Nis] **T. Nishiseki,** Planar graph problems. *Research Report Faculty of Engineering,* Tohoku University, Sendai Japan (1989).

[NW1] **A. Nijenhuis and H.S. Wilf,** *Combinatorial Algorithms.* Academic Press, New-York (1978).

[NW2] **A. Nijenhuis and H.S. Wilf,** The enumeration of connected graphs and linked diagrams. *J. Combin. Theory Ser. A* **27** (1979) 356-359.

[Ore] **O. Ore,** *The Four Colour Problem.* Academic Press (1967).

[Pen] **G.-J. Penaud,** Algorithmes de planarité. *Journées de Combinatoire et Informatique de Bordeaux* (1975) 279-303.

[Prü] **H. Prüfer,** Neuer beweis eines satzes über permutationen. *Arch. Math. Phys.* **27** (1918) 742-744.

[Rea1] **R.C. Read,** A survey of graph generation techniques. Combinatorial Mathematics VIII Geelong 1980, *Lecture Note in Mathematics* **884** (1981) 77-89.

[Rea2] **R.C. Read,** Methods for computer display and manipulation of graphs, and the corresponding algorithms. *Research Report* **CORR 86-12,** Faculty of Mathematics, University of Waterloo (1986).

[Rea3] **R.C. Read,** A new method for drawing a planar graph given the cyclic order of the edges at each vertex. *Research Report* **CORR 86-14,** Faculty of Mathematics, University of Waterloo (July 1986).

[RND] **E.M. Reingold, J. Nievergelt and N. Deo,** *Combinatorial Algorithms : Theory and Practice.* Prentice-Hall (1977).

[RT] **P. Rosenstiehl and R.E. Tarjan,** Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete Comput. Geom.* **4** (1986) 343-353.

[SIKV] **D.C. Smith, C. Irby, R. Kimball, B. Verplank and E. Harslem,** Designing the Star user interface. *Byte* **4** (1982) 242-282

[Sin] **A.J. Sinclair,** Randomised algorithms for counting and generating combinatorial structures. *Thesis University of Edinburgh* (1988).

[SJ] **A.J. Sinclair and M.R. Jerrum,** Approximate counting, uniform generation and rapidly mixing Markov chains. *Research Report Dpt of Computer Science* **CSR-24187,** University of Edinburgh (1987).

[SP] **P.-C. Scholl et J.-C. Peyrin,** *Schémas Algorithmiques Fondamentaux, Séquences et Itérations.* Masson (1989).

[Tal] **D. Tallot,** Quelques propositions pour la mise en œuvre d'algorithmes combinatoires. *Thèse 3° cycle, Université des Sciences et Techniques du Languedoc,* Montpellier (1985).

[Tar] **R.E. Tarjan,** *Data Structures and Network Algorithms.* Regional Conference Series in Applied Mathematics, SIAM (1983).

[Tin1] **G. Tinhofer,** On the generation of random graphs with given properties and known distribution. *Appl. Comput. Sci. Ber. Prakt. Inf.* **13** (1979) 265-296.

[Tin2] **G. Tinhofer,** *Zufallsgraphen.* Carl Hanser Verlag, München (1980).

[Tin3] **G. Tinhofer,** On the use of some almost sure graph properties. Graph Theoric Concepts in Computer Science, Proceeding of the International

Workshop WG' 80, Bad Honnef, June 1980. *Lecture notes in Comput. Sci.* **100**, Springer, New-York (1981) 113-126.

[Tur] **J.S. Turner**, Almost all k-colorable graphs are easy to color. *J. Algorithms* **9** (1988) 63-82.

[Ver] **D. Verove**, Un langage extensible de programmation d'algorithmes de graphes. *Thèse 3° cycle, Université de Bordeaux 1* (1979).

[Wag] **K. Wagner**, Bemerkungen zum vierfarbenproblem. *Jber. Deutsch. Math.-Verein* **46** (1936) 26-32.

[Web] **K. Weber**, Random graphs - a survey. *Rostock. Math. Kolloq.* **21** (1982) 83-98.

[Wil] **H.S. Wilf**, The uniform selection of free trees. *J. Algorithms* **2** (1981) 204-207.

[Wol] **M.S. Wolfberg**, An interactive graph theory system, *Ph. D. Thesis University of Pennsylvania* (1969).

[Wor1] **N.C. Wormald**, Generating random unlabeled graphs. *SIAM J. Comput.* **16** (1987) 717-727.

[Wor2] **N.C. Wormald**, Generating random regular graphs. *J. Algorithms* **5** (1984) 247-280.

[Wor3] **N.C. Wormald**, Enumeration of labelled graphs II. Cubic graphs with a given connectivity. *J. London Math. Soc.* **20** (1979) 1-7.

[Xuo] **N.H. Xuong**, *Graphes : Théorie, Algorithmes et Applications.* Élements de Mathématiques Discrètes pour l'Informatique, tome 3, Cours polycopié Université Joseph Fourier, Grenoble (1988).

[Zeg] **F. Zegel**, Structures de données pour un langage de traitement de graphes. *Thèse 3° cycle, Université Paris VI* (1975).

[Zin] **J.-J. Zinetti**, Définition d'un langage de traitement de graphes. *Thèse 3° cycle, Université Paris VI* (1975).

Cabri-graphes est un environnement logiciel, destiné aux chercheurs, étudiants et enseignants en théorie des graphes.

La pratique de cette discipline amène à recourir à des représentations graphiques, afin de visualiser les structures mathématiques mises en œuvre; ceci dans le but de les manipuler, de leur appliquer concrètement certaines transformations, afin de vérifier une propriété, conforter ou infirmer une idée, une conjecture.

Cette pratique, menée sur un cahier de brouillon traditionnel, souffre de limitations et les résultats ne sont que faiblement garantis. C'est pourquoi nous avons étudié et réalisé un logiciel, alliant la simplicité d'usage d'un environnement interactif à la puissance de l'ordinateur.

Cette thèse présente l'ensemble des concepts mathématiques et de génie logiciel ayant servi à la réalisation de ce projet. Nous donnons en particulier l'implémentation d'un générateur de graphes aléatoires, ainsi que quelques applications motivées et réalisées grâce à cet environnement logiciel.

Mots-clés : Théorie des graphes, Génie logiciel, Interactivité, Manipulation graphique, Génération de graphes aléatoires.