Scheduling activities under spatial and temporal constraints to populate virtual urban environments

Carl-Johan Jorgensen

2015

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale MATISSE

présentée par

Carl-Johan Jørgensen

Préparée à l'unité de recherche IRISA - UMR 6074
Institut de Recherche en Informatique et Systèmes Aléatoires

Scheduling Activities Under Spatial and Temporal Constraints to Populate Virtual Urban Environments

Thèse soutenue à Rennes le 17/07/2015

devant le jury composé de :

Céline LOSCOS
Professeur, IUT de Reims-Châlon-Charleville / Rapporteur

Tsai-Yen LI
Professeur, National Chengchi University, Taipei, Taiwan / Raporteur

Arjan EGGES
Associate professor, Utrecht University, Utrecht, Pays-Bas / Examinateur

Bruno ARNALDI
Professeur, INSA de Rennes / Examinateur

Kadi BOUATOUCH
Professeur, Université de Rennes 1 / Directeur de thèse

Fabrice LAMARCHE
Maitre de conférences, Université de Rennes 1 / Co-directeur de Thèse

.

Contents

List of Figures

List of Tables

Acknowledgements

All along my childhood, I heard my parents talking about teaching, and my father talking about computer sciences and research. So, as any child would, I promised myself I would never be a teacher, a computer scientist nor a researcher... Yet, my studies proved me wrong and I ended up doing a PhD in Computer Sciences, teaching undergraduates and, worst of all: I enjoyed it! So, now I defended my thesis, marking the end of four fascinating years of work in MimeTIC team in the IRISA lab, I think I must amend my childhood judgement: research in computer sciences is what I want to do!

Big thanks to my parents and sisters for always pushing me to choose my own ways and methods while staying supportive and surrounding me with an environment rich in cultural and scientific interests. I guess I remember listening to discussions about physics, chemistry, computer sciences, healthcare and teaching theory around the dinner table from my younger age and that such topics eventually left a trace in my curiosity about this world and its sciences.

I also want to thank all the teachers, researchers, friends, lab members and fellow students I met during these last 24 years of studies for having contributed to develop my interest and knowledge. And most especially, I think about all the members of Bunraku and MimeTIC teams, in which I spent these last years and who helped me all along my PhD, about Tsai-Yen Li, Helen and all IMlab members who welcomed me for three memorable months in Taipei, about Kadi Bouatouch who directed my PhD and who was always available to help me with my publications or paperwork, about Julian who helped greatly with my demos and about Erwan and Arthur who supported me on a daily basis at lunch. Big thanks to Bruno Arnaldi, Céline Loscos and Arjan Egges as well, who accepted to be part of my defence's jury and were immensely supportive in this last straight line.

However, among these great people, I foremost want to thank Fabrice Lamarche. First for providing me with my first introduction to behavioural animation in my second year of University, catching my interest in the subject. Second for helping me pursue my goal of working in this field and trusting me enough for offering me an internship in the Lab, then a PhD subject. Finally for the long hours we spent together talking casually, discussing passionately and even sometimes arguing about my research. He was able to leave his own opinions aside to let me sufficient freedom to explore weird ideas while keeping an eye on my work and providing with advice and guidance when needed. For all that, thanks again!

Finally, thanks to Gabi for tolerating my chaotic schedules, night work and existential crisis during these four years. She was able to support me and cheer me up whenever I felt like over-flooded by my PhD work. Luvilul!

This research was made possible by the ANR project "iSpace&Time", founded by the French National Agency for Research. Project reference: ANR-10-CORD-0023

Résumé étendu en Français

Certaines applications telles que l'analyse et la validation d'aménagements urbains nécessitent la génération de foules faisant preuve de comportements réalistes. Cependant, les modèles de simulation de foules existants se concentrent généralement sur la génération de foules visuellement crédibles en se reposant sur des règles macroscopiques, sur des animations précalculées ou sur des données capturées. Afin d'obtenir des comportements de foules réalistes, il est nécessaire de s'intéresser à la qualité des comportements individuels des agents constituant ces foules. En effet, la plupart de l'activité observable dans les foules réelles est due aux personnes effectuant leurs activités quotidiennes dans différents lieux de la ville et à leurs déplacements entre ces lieux. C'est pourquoi nous estimons que doter chaque agent d'un modèle comportemental s'intéressant à ses choix rationnels en termes d'ordonnancement d'activités et de planification de chemin permet la génération de comportements individuels plus crédibles. Ainsi, des phénomènes réalistes peuvent émerger de la combinaison des comportements individuels générés.

Dans ce document, nous proposons un tel modèle, dont la structure est présentée dans la figure 1. Ce modèle vise à simuler une partie des niveaux rationnels et cognitifs des comportements de piétons en milieu urbain. Il s'intéresse en particulier à l'analyse et à la représentation de l'environnement, à l'ordonnancement d'activités, à la sélection d'horaires et des lieux appropriés à la réalisation de ces activités et à la planification de chemins entre les lieux sélectionnés. Ce modèle repose sur quatre données d'entrée : une représentation géométrique informée de l'environnement, une base de données d'information, une description des caractéristiques individuelles des agents et une description de leurs activités envisagées. Notre modèle permet la description de plusieurs moyens permettant de réaliser une activité. Il est robuste aux variations des contraintes spatiales et temporelles associées aux activités et à l'environnement. Cela lui permet d'adapter la réalisation d'une activité à des environnements et agendas à priori inconnus, ainsi qu'aux caractéristiques des agents. Cela signifie qu'une même description d'une activité peut être utilisée pour différents agents dans divers environnements et situations. Cela signifie également que des agents partageant une activité similaire peuvent se comporter de manières totalement différentes en fonction de leurs préférences et contraintes personnelles. Cette propriété permet l'émergence d'une plus grande diversité de comportements de piétons, augmentant ainsi la crédibilité du peuplement. Dans cette section, nous donnons un aperçu des composants de note méthode et de la façon dont ils interagissent. Tout d'abord, nous détaillons les données d'entrée du modèle. Dans un second temps, nous donnons un aperçu des principaux processus qui composent ce modèle et des structures produites. Enfin, nous présentons le processus de contrôle de l'agent ainsi que la façon dont il combine les différents composants du modèle.

1 Les données d'entrée

Les comportements des individus dépendent de nombreux facteurs. Le plus important de ces facteurs est l'activité que ces individus ont l'intention d'effectuer. La façon dont cette activité est effectuée dépend beaucoup de la typologie et de la nature de l'environnement ainsi que des contraintes temporelles liées à la réalisation de cette activité. Les caractéristiques et préférences personnelles des individus ont également un impact important sur leurs comportements. Doter

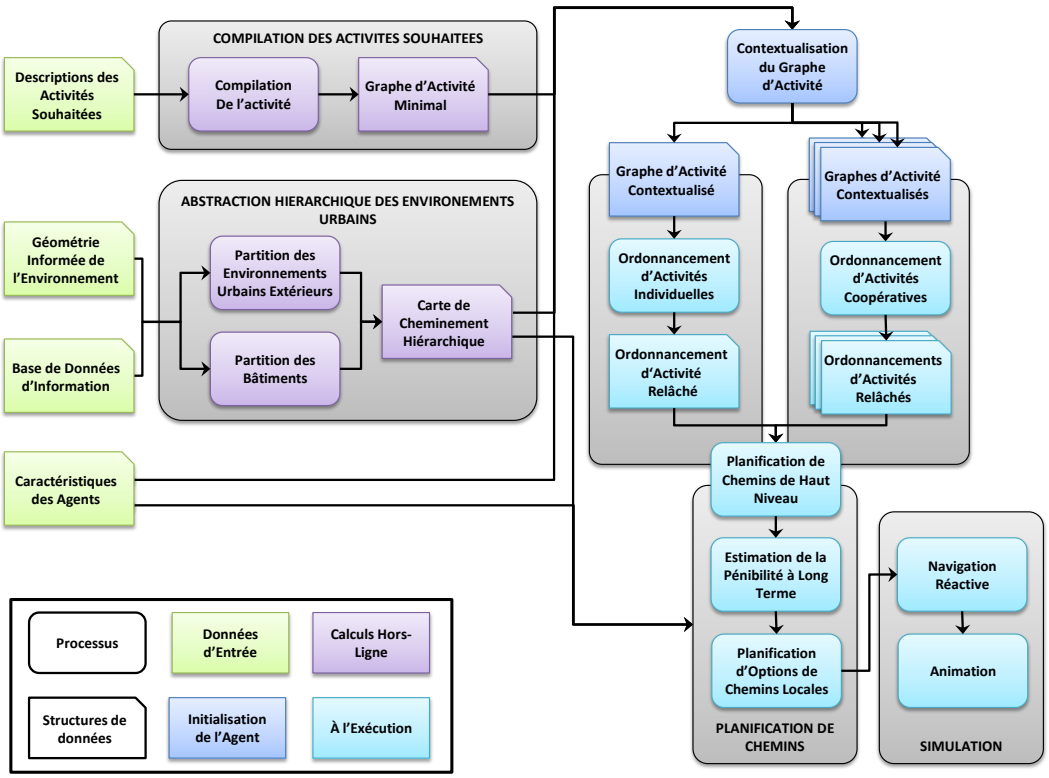


Figure 1 – Vu d'ensemble de notre modèle décisionnel.

les agents dans une foule virtuelle de comportements crédibles exige que ces facteurs soient pris en compte. Nous proposons une description de ces facteurs sous la forme de quatre structures de données, qui sont utilisées comme entrées par notre modèle :

- Une **représentation géométrique informée de l'environnement**. Elle consiste en un ensemble non organisé de triangles en trois dimensions, représentant les surfaces de navigation ainsi que les obstacles dans l'environnement. La nature des zones de navigation a un impact sur la navigation des personnes [Jaklin et al., 2013]. Notre modèle comprend donc un procédé extrayant automatiquement des zones sémantiquement cohérentes de la géométrie. Toutefois, certaines informations sont impossibles à déduire de la seule géométrie de l'environnement (par exemple, il est difficile de distinguer automatiquement les passages piétons des routes ou la fonction des bâtiments). La géométrie de l'environnement doit donc être étiquetée avec de l'information concernant la nature des zones de navigation et la typologie des lieux. Ces zones sont par exemple étiquetées comme étant réservées aux piétons ou aux véhicules, étant publiques ou privées, ou étant une boulangerie, une boucherie. . .. Cette information est conservée au cours du processus d'analyse de l'environnement afin d'être disponible aux processus de planification de chemin et d'ordonnancement d'activités. Cette information initiale peut être soit attachée manuellement à la géométrie, soit extraite d'environnements informés préexistants (Open Street Map, par exemple), soit attachée au cours du processus de génération de l'environnement (dans le cas d'environnements générés procéduralement, par exemple).
- Une **description de l'activité souhaitée des agents**. Cette description est structurée

en un arbre d'activités. Chacune de ces activités est soit une tâche (une action atomique à effectuer dans un lieu unique) soit un ensemble de sous-activités. Les tâches sont étiquetées avec des informations utiles aux processus d'ordonnancement d'activités : des horaires de réalisation admissibles, une estimation de la durée et de la pénibilité associées à la réalisation de cette tâche et une estimation de l'impact de cette tâche sur la pénibilité à long terme. Les tâches ou sous-activités qui composent une activité sont liées entre elles par des constructeurs. Ces constructeurs sont les opérateurs qui décrivent la façon dont ces éléments peuvent être combinés. Dans notre modèle, nous utilisons les constructeurs suivants : *sequence* (sous-activités à réaliser dans l'ordre), *without order* (sous-activités à réaliser dans n'importe quel ordre), *either* (choix d'une sous-activité) et *interlace* (combine les activités en les entrelaçant au niveau des tâches). Ces constructeurs permettent la description d'activités très complexes. Par exemple, les opérateurs *without order* et *interlace* permettent de décrire une activité qui peut être effectuée de différentes façons. Au sein de la description de l'activité, les tâches peuvent être contraintes à un emplacement et/ou à un intervalle de temps spécifique (par exemple, un rendez-vous nécessite que l'agent soit au bon endroit avant le début du rendez-vous). La structure de la description de l'activité est intuitive, la rendant simple à comprendre : des activités complexes pouvant être réalisées de diverses façons peuvent facilement être conçus manuellement. Cela permet également la réutilisation de sous-activités au sein d'activités différentes, réduisant ainsi l'effort nécessaire à la production de ces descriptions.

- Une **base de données d'information** relie la description de l'activité à celle de l'environnement. Elle associe à des typologies de lieux les tâches qui peuvent y être effectuées. Elle décrit également les contraintes temporelles potentielles associées aux lieux (horaires d'ouverture des commerces...). Le fait que cette information ne soit ni incluse dans la description de l'environnement ni dans celle de l'activité rend ces deux descriptions indépendante. Les activités peuvent ainsi être décrites sans aucune connaissance de la structure ou de la nature de l'environnement. Cela permet donc la réutilisation d'activités dans différentes simulations : seule la base de données d'information doit être modifiée pour adapter ces activités au nouvel environnement.

- Les **caractéristiques et préférences individuelles des agents**. Des personne souhaitant réaliser des activités similaires n'effectuent pas nécessairement les mêmes tâches dans le même ordre, ni ne sélectionnent les mêmes lieux où effectuer ces tâches. Cela est dû au fait que chaque personne possède ses propres préférences et caractéristiques, qui influent sur sa prise de décision. Afin de représenter cette diversité, nous dotons chaque agent d'un ensemble de paramètres et de préférences personnelles. La navigation entre les zones étant particulièrement importante pour l'ordonnancement d'activités, un ensemble d'allures est fourni à chaque agent, avec des vitesses associées ainsi que des préférences sur la nature des zones de navigation. Les agents peuvent aussi être dotés de préférences concernant les tâches à effectuer et les lieux ou les effectuer. Ces caractéristiques et préférences permettent la génération de multiples archétypes d'agents (par exemple des retraités, enfants, étudiants...). La génération de ces archétypes peut être effectuée sur la base d'études comportementales concernant ces groupes de personnes (par exemple, les personnes âgées ont tendance à éviter de se presser, au contraire des étudiants...). La répartition de ces archétypes dans la ville peut être effectuée sur la base d'études statistiques sur la répartition des populations dans les villes réelles. Par exemple, il est possible d'avoir une connaissance de la répartition des personnes dans des types de logements en fonction de leurs catégories socio-professionnelles. La description de ces caractéristiques et préférences personnelles permet la génération de foules dans laquelle une grande variabilité de comportements peut émerger d'un nombre limité de descriptions d'activités.

Ces données d'entrée contiennent les informations que nous pensons significatives pour l'ordonnancement d'activités et pour la planification de chemins. En outre, ces données possèdent de bonnes propriétés. Tout d'abord, la description de l'activité, de l'environnement et les agents sont indépendants les uns des autres. Cela signifie que ces descriptions peuvent être générées par des processus distincts, que ce soit à la main ou grâce à des procédés automatisés. Cela permet également la conception d'archétypes d'agents ou d'activités, qui peuvent facilement être réutilisés dans des simulations différentes. La base de données d'information est la seule partie de la description initiale qui doit être mise à jour afin de relier les différentes données d'entrée. Enfin, la description de ces données est intuitive, permettant de les décrire manuellement si nécessaire. Plus généralement, le formalisme des données d'entrée que nous proposons permet la génération de descriptions intuitives, indépendantes et réutilisables de l'environnement, des agents et de leurs activités souhaitées.

2 Processus de raffinement des données

Les descriptions de l'environnement, de l'activité et des agents présentées dans la section précédente démontrent de bonnes propriétés en termes d'indépendance, de réutilisabilité et d'intuitivité. Cependant, ces descriptions doivent être raffinées sous des formes plus appropriées à l'ordonnancement d'activité et à la planification de chemin. Notre système contient trois processus de raffinement de données : le processus d'abstraction de l'environnement, la compilation de l'activité et sa contextualisation.

2.1 Processus d'abstraction de l'environnement

La représentation initiale de l'environnement est une géométrie brute informée, à savoir un ensemble inorganisé de faces triangulaires, étiquetées avec des informations sur la nature de la zone. Une représentation plus adaptée à la planification de chemin et à l'ordonnément d'activité est nécessaire. Les décompositions exactes offrent de bonnes propriétés pour l'analyse des environnements et la planification de chemins. Par conséquent, la première partie de notre processus d'abstraction consiste à extraire une triangulation de Delaunay informée des surfaces de navigation, contrainte par des goulets d'étranglement [Lamarche, 2009]. Les environnements informés et abstractions hiérarchiques présentent également de bonnes propriétés permettant d'améliorer la crédibilité des chemins planifiés [Farenc et al., 1999] et de diminuer la complexité de la planification [Brand and Bidarra, 2011]. Notre processus d'abstraction génère une décomposition sémantiquement cohérente de l'environnement au moyen d'une hiérarchie de zones. Ces zones sont identifiées à l'aide de paramètres tels que leur forme, leur nature, la nature de leurs frontières et leur connexion aux zones voisines. Elles sont regroupées en une hiérarchie de niveaux d'abstraction. L'abstraction de l'environnement repose sur deux processus de décomposition automatiques distincts :

- Un premier processus permet de décomposer les environnements urbains extérieurs en zones correspondant à des concepts d'urbanismes communément admis. Il extrait trois niveaux d'abstraction. Le niveau des *city areas* identifie des grandes zones telles que des rues, des carrefours, des zones piétonnes et des bâtiments. Le niveau des *street sections* identifie les trottoirs et fournit des informations sur les possibilités de passage entre ces trottoirs. Enfin, le niveau des *navigation tiles* identifie de petites cellules approximativement convexes et sémantiquement homogènes qui fournissent la plus petite unité de planification de chemin.
- Un second procédé décompose les bâtiments extraits par le premier processus en quatre niveaux d'abstraction. Le niveau le plus élevé distingue les espaces intérieurs, extérieurs et

couverts. Les niveaux intermédiaires identifient respectivement les étages des bâtiments et les pièces. Enfin, le niveau inférieur identifie des *navigation tiles* similaires à celles extraites par le premier procédé.

Sur la base de ces décompositions, une carte de cheminement hiérarchique informée est extraite. Elle représente complètement la connectivité des surfaces de navigation tout en fournissant toutes les informations sur la nature de l'environnement requises par nos processus de planification de chemins et d'ordonnancement d'activités. Chacun des niveaux de la carte de cheminement est adapté à un niveau de prise de décision. En effet, la carte de cheminement de haut niveau est adaptée à la planification de chemins grossiers à l'échelle de la ville, le niveau inférieur est adapté à une planification de chemins locaux précis, enfin les niveaux intermédiaires permettent de guider la sélection de chemins locaux en fournissant des informations sur l'impact à long terme de l'utilisation de ces chemins. Ce processus d'abstraction de l'environnement est développé dans le chapitre 4.

2.2 Compilation de l'activité

La représentation de l'activité comme un arbre de sous-activités et de tâches est intuitive et facile à concevoir. Cependant, cette représentation souffre de plusieurs faiblesses lorsqu'elle est utilisée pour l'ordonnancement d'activités. Premièrement, ces arbres d'activités peuvent être partiellement redondants : deux activités différentes peuvent partager certaines sous-activités (par exemple, les activités consistant à acheter du pain dans une boulangerie ou dans un centre commercial peuvent partager une tâche consistant d'obtenir de l'argent à un guichet automatique). Cela signifie que la représentation de l'activité sous forme d'arbre n'est pas forcément minimale et que pour une même activité, plusieurs représentations équivalentes peuvent exister. Cela signifie également que, en cas d'échec dans la réalisation d'une tâche, le processus de récupération d'échec aura des difficultés à déterminer si certaines tâches ont déjà été satisfaites dans d'autres activités ou non. Deuxièmement, les algorithmes d'ordonnancement reposent sur la détection des prochaines tâches que l'agent puisse effectuer. Comme la représentation sous forme d'arbre ne mentionne pas explicitement les séquences de tâches possibles, cette information est plus complexe à obtenir. Afin de générer une description de l'activité plus adaptée à son ordonnancement, la représentation sous forme d'arbre est compilée en un graphe d'activité. Ce graphe d'activité est un automate à états fini qui décrit toutes les séquences de tâches valides menant à la réalisation de l'activité. Les états dans ce graphe représentent les situations dans lesquelles l'agent peut être. Les transitions entre ces états représentent les tâches qui doivent être effectuées pour progresser vers la réalisation de l'activité. Le processus de compilation assure que l'automate à états fini calculé contient le nombre minimal d'états et de transitions nécessaires à la représentation de l'activité. Cela signifie qu'aucun état n'est dupliqué dans le graphe, assurant ainsi qu'aucune séquence de tâches ne puisse être manquée en cas de récupération d'échec. De manière générale, le graphe d'activité permet un accès facile aux prochaines tâches auxquelles l'agent doit s'intéresser. Ce processus ne reposant que sur la description de l'activité, il peut être précalculé indépendamment de la représentation de l'environnement. Le processus de compilation de l'activité est développé dans la section 5.1.

2.3 Contextualisation du graphe d'activité

Le graphe d'activité compilé offre une représentation de l'activité adaptée à son ordonnancement. Cependant, elle ne repose que sur la description initiale de l'activité, indépendamment de la description de l'environnement. Pourtant, lorsque l'on considère des tâches à effectuer dans une ville, une partie importante des contraintes temporelles à considérer dépendent de l'environnement lui-même (horaires d'ouverture des magasins, par exemple). Pour prendre en

compte cette information, le graphe d'activité est contextualisée en utilisant les informations fournies par la base de données d'information. Les contraintes temporelles associées aux tâches sont déduites des contraintes décrites dans la description de l'activité initiale, ainsi que de celles décrites dans la base de données d'information. Cela permet le calcul de plages horaires au cours desquelles chaque état doit être atteint pour que la réalisation de l'activité soit possible. Cette information est utile car elle permet de savoir si un agent est encore capable de réaliser l'activité tout en respectant les contraintes temporelles associées au problème. L'estimation de la pénibilité à long terme occasionnée par l'exécution de tâches est également propagée dans le graphe afin d'associer une pénalité à chaque état. Cette pénalité fournit des informations utiles au processus d'ordonnancement d'activités concernant l'impact à long terme de ces tâches sur la pénibilité globale de la solution. De plus, le processus de contextualisation du graphe d'activité peut retirer certaines tâches du graph si aucune façon de les effectuer n'existe dans l'environnement, simplifiant de fait celui-ci. Le processus de contextualisation du graph d'activité est développé dans la section 5.1.

3 Eléments principaux du modèle

Dans ce document, nous proposons trois différents processus décisionnels qui simulent différents niveaux de prise de décision dans les environnements urbains. Le processus d'ordonnancement d'activités individuelles sélectionne une séquence de tâches, ainsi que des lieux et horaires auxquels l'agent doit effectuer ces tâches afin de réaliser son activité souhaitée. Le processus d'ordonnancement d'activités coopératives étend le processus d'ordonnancement individuel en traitant des tâches devant être effectuées en coopération par plusieurs agents. Le processus de planification de chemin hiérarchique sélectionne des ensembles d'options de chemins locaux, permettant à l'agent de se déplacer entre les lieux sélectionnés.

3.1 Processus d'ordonnancement d'activités individuelles

Le processus d'ordonnancement d'activités individuelles vise à simuler le processus rationnel impliqué dans le choix de la meilleure façon d'effectuer une activité. Il sélectionne une séquence de tâches valides dans le graphe d'activité conduisant à la situation but. Des lieux appropriés pour effectuer ces tâches sont sélectionnés de manière à minimiser la pénibilité globale associée à la réalisation des tâches ainsi qu'aux déplacements entre ces lieux. L'estimation de la pénibilité associée à la réalisation d'une tâche est calculée en utilisant la description de la tâche, le potentiel temps d'attente et les préférences personnelles de l'agent concernant la réalisation des tâches. La pénibilité liée aux déplacements de l'agent est calculée sur la base des distances entre les lieux, des capacités de navigation de l'agent et de leurs préférences personnelles en termes de nature des surfaces de navigation. Des intervalles de temps relâchés sont également associés à chaque tâche, indiquant quand l'agent doit exécuter ces tâches. Ils informent l'agent du délai imparti aux déplacements entre les lieux, permettant la sélection de vitesses de navigation adéquates et la détection de potentiels retards. Le processus d'ordonnancement d'activités individuelles sélectionne également un ensemble de rues à parcourir afin de rejoindre les lieux sélectionnés. Il utilise une heuristique ainsi qu'une méthode d'élagage sur la base des contraintes temporelles, afin de réduire considérablement le coût de calcul inhérent à la complexité du problème, tout en assurant de trouver une solution optimale.

Notre processus d'ordonnancement d'activités est complet, ce qui signifie que si une solution existe, elle sera trouvée. Le comportement généré fait preuve de capacités de planification à long terme, en décidant par exemple de presser le pas afin d'éviter un futur détour, en remettant des tâches réalisables à plus tard afin de ne pas être en retard à une réunion ou en passant

à son domicile afin d'y déposer des sacs de courses. Le processus proposé prend également les caractéristiques personnelles des agents en compte, générant donc des ordonnancements potentiellement différents pour des agents partageant des activités similaires. De cette façon, une grande diversité de comportements peut être générée à partir d'un nombre relativement faible de descriptions d'activités. Nous démontrons à travers une étude comment ce processus permet la génération d'ordonnancements d'activités statistiquement proches de ceux choisis par des personnes. Ce processus d'ordonnancement d'activités est développé plus en détail dans le chapitre 5.

3.2 Processus d'ordonnancement d'activités coopératives

De nombreuses activités comprennent, outre des tâches individuelles, des tâches coopératives telles que des réunions ou des échanges de documents. Ces tâches nécessitent une synchronisation entre plusieurs personnes. D'une part, si ces tâches ne sont pas limitées à un seul lieu et horaire, l'ordonnancement des activités individuelles de chaque agent ne permet pas de prendre une décision commune. D'autre part, la prise en compte de la combinaison des activités de tous les agents conduit potentiellement à des problèmes de complexité insurmontables. Pour éviter cette complexité, nous découplons l'ordonnancement des tâches individuelles et des tâches coopératives des agents. Pour réduire davantage la complexité, le processus proposé sélectionne également un sous-ensemble prometteur de possibilités de coopération et utilise un ensemble de méthodes de filtrage. Le processus d'ordonnancement d'activités coopératives est divisé en trois étapes principales. D'abord, pour chaque agent, l'algorithme d'ordonnancement d'activités individuelles est utilisé afin de calculer un ensemble de propositions de coopération. Ces propositions sont les façons optimales qu'un agent à d'atteindre un état dans lesquelles la tâche coopérative peut être effectuée. Deuxièmement, les propositions de tous les agents impliqués dans la tâche coopérative sont combinées et synchronisées. Cela crée un ensemble filtré de configurations de coopération optimales. Enfin, les plus prometteuses de ces configurations sont testées afin de sélectionner une solution offrant la pénibilité combinée la plus basse pour l'ensemble des agents impliqués.

Ce processus est en mesure d'ordonnancer des activités contenant de multiples tâches coopératives ainsi que des tâches individuelles. Les lieux et horaires auxquels les tâches coopératives sont sélectionnés en prenant en compte les activités individuelles des agents impliqués, ainsi que leurs contraintes et caractéristiques personnelles. Un compromis est réalisé entre la qualité de la solution calculée et les performances du procédé. Plus le processus sélectionne de configurations prometteuses, plus la solution calculée est proche de l'optimal, mais plus le coût de calcul de la recherche est élevé. Cependant, le processus est complet, et nous montrons que l'ordonnancement sélectionné est généralement proche de l'optimal, même avec un nombre limité de configurations sélectionnées. Ce compromis, combiné avec des méthodes de filtrage, permet de réduire considérablement le temps de calcul nécessaire à la recherche d'une solution. Ce processus d'ordonnancement d'activités coopératives est développé plus en détail dans le chapitre 6.

3.3 Processus de planification de chemin hiérarchique

L'agenda relâché produit par le processus d'ordonnancement d'activités fournit un ensemble de lieux à atteindre ainsi que des horaires associés. Le rôle du processus de planification de chemin consiste à sélectionner un chemin adapté conduisant au prochain lieu à atteindre. Cependant, lors de leurs déplacements dans les villes, les individus ne considèrent pas tous les détails de leur parcours à la fois. Au lieu de cela, ils considèrent d'abord un chemin grossier en terme de rues à traverser. Ce chemin est affiné lorsque nécessaire, en prenant en compte les données locales

accessibles. Notre processus de planification de chemin simule cette prise de décision en prenant avantage de la représentation hiérarchique de l'environnement. Il utilise cette représentation de l'environnement, l'agenda relâché et la description des caractéristiques de l'agent comme entrées. Il s'effectue en trois étapes principales :

1. La **planification d'un chemin de haut niveau.** Un chemin grossier est calculé, permettant d'atteindre le lieu où la prochaine tâche doit être effectuée. Ce chemin repose sur le niveau supérieur de la carte de cheminement, qui décompose l'environnement en rues, carrefours, zones piétonnes et bâtiments.
2. **Estimation de la pénibilité à long terme.** Les niveaux intermédiaires de la représentation de l'environnement sont explorés. Cette exploration estime l'effort nécessaire à atteindre l'objectif depuis les limites de ces zones intermédiaires. Cette estimation est utilisée par la suite pour fournir à l'agent des renseignements sur l'impact à long terme de ses décisions locales. Afin de réduire la complexité de la recherche, seuls les niveaux intermédiaires appartenant aux zones urbaines sélectionnées sont considérés.
3. **Planification d'options de chemins locales.** Au cours de la simulation, un ensemble d'options de chemins locaux est calculé à l'intérieur des zones que l'agent s'apprête à traverser. Ce calcul se fonde sur le niveau le plus bas de la carte de cheminement. Les options de chemin calculées sont organisées en un réseau de segments reliant les frontières des zones de bas niveau. Chacune de ces frontières est étiquetée avec une estimation de la pénibilité du chemin restant à parcourir pour atteindre l'objectif. Les segments du graph sont orientés de manière à converger au plus vite vers les objectifs locaux, tout en évitant les boucles et les impasses.

Ce processus de planification de chemin prend avantage de la nature hiérarchique de la représentation de l'environnement pour fournir à l'agent de multiples options de chemins locaux. Ces options sont mises à jour quand l'agent perçoit des évènements inattendus sur sa route. L'agent peut alors sélectionner l'option la plus prometteuse qui s'offre à lui à chaque frontière de zone, en prenant en compte l'estimation de l'impact à long terme de ces décisions locales. La nature hiérarchique du processus de planification de trajet remplit également la fonction habituelle de réduction de la complexité de la planification. Ce processus de planification des chemins est développé plus en détail dans la section 4.2.

4 Contrôleur de l'agent

Pendant la simulation, un système de navigation réactive est utilisé pour suivre les options de chemins sélectionnés tout en évitant d'éventuels obstacles. Ce processus fournit un chemin lissé libre de collision à travers les zones navigables. L'agent est animé le long de ce chemin, et des comportements de plus bas niveau ou des animations pré-calculées sont utilisés pour représenter la réalisation des tâches. L'agent est également doté d'un contrôleur, décrit dans la figure 2, qui est chargé de combiner les différents processus impliqués dans le comportement de l'agent. Il est également responsable de la détection d'événements inattendus et du déclanchement des processus requis à l'adaptation du comportement de l'agent à ces évènements. Dans cette section, nous allons d'abord traiter le fonctionnement normal du contrôleur, puis nous considérerons la réaction aux événements et échecs inattendus.

4.1 Fonctionnement normal

Si la connaissance qu'a l'agent de son environnement est suffisante et qu'aucun évènement inattendu ne se produit, le comportement de l'agent est initialisé comme suit :

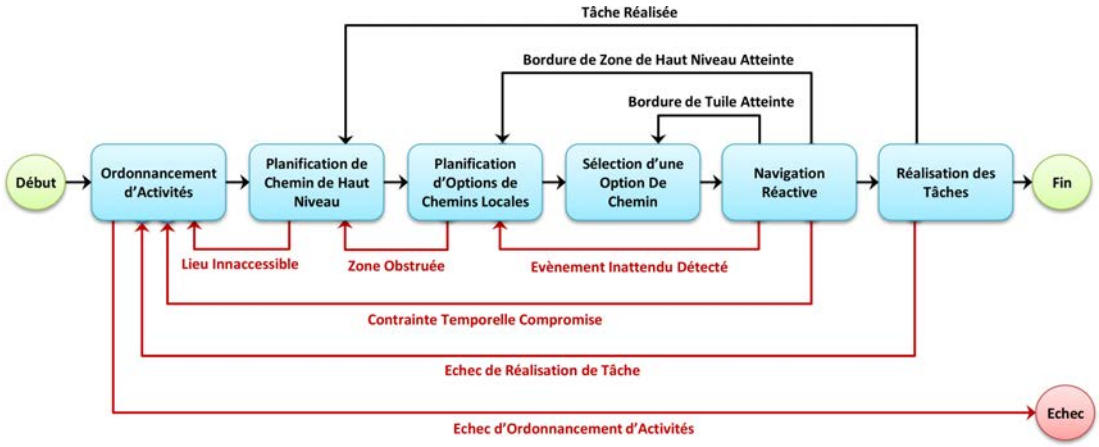


Figure 2 – Contrôleur de l'agent. Les flèches dans la partie haute représentent le fonctionnement normal du contrôleur. Les flèches dans la partie basse représentent l'adaptation aux évènements et échecs inattendus.

1. Tout d'abord, l'activité de l'agent est ordonnancée, individuellement ou en coopération avec d'autres agents.
2. Un chemin de haut niveau est calculé à travers la ville menant au lieu où la première tâche doit être effectuée, les efforts à long terme sont calculés dans les zones sélectionnées.
3. Un ensemble d'options de chemins locaux est calculé dans la première zone à traverser
4. Une première option de chemin est choisie parmi celles qui quittent la position initiale de l'agent.

Ensuite, l'agent commence à suivre le premier chemin sélectionné en utilisant le système de navigation réactive. Le fait d'atteindre des points particuliers le long du chemin déclenche certains de nos processus de prise de décision :

- Chaque fois qu'un lieu où une tâche doit être effectuée est atteint, le comportement de bas niveau associé à cette tâche est déclenché. Lorsque la réalisation de la tâche est terminée, une nouvelle planification de chemin de haut niveau est effectuée vers le lieu où la prochaine tâche doit être effectuée.
- Chaque fois que l'agent est sur le point d'atteindre une bordure de zone de bas niveau, il utilise sa perception de son voisinage et les informations fournies par l'estimation de l'impact à long terme des options de chemin proposées pour prendre une décision rationnelle sur l'option de chemin à suivre.
- Enfin, chaque fois que l'agent est sur le point d'atteindre une bordure de zone de haut niveau, une planification des options de chemins locaux est effectuée dans la zone atteinte.

Lorsque la dernière tâche que l'agent souhaite accomplir est effectuée, le processus s'arrête sur un succès.

4.2 Récupération d'erreur et réaction à des événements inattendus

Les villes sont des environnements dynamiques dans lesquels de nombreux événements imprévus peuvent se produire. Ces événements peuvent influer sur le comportement des agents, les

retarder ou même les empêcher de suivre la route prévue. Cela signifie que la faisabilité ou l'optimalité des chemins ou des horaires prévus peuvent être compromis. Voici une liste des types d'échecs et d'événement inattendu que notre algorithme est capable de traiter :

- Certains événements tels que des densités de piétons inattendues, par exemple, peuvent retarder l'agent. Pendant le trajet, le contrôleur peut détecter que le prochain objectif ne peut être atteint dans l'intervalle de temps relâché spécifié. Le **respect des contraintes temporelles étant compromis**, un nouvel ordonnancement de l'activité est réalisé.
- La réalisation d'une tâche peut échouer, par exemple, si un magasin est fermé de manière inattendue. Dans ce cas, un nouvel ordonnancement de l'activité doit être effectué, afin de trouver un autre moyen de réaliser l'activité souhaitée.
- **Des événements imprévus** tels qu'une flaque d'eau ou un groupe de personne debout sur le chemin peuvent parfois rendre la traversée d'une zone plus pénible. Dans ce cas, la pénibilité associée aux options de chemins correspondant à cette zone est augmentée. Le reste des options de chemins locales est mis à jour en conséquence, ce qui reflète la volonté de l'agent d'éviter cet obstacle, si possible.
- D'autres **événements inattendus**, comme une voiture garée sur un trottoir, par exemple, peuvent parfois obstruer complètement des options de chemins. Dans ce cas, ces options de chemin sont invalidés, les autres options de chemins sont mises à jour en conséquence, ce qui reflète le besoin d'un potentiel détour.
- De grands obstacles tels que les travaux routiers peuvent **obstruer toute une zone de haut niveau**. Dans ce cas, la planification de chemins locaux dans cette zone rencontre une erreur, car aucun chemin n'existe qui permette de traverser cette zone. Dans ce cas, un nouveau chemin de haut niveau doit être calculé, après avoir marqué le segment correspondant dans la carte de cheminement comme obstrué.
- Des zones obstruées peuvent conduire à des **emplacements inaccessibles**. Si une planification de chemin de haut niveau ne parvient pas à trouver un chemin menant au lieu où la tâche suivante doit être effectuée, cela signifie que cette zone est inaccessible. Un nouvel ordonnancement de l'activité doit être calculé en tenant compte des zones obstruées.
- Un **échec d'ordonnancement de l'activité** est possible, cela signifie qu'aucun arrangement valide n'a été trouvé permettant de respecter toutes les contraintes spécifiées dans la description de l'activité. Dans ce cas, les contraintes doivent être assouplies ou certaines tâches retirées de l'activité. Toutefois, cela est dans le domaine de la planification d'actions et non dans celui de l'ordonnancement d'activités, ce qui est en dehors du champ d'application de notre méthode.

5 Conclusion

Notre modèle simule une partie du comportement humain en environnement urbain, en mettant l'accent sur les processus de l'ordonnancement d'activités et de planification de chemins. Elle repose sur des données d'entrée indépendantes : une représentation de la structure et de la nature de l'environnement, une description d'activités complexes incluant des contraintes spatiales et temporelles et une description des caractéristiques et préférences personnelles des agents. Le modèle combine ces représentations à l'exécution afin de tenir compte de la relation étroite entre l'espace, le temps, l'activité et l'agent. La réalisation de l'activité souhaitée s'adapte automatiquement à la structure et la nature de l'environnement, aux contraintes temporelles et spatiales

spécifiques de l'activité et de ses caractéristiques et préférences de l'agent. De cette façon, une grande variété de comportements peut être générée à partir d'une seule description d'activité. La description de comportements complexes incluant des tâches coopératives nécessitant une synchronisation entre plusieurs agents est possible. Dans ce cas, les activités personnelles, les contraintes et les préférences de tous les agents impliqués sont prises en compte afin de trouver une solution offrant un bon compromis aux agents impliqués. Les trajets des agents entre les lieux où ils ont l'intention d'effectuer des tâches sont guidés par un réseau d'options de chemins. Le long de son trajet, ces options sont proposées à l'agent, qui en choisit une en se basant sur une estimation de l'incidence à long terme de ce choix sur la pénibilité globale du chemin. Cela permet à l'agent de réagir de manière transparente à un large éventail d'événements inattendus. Les problèmes que nous traitons impliquant des espaces de recherche de grande taille, des efforts particuliers ont été déployés afin de réduire les coûts de calcul des processus utilisés, grâce à l'utilisation d'heuristique, de techniques de filtrage et d'élagage. Plus généralement, les bonnes propriétés de notre modèle permettent la génération de comportements de piétons plus réalistes dans les villes virtuelles. De cette manière, des phénomènes macroscopiques réalistes peuvent émerger automatiquement de la somme des comportements individuels des agents.

Introduction

Many applications require populating virtual environment with large crowds of autonomous agents. In most of these applications, for example in films or video games, the goal is to give life to the environments by populating them with virtual crowds. In such applications, generating crowds exhibiting visually credible behaviours is sufficient. However, in some applications such as city planning validation, it is critical that the generated crowds exhibit behaviours which are consistent with real crowds' behaviours. The behavioural animation field of study aims at generating such crowds by embedding each agent with decisional processes that simulate human decision making. In cities, most of the observable human activity is due to people navigating between the locations where they perform their daily tasks. For this reason, the generation of consistent activity schedules and paths for individual agents is critical for the generation of realistic crowd behaviours. We propose a model that focuses on activity scheduling and path planning for virtual pedestrians populating urban environments. It aims at driving crowd simulations in entire virtual cities over long periods of times, up to a whole day.

We introduce our work by exploring the problem of realistic crowd simulation in virtual cities. Next, we present our main contributions and how they help handling this problem. We then discuss possible applications of our model. Finally we develop the plan of the document.

1 Problem statement

In cities, macroscopic phenomena emerge from the sum of people's behaviours. For example, the densities of pedestrians may drastically increase in front of a school around its end time and noticeable flows of people may appear between housing and working areas around the times people usually go to work and get back from it. These macroscopic phenomena are highly dependent on the tasks peoples intend to perform, and on the path they travel between the locations where they intend to perform those tasks. Therefore, in order to generate virtual crowds in which realistic macroscopic phenomena are observable, it is critical to consistently schedule virtual pedestrians' activities.

Scheduling an activity involves selecting a sequence of tasks to perform in order to realise the activity, but also choosing in which order these tasks must be performed, at which locations and when. Such decisions rely on a high number of interacting parameters. The structure of the environment and the repartition of locations where tasks can be performed have a huge impact on these decisions. Indeed, people tend to limit the distance they have to travel when scheduling their activity. For example, if a person has to buy some groceries, he may decide to go to small shops if passing by a shopping street, or to visit a mall if there is one close by. Similarly, a person may decide to shop near is his home in order to limit the distance to travel carrying bags. Time is also of great importance when scheduling an activity. Indeed, people often have to respect temporal constraints. Sometimes, for example if a person has to take a train, this activity is strongly constrained, as the person must not be late. This strongly impacts the person's activity scheduling, requiring the tasks to be performed in an order enabling the satisfaction of this hard temporal constraint. This means that temporal constraints may force the agents to select a sequence of tasks that does not optimise their

travels. Sometimes, people's activities may be loosely constrained temporally, meaning that being late is possible without much trouble. In that case, people may decide to be late in order to avoid a detour. Therefore, the space, time and activity are tightly related and cannot be considered independently when scheduling an activity. A wide variety of pedestrian behaviours is noticeable in urban environments. This diversity is due to people having different activities to perform and bearing different spatial and temporal constraints. But this diversity is also due to the differences between people's personal characteristics and preferences. Indeed, a businessman often favour the optimisation of the time required to perform his activity while a retired person may prefer limiting the travelled distances. Furthermore, people's activities often include cooperative tasks requiring coordination between multiple persons. In that case, the activity scheduling becomes extremely complex, as the personal activities, constraints and preferences of all the involved individuals must be considered.

The way people plan paths between the locations where they intend to perform tasks greatly impact their behaviour. Path planning consists in finding a suitable path, clear from obstacle, going from the person's location to the location he intends to reach. However people do not plan all the details of this path at once. A coarse path at the city level is first considered in term of streets to travel. The more local decisions are delayed until required information is available. Indeed, during navigation, a person adapts his path to the local structure of the environment and to unexpected events. This illustrates how people rely on a hierarchical representation of their environment's structure [Hirtle and Jonides, 1985]. Indeed, people consider different levels of abstraction, decomposing their environment into commonly used city planning concepts such as city blocks, streets, crossroads, sidewalks and crosswalks. People use similar representations of buildings, considering established architectural elements such as courtyards, floors, rooms and balconies. For example, when planning a path in a building, people first aim at reaching the right floor before searching to reach the right room in this floor, and finally the goal location in this room. However, people do not solely rely on the structure of their environment when planning a path. They also consider semantic information such as the nature of navigation zones or the right they have to access to private areas. For example, people avoid walking on the road and do not usually go through their neighbours' house, even if it is the shortest path. People's personal preferences may also impact their path planning. For example, some people are more likely to cross streets out of crosswalks than others and some people prefer travelling pedestrian streets or parks, even if it means travelling longer distances.

We believe that handling the tight relationship between space, time, activity and individual preferences is required in order to consistently schedule activities. The complex compromise involved in the scheduling of collaborative activities must also be considered. The hierarchical nature of the environments' structure must be taken into account in order to plan credible paths enabling the realisation of the scheduled activities. This way, by properly modelling pedestrian behaviours, realistic macroscopic phenomena should emerge from the combination of the consistent individual schedules and paths of all the agents. However, none of the existing decisional models takes the complex interactions between all these parameters into account.

2 Our contributions

In this thesis, we propose a model that aims at generating credible behaviours for virtual pedestrians in urban environments. This model focusses on producing activity schedules consistent with the ones people choose and on planning sets of path options the agent can dynamically pick from. These schedules and paths are used to drive crowd simulation systems, in order to produce crowds in which realistic phenomena emerge from the sum of individual behaviours. Our model relies on intuitive and independent representations of the environment, of the agent and of their intended activity. It consists of four original independent processes that can be

combined into a consistent model or used separately. These processes are, respectively:

1. An environment abstraction process able to automatically extract a semantically coherent hierarchical representation of a virtual city. It relies on two independent methods, which respectively decompose outdoor urban environments and buildings into multiple levels of semantically coherent zones inspired by city planning and architecture concepts such as streets, crossroads, sidewalks, courtyards, floors and rooms. An informed hierarchical representation of the environment is extracted from these two decompositions. Each level of environment abstraction provided by this representation is adapted to a different level of decision-making. It makes this representation especially suitable for hierarchical path planning purpose.

2. A hierarchical path-planning process that takes advantage of the hierarchical nature of this environment representation to delay local decisions until suitable information is available. At first, it selects a general path in term of streets to travel. This path is refined during navigation in a network of local path options the agent can dynamically choose from. This process considers the structure and nature of the navigation surface and takes the personal preferences of the agent into account. It improves the path selection of the agent, enabling a better adaptation to unexpected events, while limiting the computation cost of the path planning.

3. An individual activity scheduling process that handles the tight relationship between time, space and activities. It is able to schedule agents' intended activities in large environments such as virtual cities, and over long periods of time, up to a whole day. It selects a sequence of task to perform compatible with an intended activity description. Locations in the environment where these tasks must be performed are selected and a relaxed temporal schedule is produced. The typology and nature of the environment are considered, as well as spatial and temporal constraints associated with the agents' intended activities. This process also considers the agents' personal characteristics and preferences. This way, variability in the execution of similar activities is automatically produced. We demonstrate that the generated activity schedules are statistically consistent with the ones produced by humans in the same situation.

4. An extension of the individual activity scheduling process to activities that include cooperative tasks requiring coordination between multiple agents. In order to avoid the complexity issues inherent to the high dimensionality of the problem, this process schedules the individual parts of agents' activities independently and relies on filtering and pruning methods. This process is able to find an optimal solution to the problem, selecting the best locations and time to perform the cooperative tasks. However, it is also capable of finding a close-to-optimal solution in a fraction of the time by considering only a small number of promising solutions.

The model we propose combines these processes in order to handle the rational part of pedestrians' navigation behaviour in virtual cities, from the analysis of the environment to the low-level path planning. As it considers the tight relationship between the space, the time, the activity and the agents' characteristics, it is able to produce credible individual behaviours. Our model is made in such a way is that the descriptions of the environment, of the activity and of the agents are independent. It means that new agent archetypes and activities can be seamlessly added to the simulation. This enables the generation of credible populations and typical activities from statistical data. It also means that an activity can be scheduled in a-priori unknown environments, for example procedurally generated ones.

3 Domains of applications

Our model produces consistent activity schedules and paths in virtual cities. Using this model to drive simulations enables the emergence of more realistic individual behaviours, enhancing the global quality of virtual crowds. However, the unique properties of our model make it especially adapted to some specific applications, such as urban planning validation, automatic virtual storytelling and some video games. The domain of **urban planning validation** aims at detecting design faults in city planning blueprints such as insufficiently wide passageways, ill-placed doorways in a building or a lack of crosswalk. Populating 3D models of planned urban areas with crowds of people enable the detection of such faults by analysing virtual agents' flows. This way, design faults can be fixed before the city area is actually built, avoiding potential modification costs. However, the detection of such design faults requires that the generated flows are representative of the ones that would exist in the built environment. This implies taking into account the locations of important location in the city, such as shops, public buildings, housing areas and work areas. It must also take into account the temporal variations of pedestrian flows, for example the fact that people move between housing and working areas at fixed hours and that more people wander in shopping areas on Saturdays afternoons than on Sunday mornings. As our model considers the tight relationship between space, time and agent's activities, it is able to generate a credible behaviour for each agent. Realistic flows and densities should emerge from the sum of all these individual behaviours, enabling a correct identification of city design faults. Furthermore, our model permits an easy generation of archetypes of agents with different characteristics and preferences. This enable the generation of groups of people consistent with the actual population of the tested city area, based on statistical studies, for example.

The **virtual storytelling** domain aims at generating complex stories from high-level scenario descriptions. Virtual actors act in accordance with the scenario description, navigating in the environment, performing actions and interacting with other actors. As our model enables the description of complex activities including cooperative tasks, it can be of great help for such applications. Indeed, a story can be authored at a very high level by describing a complex activity for each actor, including meetings or other coordinated actions. Our cooperative activity scheduling process is able to automatically select locations in the environment where the actors may perform their actions or meet with other actors. Continuity in the actors' behaviours is thus ensured, avoiding some unrealistic situations such as actors teleporting to reach the next scene, for example. Our method also ensures that an authored scenario can be automatically adapted to unknown environments, and that new actors or actions can be seemingly added to the scenario. This way, the story designer may easily edit his scenario to observe the obtained stories. Furthermore, our method enables the addition of temporal and spatial constraints to the actors' activities, giving a greater control of the generated story to the scenario designer. In the domain of **video games**, last years have seen the emergence of procedurally-generated environments. These environments enable a greater replayabilty of the game and relieve the game designers of the tedious task of hand-building a world. However, as the structure of these worlds is a-priori unknown and sometimes even evolves with the player's actions, NPCs (Non-Player Characters) behaviours gets more complex to design. We believe that our model would bring a solution to this problem. Indeed, our model adapts the agent's behaviour to the environment at runtime. The game designer just has to embed NPCs with high-level activity descriptions and let the model adapt this activity to the environment. Adding some virtual storytelling elements to this solution could enable the automatic generation of complex quests from high-level descriptions. Indeed, many quests or stories can be expressed as an interaction between the player and NPCs performing a mix of individual and collaborative tasks, with

associated spatial and temporal constraints. For example, the player may have to prevent a document to be exchanged between two characters, pursue a thief escaping a castle or spy on a cult gathering. Our model is able to automatically select locations where these events could take place and adapt the NPCs behaviours accordingly, providing the player with a potentially infinite number of different quests from a limited number of high-level descriptions.
Overall, we believe that the good properties of our method make it a real interest for applications that require agents subject to multiple spatial and temporal constraints to perform consistent activities in virtual environments. It is able to handle a wide range of applications, from the simulations of thousands of agents performing archetypal daily activities to the realisation of complex stories including multiple interactions between virtual actors.

4 Document organization

In this document, we present the different processes composing our model. The first two chapters describe the context and related works. They respectively cover environment representation, path planning methods and decisional models. In chapter 3, we give an overview of our model, describing its different components and the way they are linked together. Chapter 4 gives a more precise description of our environment abstraction process and of our path planning process. Chapter 5 further details our individual activity scheduling process and chapter 6 describe the extension of this process to cooperative activities. Finally, we conclude our document with a discussion on the good properties of our model and on the future research topics it could lead to.

Environment representation and path planning

1

Navigation is one of the most essential interactions between an entity and its environment. Indeed, interacting with an object or another entity usually requires reaching it in the first place. For this reason, agents' navigation behaviours have been extensively investigated in the robotics and crowd simulation fields of study.

When an agent has to reach another position in its environment, its navigation behaviour can be decomposed in two main phases. First, a free path linking the current position of the agent to its goal is determined. Second, the agent follows this planned path while avoiding dynamic obstacles. The initial path planning phase relies on the agent's capability to reason on the structure of its environment. However, virtual environments are primarily defined as raw geometries, usually a set of triangles localized in space. Therefore, planning an adequate path in these environments depends on the capacity to extract an exploitable representation of this geometry. Three main categories of representations have been extensively studied: *potential fields*, *roadmaps* and *cells decompositions*. However, exploring these representations may prove resource-consuming in complex environments. Hierarchical abstractions of environments have been proposed to improve the path planning performances. Furthermore, people do not only take the geometrical structure of their environment into account but also consider navigation surfaces based on semantic information. As this semantic information cannot always be automatically deduced from the geometry of the environment, the notion of informed environments was proposed. In these environments, navigations zones are labelled with semantic information, guiding the environment analysis and path planning processes.

In this Chapter, we discuss the topics of environments analysis and path planning, organised as follows. First, an overview of environment representations is presented. Second, we introduce some path planning methods using these representations. Then, the idea of informed environments is explained. Finally, we discuss the concept of hierarchical representation of the environment and its properties in term of navigation.

1 Environment geometric representation

A virtual environment is foremost described as a geometry, usually a set of triangles localized in space. This geometry describes navigation surfaces and obstacles. The notion of configuration space [Lozano-Perez, 1983] can be used to define the navigations surfaces accessible to the agent. A configuration is a set of values that define the state of the degrees of freedom of a system. The configuration space possesses one dimension for each of these degrees of freedom, thus containing all the configurations of this system. The workspace divides the configuration space in two distinct parts: the free space C_{free}, composed of all admissible configurations, and the obstacle space $C_{obstacle}$, composed of all the impossible configurations. For example, for a robotic arm, a configuration would contain the value of every joint's angle and the workspace would separate the admissible arm configurations from the ones that generate

collisions with the environment. In the case of path planning, a configuration is usually defined as a 2D position of the agents in a two dimensional map of the environment. The workspace distinguishes the navigable surfaces in this environment from obstacles. Planning a path in the environment consists in finding a suitable list of configurations belonging to C_{free} and linking the initial configuration to the goal configuration. In order to perform this path planning, a suitable representation of the free space has to be extracted. In the domain of robotics, in which navigation is compulsory, many methods have been proposed to describe the structure of environments [Latombe, 1991][Choset, 2005][LaValle, 2006]. Three main approaches can be distinguished: the potential fields approach, the roadmap approach and the cell decomposition approach. In this section, we focus on the application of these methods to path planning in two dimensional configuration spaces.

1.1 Potential fields

The *potential fields* method is an easy and intuitive way to represent the interaction between an agent and its environment. It offers a representation that can be directly used for agent navigation. In this method, repulsive forces are associated with obstacles while an attractive force is associated with the goal of the agent [Reich et al., 1994] [Treuille et al., 2006]. A global gradient of forces, called potential field, is deduced from the sum of all these forces (see Figure 1.1). The agent can then navigate in this potential field by weighting it with the distance from the goal then following a gradient descent method until its goal is reached.

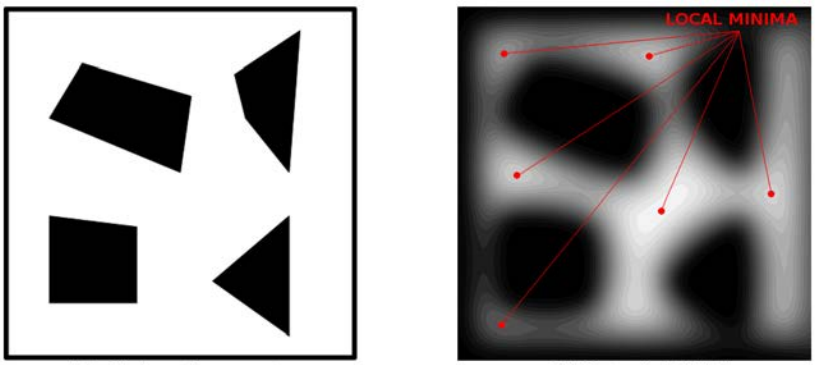


Figure 1.1 – Potential field (b) computed from the environment geometry (a) Greyscale represent the strength of the repulsive force applied on the agent. Local minima are displayed.

Though this method enables a simple and efficient agent navigation, it suffers of a main weakness: local minima. These local minima are zones of the environment where an isolated lowest value of potential field appears despite not being the goal of the agent (see Figure 1.1). During its navigation, if an agent reaches one of these points, it will be trapped at this location, hence never reaching its goal. Methods were proposed to deal with this issue. Among them, the *Random Path Planner* method [Barraquand and Latombe, 1991] uses a random configurations selection to generate a graph of inter-linked local minima. The agent can use this graph in order to escape local minima and reach its goal.

Potential fields are generally not well adapted to the simulation of virtual humans' navigation, due to their lack of controllability. It does not enable reasoning on the structure of the environment but directly guides the agent navigation. Furthermore, the chosen path can be highly sub-optimal. However, this method has been shown to be efficient to simulate very large crowds [Gloor et al., 2004] in which the focus is on global flows rather than on individual

behaviours.

1.2 Roadmaps

Environment representation methods based on roadmaps tend to capture the connectivity of the free space thanks to sets of standardized paths. Configurations belonging to the free space are labelled as waypoints. These waypoints are linked together if a free path exist between them i.e. the interpolation of these two configurations do not intersect $C_{obstacle}$. In a 2D geometry, this means that it is possible to travel between these waypoints in a straight line without colliding with an obstacle. Multiple methods exist, using different strategies to place waypoints and to link them.

Probabilistic roadmaps. The *Probabilistic roadmaps* method randomly selects configurations belonging to the free space [Overmars, 2002]. Edges are then added between each waypoint and its k closest visible neighbours i.e. neighbours that can be reached by a free path (see Figure 1.2). This method suffers from a major issue concerning the density of waypoints: the higher is the waypoint density, the best are the chances of representing the connectivity of the configuration space. However, whatever the chosen density, the completeness of the representation cannot be guaranteed. Furthermore, the higher the waypoint density is, the more complex the graph becomes. In the case of large open environments, a high waypoint density means a huge number of paths that are not needed to represent the structure of this environment. This means that, when selecting a waypoint density, a trade-off has to be made between the graph complexity and the chances of completeness. In the case of environments composed of both large open zones and more intricate zones like corridors, this trade-off is hard to make. In their work, Simeon et al. propose the *visibility roadmaps* as a solution to this issue [Siméon et al., 2000]. In this method, the waypoints are randomly placed in a way that none of them are inter-visible. Then, extra waypoints are added to enable a connection between these first waypoints. This way, the density of generated waypoints is dependent on the complexity of the obstacles. However, this method still does not guaranty to completely represent the connectivity of the configuration space.

Another issue lie in the fact that probabilistic roadmap methods offer no guarantee concerning the representativeness of the generated paths. Indeed, the waypoints being placed at random, the generated path do not accurately represent the real distances to be travelled. An optimal path between two points in the roadmap is not necessarily the optimal path in the environment. Furthermore, the generated path is usually noisy. A path optimization phase is required to smooth it and to produce a more suitable route. The borders of obstacles not being explicitly represented in the roadmap, numerous queries have to be made on the geometry to check that the produced optimized path does not collide with obstacles. Therefore, the path optimisation is a costly, yet necessary, process.

The lack of representativeness of generated path and the cost of the path optimisation impede the method reliability for agent path planning in large environments. However, it has shown good properties to plan complex locomotion movements (walking on poles, moving along complex obstacles, jumping...) [Choi et al., 2003][Pettre et al., 2003] or to deal with navigation behaviours of flocks of agents [Bayazit et al., 2002].

Rapidly exploring Random Trees. Rapidly exploring Random Trees (RRT) are designed to explore the configuration space by rapidly producing a random exploration tree [LaValle, 1998]. The tree is developed from the original position of the entity by randomly picking waypoints in the environment (see figure1.3). By limiting the maximal expansion of the tree at each exploration step, the method tend to explore areas that were not explored yet, thus being able to rapidly produce a tree that effectively covers the free space. The search stops

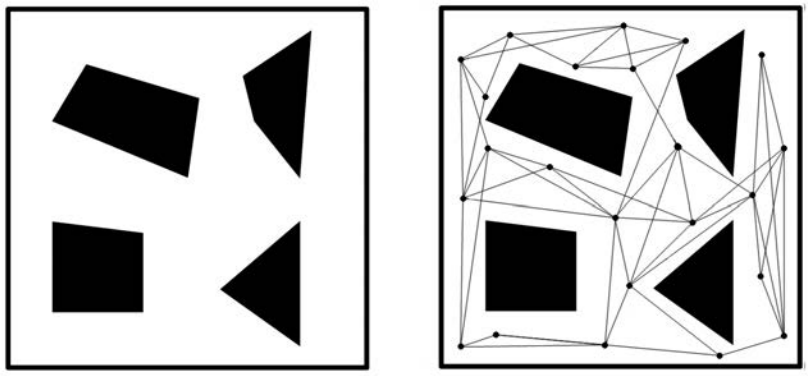


Figure 1.2 – An example of probabilistic roadmap (b) computed from the environment geometry (a). Note that one path has been missed in the top-right corner of the environment, demonstrating the non-completeness of the method.

when the goal of the agent is reached, providing a free path to the entity. This efficiency of the method was improved in [Kuffner and LaValle, 2000] with the RRT-connect method. It relies on developing a second tree from the goal of the entity. Both trees are developed in parallel until a free path is found between them. This way, a free path between the origin and the goal can be found more rapidly. These methods are especially adapted to the quick planning of paths in heavily constrained environments, without requiring any pre-computation. However, the planned tree is not stored and a new search has to be performed for every new goal. In order to avoid restarting from zero every time a path has to be planned, the Reconfigurable Random Forest (RRF) method was proposed in [Li, 2002]. This method aims at storing the previously computed trees, regularly simplifying their structure by suppressing redundant waypoints in order to keep a compact representation of the environment. Every time a path has to be planned, a solution is first searched in the existing forest of RTT. If no solution is found, new trees are computed and added to the representation. This way, the representation of the environment is enhanced throughout the simulation depending on the needs of the entity.

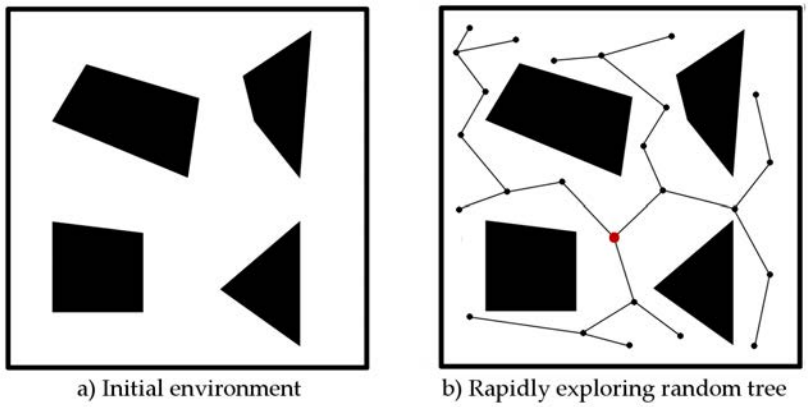


Figure 1.3 – An example of rapidly exploring random tree (b) computed from the environment geometry (a). The origin of the exploration is highlighted in red.

Visibility graphs. The visibility graph method aims at completely representing the connectivity of the free space and guaranteeing the accuracy of the estimated distances. In this method, waypoints are placed on the vertices of the polygons describing the shape of the obstacles. These waypoints are then linked if inter-visible (see Figure 1.4). This means that obstacle edges also belong to the visibility graph. The good property of this method is that it always contains the shortest paths between two waypoints [Arikan et al., 2001]. However, the size of the graph depends on the number of inter-visible vertices in the environment geometry. Big open environments, featuring punctual obstacles or complex geometry will tend to generate huge numbers of edges in the graph. This makes visibility graphs an inefficient method in such kind of environments.

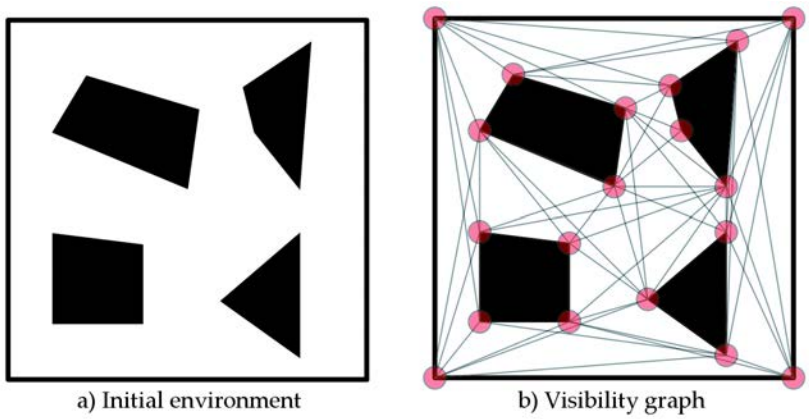


Figure 1.4 – Visibility graph (b) computed from the environment geometry (a). waypoints are placed on obstacle vertices (highlighted in red).

Generalized Voronoï diagrams. The generalized Voronoï diagram method builds paths as sets of configurations equidistant from surrounding obstacles (see Figure 1.5). Waypoints are defined as the configurations where multiple paths intersect. These configurations have the property of being equidistant from at least three obstacles. The obtained graph completely represents the connectivity of the configuration space. Obtained paths also have the good property of maximizing the distance to obstacle, which limits the risk of collision between the agents and obstacles. This property is very important in robotics applications, in which obstacle avoidance is primordial. The computation of the generalized Voronoï diagram is expensive. However, methods have been proposed that use video cards capabilities in order quickly compute an approximation of the Voronoï diagram [Hoff III et al., 1999].

Corridor maps. The corridor map method provides a way for fast computation of high quality path in the environment [Geraerts and Overmars, 2007]. It relies on a reachability roadmap method[Geraerts and Overmars, 2006], which provides a backbone roadmap close to a generalised Voronoï diagram. Then, it uses the concept of *clearance* introduced in [Kamphuis and Overmars, 2004] to compute corridors along this backbone roadmap (see Figure 1.6). When an agent has to plan a path in the environment, he selects a corridor to travel. In this corridor, navigation is controlled by a potential field, generating smooth and short paths. This kind of method offers a better representation of the navigable surface than classical roadmap methods. They prove to be adapted to dynamic obstacle avoidance and to dealing with the navigation of groups of agents[Geraerts and Overmars, 2007].

Roadmaps provide simple and condensed representations of the free space. They offer a straightforward way to plan paths, as path planning method can directly be used on the

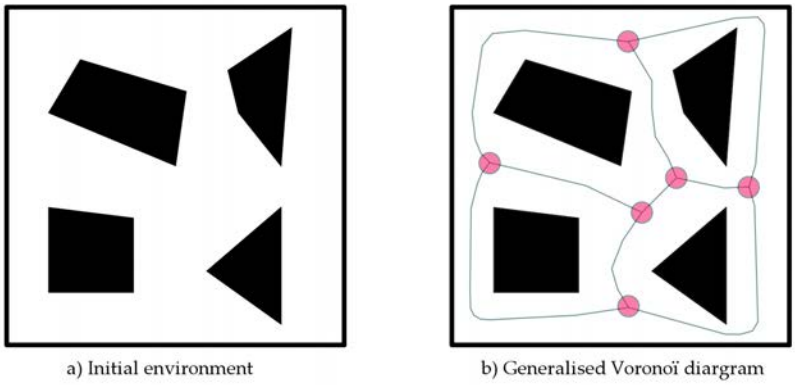


Figure 1.5 – The Generalized Voronoï Diagram (b) computed from the environment geometry (a). Waypoints are highlighted with red circles.

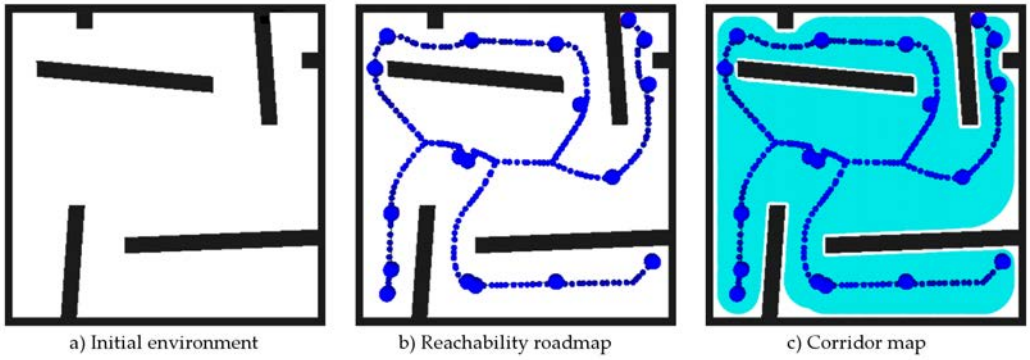


Figure 1.6 – The reachability roadmap (b) and corridor map[Geraerts and Overmars, 2007] (c) computed from the environment geometry (a). A set of corridors is determined, providing the agent with knowledge on the available clearance around the backbone roadmap.

generated graph. However, these methods do not explicitly model the borders of obstacles. Yet, this knowledge is required when optimizing the computed path, to ensure that it does not collide with an obstacle. This is especially true for probabilistic roadmap, in which the optimized path can be distant from the originally planned path. Overall, roadmaps provide an intuitive structure adapted to path-planning, but lack an explicit representation of the free space.

1.3 Cell decomposition

Roadmap-based approaches provide a set of paths representing the navigation opportunities through the free space. However, they lack knowledge on the positions of obstacle, requiring costly queries every time the agent moves away from the path. In order to describe explicitly the shape of the free space, cell decomposition approaches tend to represent the free space as a set of connected cells, either exactly or approximately. The agent is thus able to reason on navigable areas, free of obstacle, instead of free paths. This approach offers more freedom of movement to the agent and avoids costly path optimisation queries.

Approximate cell decompositions. Approximate cells decomposition methods represent the environment as a set of predefined cells shapes which union is strictly included in the

free space. These methods are widely used for agent navigation, due to their simplicity of construction. Uniform grids are the most common approximate cells decompositions methods. They represent the free space as union of squares[Shao and Terzopoulos, 2005a]. These squares are labelled as occupied if partially or completely covered with static obstacles, and free if not (see Figure 1.7). This method provides a basic approximation of the navigable areas in the environment. However, it suffers from a precision issue: the connectivity of the free space may not be completely represented, depending on the size of the cells. Indeed, if the definition of the grid is not high enough, some narrow paths between two obstacles might be missed. On the other hand, the smaller the cells, the more complex and memory-consuming the representation becomes [Thrun and Bücken, 1996]. Similarly to probabilistic roadmaps, a trade-off has to be made between the precision and the complexity of the representation. This trade-off is also more difficult in environment featuring large free areas as well as smaller zones constrained with obstacles. Regular grids were used in [Loscos et al., 2003] to manage the collision avoidance between virtual agents. It enabled the population of large urban environments with up to 10000 autonomous pedestrians.

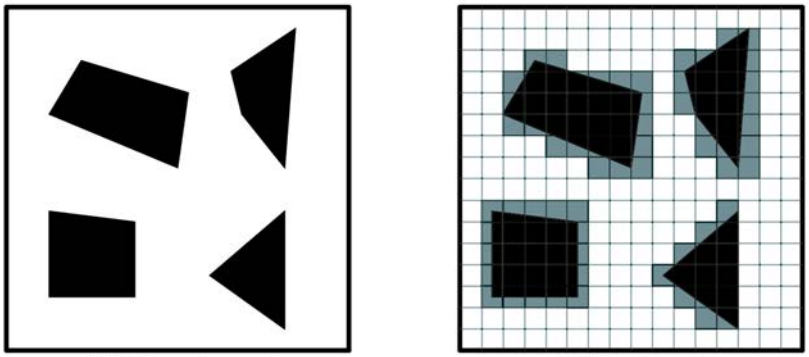


Figure 1.7 – The regular grid (b) computed from the environment geometry (a). White cells are free cells while grey are partially occupied and black are obstacles.

In order to deal with the precision issue of the regular grids, hierarchical decompositions were proposed [Yahja et al., 1998, Shao and Terzopoulos, 2005a]. They decompose free space as a tree of recursively divided cells. In the case of two-dimensional environments, quadtrees are commonly used [Shao and Terzopoulos, 2005b]. They are computed starting from a classical regular low-resolution grid decomposition of the environment (see Figure 1.8.b). Then, every partially occupied cell is divided into four sub-cells, themselves labelled as free, partially occupied and totally occupied (see Figure 1.8.b). This process of decomposition is recursively applied to all sub-cells, until a predefined depth limit or minimum size of cell is reached. By using a more precise decomposition only when needed (along obstacles borders), hierarchical grids offers a better compromise between precision and memory consumption. This proves especially true in the case of large environments containing vast free areas and complex obstacle shapes. However, this method still does not ensure that the connectivity of the free space in completely represented. The memory consumption can still be high, for example if many punctual obstacles exist in the environment.

Other methods were proposed, using different geometrical shapes as a basic cell. For example, in his work, Pettre proposes to represent the free space through a set of navigation cylinders [Pettre et al., 2005]. On a way quite similar to corridor maps [Geraerts and Overmars, 2007], this method relies on a backbone path close to the generalized Voronoï diagram. Cylinders are placed, centred on this backbone, with a radius defined by the closet obstacle. These cylinders represent the clearance around the central path. Navigation corridors, free of obstacles

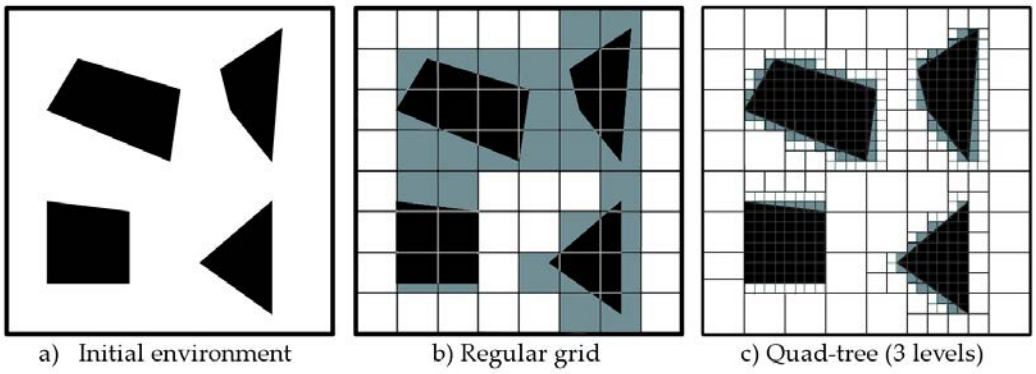


Figure 1.8 – The regular grid (b) and its hierarchical decomposition into a quadtree (c) computed from the environment geometry (a). White cells are free cells while grey are partially occupied and black are obstacles.

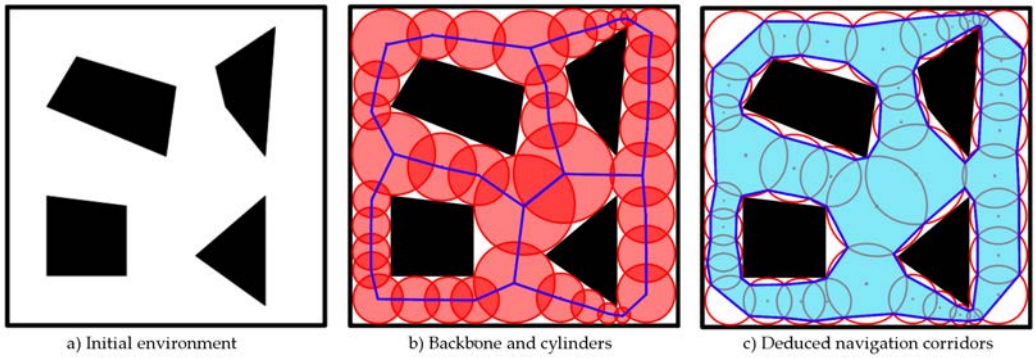


Figure 1.9 – Cylinder decomposition and backbone (b) computed from the environment geometry (a). Navigation corridors deduced from the cylinder decomposition (c)

can be deduced from the intersection between the cylinders. This representation offers a good approximation of the shape of the free space, with a limited number of elements. Once again, this number can be increased to increase the precision of the representation, but leading to a higher memory consumption. This environment representation method was used to simulate large crowds of people in real time [Pettre et al., 2006].

Overall, the easy implementation of approximate cells decomposition methods and the explicit representation of the navigable surface they provide to the agent make them suitable for behavioural animation purpose [Tecchia and Chrysanthou, 2000, Tecchia et al., 2002]. However, these methods require a trade-off to be made between the precision of the representation and the complexity of the generated structure. They do not ensure to represent the complete connectivity of the free space, nor a reasonable memory consumption. Furthermore, none of these methods represent the exact position of obstacle borders, leading to the same optimisation issues as roadmap-based methods: queries on the geometry has to be done when optimizing a path computed over multiple cells or partially covered cells to determine an agent trajectory that do not collide with an obstacle.

Exact cell decompositions. On the opposite of approximate cell decomposition methods, exact cell decomposition methods represent navigable areas as a set of convex cells whose union exactly match the free space. A good characteristic of these decompositions is that all cell vertices are carried by an obstacle border. This means that an edge in this decomposition

is either constrained, representing an obstacle border, or free, representing a passage between two obstacles. Furthermore, these methods do not suffer from the precision issue of exact cell decompositions. Indeed, the size of cells is only determined by the geometry, and the connectivity of the free space is guaranteed to be entirely represented.

Although some methods were proposed based on trapezes [Latombe, 1991], most of exact cell decompositions are based on triangular cells. Among these methods, the constrained Delaunay triangulation [Chew, 1989] is the most popular [Kallmann et al., 2003] [Lamarche, 2009] [Mekni, 2010]. This method exactly decomposes the free space as a set of triangular cells. It is based on the classical Delaunay triangulation method [Delaunay, 1934][Boissonnat et al., 1998]. In a two dimensional environment, this method, links a set of points in a plane in order to produce a triangular partition of the geometry defined by convex hull of these points. Two simple rules are respected: all triangle vertices are supported by a point of the set and the circumscribed circle of every triangle contains no other vertex than the three vertices of the triangle. This tends to avoid elongated triangles in the decomposition and guaranties that every point is linked to its closer neighbour by a triangle edge. Constrained Delaunay triangulations adapts this method to the partition of environments constrained with obstacles [Chew, 1989]. The second rule is modified so those only visible vertices are taken into account when checking that no other vertex is included in the circumscribed circle of the triangle. This has the effect of forcing the triangulation to include the edges of obstacles (see Figure 1.10.b).

Constrained Delaunay triangulations demonstrate good properties. First, the representation is exact: if a path exists between two points in the environment, it will exist in the representation. Second, the complexity of the decomposition is linearly dependent on the complexity of the environment geometry: if the model complexity is supported by the system, the presentation complexity will also be. Third, as all cells vertices rely on obstacles, edges either represent obstacle borders or a passage between two obstacles. As a result, a path in the environment is represented by a unique set of cells in the representation, which simplify some navigation queries and spatial analysis. These properties makes constrained Delaunay triangulations a customary choice for virtual human navigation [Kallmann et al., 2003][Lamarche, 2009][Mekni, 2010].

Multiple extensions to constrained Delaunay triangulations were proposed in the domain of environment representation. A method was used to add bottleneck edges to the triangulation [Lamarche and Donikian, 2004] (see Figure 1.10.c). These bottlenecks are used to determine if a path is wide enough to enable the passage of an agent. The obtained constrained Delaunay triangulation demonstrates good properties in term of navigation queries [Kallmann, 2010] and path planning [Demyen and buro, 2006].

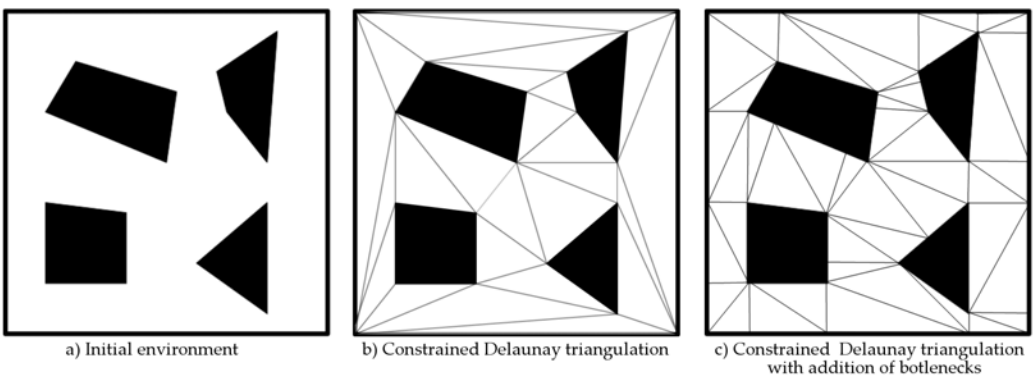


Figure 1.10 – The Delaunay triangulation (b) and a Delaunay Triangulation constrained with bottlenecks (c) computed from the environment geometry (a).

Overall, cell decomposition approaches offers a way to explicitly describe the shape of the free space. Approximate cell decompositions are easy to compute, but they suffer from definition issues: a trade-off has to be made between the precision and the complexity of the representation. Exact cell decompositions completely and exactly represent the free space, offering good properties for agent navigation. Furthermore, even though complex obstacle borders may generate high number of cells, the complexity of the generated representation remains linear depending on the complexity of the environment's geometry.

2 Planning Algorithms

Simulating navigation behaviours require the ability to efficiently plan a path from the actual position of the agent to its goal. Path planning algorithms are graph search algorithms. They require a path planning graph to be extracted from the environment representation. The path planning graph is a set of nodes (waypoints), linked together by edges (accessibility between these waypoints). A cost function associates a cost to each edge, for example the distance to travel, or the energy consumption in case of robot navigation. Path-planning methods consist of graph-exploration algorithms selecting a succession of edges linking the initial node to the goal node while optimizing the sum of the selected edges' cost. Two main approaches can be distinguished: depth-first approaches tend to explore a succession of entire paths while breadth-first search methods prefer exploring all paths in parallel.

2.1 Path planning graphs

Path planning methods are graph-exploration algorithms. The path planning graph represents the environment structure and supports the path planning process. The nodes of the graph stand for waypoints or zones of the environment. The edges in the graph model the accessibility between these waypoints or zones. This graph is usually extracted from the environment representation.

Extracting a path planning graph from a roadmap representation of the environment is simple. Every waypoint and edges in the roadmap are respectively turned into nodes and edges in the path-planning graph. Extracting a path planning graph from a cell decomposition of an environment can be done on multiple ways. A first idea is to place nodes at the centroid of cells (see Figure 1.11.b). This way, every node stand for a cell of the representation and each edge stand for a common border between two cells. These properties are suitable for reasoning on the structure of the environment. It also guarantees a limited number of edges, always equal to the number of free edges in the decomposition. However, this method makes the computation of travelled distances harder, as the straight line between two connected cells centroids can pass through an obstacle. A second way of extracting the path planning graph from a cell decomposition is to place nodes at the centre of free borders of the decomposition. Then, all borders belonging to the same cell are linked together with edges. This method offers an easier way to compute the travelled distance, as, given convex cells shapes, all edges only cross free space. However, the number of edges in a cell is quadratic over the number of free borders of this cell. This implies graph complexity issues in case of decompositions using convex cells more complex than triangles. Path planning algorithms require a cost to be associated with each edge in the path-planning graph. A cost function is defined that enable the computation of these costs. This cost function is specific to the problem, depending of the objective of the agent. In many applications, the distance between nodes is used as a cost function enabling the selection of the shortest path. In robotics, the goal is more often to optimise the resource consumption, leading to cost functions that take the slope of the path or its nature into account.

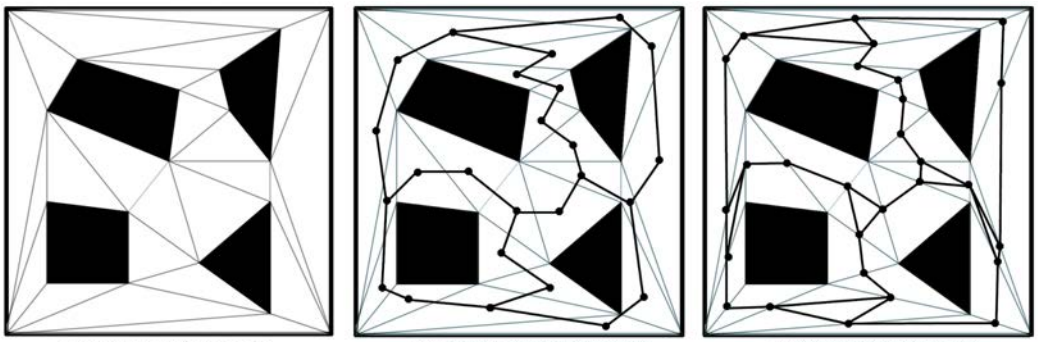


Figure 1.11 – Two examples of path planning graphs extracted from a Delaunay triangulation. In b), nodes of the graph are placed at the centroid of cells. In c), nodes are placed at the centre of free edges.

2.2 Dijkstra algorithm

The Dijkstra algorithm (see Algorithm 1), is a general breadth-first search method that produce the graph's shortest path tree from a given origin node. This tree describes the shortest paths from the origin to every other node of the graph. This graph is produced by expending a search area from the origin until all nodes are visited. This is done by storing a list of exploration states - already explored nodes, associated with the minimal cost needed to reach them and their predecessor state on the path. This list is expanded by iteratively:

1. Picking the node associated with the lower cost in the list;
2. Marking it as visited;
3. Exploring the unvisited neighbours of this node;
4. Either adding the neighbour node to the list if not already in, updating its cost if already in with a higher cost or discarding it if already in with a lower cost.

This process is repeated until the list of exploration states is empty. The returned structure stores the costs and optimal predecessors of each state, enabling to deduce the shortest path tree in the graph.

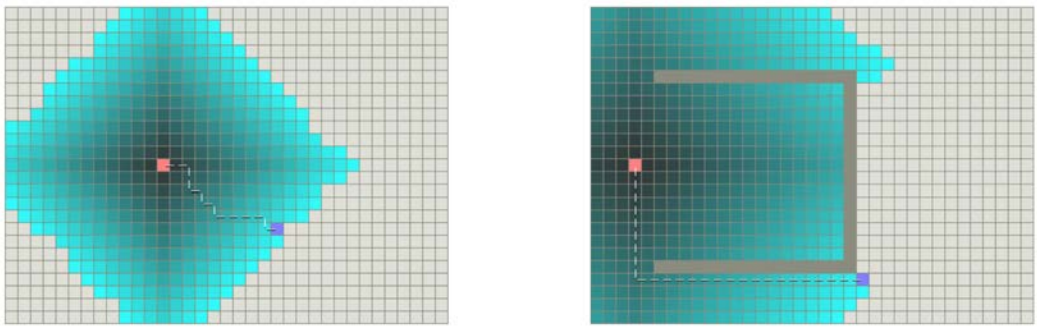
The Dijkstra algorithm can be used to find the shortest path between the origin and a goal node. In that case, the algorithm is executed until the goal node has been marked as visited (see Figure 1.12.a). The search then succeeds and the optimal path is found by iteratively retrieving the predecessor states of this final state. The Dijkstra algorithm is optimal: if it exists, the path minimizing the cost function is found. However, the Dijkstra algorithm is a costly path planning method, as it explores the entire graph. Even when stopping the search when the goal node is found, it explores all the nodes that can be reached by a path with an associated cost lower than the one of the solution. In constrained environments including dead-ends (see Figure 1.12.b), this can lead to exploring most of the environment, inducing a high computational cost.

2.3 Planning with heuristics

The Dijkstra method is not efficient when searching for the shortest path between two nodes, as it explores a large part of the environment. To offer better path planning performances,

Algorithm 1 Dijkstra algorithm

```
1:  function DIJKSTRA(Graph, source)
2:      dist[source] ← 0
3:      for all vertex v ∈ Graph do
4:          if v ≠ source then
5:              dist[v] ← ∞
6:              prev[v] ← NULL
7:          Q.add_with_priority(v, dist[v])
8:      while Q ≠ ∅ do
9:          u ← Q.extract_min()
10:         mark u as visited
11:         for all neighbor v of u do
12:             if v is not yet visited then
13:                 alt = dist[u] + length(u, v)
14:                 if alt < dist[v] then
15:                     dist[v] ← alt
16:                     prev[v] ← u
17:                     Q.decrease_priority(v, alt)
        return dist[], prev[]
```



a) Dijkstra (non-constrained environment)

b) Dijkstra (constrained environemnt)

Figure 1.12 – Representation of the nodes explored by a Dijkstra algorithm, in a non-constrained environment (a) and in a contained environment including a dead-end (b). The colour gradient represent the cost to reach the node (darker means lower cost)

the A* algorithm can be used [Hart et al., 1968]. It uses a heuristic function to guide the exploration of the path-planning graph toward the goal. This heuristic function provides an estimation of the remaining cost to reach the goal from an exploration state. For example, if the cost function is the distance to travel, a good heuristic is the length of the straight line between the actual position and the goal. At every step of the search, the state with the lowest estimated global cost (cost plus heuristic) is picked, favouring exploration states more likely to quickly lead to the goal (see Figure 1.13.a). The use of the heuristic drastically reduces the number of explored nodes, even in constrained environments (see Figure 1.13.b). The closer the estimation is from the real remaining cost, the quicker the search converges. If the heuristic is admissible, i.e. the estimation is never overestimate the remaining cost, the A* algorithm finds the optimal solution. If the heuristic is not admissible, overestimating the real cost by a factor α, the planned path will have a worst cost of $\alpha.c$, c being the cost of the optimal path. The main issue of the A* algorithm is thus to find the admissible heuristic which estimations

are the closer to the real costs. This can prove complex when the cost function is not based on the distance to travel.

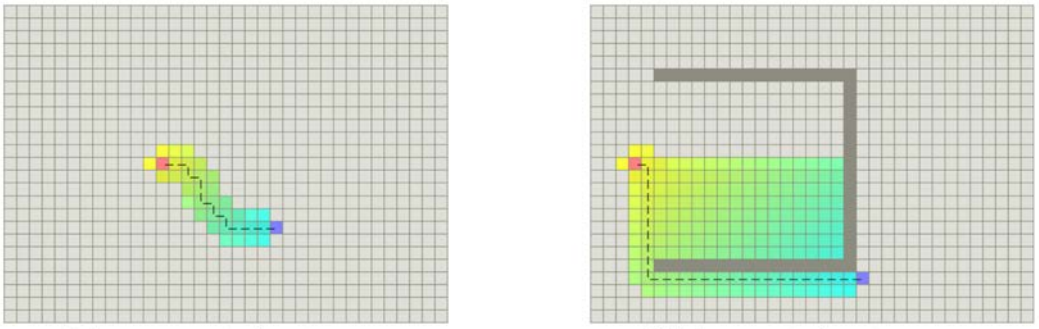


Figure 1.13 – Representation of the nodes explored by an A* algorithm, in non-constrained (a) and contained environments (b). The colour gradient represent the estimated remaining cost for each node (yellow means higher cost)

The IDA* (Iterative deepening A*) algorithm relies on depth-first searches bounded by an estimation of the goal distance [Korf, 1985]. The depth limit is updated with each iteration until the goal is reached. This method suffers from a higher computation complexity than a standard A*, as it visits the same nodes multiple times. However, it does not store all the visited nodes, thus requiring less memory. It makes this algorithm suitable for application in which the available memory is insufficient to handle the planning problem with a traditional A*.

The choice of a path-planning algorithm is dependent on the nature of the problem. The A* algorithm and its derivatives efficiently reduce the computation time of the path planning, making them suitable for real-time applications. However, the planned path totally depends on the chosen cost function while the computation time reduction depends on the chosen heuristic function. Therefore, the main difficulty lies in finding suitable cost and heuristic functions to fit the planning problem. If the nature of the problem makes that no suitable heuristic can be determined, the Dijkstra algorithm can be used. The Dijkstra algorithm is also an efficient method if the problem requires a number of agents to reach a small number of common goals (for example, reaching the closest exit of a building in case of an emergency evacuation). In that case, a single Dijkstra algorithm executed for each goal enable the computation the shortest path from any point to its closest exit, avoiding to plan an individual path for each agent. The IDA* algorithm is especially interesting when the memory consumption of the search has to remain limited.

3 Informed environments

As presented in previous sections, most common environment representation and path planning methods rely solely on the geometry of the environment. Yet, people do not only consider the structure of their environment when choosing a path to travel. For example, an individual do not behave the same way when walking on the road or on pedestrian areas, nor he interacts the same way with different objects. This is due to the fact that people take their knowledge on the nature of their environment into account when navigating. The notion of informed environments was proposed to represent this knowledge, relying on the notion of affordances [Gibson, 1979]. This notion describes the perception of objects in the environment through the action opportunities they offer to the agent. For example, a chair would be perceived as an

object offering the opportunity to sit down.

3.1 Interaction with objects

The concept of affordances is an interesting way to express the possibilities of interaction between the agent and objects in its environment. Relying on this concept, formalisms were proposed, offering practical ways of describing these interactions.

Parametrized Action Representation model. In their work, Badler et al. proposed the Parametrized Action Representation model [Badler et al., 2000]. This model aims at offering a natural description of agents' possible actions in their environment. This description is composed of:

- **Atomic actions** that can be performed by the agent.
- **Complex actions** that are composed of multiple atomic actions to perform, simultaneously or consecutively.
- **Objects** related to actions, described with a set of properties.
- **The agent** that perform the actions. It is embedded with interaction abilities.
- **Application conditions** that have to be matched in order to perform actions.
- **Termination conditions** that stop the action when matched.
- **The effects** of actions, that modify the state of the world, objects or agents.

The description of the actions is implemented as parallel finite-state machines[Badler et al., 1997]. These machines supervise the agent's interactions with objects.

The PAR model enables an intuitive description of actions. However, in this model, the information is not explicitly attached to objects, but described in a large database representing the general knowledge of the agent. This imposes a constant browsing of the database for the agent to find interactions opportunities in its environment.

Smart Objects. The concept of Smart Objects, proposed by Kallmann and Thalmann is directly based on the theories of J.J.Gibson [Kallmann and Thalmann, 1999]. Smart objects are structures that contain everything needed for interaction [Kallmann and Thalmann, 2002]:

- Objects properties (size, shape...);
- Information on the interaction itself (where the agent must be placed to be able to interact, which part to interact with...);
- Objects internal behaviours, triggered by interactions (a lock will unlock if activated with a key);
- Agent's behaviour during interaction.

When an agent wishes to interact with an object, these characteristics are delivered to it. Its behaviour is then controlled by a finite-state machine, triggering basic interaction animations and adapting its posture depending on the object's position.

Smart Objects' main goal is to produce credible interaction animations, not to deal with the long term activity of the agent. However, by combining this approach with action planning methods provides a more global control of the agent's behaviour [Abaci et al., 2005]. This process is performed in four main steps:

1. The agent collects data related to its interactions.

2. The agent produces a representation of the global problem.

3. A decision making process produces a set of plans and the resulting environment states.

4. The plan enabling the agent to reach its goal is performed.

Smart objects were also used to handle collaborative work between agents [Abaci et al., 2004]. A supervisor is used, endowed with the ability to transfer tasks between agents. If an agent is unable to reach its goal because a task cannot be performed, this task is transferred to another agent. The goals of both agents are modified accordingly.

STARFISH model. The Synaptic-object Tracking Actions Received From Interactive Surfaces and Humanoids (STARFISH) model was introduced in [Badawi and Donikian, 2004]. It aims at describing the physical interaction between the agent and objects, or between two agents. This model describes interactions based on eight atomic actions, based on Schank's theory of conceptual dependency [Schank et al., 1972]: *give*, *transfer*, *displace*, *move*, *grasp*, *ingest*, *speak* and *attend*. More complex actions are modelled by assembling these atomic actions through finite-state automata. The actions are linked to objects through interaction surfaces. The position, size and orientation of these interaction surfaces depend on the manipulated objects' characteristics (their shape, size or weight). This way, very general interactions can be described (hold, open...), that automatically adapt to the object it is applied to (see Figure 1.14). This model is really efficient for managing the visual representation of interactions with objects. However, it does not address the behavioural aspect of the interaction.

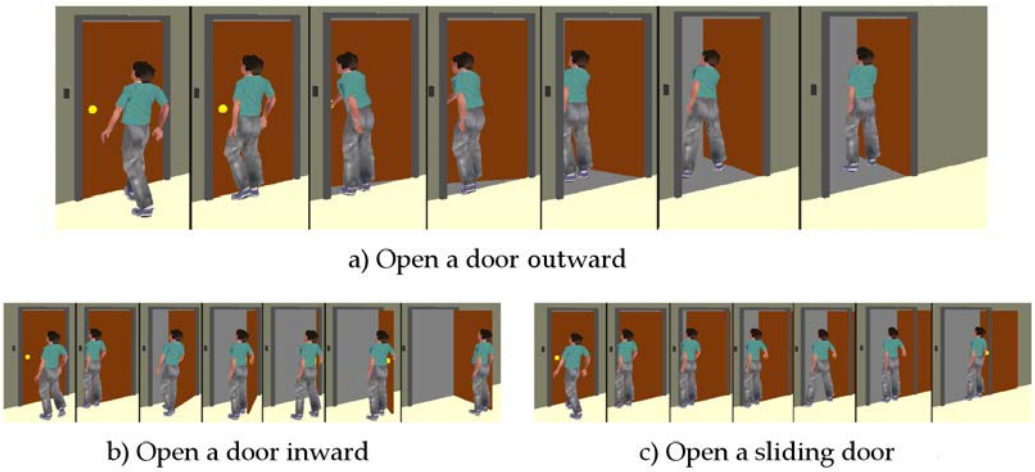


Figure 1.14 – Opening a door with STARFISH model. The physical interaction with the door automatically adapt to the door characteristics. No modification of the action description is needed.

Overall, these methods offer different formalisms to describe the interaction between an agent and its environment. A wide range of interaction characteristics are described, from the conditions and effects of the interaction to the animation of the agent. However, these methods only consider the interactions with objects as individual elements, not as parts of a more global behaviour. The main interest of these methods lies in their combination with decision making and path planning methods. They manage the thin interaction with object, leaving the action planning and navigation to the rest of the system.

3.2 Informed environments for agent navigation

People navigating in urban environments share a set of behavioural rules, either explicitly (traffic rules...) or implicitly (respect of personal space...). For example, people avoid navigating on roads and do not cross private areas to reach their goal, even if it is the shortest path. To represent this common knowledge, the notion of informed environment has been introduced. It associates data related to the behaviour of agents to regions of the environment [Farenc et al., 1999]. In their work, Thomas et al. use a city modeller to produce large-scale informed urban environments[Thomas and Donikian, 2000]. In this environment representations, navigation areas are labelled with their nature (see Figure 1.15). This representation was used to populate virtual city centres with different kinds of actors (pedestrians, vehicles and public transportation systems). It was demonstrated that the semantic information attached to the navigation zones helps to create more credible navigation behaviours. Subsequently, informed environments have been included in many crowd simulation systems, with good results in the simulation of public buildings' activity, for example [Shao and Terzopoulos, 2005a, Paris et al., 2006].

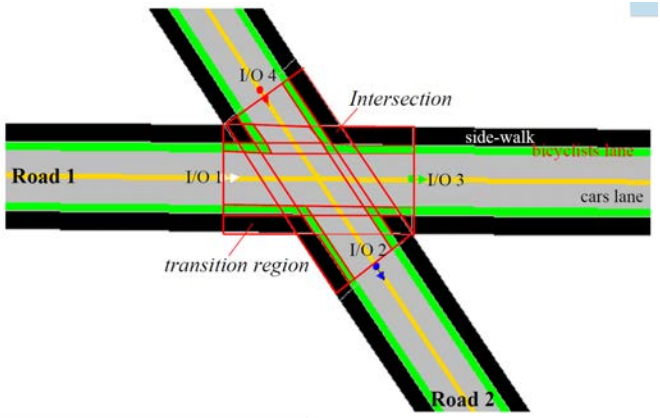


Figure 1.15 – Informed representation of a crossroad [Thomas and Donikian, 2000].

More recently, informed environments were used to take navigation preferences into account. In their work, Jaklin et al. embed agents with different navigation preferences and place them in an environments labelled with the nature of navigation zones [Jaklin et al., 2013]. The path planning method they propose, MIRAN: Modified Indicative Routes and Navigation, uses the nature of zones to weight the travelled distances in the cost function. It produces terrain-dependent paths, taking the agents' personal preferences into account. For example, an adult would avoid puddles as much as possible while a child would attempt to cross as many of them possible (see Figure 1.16).

Informed environments provide the agent with additional semantic information on its environment. This information can concern the possible interaction with objects, enabling the generation of more credible interaction animations. The provided semantic information can also

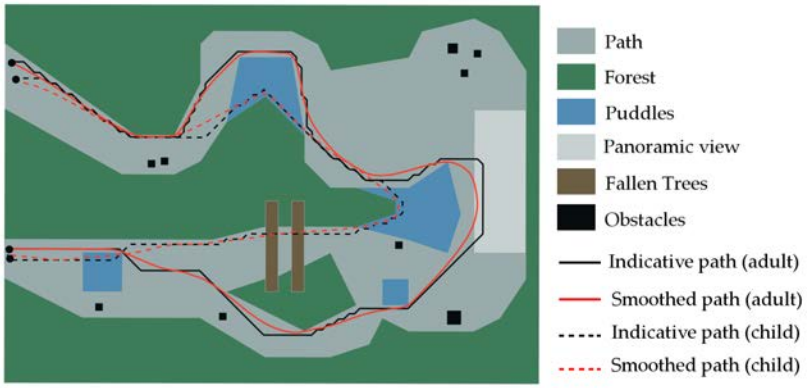


Figure 1.16 – Terrain dependent indicative and smoothed paths produced by MIRAN path planner [Jaklin et al., 2013] for an adult and a child.

describe the nature of navigation zones. It offers the possibility to extend existing path planning methods by taking this nature into account in the path planning process. More credible paths can be planned, some agents favouring some navigation zones and avoiding some others. Furthermore, informed environments enable the generation of navigation behaviours that takes into account information that cannot be deduced automatically solely from the environment's geometry. For example, it is complex to automatically distinguish crosswalks form the road or to deduce the function of a building from its geometry.

4 Hierarchical representations

In the previous sections, we presented environment representations methods and how they were used to plan path in the environment. We saw that cell decompositions, and more particularly exact cell decompositions offer good properties in term of navigation queries. However, these methods tend to generate huge numbers of cells when applied to big environments featuring complex geometries. The complexity of the path planning graphs extracted from these decompositions leads to high computation times. In order to perform efficient path planning in complex environments, more abstract representations of theses environments can be extracted [Botea et al., 2004]. One proposed solution aims at extracting abstract regions from the environment decomposition by merging cells from this decomposition into zones. The resulting path planning graph is reduced in consequence. In his work on Parallel Ripple Search [Brand and Bidarra, 2011], Brand et al. demonstrate that using such an abstracted representation drastically decrease the planning complexity. The idea of hierarchical environments [Botea et al., 2004] was proposed to describe multiple levels of abstraction of an environment. Such representations were used to perform hierarchical path panning [Wong and Loscos, 2008], planning path at interactive rates for thousands of agents.

4.1 Topological abstractions

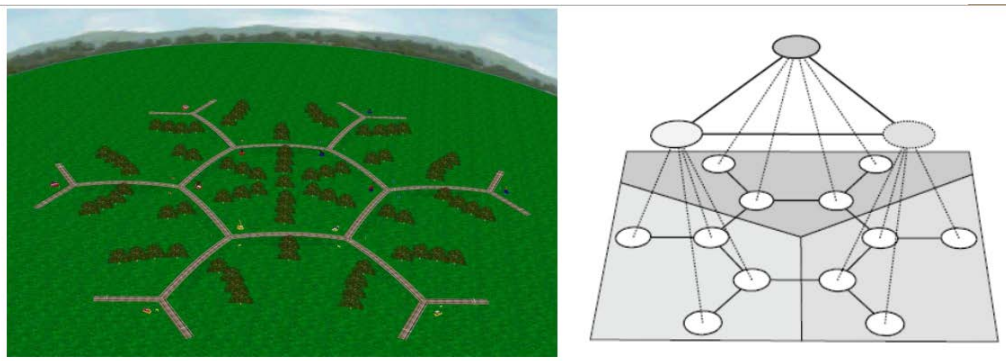
In order to cut down the computational cost of path planning in complex environments, the number of cells in the environment representation must be reduced. To do so, topological abstraction methods relies on the structure of the environment. The geometrical shape of cells and accessibility between these cells are the main elements taken into account. In [Lamarche and Donikian, 2004], Lamarche et al. propose a method to extract a hierarchical abstraction from a set of convex cells. It uses accessibility relations between cells to qualify

them as dead ends (one neighbour), corridors (two neighbours) or crossings (three or more neighbours). Then, these cells are recursively merged following rules such as "two neighbour corridors can be merged in one unique corridor" or "a dead-end can be merged to a neighbour corridor as a unique dead-end". This reduces the graph complexity without losing any path option. The work Paris et al. extends this method by generating a hierarchy of three levels of environment abstractions[Paris et al., 2006]. The first level is the initial cell decomposition. The generation of the second level of decomposition uses two heuristics to guide the topological abstraction process. The first heuristic aims at maximizing the convexity of the abstract zones while the second tend to maximise the size of crossroads. The third level of hierarchy abstracts the second level by further merging its zones depending on their accessibility relations. This model generates a hierarchical abstraction of the environment's structure, efficient in access time, which can handle large and complex environments. This makes it a suitable representation for microscopic crowd simulation in urban environments.

Topological abstraction of environment representations can drastically improve the computation time of planning algorithm, making them suitable for real-time crowd simulation. However, they only rely on the geometry and topology of environments. They lack some semantic information to extract zones that match the human representation of environments.

4.2 Semantic abstractions

As pointed out in [Hirtle and Jonides, 1985], people tend to hierarchically organise their mental representation of the environments (see figure 1.17). These representations are based on different levels of abstraction, which influence different steps of navigation [Wiener and Mallot, 2003]. These levels of abstraction do not solely rely on the topology of the environment. Indeed, we saw on the previous section that people's navigation behaviour rely of the nature of zones of their environments, like sidewalks and crossroads. Their mental abstraction of their environment thus rely more on their knowledge of city planning and traffic rules than on the environment's geometry. Informed environments provide the agent with semantic information on the nature of navigation zones. Using this semantic information enables to automatically extract abstractions of virtual environments that better match the human abstraction.



a) Initial environment b) Hierarchical representation

Figure 1.17 – Hierarchical abstraction (b) of the environment topology (a) [Hirtle and Jonides, 1985].

Some cell decomposition abstraction methods relying on semantic information were proposed. In their work, Farenc et al. propose methods and tools for creating a hierarchical decomposition of an urban environment from an informed environment[Farenc et al., 1999]. This decomposition relies on commonly used city-planning concepts like quarters, blocs, streets,

buildings... It uses the concept of *Environment Entities* providing geometrical data as well as semantic information. Using such an informed representation enables a more realistic simulation of human navigation behaviour. Theses decompositions being tedious to describe by hand, Jiang et al. proposed a method to automatically extract hierarchical decomposition of multi layered environments with floors and stairs [Jiang et al., 2009]. This kind of representation is useful to decompose the path planning in multiple steps, like searching to reach the right floor before trying to reach the right room in this floor. It improves the path planning performances, and was used to generate visually convincing paths for thousands of virtual characters in real-time [van Toll et al., 2011]. In the field of geographic information systems, informed environments were also used to provide hierarchical representation of virtual geographic environments featuring several overlapping layers of semantic information[Paris et al., 2009]. Hierarchical path-planning process in such environments were shown to reduce the computation time and enhance the quality of the computed paths [Mekni, 2010].

The use of topological abstractions of virtual environments enables the reduction of the path planning complexity. However, relying only on the geometry is not sufficient to identify semantically meaningful zones like crossroads and sidewalks in cities, for example. Yet, people rely on such zones to plan their path in their environment. The use of semantic information in the abstraction process enables to extract abstract representations closer to human mental representations, therefore enhancing the quality of virtual human navigation behaviour.

Conclusion

Navigation behaviour is one of the most essential interactions between humans and their environment. In robotic and virtual human animation, this behaviour relies on the ability to plans a path from an initial location to a goal. However, as virtual environments generally consist in raw geometries, an analysis of the structure of these geometries is required prior to the path panning process. Many environment representation methods were proposed, representing the free space as potential fields, roadmaps or cell decompositions. Among these methods, exact cell decompositions such as constrained Delaunay triangulations offer good properties for agent path planning. Indeed, these methods exactly represent the free space, and the complexity of the computed representation is linearly dependant on the complexity of the environment. However, in big environments featuring complex geometries, the number of generated cells can be huge, leading to high computation times. Solutions were proposed to extract more condensed cells representations by merging groups of cells into more abstract zones. Using these abstracted graphs as a base for the path planning greatly improves its performances [Lamarche and Donikian, 2004]. The concept of hierarchical environments provides multiple levels of such abstractions. By using the semantic information provided by informed environments, it is possible to extract an abstraction of the environment structure closer to the human representation. Using such an abstraction also enables the use of reasoning processes inspired by humans' behaviours. These reasoning processes provide the agent with more credible navigation behaviours[Mekni, 2010].

The validation of urban environments require the generation of high-quality paths that closely match humans' decisions. Due to their good properties in terms of abstraction and expressiveness, informed hierarchical representations are suitable for activity scheduling and path planning purpose. However, none of the proposed abstraction method focus on extracting semantically meaningful zones inspired by urbanism or architecture concepts such as streets, crossroads, sidewalks, floor and rooms. Yet, we believe that such a representation would allow the generation of more realistic paths, adapted to the validation of urban environments.

Behavioural animation 2

In the previous chapter, we discussed how essential is the simulation of virtual humans' navigation behaviours. However, navigation is usually not a goal on its own: it serves the purpose of reaching a location where some tasks should be performed. It means that the behaviour of people mainly depends on the numerous decisions they take while performing their activity. These decisions are taken at different levels, from the simple reaction to a perceived event to the complex scheduling of a daily activity. The generation of credible agents' behaviours in virtual cities relies on the simulation of these decision making processes. In microscopic simulation approaches, this process is simulated by embedding each agent with an individual decisional model managing different levels of interactions between the agent and its surroundings [Shao and Terzopoulos, 2005a]. The *behavioural animation* field of study focuses on developing such models. Among these models, we consider two main categories: reactive models and goal-oriented models. Reactive models tend to select actions in direct reaction to perceived stimuli. Goal oriented models, on the contrary, rely on more abstract representations of the environment to select sequences of actions fulfilling a long term goal. Such decisions rely on a huge number of parameters, from the topology and nature of the environment to the temporal and spatial constraints associated with the agent's intended activity. The behaviour of the agent can also be altered by its personal characteristics and preferences, as well as by its interaction with other agents in the simulation.

In this chapter, we give an overview of different methods developed in behavioural animation. First, we present the Perception-Decision-Action loop. Then, we study the properties of some of the main decisional models proposed in the literature. Finally, we study different kinds of parameters that can alter the decisions of an agent.

1 Perception-Decision-Action loop

All living organisms interact with their environment. The complexity of this interaction is extremely variable, going from the capability of moving in the environment possessed my most organisms to the complex social interactions between individuals characteristic of the most evolved species. Yet, regardless of the level of complexity of these interactions, they rely on common functions. First, all living organisms possess some senses enabling them to perceive their environment. These senses can be simple, like the sensation of pain or as complex as the sense of sight. Second, all organisms possess effectors enabling them to perform actions modifying the state of their environment or their own state. The activation of these effectors is controlled by the organism's decisions making. Most virtual humans' behavioural models rely on a similar model. A representation of this behavioural process as a perception-decision-action loop, illustrated in figure 2.1 was proposed [Mallot, 1997]. The components of this representation are defined as follow:

- The **perception** uses the organism captors (eyes, ears...) to extract raw information on the state of the environment. These captors are restricted to the capture of a given type of data, and to a limited acquisition zone, usually the surroundings of the organism. Therefore, the organism only possesses a limited perception of his environment.

- The **action** uses the organism effectors (limbs, voice...) to modify the state of the world. This modification may concern elements of the environment or the agent's own state in this environment.

- The **decision** process links the perceived information to the actions performed by the agent. It sorts and analyses the information provided by the captors. Then, it selects suitable actions to perform and activate the effectors in order to perform these actions.

- The **homoeostasis** is the internal regulation process of the organism. The body constantly adapt its biological parameters (heartbeat, internal heat...) to the variations of the environment. It is not considered as a part of the behaviour.

- The **acquisition** is the behaviour of enhancing the perception in order to extract more accurate information on the environment. For example, focussing on an object or trying to hear a distant noise.

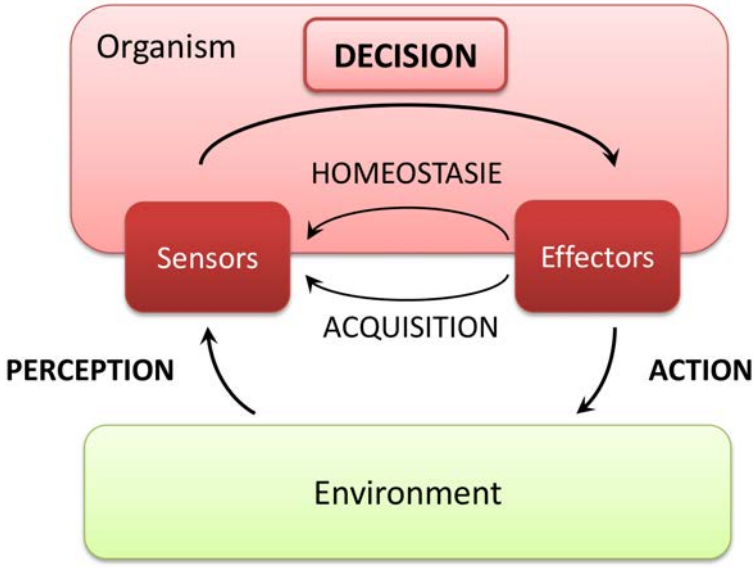


Figure 2.1 – Perception-decision-action loop

This model has the good property of being able to represent any kind of interaction between an organism and the environment. However, all interactions do not have same level of complexity. Indeed, the decisional processes of complex organisms differ by the abstraction level of the manipulated knowledge and by the time extent of the decision. In his work, Allen Newell [Newell, 1994] proposed a hierarchical decomposition of the human decision process into four levels of abstraction. Higher levels of decisions are characterized by the use of more abstract knowledge, longer decision times and better estimation of the long-term impact of selected actions. The four decisional levels identified by Newell are:

- The **biological level** relates to the functional units formed by neural networks. These highly parallelised processing units represent functions arbitrarily linking an output message to an input stimulus. These units have an approximate processing time of ten milliseconds and are considered as the atomic unit of decision. The biomechanical process of walk or the sense of balance are handled at this level, for example.

- The **reactive level** regroups pre-recorded simple actions. These actions are performed as reflex reactions to external stimuli, with a processing time inferior to one second. An example of reactive process is the avoidance of others pedestrians during navigation.
- The **cognitive level** relates to deliberate actions, with a processing time of few seconds. It relies on an abstract representation of the environment. The decision is taken consciously, but do not involve complex reasoning on the impact of the action or on its place in a larger plan. The decision to cross a road in order to avoid an obstacle belongs to this category, for example. Indeed, it requires an abstract representation of the street and implies a short-time analysis of the situation, but does not require a complex reasoning on the long-term impact of the decision.
- The **rational level** considers complex representations of the world to build plans aiming at the completion of a final goal. The reasoning, which can last from few second to minutes or hours, takes long-term effects of action into account, and considers the memories of previous experiences. Planning a path in a city is a decisional process carried out at the rational level, for example. Indeed, it requires an abstract representation of the environment as a network of streets and requires taking into consideration the long-term impact of decisions.
- The **social level** deals with long term interactions and relations between individuals.

When simulating a human behaviour, it is necessary to provide the decision process with a suitable representation of the world. For example, an agent's locomotion process does not require reasoning on a complex representation of the environment. On the opposite, an activity scheduling process will need an abstracted representation of the environment as well as look-ahead capabilities to generate credible results.

2 Decisional models

Human activity consists of a succession of actions. As we presented in the previous section, these actions belong to different levels of decision making. Decisional models offer automatized decision making processes. Each of these models, depending on its characteristics, is suitable for the simulation of a different human decision level. Decisional models rely on descriptions of the agent's possible actions and on information concerning the state of the world. Two main categories of models can be distinguished: reactive models, and goal-oriented models. Reactive models select the best next action to perform, in accordance with their immediate perception of the environment. They are suitable for the simulation of the reactive and cognitive levels of decision making. Goal-oriented models aim at finding a suitable sequence of actions leading to the fulfilment of a goal. This goal can be expressed either as a situation, i.e. as a state of the world to reach or as an activity, i.e. a set of tasks to perform. Goal-oriented decisional models are more suitable for the simulation of the cognitive and rational levels of human decision. In this section, we present the main categories of reactive models and goal oriented models, with either situations or activities as goals.

2.1 Reactive models

Reactive models aim at selecting an immediate action in reaction to a perceived stimulus. They consider only the short-term impact of decisions, adapting the agent's behaviour to the present state of the world. It makes these methods especially suitable for the simulation of reactive decision making and of low-level cognitive decision making. In this section, we describe

three main reactive approaches: stimuli-responses models, rule-based models and finite state machines.

2.1.1 Stimuli-responses models

Stimuli-responses models are the most direct representation of the perception-decision-action loop applied to the reactive level of human decision[Brooks, 1995]. They are directly inspired by biological neural networks that control animal reflexes. The decisional system is composed of a multi-layered network of weighted node called *artificial neurones*. It generates an output signal for any input signal. This output signal directly controls the effectors of the agent. The behaviour of the agent is thus directly correlated to the perceived information through a mathematical function which complexity is dependent on the number of layers in the network and on the number of nodes in each layer. In order to generate an adequate reaction to a given input, stimuli-response systems must be trained over large sets of recorded data.

Multiple adaptations of stimuli-response systems to agent animation were proposed. The Neuroanimator system [Grzeszczuk et al., 1998] uses a hierarchy of artificial neural networks to simulate agents' behaviours. Every sub-network is associated with a precise action (moving an arm...) while upper-level networks control the global behaviour. This system enables to independently train specific actions. In [Granieri et al., 1995], nodes in the network are specialised in three categories of higher level nodes. *Perceptual nodes* represents the agent perception by generating a more abstract output than the one edited by the raw sensors. They also limit the knowledge of the system to what the agent can physically perceive. *Motor nodes* control the effectors of the agent. Some of them directly generate a simple action. Others manage more complex actions (walking...) using separate asynchronous execution modules. *Control nodes* link perceptual nodes to motor nodes. A higher level of control of the agent's behaviour is achieved through this specialisation of the neural network's nodes.

Stimuli-responses decisional systems enable a quick reaction to the perceived information. However, they are not adapted to simulate more abstract behaviours. Furthermore, every independent behaviour need to be trained over a large example set. Another weakness of the method is that neural networks work as "black box" systems: the values of the weight associated with nodes is not predictable and does not able logical deduction on the nature of the problem. This means that it is impossible to extend the system without staring over the learning process. Yet, These method are efficient for automatic learning of specific simple behaviours which description would not be easy, for example the behaviour of insect agents [Beer and Gallagher, 1992].

2.1.2 Rules-based models

Rule-based models describe the agent behaviour as a set of specific rules depicting how to react to given situations. Compared to stimuli-response decisional models that rely on learned reactions without any notion of the meaning of produced behaviours, rule-based models explicitly describe the desired reaction to a stimulus. For this reason, they offer a better control on the generated behaviour. Furthermore, the system is easy to extend, as new rules can be seamlessly added to the existing set.

In his work on flocks, Reynolds simulate the behaviour of a flock of birds through a rule-based method [Reynolds, 1987]. Each entity in the flock is embedded with three simple rules: keeping distance from neighbours to avoid collisions, following neighbours to keep the flock's coherence and adjusting its own speed to the neighbours' one. A credible global flock emerge from these rules, but none of the entity really demonstrates an autonomous behaviour.

Rule-based models have been used as parts of various agent behaviour systems. For example, *if - then* rules were used as atomic actions in hierarchical descriptions of behaviours

[Laird et al., 1987]. They were also used to choose between branches in decision trees, eventually selecting the most suitable behaviour [Coderre, 1987]. For example, in [Blumberg, 1996], this method was used to handle the behaviour of a dog agent able to react to the user's gestures (see figure 2.2).

Figure 2.2 – A dog agent able to react to the user's gestures using rules to select between the branches of a decision tree [Blumberg, 1996].

Such models are intuitive to design and new rules can easily be added to generate new behaviours. However, these models require that the set of expressed rules is sufficient to cover all possible situations. Furthermore, they suffer from a difficulty to treat conflicting rules. Indeed, if a situation arises in which incompatible rules could be applied, choosing one of these rules can prove complex.

2.1.3 Finite state machines

Most of behaviours can be modelled as sequences of actions being performed, with variations depending on the situation. Finite state machines represent the agent's possible behaviours as an oriented graph composed of independent actions. Transitions between these actions are associated with conditions that have to be met for the next task to be performed. Compared to stimuli-response and rule-based models, an interesting property of finite state machines is that decisions do not only depend on the state of the world, but also on the previous actions performed by the agent. Figure 2.3 shows a finite state machine used in [Moreau, 1998] to control the reaction of a driver agents to traffic lights.

However, elaborate behaviours may require the design of extremely complex finite state machines. Multiple variations of the model were proposed to describe more elaborated behaviours while avoiding too high finite state machines complexities. The finite state machine stacks model is a transposition of the procedure call method to finite-state machines. It enables the alteration of the agent's behaviour depending on the state of the world by stacking finite state machines [Noser and Thalmann, 1998]. The state machine on the top of the stack is the one to be executed. Some of the executed actions can have as an effect to stack more machines that describe the realisation of this action at a lower level. Changes in the environment can also trigger stacking of new finite state machine, hence modifying the behaviour of the agent. Once these finite state machines have been executed, the previous behaviour can be resumed, maintaining the coherence of the behaviour over time. Parallel Transition Networks [Badler et al., 1995] use multiple parallel finite state machines to simulate more complex behaviours. Each of these finite state machines can either describe the resolution of a specific action, generate new finite state machines or reason on the activity to perform. In order to avoid incoherence in the global behaviour, parallel state machines communicate through messages and use semaphores for synchronisation. This way, different levels of decision can be carried on simultaneously. For example, the navigation process of the virtual human can be

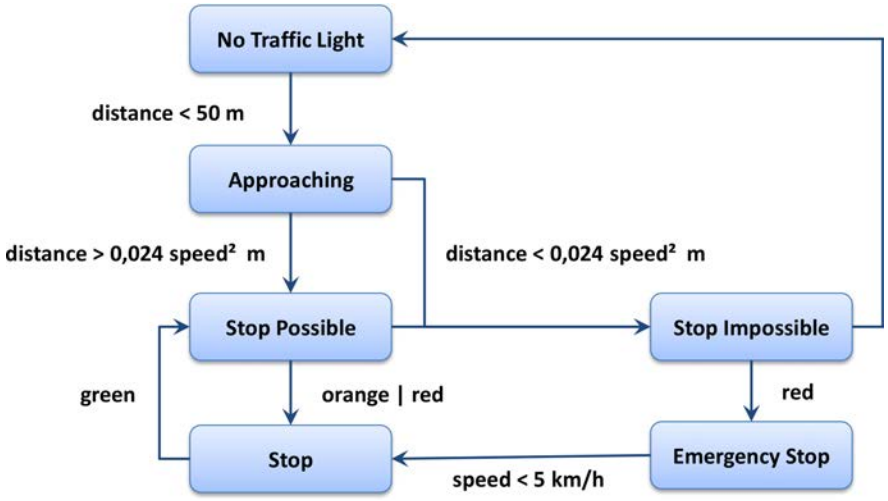


Figure 2.3 – Finite state machine controlling the reaction of a driver agents to traffic lights [Moreau, 1998].

managed by a finite state machine while higher-level decisions on the destination are dealt by another. Parallel Transition Networks enable the description of complex behaviour while keeping a reasonable representation complexity. This model was extended by the Hierarchical Concurrent State Machines model [Cremer et al., 1995] and by the Hierarchical Parallel Transitions Systems model [Donikian, 2001]. Both these models enrich parallel state machines with the notion of hierarchical state machines. Each state machine, representing a specific behaviour, can possess sub-state-machines that describe lower level aspects of this main behaviour. The main finite state machine combines the outputs of the sub-states-machines using concurrence rules. Such models has been shown to give realistic results in human behaviour simulation [Donikian, 2001]. This last system was extended in [Lamarche and Donikian, 2002] to manage the exclusive use of resources and synchronisation of hierarchical parallel state machines. It was demonstrated through the example of an agent that has to read, smoke and drink, these actions requiring conflicting use of the agent's hands and mouth. The system manages the exclusive use of these resources in order for the agent to be able to organise the completion of the desired actions.

Finite state machines provide an easy and intuitive way of managing the execution of sequences of actions. The impact of the world state on behaviour is easy to manage through conditional transitions. Complex behaviours can be represented thanks to parallel and hierarchical finite state machines. However, this decisional model require all possible sequences of actions to be explicitly described, which can lead to high complexities of descriptions.

Overall, reactive models enable the quick selection of the seemingly best action in reaction to a stimulus and depending on the current state of the world. However, these systems lack the look-ahead capabilities required to consider the long-term impact of the decision. This makes reactive models great tools for modelling the reactive and low-level cognitive decisional levels that do not need to reason over long period of time.

2.2 Goal oriented decisional models with situation-type goals

Goal oriented models aims at selecting a course of action that fulfils a long term goal. In the case of situation goals, the goal is expressed as a state of the world to reach. An action planner is used, aiming at the selection of actions that modify the current state of the world in order to

reach the desired state. Multiple sequences of actions may lead to the satisfaction of this goal, with more of less efficiency. A goal oriented relies on look-ahead capabilities in order to select a satisfying sequence of actions among the possible ones.

2.2.1 Situation Calculus

The *situation calculus* model offers a formalism to describe the state of the world and its possible alterations [McCarthy and Hayes, 1968]. It enables reasoning on the agent's actions and their effects on the state of the world. The situation calculus formalism relies on multiple concepts:

- Each **situation** describe a state of the world a given instant. This state is characterised by a set of properties.
- **Fluents** are functions used to interrogate some properties of the world that can vary over time. These fluents can be either facts (it is raining), numerical values (money in the agent's possession), or even relations between elements of the world (the agent is at home).
- **Causalities** describe how fluents can be combined to deduce information on the world that is not explicitly expressed in the situation. For example, if two fluents respectively express the facts that it is raining and that the agents is outdoor, then, the information that the agent is wet can be deduced as a causality.
- **Actions** are used to apply modifications to a situation. An action is defined by a pre-condition which has to be satisfied for the action to be performed, and effects, that describe the modifications applied to the situation. These modifications basically change the values of some fluents associated with a situation. Conditional effects can be also expressed. In such a case the effects of the action depends on the situation.
- **Strategies** are finite sequences of actions. They enable the description of more complex behaviours required to reach a situation. As actions, strategies can be summarised as a set of pre-conditions and effects.
- **Knowledge** expresses the idea that an agent is not omniscient and may ignore the value of some fluents. These values can only be determined using perceptive actions, for example looking at a sign to know the name of a street.
- The **Goal** of the agent is expressed as a set of propositions over fluents. If all these propositions are satisfied in a situation, this situation is a suitable solution to the action planning.

Through these concepts, sets of possible future situations can be computed as results of actions or strategies. A planner is used to select a suitable strategy leading to a situation satisfying the goal of the agent.

Several implementations of this model were developed for autonomous agent simulation such as GOLGOL (alGOL in LOGic)[Levesque et al., 1997] and CML (Cognitive Modelling Language) [Funge, 1998]. Both these planers offer a syntax to express situation calculus elements, as well as classical control structures (loops, conditions...). More specifically, CML offers ways to express uncertainty over numeric values and features non-deterministic action selection[Funge, 1998]. The Situation Calculus model is intuitive and expressive: complex worlds and actions can be described as a natural set of conditions and effects. Reasoning process can take advantage of this description to generate convincing behaviours demonstrating look-ahead capabilities. The notion of incomplete knowledge enhances the credibility of the simulation by requiring the agent to perceive his environment before making choices. It can

also induce exploration behaviours when the agent does not have sufficient information to carry out his activity. However, descriptions of the world using situation calculus formalisms are complex. The exploration of the huge number of possible situations leads to high computational costs. This language was used in [Funge et al., 1999] to simulate a struggle for territory between a T-rex and raptors (see figure 2.4). This demonstrated the model ability to simulate decision making involving look-ahead and long-term strategies. Yet the complexity of the search space grows exponentially with the number of possible actions, making this model unsuitable to the simulation of elaborate behaviours.

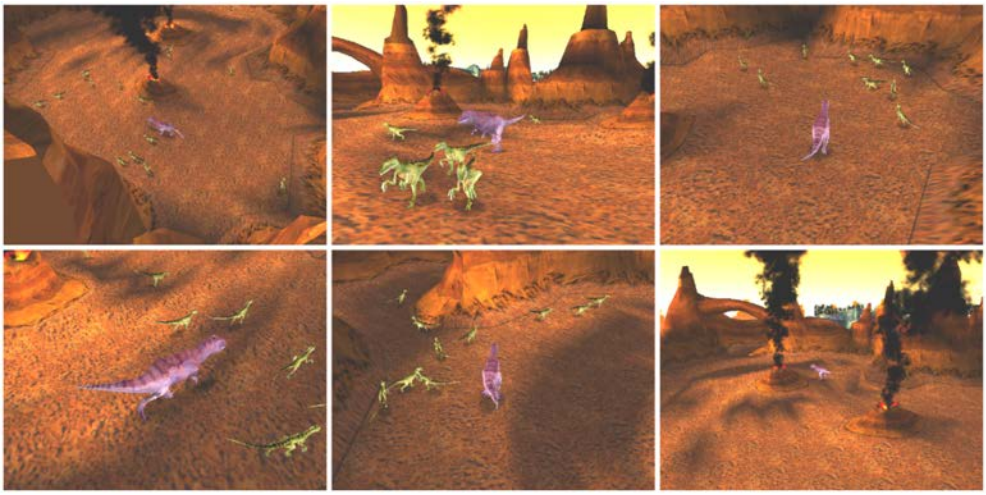


Figure 2.4 – Example application of the Situation Calculus, involving a struggle for territory domination between a T-rex and raptors.

2.2.2 STRIPS

The STRIPS (Stanford Research Institute Problem Solver) language offers a formalism to describe a subset of the situation calculus formalism [Fikes and Nilsson, 1972]. The state of the world is described as a set of boolean propositions (having money or not, being at home or not...). Operators, similar to the actions of the situation calculus formalism, are defined by a set of pre-conditions and effects. However, fluents values being exclusively boolean, the description of these operators is more simple. Indeed, conditions are limited to a constraint on the presence or absence of some propositions in the situation. Operators' effects are described through two lists: propositions added by the operator and propositions deleted by it. The planning is also simplified as it consists in selecting a sequence of operators that adds or removes the desired propositions from the situation to reach the goal. Even if the narrower formalism of STRIPS limits the planning computation cost, the exploration of all the possible situations remains complex. Several approaches have been proposed, aiming at limiting this complexity.

GRAPHPLAN is a planning algorithm method that relies on a planning graph [Blum and Furst, 1997]. The graph is composed of successive layers of operators and proposition nodes, beginning with the initial state of the world. Edges in this graph represent pre-conditions, add and delete relations between propositions and operators. Each action layer contains all the operators which pre-conditions are satisfied by the propositions in the previous layer. Exclusions between operators or propositions are added to the graph to deal with conflicts (for example, if an operator deletes a pre-condition of another operator). When a proposition layer satisfy all the goal conditions without inter-exclusions, a backtrack algorithm searches the graph for the sequence of operators that led to adding these propositions. This search

algorithm possesses good properties. First, it guaranties to finds a solution that optimises the number of layers required to reach the goal. Second, it is able to detect which operators can be executed in parallel. Finally, as it does not develop the entire planning tree but only a combined representation of each step, the computational cost and memory consumption of the planning is reduced. However, managing the inter-exclusions can prove complex and the whole backtracking process remains costly.

In order to further reduce the planning complexity of STRIP planners, methods relying on heuristics were proposed, such as HSP (Heuristic Search Planner) [Bonet and Geffner, 2000] or FF (Fast-Forward planner) [Hoffmann and Nebel, 2001]. Both these methods rely on the computation a heuristic, estimating the number of operators that must be performed in order to reach the goal. This heuristic is used to guide the planning, thus decreasing the number of explored situations. In HPS, this heuristic is computed by taking only the *add* effects of operators into account and ignoring the *delete* effects. Without *delete* effects, inter-exclusions can also be ignored. This way, a minimum number of operators required to reach any of the goal propositions is computed. By summing these estimations for all the goal propositions, the distance between the goal and the current situation is estimated. FF relies on the graph produced by GRAPHPLAN to estimate the minimum number of operations required to reach the goal. This estimation is closer to the real number of required actions, enabling the planning to converge much quickly toward the solution. However, both these methods tend to over-estimate the real number of actions required to reach the goal situation. Therefore, these methods are not guaranteed to select an optimal solution in terms of number of actions to perform. Furthermore, when computing a plan, STRIPS planners considers that the agent is the only entity able to modify the state of the world. In dynamic environment or when multiple agents are performing actions concurrently, this can lead to an unexpected fluent modification invalidating the computed plan.

2.3 Goal oriented decisional model with activity-type goals

Not all activities have the modification of the environment as an objective. For example, actions as "having a walk" do not aim at modifying any property of the world: performing the action is a goal in itself. Goal oriented action planning methods that use an activity to perform as a goal take this fact into account. They still aim at selecting a suitable sequence of action to reach a goal situation, but this goal is expressed as a set of tasks to perform instead of a state of the world to reach.

2.3.1 Beliefs-Desires-Intentions model

The BDI (Beliefs-Desires-Intention) action selection model aims at simulating the hierarchical nature of the human decision making [Bratman, 1987]. It is inspired by studies on the cognitive process supporting decision, which relies on three main concepts: Beliefs, Desires and Intentions.

- **Beliefs** represent the agent's knowledge on the state of the world. This knowledge can be incomplete or even wrong, due to the limited perception capabilities of the agent and to unexpected modifications of the state of the world.
- **Desires** are the agent's objectives. They express a goal to reach in terms of activities to perform. For each desire, the agent possesses one or multiple plans describing ways of satisfying this desire, with their own pre-conditions.
- **Intentions** are the plans the agent can use to satisfy his desires.

When an agent wishes to satisfy a desire, a suitable plan is selected among those possessed by the agent. This plan must have its pre-condition satisfied to be added as the agent current

intention. Once a plan is selected as the new intention of the agent, it is executed step by step, each step triggering either:

- the addition of a new desire for the agent to satisfy;
- an update of the agent's knowledge, acting as a perception action;
- the execution of an atomic action.

The addition of new desires as part of a plan generates a hierarchical decomposition of the plan into sub-plans. The addition of each new desire triggers the selection of a plan, taking the present beliefs of the agent into account. This way, the BDI agent easily adapts his behaviour to the evolution of his knowledge. This reflects the human decision process, as people first plan their global activity and delay the more local decisions until the required information is available. However, this makes BDI models subject to planning failures. Indeed, the plans being gradually decomposed, such failures can remains undetected until a sub-desire is added for which no plan has its pre-conditions satisfied. The lack of formalism in the desire description can also cause unexpected issues, as incompatibilities between desires remains undetected [Rao et al., 1995]. These properties make BDI models great for simulating decisions belonging to the high-level cognitive decisional process, aiming at the realisation of a goal without considering the precise long-term impact of the taken decisions. It makes this model especially adapted to goal-oriented action planning in dynamic environments.

2.3.2 Hierarchical Task Networks

The HTN (Hierarchical Task Network) method uses a hierarchical representation of the agent activity [Erol et al., 1994]. The world description is similar to the one used in the STRIPS formalism: as set of propositions. HTN operators are also similar to STRIPS operators, defined with pre-conditions and effects on the world. However, instead of searching to combine these operations to reach a goal situation, HTN aims at selecting a valid decomposition of a goal task that has to be performed. The structure of HTN graphs, illustrated in figure 2.5, relies on the concepts of tasks and methods:

- **Tasks** are goals to reach expressed as an activity that must be performed. Two types of tasks can be described. *Primitive tasks* express the desire to perform a simple action, and are satisfied with the execution of the corresponding operator. *Compound tasks* are more complex tasks that decompose into multiple sub-tasks through the use of methods.
- **Methods** describe a way to perform a task as a sequence of sub-tasks. Like operators, methods possess pre-conditions. The interest of HTN planning resides in the fact that several methods can be described to perform a given task, with possibly different pre-conditions. For example a "buy groceries" task can either be performed by going to a supermarket or to small shops. In the second case, the agent may need to carry cash as a pre-condition.

The use of methods represents the knowledge of the agent on how to perform tasks. They create a hierarchy of tasks that can be explored to select a suitable plan leading to the satisfaction of the agent's goal. As the goal is expressed as a task to perform and not a set of conditions to satisfy, the HTN formalism enables the expression of goals that do not necessarily aim at modifying the state of the world (for example, having a walk can be a goal in itself, regardless of the effects of this action). However, if no valid plan was described enabling to reach the goal from the current situation, the model will not be able to infer a new plan. Overall, the HTN formalism was proven to offer more expressiveness than the STRIPS formalism, but do not provide the same flexibility in the selection of plans [Charles et al., 2003].

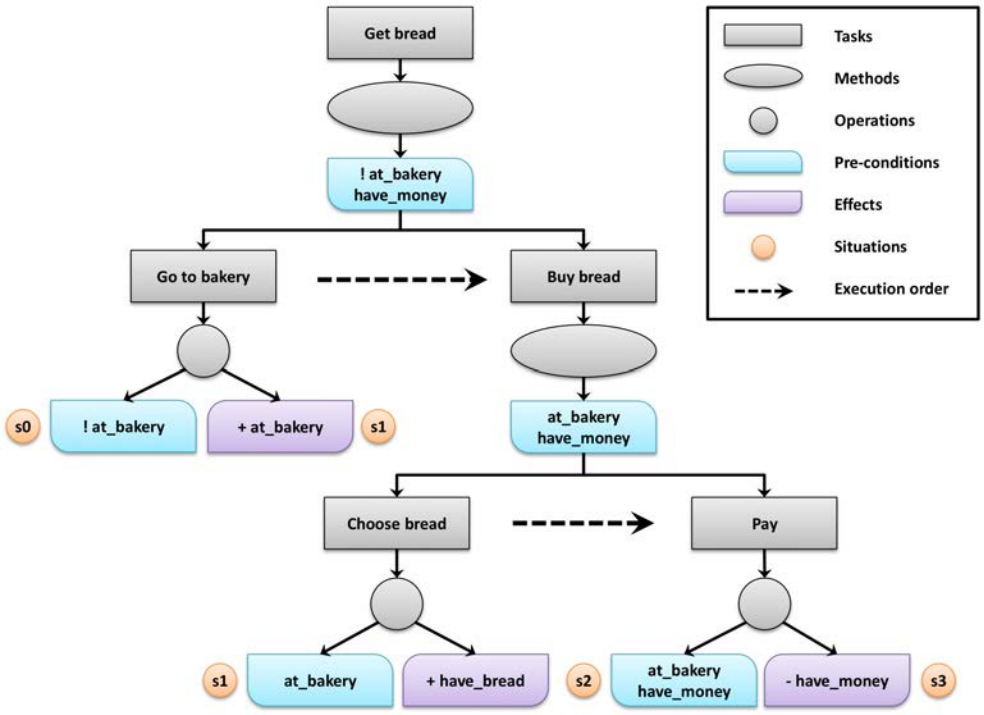


Figure 2.5 – Simple example of HTN graph.

The HTN model suffers from the same adaptability issue as STRIPS planners. Indeed, plans are computed considering the agent only. If a dynamic event or another agent modifies a fluent, it may invalidate the computed plan, leading to a plan execution failure. For the same reason, it is also incapable of opportunist behaviours, as it does not reconsider the computed plan when discovering new elements, possibly missing a more optimal way of reaching the goal. For example, an agent travelling to a far-away ATM and discovering a yet-unknown ATM on his way will not consider using this one instead and will resume travelling to the distant one. In his work, Rannou propose a way to tackle this issue by combining the HTN planning to failure anticipation and opportunism capabilities [Rannou et al., 2012]. The state of the world is monitored, checking for any fluent modification that could impact the computed plan execution. If a detected fluent modification invalids the computed plan or enables it to be enhanced, the plan is adapted, altering the agent's course of action.

summary

Decisional models aim at selecting actions an agent should perform, either in reaction to perceived stimulus or aiming at reaching a long-term goal. Depending on their characteristics, these models are adapted to different levels of decision making. For example, rule-based models are efficient at simulating reactive decisions while BDI models simulate decisions taken on the cognitive level. Due to their high abstraction and their look-ahead capabilities, goal oriented models such as HTNs are able to simulate the rational behaviour of human beings. Decisions are taken involving long-term planning and knowledge management. These goal oriented decisional models either build sequences of tasks from individual actions descriptions (situation calculus, STRIPS ...) or rely on described plans (HTN, BDI ...). Relying on such plans enables a better authorial control of the simulation, but causes a lack of flexibility when con-

fronted to unexpected situations. Furthermore, these models, BDI models excepted, suffer from the "closed world" issue: they only consider the agents action when computing a plan, leading to possible plan execution failures in dynamic environments. Methods were proposed to repair invalidated plans in dynamic environments [Ayan et al., 2007] and in the case of cooperative activities [Gateau et al., 2013]. Yet, BDI models, by developing the plan as the agent executes it, better handles dynamic environments, at the cost of its look-ahead capabilities.

3 Influence of external factors on the decision making

Most of decisional models we presented in the previous section rely on high-level observations on the state of the world to select suitable actions to perform. However, people decisions do not only rely on such observations. Many external factors such as the structure and nature of the environment, the time, personal preferences or interactions with other agents may also influence the decision making process. The presented decisional models do not properly handle all these factors. In this section, we study the impact of these factors on the decision making process and we present models aiming at taking some of them into account.

3.1 Impact of the spatial and temporal constraints

People's behaviours are highly dependent on the space and time constraints associated with their activity. Indeed, a lot of tasks must be performed in specific locations (for example, buying bread requires to be in a bakery or a grocery store) or during specific time intervals (Shopping must be done during open hours). Some tasks, such as appointments, even require being performed at a given time at the right location. Such constrains have a huge impact on people decisions. For example, if a person's activity specify he must shop and go to an appointment, his decision on the order these tasks must be performed mainly depends on the time and on the distances between its current position, shops locations and the appointment's location. Yet, describing these factors through the use of fluents would lead to high dimensional search spaces, inducing complexity issues. Methods were proposed to take some of these factors into account.

The main spatial constraints impacting people's behaviour are the repartition of locations where tasks must be performed and the distances between these locations. As an example, if someone has to visit a bakery and a butcher on his way home, he will probably choose to visit these shops in the order that minimises his global effort. In the case multiple shops options are available, the ones to visit are also selected in order to reduce this effort. In the field of combinatorial optimization, the travelling salesman problem focus on the spatial aspect of decision making [Kruskal, 1956]. This problem emphasizes on the complexity of determining an optimized path that visits a set of positions in a graph. The path optimisation mainly relies on the distance travelled between these positions. However, this problem focuses only on the visited positions and not on the order in which they are visited. Therefore, it is not suitable for the selection of the best locations to visit in order to optimize the realisation of a sequence of tasks.

Temporal constraints also play an important role in the decisional process. Indeed, most tasks are associated with time intervals such as opening hours, work hours, appointment times... As these tasks should not be realised out of these intervals, decisions must take these time constraints into account. In their work, Do et. al. describe the challenges posed by the integration of time constraints in planning problems [Do and Kambhampati, 2003]:

- The search space tends to get significantly larger due to the addition of the time dimension.

- The planner has to deal with many additional types of constraints involving time.
- The objective of temporal planners cannot be expressed as finding least cost path. Indeed, the user may be interested in optimizing the duration of the plan instead, or to find a good compromise of both metrics. This makes temporal planner multi-objective problems.

They introduce Sapa, a domain-independent planner using a heuristic mixing the cost and duration of actions. This method is able to deal with durative actions, metric resource constraints, and deadline goals. It handles the multi-objective nature of temporal planning. However, it is not able to consider all kind of temporal constraints, like tasks limited to given time intervals.

Halsey et. al. introduced the CRICKEY temporal planner in [Halsey et al., 2004]. This planner deals with the problems in which the spatial and temporal planning problems are partially coupled. These two problems are handled separately, but the model is able to detect the parts where they interact. In these parts, the spatial and temporal planning are combined and performed simultaneously. This way, most of the complexity of temporal planning can be avoided, improving performances. This planner was shown to be quicker than the Sapa planner, but produces lesser quality plans. The problem of plan quality was discussed in [Cushing et al., 2007]. They note that most models handle the temporal planning complexity by restricting the start time of actions to a small subset of possibilities, leading to non-completeness issues. They propose a state-space temporal planner that improves the heuristic search in order to achieve complete and high-performance temporal planning.

In the field of robotics, mixing navigation and mission fulfilment is fundamental. Strong spatial and temporal constraints must be taken into account. For example, a robot has to reach a recharge station before its batteries run up or to visit a set of given locations. Planning paths for these robots requires taking into account the strong interaction that exist between space, time and activities. In their work, Smith et. al. handle this problem by producing an automaton describing the problem. This automaton is defined as the product of a roadmap representing the environment and a linear Temporal Logic Graph specifying the mission parameters. Running this automaton automatically generates an optimal robot paths satisfying the high level mission specifications [Smith et al., 2011]. This approach attests the importance of taking both the environment topology and time constraints into account in decision making problems. However, they focus on shortest path computation, while humans take more complex parameters into account, such as the nature of the visited zones, as we saw on the informed environments section.

Scheduling virtual humans' activities requires considering the tight interaction between space, time and their intended activities. However, combining the temporal and spatial dimensions of a problem tends to lead to huge search spaces. Planning methods used to find a solution in such search spaces must handle this complexity.

3.2 Impact of cooperation with other agents

People's activities often include tasks that should be performed in cooperation with multiple individuals. Meetings and appointments are good examples of such tasks. Some of these tasks, appointments for example, are easy to handle as they are usually constrained in advance to a given location and time. But some other cooperative tasks, such as exchanging an object between two persons, may not require to be performed at a specific time or location. In that case, the task planning involves the selection of a common location and time compatible with both agents' intended activities. Solving this problem implies exploring a huge search space composed of the combined situations and locations of both agents, with the addition of a temporal dimension. Several approaches were proposed to tackle the complexity intrinsic to such a scheduling problem.

3.2.1 Approaches in automated personal agendas

The topic of cooperative activity scheduling has been widely studied in the domain of automated personal agendas. Initial approaches mainly focused on temporal constraints, not considering the traveling distances between locations, for example [Mattern and Sturm, 1989, Jennings and Jackson, 1995, Garrido and Sycara, 1996]. In order to find meeting times and locations compatible with several agendas, these approaches rely on distributed techniques: each agenda possess its own scheduling agent, that communicate with other agents in order to find meeting times suitable to the higher number of persons. In [Macho et al., 2000], a distributed meeting planner was proposed taking transportation schedules into account. This way, the system is able to consider the persons' traveling times when selecting meetings locations and times. However, only high level transportation information (flight schedules and durations) are taken into account. Moreover, meetings are limited to a list of possible times and locations. Methods taking personal habits and preferences into account, have been proposed in [Bowring et al., 2005]. They are designed to select the best day, time and place to hold the meeting [Zunino and Campo, 2009]. However, no general solution handling cooperative and individual activity scheduling has been proposed yet.

3.2.2 Approaches in robotics

In the domain of robotics, many approaches focus on the problem of planning paths for multiple objects while avoiding inter-collisions [Erdmann and Lozano-Perez, 1987]. This problem is complementary to the scheduling of cooperative tasks. The multiple moving objects problem is described as very challenging [Li et al., 2005]. To solve this problem, some approaches use decoupled methods [Li et al., 2005]. Each robot plans a path in the environment without taking the other agents into account. Then, the speed along the path is tuned to avoid inter-collisions. Some authors argue that decoupled methods lead to high failure rates [Saha and Isto, 2006]. To avoid these failures, they propose to use centralized approaches. To handle the problem of handling robots cooperative missions, Williams et. al. propose the use of Temporal Plans Networks to drive a Rapidly exploring Random Tree path planning [Williams et al., 2001]. These Temporal Plans Networks are similar to those used by temporal planners, but extended to symbolic constraints and decisions. This method enables multiple automated vehicles to achieve elaborate cooperative missions within uncertain environments. They demonstrated this ability through an example featuring four Martian rover modules exploring larges areas in cooperation. However, these methods do not allow the specification of spatial or temporal constraints over tasks. In [Bhattacharya et al., 2010], the problem of multiple robots path planning with constraints on the distance from each other, coupled with multiple task realizations, is tackled. With this approach, distance constraints can be used to model cooperation between robots. The problem is solved by iterating over each robot planning using other robots plans as constraints, until an acceptable solution is found. However, in this model, task times are known in advance and each task is precisely located in the environment. Moreover, the execution order of multiple tasks cannot be constrained. Finally, the user cannot express time constraints on task realization.

3.2.3 Approaches for virtual agents

The problem of cooperative activities has been discussed in the virtual agent community. A model was proposed to mix the individual activities with activities common to multiple agents [Pelechano et al., 2008]. It relies on including both groups and individual goals into a comprehensive computational model. However, this model focuses more on the choice of destinations and paths than on the scheduling of tasks. A method able to schedule more complex agent cooperative tasks was proposed in [Kapadia et al., 2011]. In this approach, groups of agents

are identified in order to create composite search domains which aim at reducing the global search complexity. In each composite domain, a planner search for a solution in the product of each agent personal search domain. This avoids the complexity of modelling communication methods between agents. However, as individual and cooperative tasks are included in this composite search space, high complexity issues arise in case of complex scenarios.

3.2.4 Approaches in virtual storytelling

In the field of virtual storytelling, Porteous et. al. [Porteous et al., 2011] outline the importance of time for the coordination of activities between multiple agents involved in a narrative. They decompose the narrative generation into a sequence of sub problems, each of them being solved using the CRIKEY temporal planner [Coles et al., 2009]. This planner plans concurrent actions, taking realization duration into account. In [Porteous et al., 2011], this planner is used to solve problems in which meeting locations are known in advance with regards to the narrative nature of the problem. Moreover, due to the problem decomposition into sub problems, timings are relative but do not include hard temporal constraints. In [Shoulson et al., 2013], Shoulson et. al. offer a way to design narratives involving multi-agent interactions. Rather than describing the search space as a combination of the agents' activities, they describe an event space in which each collaborative task is an event, with its own preconditions and effects. A solution is first planned in the event space. Then, each agent plans its own activity in accordance with the planned events. This way, most of the complexity of the multi-agent search is avoided, enabling the planning of complex activities involving high number of agents. They demonstrated the efficiency of their method through an example scenario (see figure 2.6). This scenario features three agents escaping from a prison guarded by multiple guards and security systems. The agents plans several synchronised actions such as two agents hiding while a third one lure the guards away (see figure 2.6.a) or an agent pressing a button at the right time in order to trap the guards (see figure 2.6.d). This enables the precise authoring of complex scenarios featuring multiple cooperative activities involving multiple agents while avoiding the high complexity of the problem. However, this method requires the description of high-level events and is thus not able to come up with strategies that were not explicitly described.

Figure 2.6 – Example scenario demonstrating the effectiveness of event-centric approaches for real-time narratives [Shoulson et al., 2013].

3.3 Impact of personal characteristics and preferences

A great variability exists between people's behaviours. A part of this variability is due to the differences between individuals' activities, agendas and constraints. However, two persons sharing the same intended activity and constraints may act differently: an important part of the variability is due to personal characteristics and preferences. Even in a crowd in which all individuals have the same global goal and constraints (for example, people entering a stadium),

a variety of individual behaviours is observable. Some proposed approaches aim at producing such a variability of behaviours by embedding agents with personal characteristics.

The variations in people behaviours is determined by their personality traits, including a number of interplaying factors like emotions, moods, personalities, cultures, roles, status, needs, perceptions, goals, relationships... [Li and Allbeck, 2011]. However, relying on personality traits in agents' decisional process requires a formal way of describing these traits. The OCEAN models offers such a description formalism [Wiggins, 1996]. It describes an individual personality as a mix of five main personality traits: openness, conscientiousness, extroversion, agreeableness, neuroticism. The agent's personality can be expressed as a set of values describing how much the agent is subject to each personality trait [Durupinar et al., 2008]. For example, the leadership of an individual can be defined as a high value of consciousness, medium values of extroversion and neuroticism, and a low value of agreeableness. In their work, Li et. al. use the OCEAN formalism to control the selection of social roles for the agent[Li and Allbeck, 2011]. A social role defines the agent current goals and behaviours. An agent may switch from a role to another depending on its desires and present needs, as well as on its mental status and personality traits. The OCEAN model was also used to handle conversational interactions between agents, impacting the activities of these agents [Egges et al., 2004]. For example, a situation is presented, in which an agent helps the other or not, depending on the personality traits of this agent. The location preference of individuals was also used to improve the credibility of agents behaviours [Li et al., 2012]. The agent is able to choose a location in his environment depending on the local densities and on his characteristics (personality traits, needs and interests). Personality traits being specific to each individual, variability in the behaviour of agents automatically emerges from agents descriptions. This improves the credibility of the generated crowds.

summary

People's behaviours rely on a huge number of different parameters. The order in which people perform tasks, where and when they perform these tasks is highly dependent of their temporal constraints (work hours, appointments...) and spatial constraints (typology of the environment, workplace and home locations...). The influence of these temporal and spatial constraints is tightly related: they cannot be considered separately in the scheduling process. When handling activities involving cooperation between multiple agents, the personal intended activities and constraints of all the involved agents must be taken into account. This leads to huge search space, requiring the use of algorithms that drastically limit the scheduling complexity. People's behaviours also depend on their personal characteristics and preferences. In order to generate a credible variability in agent's behaviours, scheduling processes must take these parameters into account. However, even if many models were proposed to take some of these parameters into account, none completely considers the tight relationship that exists between space, time, agents' activities and their personal characteristic and preferences.

Conclusion

Decisional processes were proposed to select suitable actions an agent should perform in reaction to a stimulus or in order to reach a goal. Depending on the environment's level of abstraction they rely on and their look-ahead capabilities, these processes are more adapted to the simulation of given levels of human decision. For example, rule-based models enable the simulation of reactive behaviours while BDI models are good at simulating decisions taken at the cognitive level and HTN focus on the rational level of decision. The look-ahead capabilities offered by

goal-oriented decisional models are necessary to the simulation of complex behaviours in which the long-term impact of actions must be considered. Many parameters impact people's decisions. Indeed, the order of actions they choose to perform and the choice of locations and times to perform these actions is highly dependent on the topology of the environment, as well as on the person's temporal constraints. The influence of these spatial and temporal constraints is tightly related in the decisional process. People' decisions are also impacted by their personal characteristics and preferences. Several global agents' behavioural animation systems were proposed, handling the different levels of decision of the agents. In [Tecchia et al., 2001], a whole city block is populated with thousands of agents through the use of four layers of description of the environment, each handling a different aspect of agents' behaviour. In the field of geosimulation, the MAGS project [Moulin et al., 2003] and extensions [Moulin and Larochelle, 2010] embed agents with a set of goals to satisfy to model crowds activity. The whole simulation can be driven by a user specified scenario [Kapadia et al., 2011]. Paris et. al. use the concept of affordance [Gibson, 1979] in order to generate activity driven navigation [Paris et al., 2009]. In [Shoulson et al., 2013], Shoulson et. al. propose an event-centric framework handling complex narratives involving the realisation of cooperative tasks. We believe that, in order to be able to generate agent's behaviours statistically consistent with real human behaviours, it is required to take into account the spatial and temporal constraints associated with the agent's activity, as well as the influence of personal characteristics. We also believe that special models must be proposed to handle agents' cooperative activities. However, none of the proposed models completely handle the tight interaction between space, time, activities and agent's personal characteristics. The literature also only offers partial solutions to address cooperative activities scheduling issues that fit the needs of the domain.

Overview

3

Most crowd simulation models usually focus on generating visually credible crowds. Many of these systems rely either on macroscopic rules, on precomputed animation patches or on captured data. However, some applications such as the analysis and validation of urban planning arrangements requires the generation of crowds showing realistic behaviours. Focussing on the generation of more credible individual agents' behaviours is required to obtain such realistic crowd behaviours. Indeed, the global activity in a city is the sum of the behaviours of the individuals in this city. Most of these behaviours consist of people performing their daily activities in different locations of the city and navigating between these locations. Thus, we believe that embedding each agent with a model handling the rational choices of the agent in term of activity scheduling and path planning enables the generation of more credible individual agent behaviour. Thus, more realistic crowd behaviours can emerge from the combination of the generated individual behaviours.

In the present work, we propose such a model, which structure is presented in the figure 3.1. This model aims at simulating a part of the rational and cognitive levels of pedestrians' behaviours in urban environments. It especially focusses on the environment analysis and representation, the scheduling of daily activity, the selection of suitable times and locations to perform this activity and the planning of a path between the selected locations. This system relies on four inputs: an informed geometry of the environment, an information database, a description of the agents' characteristics and a description of the agents' activities. Our model enables the description of multiple ways of performing an intended activity. It is robust to variations of spatial and temporal constraints associated with the activity and environment. This makes it able to adapt the activity realisation to a-priori unknown environments and agendas, as well as to various agents' characteristics. This means that a same activity description can be used for different agents in various environments and situations. This also means that many agents sharing a similar activity description may behave in totally different ways depending on their personal preferences and constraints. This property enables the emergence of more diversity in pedestrians' behaviours, hence increasing their credibility. In this section, we give an overview of the components of our model and how they interact. First, we give more detail on the inputs of the method. Second, we give an overview of the main components of the model and their outputs. Finally, we discuss the runtime controller that drives the agent during the simulation and how it uses the different components of the model.

1 Input data

As we discussed in the previous chapter, people's behaviours depend on numerous factors. The most important of these factors is the activity these people intend to perform. The way this activity in performed is highly dependent on the topology and nature of the environment as well as on the temporal constraints associated with the realisation of this activity. People's personal characteristics and preferences also impact their decisions. Providing agents in a virtual crowd with a credible behaviour requires that such factors are taken into account. We propose four

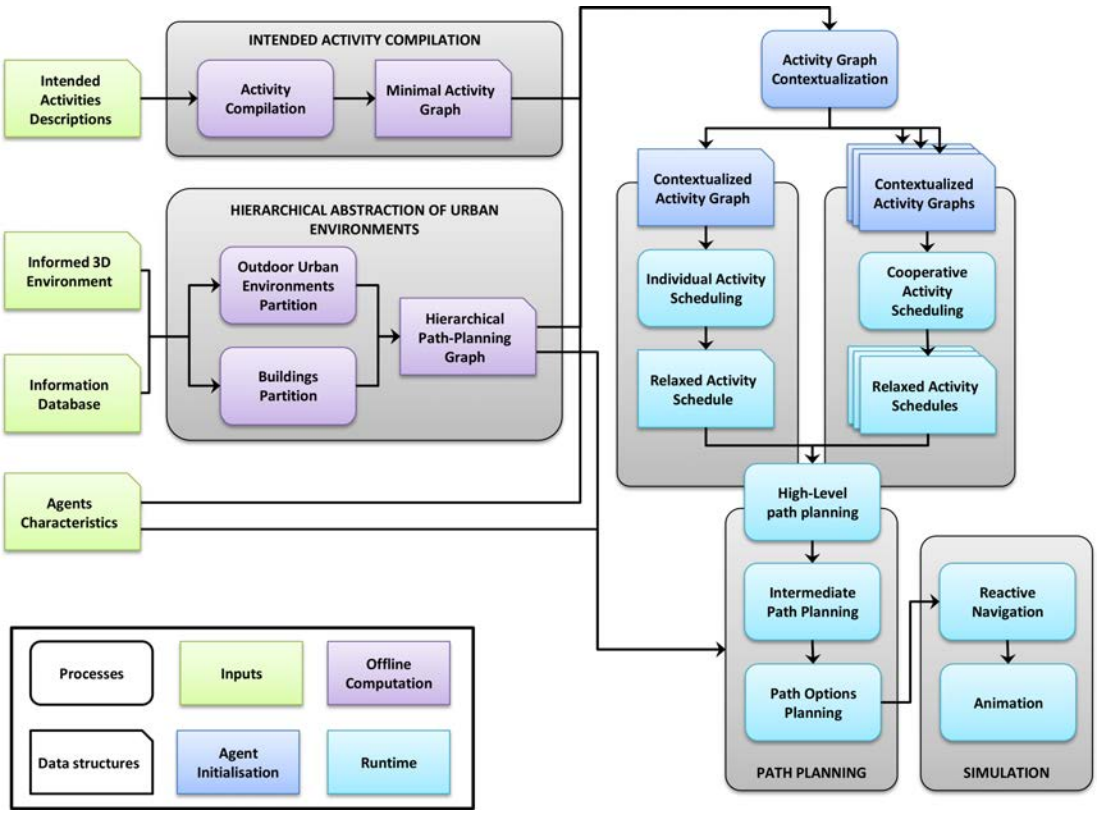


Figure 3.1 – Overview of our model for virtual pedestrians in urban environments

data structures enabling the description of these factors, which are used as an input by our model:

- An **informed geometry of the environment**. It consists of an unorganised set of 3D triangular faces, depicting the navigation surfaces as well as the obstacles in the environment. As stated in the previous chapters, the nature of the navigation zones has an impact on navigation decisions. Our model includes an automatic process extracting semantically coherent zones from the geometry. However, some information is hard to deduce from the geometry only (for example, distinguishing crosswalks from roads or the function of buildings). The geometry of the environment is thus informed with knowledge on the nature of navigation zones and typologies of locations. These zones are for example labelled as reserved to pedestrians or to vehicles, being public or private, or being a bakery, a butcher.... This information is kept through the environment analysis process in order to be available to the path planning and task scheduling processes. This initial information can either be manually attached to the geometry, extracted from existing informed maps (open street map, for example) or attached as a part of the environment generation process (in the case of procedurally generated environment, for example). An example informed geometry is shown in appendix A.1.

- A **description of the agents' intended activity**. It is designed as a tree of activities. Each of these activities is either a task (an atomic action to perform in a single location) or a set of sub-activities. The tasks are labelled with information useful to the scheduling processes: admissible realisation times, estimations of the duration and effort associated

with the realisation of this task and an estimation of the impact of this task on the long-term effort. The tasks or sub-activities composing an activity are linked together by constructors. They are operators describing how these elements can be combined. In our model, we use the following constructors: *sequence*, *without order*, *either* (choice of one sub-activity) and *interlace* (combines activities by interlacing them at the task level). They enable the description of very complex activities. For instance, *without order* and *interlace* constructors enable the description of activities that can be performed in several different ways. Tasks in the description can be constrained with a specific location and/or time interval (for example, an appointment requires the agent to be at the right location before the appointment begins). The structure of the activity description is intuitive, making it simple to understand. Complex activities enabling many possible realisation variants can easily be designed by hand. The description also enables sub-activities to be reused in different activities description, even reducing the description effort. Example task and activity descriptions are shown in appendix A.3 and appendix A.4.

- An **information database** links the activity and the environment descriptions. It associates tasks with the typologies of locations where they can be performed. It also describes the potential temporal constraints associated with locations, such as shops opening hours, for example. The fact that this information is neither included in the environment or in the activity descriptions makes these two descriptions independent. Activities can be created without any knowledge on the structure and nature of the simulation environment. It enables an activity description to be reused in different scenarios: only an update of the information database is required to adapt to the new environment. An example information database is shown in appendix A.2.

- **Agents' characteristics and preferences**. Different humans intending to perform similar activities do not necessarily perform the same tasks in the same order nor select the same locations to perform these tasks. This is due to the fact that each human being has their own preferences and capabilities impacting his decision. In order to model this diversity, we embed each agent with a set of personal parameters and preferences. As navigation between zones is especially important in the scheduling of activities, a set of paces is given to each agent, with associated speeds as well as preferences over the nature of navigation zones to travel. Agents are also embedded with preferences concerning their favourite tasks, locations or paces. These characteristics and preferences enable the generation of multiple archetypes of agents such as retired people, children or students. The generation of these agents archetypes can be achieved using behavioural studies on the associated groups of people (older people tend to avoid hurrying, on the contrary of students, for example). Assigning these archetypes to agents in the city can be achieved by using statistical data over the repartition of people in actual cities. For example, it is possible to know the statistical repartition of socio-professional categories of people given the type of housing. The description of such personal characteristics and preferences enables the generation of crowds in which a wide variability of behaviours emerge from a limited number of activities descriptions.

These input descriptions contain the information we believe meaningful for activity scheduling and path planning purpose. Furthermore, they possess good properties. First, the descriptions of the activity, of the environment and of the agents are independent from one another. This means that these descriptions can be generated separately, either by hand or by automated processes. This also enables the design of archetype of agents or activities, which can easily be reused in different setups. The information database is the only part of the description that must be updated in order to link the different inputs. Second, the description of the inputs is intuitive, enabling an easy manual design of these elements. Overall the inputs descriptions

we propose enable the generation of intuitive, independent and reusable descriptions of the environment, of the agents and of their intended activities

2 Data refinement processes

The environment, activity and agent descriptions presented in the previous section demonstrate good properties in term of independence, reusability and intuitiveness. However, these descriptions need to be refined into representations more suitable for path planning and activity scheduling purpose. Our system contains three data refinement processes: the environment abstraction process, the activity compilation and its contextualisation.

2.1 Environment abstraction process

The initial representation of the environment is an informed raw geometry, i.e. a set of unorganised triangular cells, labelled with information on the nature of the zone. A representation more suited to path planning and activity scheduling purpose is required. Exact cell decompositions offer good properties for planning and environment analysis. Therefore, the first part of our environment abstraction process consists in extracting an informed Delaunay triangulation of the environment's navigation surfaces, constrained with bottlenecks [Lamarche, 2009]. As explained in the previous chapter, informed environments and hierarchical abstractions also exhibit good characteristics for enhancing the credibility of planned paths and the decreasing the planning complexity. We propose an environment abstraction process generating a semantically consistent decomposition of the environment as a hierarchy of zones. These zones are identified using parameters such as their shape, their nature and the nature of their borders. Our environment abstraction process relies on two distinct automatic decomposition processes:

- A first process identifies zones in outdoor urban environments that match usually accepted urban entities. It extracts three levels of abstraction. The city areas level identifies large areas such as streets, crossroads, pedestrian areas and buildings. The street sections level identifies sidewalks and provides information on the crossing opportunities between them. Finally, the navigation tiles level identifies small semantically homogeneous near-convex cells that provide the smallest unit of path planning.
- A second process decomposes the buildings extracted by the first process into four hierarchical decomposition levels. The higher levels separate outdoor and indoor areas. The intermediate levels respectively identify the buildings floors and rooms. Finally, the lower level identifies navigation tiles in buildings.

From these decompositions, a hierarchical informed roadmap is extracted. It completely represents the connectivity of the navigation surfaces while providing all the information on the nature of the environment required by our path planning and activity scheduling processes. Each of the extracted roadmap levels is adapted to a different level of decision making. Indeed, the higher level roadmap is adapted to coarse path planning at the city level, the lower level is adapted for precise local path-planning, and the intermediate levels are useful to guide the local path planning by providing information on the long-term impact of decisions. This environment abstraction process is further developed in chapter 4.

2.2 Activity compilation

The representation of the agent's intended activity as a tree of activities is intuitive and easy to design by hand. However, this representation suffers from several weaknesses when used

for activity scheduling purpose. First, such trees of activities can be partially redundant: two concurrent activities may share some sub-activities (for example, the activities consisting of buying bread in a bakery or in a mall may share a task consisting of getting some cash at an ATM). This means that the tree representation is not minimal. For a same activity, several equivalents tree representations may exist. It also means that in case of a failure in the resolution of an activity, the failure recovery process will have trouble to determine if a sub-activity have already been satisfied as a part of another activity or not. Second, our scheduling algorithms rely on the knowledge of the next tasks that the agent may perform. As the tree representation does not explicitly list the possible task orderings, such data is more complex to obtain. In order to provide the activity scheduling process with a more suitable description of the agent's intended activity, the tree representation is compiled into an activity graph. This activity graph is a state machine which describes all independent valid sequences of tasks leading to the completion of the activity. States in this graph represent the situations the agent can be in. Transitions between these states represent the tasks that need to be performed to progress toward the activity completion. The compilation process ensures that the computed state machine contains the minimal number of states and transitions required to completely represent the described activity. This means that no state is duplicated in the graph, ensuring that no task sequence is missed in case of failure recovery. As the activity compilation process only relies on the description of the activity, its computation can be carried out offline, independently of the environment representation. The activity compilation process is further developed in section 5.1.

2.3 Activity graph contextualisation

The computed activity graph offers a representation of the activity suitable for activity scheduling purpose. However, it only relies on the initial description of the activity, which is independent of the environment description. Yet, when considering tasks to perform in a city, an important part of the time constraints to consider depends on the environment itself (opening hours of shops, for example). To take this information into account, the activity graph is contextualized using the information provided by the information database. Temporal constraints associated with tasks are deduced from the constraints described in the initial activity description as well as those described in the information database. This enables the computation of time limits before which each state must be reached for the completion of the activity to be possible. This information is precious as it enables to know if an agent can possibly complete the activity while respecting the temporal constraints of the problem. Estimations of the long-term efforts occasioned by the completion of tasks are also propagated in the graph in order to associate an effort penalty to each state. This effort penalty provides the activity scheduling process with useful information on the long-term impact of these tasks on the global effort of the solution. This activity graph actualisation is also able to remove some tasks from the graph if no location in the environment enables their completion. The activity graph contextualisation process is further developed in section 5.1.

3 Model's main components

In this work, we propose three different decisional processes, which simulate different levels of rational decision making. Our Individual task scheduling process selects locations and times to perform tasks in order to complete an intended activity. Our cooperative task scheduling process extends the individual scheduling by considering tasks that must be performed in cooperation by multiple agents. Our hierarchical path planning process selects sets of path options in the

environment enabling the agent to navigate between the locations selected by the scheduling processes.

3.1 Individual activity scheduling process

The individual activity scheduling process aims at simulating the human rational process involved in the choice of the best way of performing an intended activity. It selects a valid task sequence fulfilling the activity. Suitable locations where to perform these tasks are selected in a way that minimizes the agent's global effort. This effort is either associated with the realisation of tasks and to the navigation between the selected locations. The estimation of the effort associated with the realisation of tasks is computed using the description of the task, the potential waiting time required and the personal preferences of the agent over task realisation. The effort associated with navigation is computed based on the distances between locations, on the navigation capabilities of the agent (paces ...) and on its personal preferences in terms of nature of the navigation surfaces. Relaxed time intervals are also associated with each task, indicating when the agent should perform the selected tasks. They inform the agent of the allowed time to navigate between locations, enabling the selection of adequate navigation speeds and the detection of potential failures. The individual activity scheduling process also selects a set of city areas to travel between the selected locations. It uses a heuristic as well as pruning methods in order to drastically reduce the complexity of the search while ensuring to find an optimal solution. The proposed individual activity scheduling process is complete, meaning that if a solution exists, it will be found. The resulting agent behaviour demonstrate long-term planning capabilities, for example deciding to hurry in order to avoid a future detour , passing by home to drop grocery bags or delaying other tasks in order not to be late at a meeting. The proposed process also takes the agents personal characteristics into account. This potentially generates different schedules for agents sharing similar intended activities. This way, a wide diversity of behaviours can be generated from a relatively small number of activity descriptions. We demonstrate through an experiment how this process enables the generation of agents' schedules statistically consistent with humans' decisions. This activity scheduling process is developed in details in section 5.2

3.2 Cooperative activity scheduling process

Many activities include, among individual tasks, cooperative tasks such as, for example, meetings or exchange of documents. These cooperative tasks require a synchronisation between two or more individuals. On the one hand, if these tasks are not constrained to a single location and time interval, scheduling activities individually for each agent does not allow the selection of a common decision. On the other hand, considering the combination of agents' activities as a whole leads to overwhelming complexity issues. To avoid this complexity, we decouple the scheduling of individual tasks from the scheduling of cooperative tasks. To further reduce the complexity, the proposed process only explores a promising subset of the potential solutions. The cooperative activity scheduling process is composed of three main steps. First, for each agent, the individual task scheduling algorithm is used to compute a set of cooperation proposals. These proposals are the potentially optimal ways an agent reaches situations allowing the realisation of the cooperative task. Second, the proposals of all agents involved in the cooperative task are matched and synchronized. This creates a filtered set of possibly optimal cooperation configurations with associated locations and times. Finally, the most promising configurations are tested in order to select a solution offering the lower combined cost for all involved agents.

This process is able to schedule multiple intended activities containing cooperative tasks as

well as individual tasks. The locations and times to perform cooperative tasks are selected by taking into account the intended activities, constraints and characteristics of each agent involved in these tasks. An activity schedule is produced for each agent, with associated locations and relaxed time intervals. A compromise is made between the quality of the computed solution and the performances of the process. The more promising configuration are selected, the closer to the optimal is the computed solution, but the higher is the computational cost of the search. Yet, the process is complete : if as solution exist, it will be found. Furthermore, we show that the selected schedule is usually close to the optimal, even with a limited number of selected configurations. This compromise, combined with filtering and pruning methods, drastically reduce the computation time of the search. The cooperative activity scheduling process is developed in details in the chapter 6.

3.3 Path planning process

The relaxed schedule provided by the individual or cooperative activity scheduler provides a set of locations to reach with associated time interval. The role of the path planning system is to select a suitable path leading to the next selected location. However, when navigating in cities, people do not consider all the details of their path at once. Instead, they first consider a coarse path in terms of streets to travel. Then ,they refine this path when needed, taking local information into account. Our path planning process simulates this decision making by taking advantage of the hierarchical representation of the environment. It uses the environment representation, the relaxed activity schedule and the description of the agent characteristics as inputs. Our hierarchical path planning process is divided in three main steps:

1. **High-level path-planning.** A coarse path in terms of streets to travel is planned between the current location of the agent and it goal. The path planner uses the higher level of environment abstraction, which decomposes the environment in city areas.
2. **Long-term effort estimation.** An exploration of the intermediate levels of environment abstraction is performed. It estimate the remaining effort required to reach the goal from intermediate zones' limits. These estimations are used to provide the agent with information of the long-term impact of its local decisions. In order to reduce the complexity of the search, only the intermediate levels zones belonging to the selected city areas are considered.
3. **Local path options planning.** A set of low level path options is computed during navigation inside the areas the agent is about to enter. This computation relies on the lower level of environment abstraction, which describes the environment as a set of navigation tiles. The computed path options are represented as a network of oriented edges linking tiles borders. Each of these options is labelled with an estimation of the remaining effort required to reach the goal, guiding the local decisions of agents.

This path planning process takes advantage of the hierarchical nature of the environment description to delay the local path-planning, reducing the global planning complexity. It provides the agent with multiple path options. These options are updated as the agent navigates through the environment, providing an estimation of the long-term impact of local decisions. This path panning process is developed in details in the section 4.2.

4 Runtime agent controller

During navigation, a reactive navigation process is used to follow the selected path options while avoiding dynamic obstacles. This process provides a smoothed path free of collision through

the navigation tiles. The agent is animated along this path, and lower-level behaviours or pre-computed animations are used to represent the realisation of tasks. The agent is embedded with a runtime controller, described in Figure 3.2, which is in charge of executing the different processes involved in the agent's behaviour. It is also responsible for monitoring the perception of the agent and detecting unexpected events. In this section, we will first treat the normal control workflow, then we will consider the reaction to unexpected events and failures.

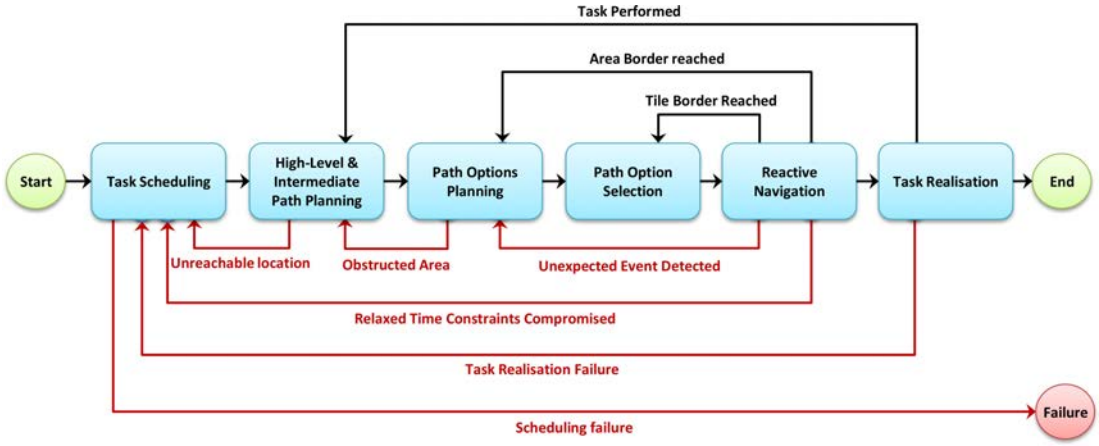


Figure 3.2 – Runtime controller for one agent. The upper arrows represent the normal process although the lower arrows represent the error recovery and adaptation to unexpected events.

4.1 Normal workflow

If the knowledge of the agent on his environment is sufficient and nothing unexpected happens, the initial agent's behaviour is computed as follows:

1. The agent's intended activity is scheduled, individually or in cooperation with other agents.
2. A high-level path is computed through the city leading to the first location where the first task must be performed.
3. A set of low-level path options is computed in the first city area to travel.
4. A first path option to follow is selected among the ones leaving the initial position of the agent.

Then the agent begins to follow the first selected path using the reactive navigation process. Reaching some locations along the path triggers the executions of some of our decision making processes:

- Every time a location where a task must be performed is reached, the low-level behaviour associated with this task is triggered. When the task realisation is over, a new high-level path planning is performed toward the location associated with the next task.
- Every time the agent is about to reach a navigation tile border, it uses its perception of its vicinity and the information provided by the path options graph to make a rational decision on the next path option to follow.

- Finally, every time the agent is about to reach a city area border, a new low-level path options planning is performed in the next area to travel.

When the last task the agent intended to perform is completed, the process stops with a success.

4.2 Failure recovery and reaction to unexpected events

Cities are dynamic environments in which many unexpected events can occur. These events may impact the behaviour of the agents, delaying them or even preventing them from fulfilling the activity as scheduled. This means that feasibility or optimality of the planned paths or schedules can be compromised. Here is a list of the kind of failures and unexpected event our algorithm is able to deal with:

- Impeding events such as an unexpected density of pedestrians can delay the agent. During navigation, the controller may detect that the next goal location cannot be reached within the specified relaxed time interval without switching to a higher pace. The **respect of temporal constraints being compromised**, a new activity scheduling is performed.
- A **task realisation failure** may occur, for example if a shop is unexpectedly closed. In that case, a new activity scheduling must be performed, selecting another way of fulfilling the intended activity.
- **Unexpected events** such as a puddle of water or a group of person standing on the path may sometime make the crossing of a tile less attractive. In that case, the effort associated with the navigation of the corresponding path options is increased. The remaining of the graph is updated accordingly, now reflecting the agent's will to avoid this obstacle, if possible.
- Other **unexpected events**, like a car parked on a sidewalk, sometimes completely obstruct a tile. In that case, the corresponding path options are invalidated in the graph, which is updated accordingly.
- Large obstacles such as road works can **obstruct a whole city area**. In that case, the low-level path planning in this zone encounters an error as no path exist from the entrance border to the exit border of the zone. In that case, a new high-level path must be computed, taking into account the incapacity to travel this area.
- Obstructed areas may create **unreachable locations**. If a global path planning fails at finding a path through city areas leading to the location where the next task must be performed, it means that this area is unreachable. A new activity scheduling is computed, taking into account the incapacity to travel the obstructed areas.
- An **activity scheduling failure** may happen, meaning that no valid task arrangement was found. In that case, the activity constraints must be relaxed or some tasks removed from the activity. However, this is in the domain of activity planning and not activity scheduling, which is outside the scope of our method.

5 Conclusion

Our model simulates a part of the human rational behaviour, focussing on the activity scheduling and path planning processes. It relies on independent inputs: a representation of the environment's structure and nature, a description of complex activities including spatial and

temporal constraints and a description of agents' personal characteristics and preferences. The model combines these representations at runtime in order to take into account the tight relation between space, time and activities. The activity realisation is adapted to the structure and nature of the environment, to the agent's specific temporal and spatial constraints and to its personal characteristics and preferences. This way, a wide variety of behaviours can be generated from a single activity description. The description of complex behaviours that involve cooperative tasks requiring synchronisation between multiple agents is also possible. In that case, the personal activities, constraints and preferences of all the involved agents are considered in order to find a solution offering a good compromise. The navigation of agents between the locations where they intend to perform tasks is guided by a network of path-options. Along his path, these options are offered to the agent, which may choose between them, relying on an estimation of the long-term impact of these choices. This enables the agent to seamlessly react to a wide range of unexpected events. As the problems we handle imply huge search space, specific efforts were deployed to reduce the processes computation costs through the use of heuristics, filtering and pruning techniques. In the following, we will describe more in detail the components of our model.

Path planning in hierarchical representations of urban environments

4

When planning a path in their environment, pedestrians do not consider every detail at once, but first plan a coarse path to their goal. For example, in a city, people first choose which streets to travel. Local decisions such as where to cross a street or on which side to bypass a pole are delayed to the moment such decisions are required. Inside buildings, people demonstrate similar behaviours, first reaching the right floor before trying to find a room on this floor. These behaviours illustrate the fact that people consider several levels of abstraction of their environments [Hirtle and Jonides, 1985]. When figuring out a path in a city, people usually rely on commonly accepted city planning concepts such as "streets", "sidewalks" or "crosswalks" and on architectural concepts such as "buildings", "floors" and "rooms". These zones are identified considering characteristics such as their shape, their connectivity to other zones, their nature and the nature of their borders.

In computer science, hierarchical representations of environments have mainly been used to improve planning algorithms' performances [Botea et al., 2004][Brand and Bidarra, 2011]. Yet, we believe that using a hierarchical environment representation relying on concepts familiar to humans enables the generation of smarter navigation behaviours. Indeed, such a representation offers the possibility to delay agents' local decisions until suitable information is available. The identification of semantically meaningful zones also enables the adaptation of agents' behaviours to the type of zone they go through. Indeed, people do not behave the same way when walking in a park, on a sidewalk or when crossing a street. Therefore, we believe that, in order to generate credible pedestrian behaviours in virtual cities, the environment must be partitioned into semantically coherent zones similar to the ones people use, such as streets, crossroads, floors and rooms.

However, in the literature, the hierarchical decomposition of the environment is usually computed using purely geometrical approaches, focussing solely on decreasing the path planning cost. We propose a method that automatically generates semantically coherent hierarchical representations of virtual urban environments. This method relies on two original environment decomposition processes. The first process extracts a hierarchical decomposition of an informed urban environment. This representation is composed of three levels of decomposition, relying on commonly accepted city planning concepts such as streets, sidewalks and crosswalks. The second environment decomposition process extracts a hierarchical decomposition of a building using commonly accepted architecture concepts such as floors, stairs and rooms. In these hierarchies, each abstraction level is adapted to a different level of decision making. We also propose a method that takes advantage of this hierarchical decomposition to generate more credible paths. It first plans a coarse path to a goal as a sequence of streets and buildings to travel. Then, it relies on the lower levels of the decomposition to refine this coarse path into a set of local path options during navigation. Those options can be used in real time to adapt the planned path to the perceived situation and take a detour if necessary.

In this chapter, we first describe our environment decomposition processes and the produced path planning graphs. Then, we discuss our hierarchical path planning method and how it takes advantage of the environment's hierarchical representation. Finally, we show the results obtained by both our decomposition processes and our path planning algorithm.

1 Environment representation

A virtual environment is usually defined as raw geometries composed of an unorganised set of geometrical faces. In order to navigate in it, virtual agents rely on a representation of this environment. Exact cell decomposition methods demonstrate good properties for real time path planning problems [Kallmann, 2010]: they exactly represent the connectivity of the free space, they explicitly represent the obstacle borders and they guarantee a representation complexity linear with the complexity of the initial geometry. However, when navigating in their environment, people do not only rely on its geometry. First, people consider the nature of navigation zones. For example, people tend to walk on sidewalks and use crosswalks to cross streets. Informed environments provide the agent with information on the nature of zones and objects [Thomas and Donikian, 2000]. Second, people reason on a hierarchical representation of their environment Hirtle1985. Providing the agent with a hierarchical informed representation of the environment enables the generation more credible navigation behaviours [Farenc et al., 1999]. However, manually creating this kind of representation is a tedious process. We propose a method that automatically extracts an informed hierarchical representation of urban environments. It relies on an exact cell representation of the environment's geometry labelled with general information on the nature of navigable areas. This method consists of two independent environment decomposition processes. The first one extract city planning areas such as crossroads and streets form outdoor environments. The second one extracts meaningful zones such as rooms and stairs from buildings. A unified hierarchical path planning graph is created from the output of these two processes. This graph greatly helps the information process and provides a suitable representation for real-time path planning.

In this section, we first give an overview of the hierarchy of semantically meaningful zones we use to decompose virtual cities. Then, we describe the initial cell decomposition of the environment and we detail the two decomposition processes, extracting meaningful zones from outdoor urban areas and buildings. Finally, we describe the extracted environment representation.

1.1 Semantically meaningful hierarchical representation of virtual cities

In order to generate more credible agent paths, one solution is to design path planning processes inspired by actual pedestrian behaviours. To do so, we rely on a hierarchically organised representation of urban cities. It describes different levels of abstraction of the environment as partitions of semantically meaningful zones inspired by city planning and architecture concepts. In this section, we define the notion of meaningful zone, we propose a hierarchy of meaningful zones to abstract urban environments and we discuss the information required by our decomposition process.

1.1.1 Initial environment

A virtual environment's geometry consists in an unorganised set of triangular cells. Our method automatically extracts semantically coherent zones from this geometry. However, not all information can be deduced only from the geometry. For example, it is almost impossible to distinguish a crosswalk from the surrounding road or sidewalk solely using the geometry of the ground. It is also complex to determine the nature of a building considering only its shape.

Yet, information on the nature of navigation zones enable a better credibility of planned paths [Jaklin et al., 2013]. It is also useful to possess information on whether zones are public or private when planning a path, as we do not want agents to cross their neighbour's house to avoid a detour. Furthermore, scheduling of an activity may require additional information that cannot11 be automatically deduced. For example it requires information on the nature of buildings or rooms to determine where tasks can be performed. Therefore, our model requires an initial labelling of the geometry with basic semantic information. We define three kind of such semantic information:

- **Nature of navigation zones.** The navigation zones in the geometry are labelled as buildings, reserved to pedestrians, reserved to vehicles or crosswalks (pedestrian-allowed zones crossing a vehicle-reserved lane). This information is useful to guide the path planning.
- **Access allowance.** Navigation areas are also labelled as public, being accessible to any agent, or private, being restricted to some agents.
- **Typology of buildings.** Buildings are labelled with a typology (school, grocery store, home...). This typology is later used to determine which tasks can be performed in these locations.

These labels are kept through the decomposition process in order to provide the required information to the path planning and task scheduling processes. There are many ways of providing this initial information. If the environment geometry was built by hand or automatically captured by 3D vision methods, this information must be manually attached to the geometry. However this information can also be automatically generated or extracted from informed maps (for example, we use Open Street Map to provide information on the nature of navigation zones and buildings). In procedurally generated environments, this information can be attached during the generation process. An example informed geometry is shown in appendix A.1.

1.1.2 Notion of meaningful zones

We propose subdivision processes that identify semantically meaningful zones in the environment, with different levels of abstraction. The identification of these zones relies on four main criteria: their shape, their nature, the nature of their borders and their connectivity to other zones.

- **Zone shape criterion.** The identification of convex zones in the environment offers several advantages for path planning purpose. First, the convexity of a zone convex ensures that an agent in this zone is able to perceive it entirely. For this reason, it is safe to consider that the agent is aware of dynamic information such as pedestrian densities or traffic lights inside this zone. This means that the agent is able to plan a path through this zone in accordance with the perceived information. Second, the convex nature of zones guaranties that they can be crossed in straight line, making the length of this path easy to compute. For these reasons, convex zones are especially suitable for path-planning purpose.

  However, in real situations, small obstacles such as city furniture, as well as the exact shapes of walls do not strongly affect the pedestrian perception or navigation. Therefore, an approximation of zones' convexity is sufficient when partitioning environments. In fact, people tend to naturally decompose complex shapes into near-convex parts [Ren et al., 2011]. However, the notion of approximate convexity does not have a proper mathematical definition. A proposition of definition of near-convexity was introduced in

[Lien and Amato, 2004]. Holes and concave edges are detected by comparing the geometry to its bounding box. A concavity factor of the geometry is then estimated by summing the square distances between the vertices of the geometry and the edges of its bounding box (see figure 4.1). In the case of holes, the concavity factor is computed considering the distance between the vertices of the hole's border and the backbone of the hole's geometry. A concavity tolerance is specified: a zone whose concavity factor is under this tolerance is considered as near-convex. It means that a geometry that contain a border vertex very distant from the bounding box (see figure 4.1.c) will be less likely to be considered near-convex than a geometry with several vertices close to the bounding box (see figure 4.1.b). This guaranties that the travel distance required to cross a near-convex zone can be estimated as the straight line distance, with a minimum error.

Allowing an approximation of the convexity criterion enables the extraction of more meaningful zones and greatly reduces the number of elements in the decomposition. Yet, when planning a path through such zones, they can be considered as convex with a minimum error. For these reasons, we use the "near-convexity" criterion to identify meaningful zones in virtual cities. We also use the concept of "improving a zone convexity". It means that by merging a set of cells to a zone, a better convexity factor that the one of the initial zone is obtained.

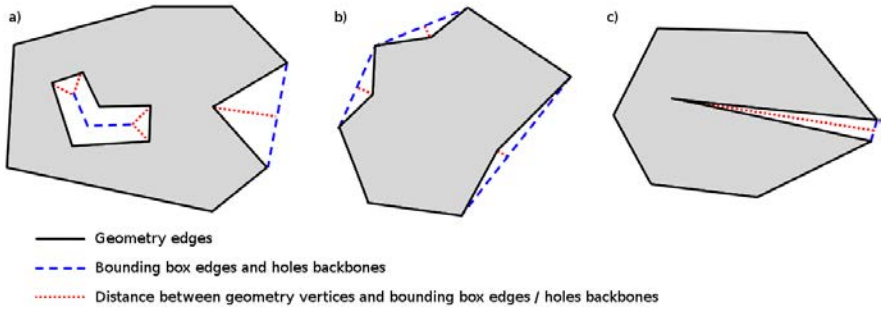


Figure 4.1 – Method used to estimate a geometry's concavity (a) and two example geometry, being near-convex (b) or not (c) for a given concavity tolerance.

- **Semantical homogeneity criterion.** Most commonly accepted city planning concepts are identified by the nature of the navigation surfaces. For example, a crosswalk indicates a zone allowing pedestrians to cross a street and a square is a pedestrian-reserved area. Merging together neighbour cells of the geometry of same nature thus makes sense when aiming at the identification of consistent zones. Furthermore, a semantically homogeneous zone ensures that all paths through this zone are only going through semantically identical cells, thus making the evaluation of the cost of the path easier. However, the notion of nature of zone is also dependent on the considered abstraction level. For example, a street can be considered as semantically homogeneous at a high level of abstraction, but is composed of roads, pedestrian areas and crosswalks when considering it as a more precise level. This has to be taken into account in the decomposition process.
- **Zone borders' nature criterion.** The nature of borders in the environment is useful for the identification of meaningful zones. For example, in [Lamarche and Donikian, 2004], bottlenecks and steps edges are identified to help the environment's decomposition. Indeed, floors in a house can be distinguished by cutting the building navigation surface at step edges. Bottlenecks can also be a good indicator of a border between two zones. Doors, for example, are by nature defined by bottlenecks. Identifying zones separated by bottlenecks can also ensure that no punctual obstacle exists in zones.

- **Zone connectivity.** The connectivity of a zone, i.e. its number of neighbours, is a useful criterion to identify a zone. For example, crossroads can be identified as street sections connected to more than two other street sections. This criterion was used in [Lamarche and Donikian, 2004] and [Paris et al., 2006] to generate coherent topological abstractions of virtual environments. We use this criterion to distinguish some meaningful zones during our decomposition process.

In our environment decomposition processes, we use those four criteria to identify semantically meaningful zones, suitable for path planning purpose. Since all commonly used city planning and architecture concepts do not rely on all these criteria, we select the most suitable criteria to use at each step of the decomposition.

1.1.3 Hierarchical decomposition

We propose a method that automatically extracts an informed hierarchical representation from an informed geometry of the environment. This method relies on environment decomposition processes that identify semantically meaningful. Figure 4.2 depicts the hierarchy of meaningful zones that we propose.

Our model decomposes virtual cities into several hierarchical levels: the city area level, the intermediate sections levels and the navigation tiles level:

- The **city area level** is the higher level of abstraction in our representation. It decomposes the environment into city areas: large zones describing the connectivity of the city as a network of streets and buildings. It enables the planning of a coarse path through the city. Four kinds of areas are distinguished:
  - *Pedestrian areas* are large near-convex zones reserved to pedestrians, such as pedestrian streets or squares.
  - *Streets* are near-convex urban areas composed of roads, sidewalks and crosswalks, and not connected to more than two other streets or crossroads.
  - *Crossroads* are near-convex urban areas composed of roads, sidewalks and crosswalks, and connected to a least three other street or crossroad areas.
  - *Buildings* are zones composed of constructions and their associated exterior areas.
- The **intermediate sections levels** decompose city areas into smaller semantically homogeneous zones such as sidewalks, road sections or buildings floors. Their goal is to provide the agent with a more detailed knowledge on the navigation options along its path. They enable the agent to estimate the long-term impact of its local decisions. As streets and buildings do not share a common structure, different intermediate levels are used to decompose buildings and outdoor city areas. In outdoor city areas, the important knowledge concerns the presence of pedestrian-allowed areas and the crossing possibilities between them. For this reason, we distinguish three kinds of street sections at this level:
  - *Pedestrian section* represent either near-convex parts of sidewalks or of others pedestrian areas.
  - *Crossable road sections* indicate near-convex vehicle-reserved lanes that can be crossed using a crosswalk.
  - *Road sections* are near-convex vehicle-reserved lanes that cannot be crossed using a crosswalk.

  The structure of buildings is more complex, as they often contain multiple floors and complex connectivity between rooms. We define three intermediate levels of decomposition in building areas, defined as follows:

  - The higher level of intermediate building sections distinguish the *indoor sections* of building from their associated *exterior sections* (gardens, courtyards, balconies...) and from their *covered exterior sections* (archways ...).
  - The next level of intermediate building sections decomposes indoor sections into *floors*, linked by *stairs*.
  - The last level of intermediate building sections decomposes floors into *rooms*.

- The **navigation tiles level** is the lower level of abstraction in our representation. It decomposes the intermediate sections into sets of semantically homogeneous near-convex zones: navigation tiles. It provides knowledge on the structure of streets and buildings in term of pedestrian-reserved zones and interconnection between them. It offers a compact representation of the environment's geometry while precisely describing its connectivity. These properties make this level suitable for local path planning. We distinguish ten kinds of tiles, five in buildings and five in outdoor urban areas:

  - *Pedestrian tiles* are pedestrian-reserved zones such as sidewalks, squares or pedestrian streets.
  - *Road tiles* are vehicle-reserved lanes. The agents are not meant to walk on these zones, but they are given this capacity. This enables agents to walk on the road to bypass eventual obstacles.
  - *Crosswalks* regroup all the cells of a single crosswalk. These zones offer an allowed crossing point of a road. Crosswalks with complex shapes are decomposed into several crosswalk tiles.
  - *Crosswalk accesses* represent the portions of pedestrian zones allowing access to a crosswalk. These zones are useful to determine the connectivity between two sidewalks, and give an indication on where to wait for crossing a road.
  - *Building accesses*, on a similar way, indicate the portions of pedestrian zone allowing access to a building. These zones also give knowledge on the environment's connectivity and can also be used to model behaviours such as "waiting at the door".
  - *Room tiles* identify near-convex parts of rooms.
  - *Doors* are small tiles that identify limits between rooms.
  - *Steps* are parts of stairs separated by step borders.
  - *Exterior* tiles are near-convex parts of exterior building sections.
  - *Covered* exterior tiles are near-convex parts of covered exterior building sections.

As it relies on commonly accepted concepts, this hierarchy is close form the one used by humans, and thus enables more credible agents' paths to be planned.

1.2 Navigation mesh generation

In order to be able to reason on the structure of the environment, we generate a representation of the free space. Given the good properties of exact cell decomposition for real time agent navigation and space analysis, our work relies on such a method. We use the prismatic subdivision method introduced in [Lamarche, 2009] to generate the informed navigation mesh that is used as a basis for our environment decomposition processes.

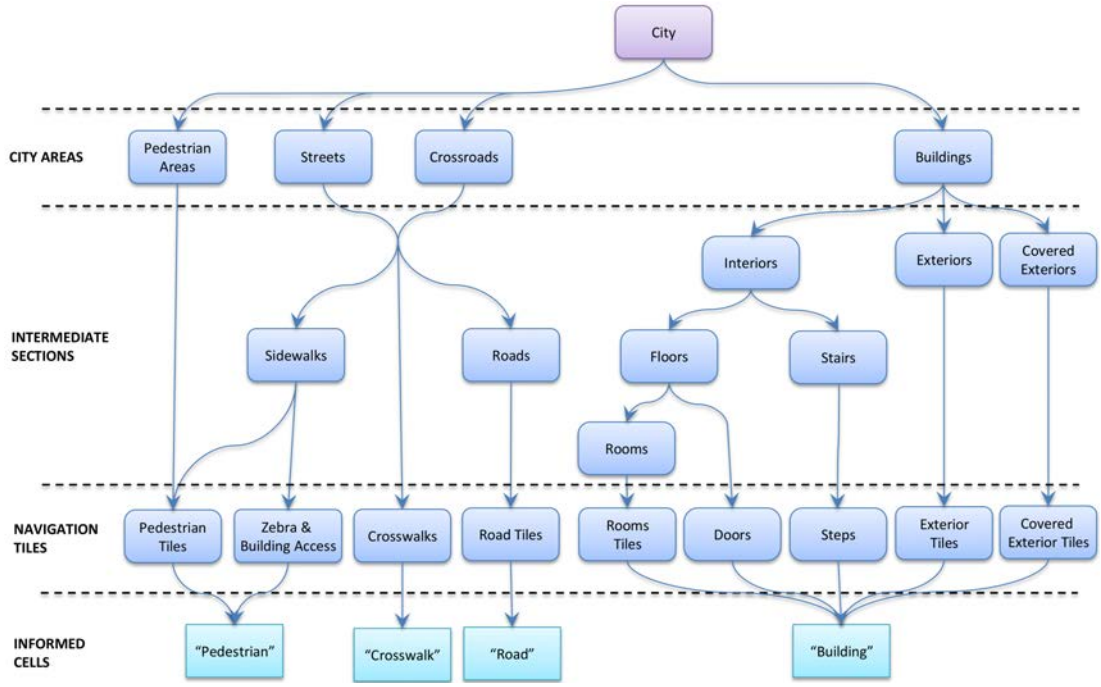


Figure 4.2 – Proposed hierarchy of meaningful zones

1.2.1 Prismatic subdivision

The Prismatic subdivision method introduced in [Lamarche, 2009] aims at organizing a set of 3D polygons in order to capture ground connectivity and identify floor and ceiling constraints. The prismatic subdivision method, depicted in figure 4.3 is performed in three steps. First, all the edges of the geometry are projected on the ground plan, producing a 2D geometry. Second, a constrained Delaunay triangulation of this 2D geometry is computed. Finally, triangular prisms are extruded from this triangulation, intersecting the faces of the geometry and decomposing them into layers of 3D triangular cells. The obtained decomposition offers good properties in term of spatial analysis: cells belonging to the same prism exactly overlap, enabling an easy computation of floor-ceiling relations. By considering the connectivity relations between prisms, this subdivision also enables the deduction of connectivity relations between cells at different heights, like stairs steps for example. The figure 4.4 depicts the prismatic subdivision of a simple building environment.

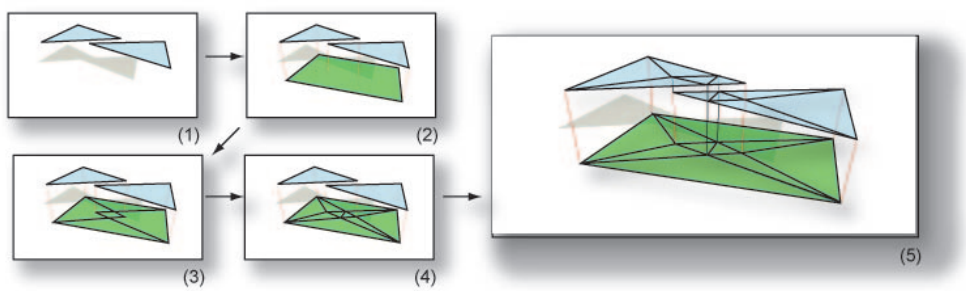


Figure 4.3 – Prismatic subdivision computation. [Lamarche, 2009]

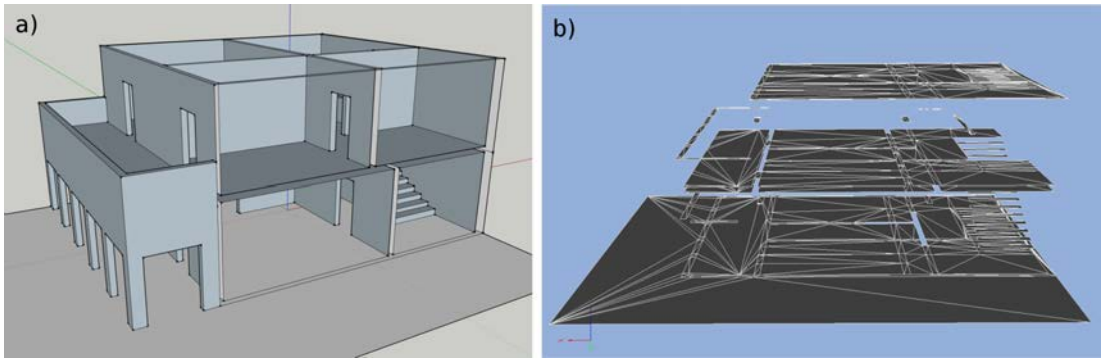


Figure 4.4 – a) A simple example building (wall and ceiling masked for more visibility). b) Its prismatic decomposition as generated by TopoPlan [Lamarche, 2009]

1.2.2 Informed navigation mesh

In order to make it usable for agent's navigation purpose, the decomposed geometry provided by the prismatic subdivision is filtered given a set of characteristics related to the agent capabilities, like the minimum required floor-ceiling distance and the maximal navigable slope. The remaining connected cells exactly represent navigable surfaces. However, many of these cells do not rely on the actual obstacles but on overlapping geometries, causing an unnecessary decomposition complexity. In order to reduce this complexity, all cells belonging to the same surface (i.e. not separated by steps edges) are merged. Then a new constrained Delaunay triangulation is applied to these zones, only considering the actual obstacles. Bottlenecks are added to this triangulation [Lamarche and Donikian, 2004]. The obtained navigation mesh is labelled with information provided by the geometry. In the partitioning processes, we principally consider the nature of navigation zones. We thus identify cells as either *"pedestrian cells"*, *"road cells"*, *"crosswalk cells"* or *"building cells"*. Edges of this mesh are either labelled as *"obstacle borders"*, *"steps borders"* or *"free edges"*. The obtained informed navigation mesh demonstrate good properties in terms of environment analysis: it exactly represent the connectivity of the navigable free space, it explicitly identifies obstacle borders, steps borders and bottlenecks and provides information on the nature of the navigation zones. Due to these good properties, we use this mesh as the lowest level of our environment hierarchical representation and as a basis for our environment decomposition processes.

1.3 Hierarchical partition of urban outdoors

The first of our environment decomposition processes focus on the decomposition of outdoor urban environments, without going into the details of building's structure. It relies on the identification three levels of environment abstraction composed of semantically meaningful zones inspired by city planning concepts. We illustrate the different steps of the decomposition process using an example urban environment, shown in figure 4.5. It begins with the generation of the navigation tiles level (see figure 4.6.b). Then the city areas level is extracted (see figure 4.6.d). Finally, the street sections level is deduced from the two other levels (see figure 4.6.c).

1.3.1 Navigation tiles level generation

The navigation tiles level is designed to provide a precise representation of the environment suitable for local path-planning purpose (see figure 4.6.b). It abstracts the navigation mesh

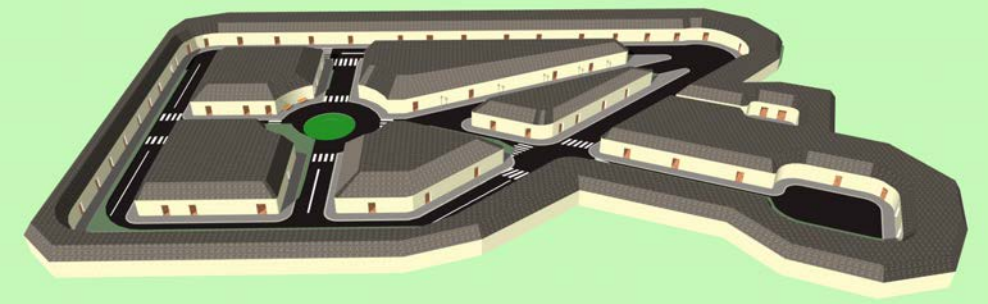


Figure 4.5 – An example urban environment.

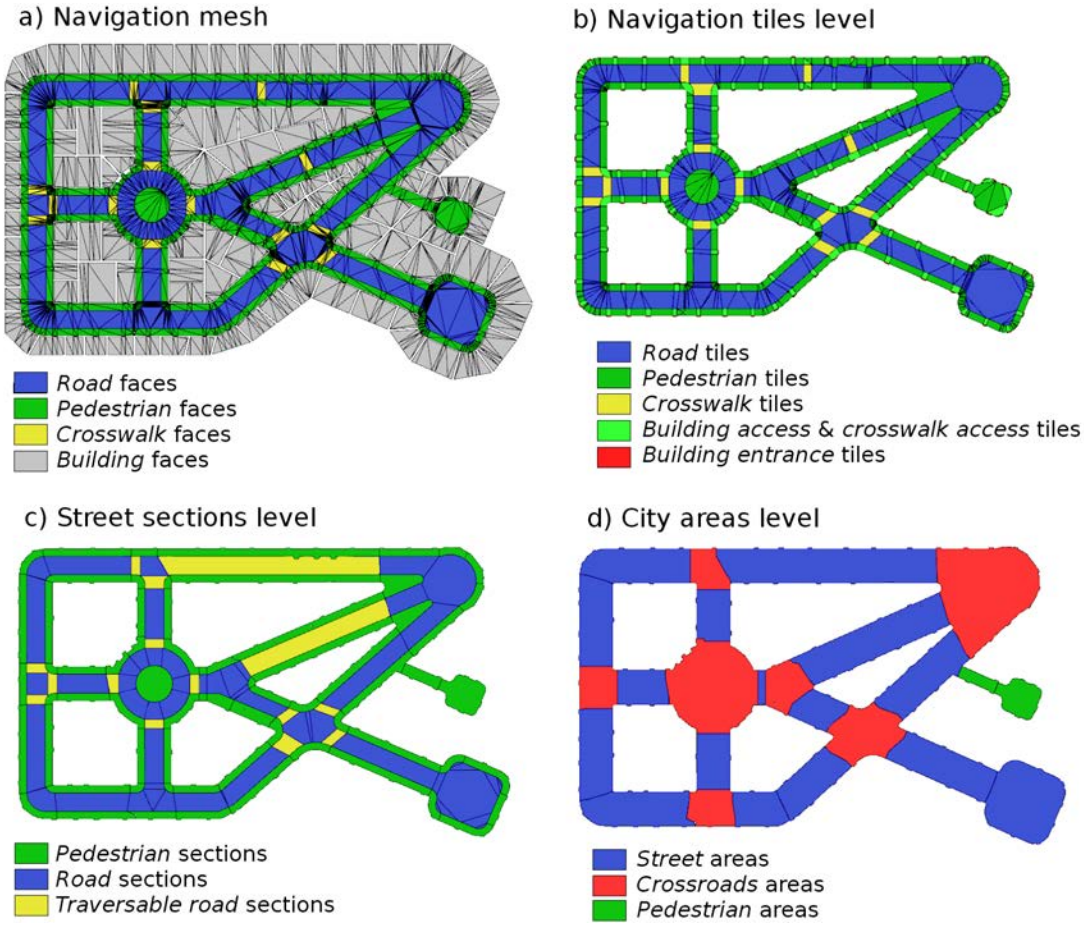


Figure 4.6 – The navigation mesh extracted from the example environment and the three computed levels of abstraction of this environment.

while completely representing the low-level connectivity in the environment. The environment is partitioned into near-convex groups of adjacent triangular cells carrying the same information. This near-convexity factor guarantees that a unique simple path links each pair of tiles borders. At this level, five different kinds of tiles are identified: *road*, *crosswalk*, *pedestrian*, *crosswalk access* and *building access* tiles. The partitioning, described in figure 4.7, is achieved by merging

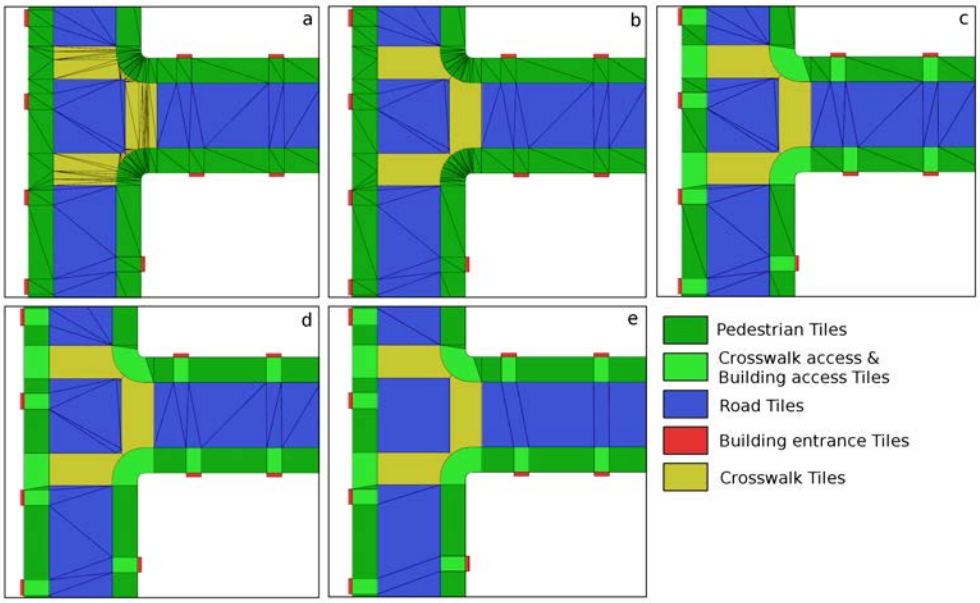


Figure 4.7 – The five steps of navigation tiles generation.

the navigation mesh cells following a set of rules:

1. As this process focus solely on exterior environments, only building entrances are kept in the decomposition (see figure 4.7.a). *Building entrances* are defined as near-convex unions of building cells, adjacent to at least one non-building cell. These zones do not belong to the decomposition: their purpose is to keep track of the building connection to the other zones.

2. *Crosswalk* tiles are identified by merging adjacent *crosswalk* cells into near-convex zones (see figure 4.7.b). As most crosswalk are roughly of trapezoidal shapes, the approximate convexity criteria enables the extraction of crosswalks, but distinguish crosswalks from each other's, even if they are connected by an angle, as in many crossroads.

3. The notion of connectivity is important when planning a path. Knowing which parts of a sidewalk enables the access to a building or crosswalk is useful when considering the connectivity of the streets elements. All *Pedestrian* cells adjacent to a *crosswalk* tile or a building tile are respectively merged into *crosswalk access* tiles or *building access* tiles (see figure 4.7.c). In order to obtain more consistent *crosswalk access* tiles and *building access* tiles, adjacent *pedestrian* cells are added to these tiles if it improves their convexity.

4. *Pedestrian* tiles are identified by merging adjacent *pedestrian* cells that do not belong to a *crosswalk access* or to a *building access* tile (see figure 4.7.d). These cells are merged if they are not separated by a bottleneck and if the resulting zone is near-convex. Using the bottleneck information to separate tiles ensures that these tiles do not contain any punctual obstacle and thus completely describe the connectivity of the pedestrian zones: If a punctual obstacle exists on a sidewalk, the paths going on both sides of it are identified.

5. In order to be able to consider the different ways of crossing roads, segments of roads that link opposite *pedestrian* tiles are identified (see figure 4.7.e). First, vertices shared by at least two *pedestrian* tiles and a *road* cell are selected. Then, for each of these vertices,

the shortest edge linking to such a vertex on the other side of the road is identified as "separating edge". If no such edge exists, the shortest edge linking to any vertex on the other side of the road is identified as "separating edge". Finally, all adjacent *road* cells are merged if they are not separated by a "separating edge" and if the resulting tile is near-convex.

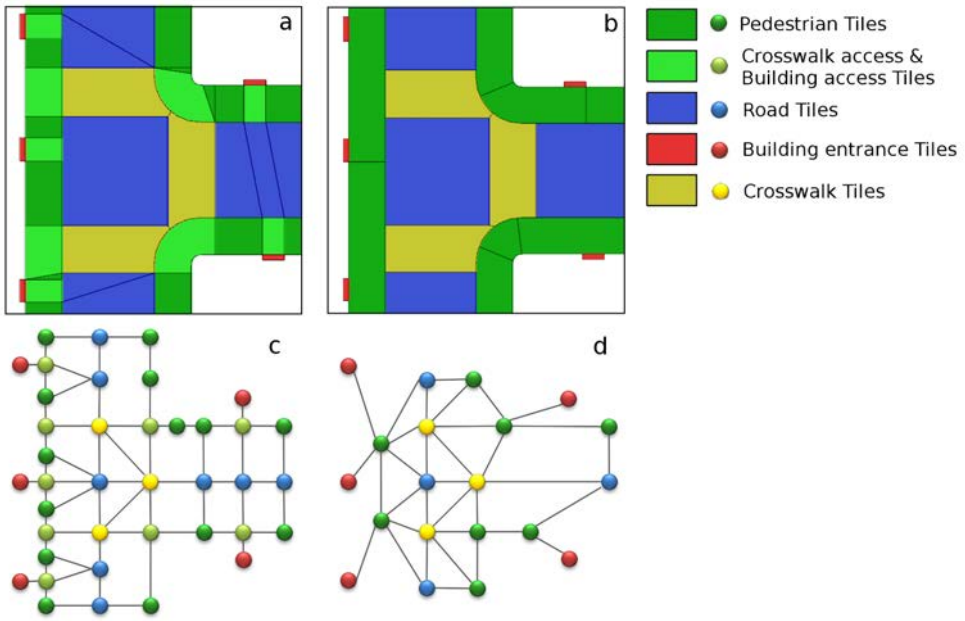


Figure 4.8 – An example of a tile decomposition using our method (a) or only extracting semantically homogeneous near-convex zones (b) and the resulting connectivity graphs (respectively c and d). Our method offers a clearer representation of the navigation zones connectivity.

The *navigation tile* level of decomposition of the environment is composed of near-convex zones, ensuring the visibility of most of the zone to the agent and guaranteeing to find a simple path from one side of the tile to the other. This decomposition process aims at abstracting the navigation mesh while entirely describing the connectivity of the environment. First, the fact that accesses to buildings and crosswalks are explicitly represented aims at making each crossroad or building to be connected to sidewalks by a single tile. Second, the method determining road tiles tends to connect sidewalk tiles on both sides a road with a single road tile. The figure 4.8 compares the decomposition obtained using our method to a simple extraction of near-convex semantically homogeneous zones. Our method generates a more complex decomposition, but offers a clearer representation of the environment's structure. For example, a local path in this representation can be expressed as natural directions such as "Go past the bakery and the bookstore then keep going until you reach the next crossroad, cross it and you will reach the school entrance".

1.3.2 City area level generation

The city area level aims at providing a coarse abstraction of the environment suitable for high-level path planning purpose. It decomposes the environment into street, crossroads and pedestrian areas (see figure 4.6.d). The decomposition process relies on the navigation tiles previously computed. It creates areas by merging these tiles together through a five-step process illustrated by the figure 4.9.

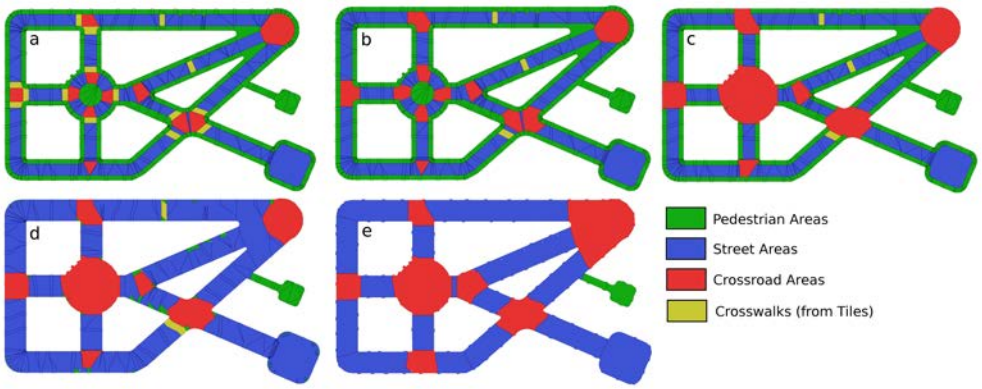


Figure 4.9 – The five steps of city areas generation.

1. Each *road* and *crosswalk* tiles is identified as "crossroad" if contiguous to strictly more than two *road* or *crosswalk* tiles and "street" if contiguous to one or two *road* or *crosswalk* tiles (see figure 4.9.a). Due to the irregularities in the shapes of roads, some small dead-end tiles may exist on road borders, generating improperly identified crossroads. To avoid this, each small dead-end tile is merged with its contiguous tile, transforming improperly identified crossroads tiles into streets areas.

2. It seems more natural to consider the crosswalks surrounding a crossroad to belong to this crossroad than to the connected streets. It enables the agent to delay the decision of crossing a street until the crossroad is reached and more options are available. *Crosswalk* tiles are merged with adjacent areas labelled as crossroads (see figure 4.9.b).

3. It is important for the environment's representation consistency that crosswalk accesses remains attached to their associated crosswalks. For this reason, *crosswalk access* tiles are merged with their adjacent *crossroad* areas. Moreover, in order to improve the geometrical consistence of crossroads, *pedestrian* tiles are added to their adjacent crossroad areas if it improves these areas' convexity (see figure 4.9.c).

4. In order to generate areas with the simplest border shapes, a first step generates "street slices" by merging *road* and *crosswalk* tiles that do not already belong to a crossroad area to their adjacent *pedestrian* or *building access* tiles (see figure 4.9.d).

5. Finally, these street slices are either merged together into street areas or merged to their adjacent crossroad areas. For the same reasons as in step 2, slices containing crosswalk are preferentially merged to crossroad areas. The remaining *pedestrian* and *building access* tiles are merge into near-convex zones that become the *pedestrian* areas (see figure 4.9.e).

All these steps aim at generating the most geometrically-consistent high-level zones with the simplest borders possible. Crossroads are well defined and include their associated crosswalks, enabling the agent to delay its crossing decision until the crossroad is reached. The fact that we use the navigation tiles decomposition as a basis for this process ensure that the city areas limits do not cut a meaningful zones such as crosswalks or a building accesses in two parts.

1.3.3 Street sections level generation

When taking local path planning decisions, questions arise such as "Is it better to cross the road now or will I have a better opportunity in the next street?". To answer such a question,

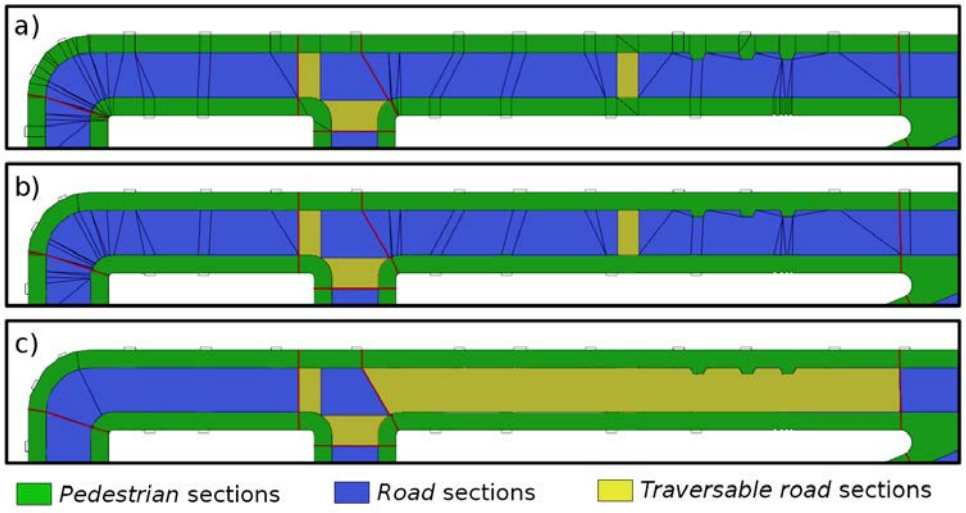


Figure 4.10 – The three steps of street sections generation.

it is required to possess knowledge on future crossing opportunities along the high-level path. The agent can obtain this knowledge by considering a rough representation of the city as a network of sidewalks separated by road section, being crossable or not. This representation is provided by the *street* sections level (Cf. figure 4.6.c). This level partitions the environment into *pedestrian* sections, *road* sections and *traversable road* sections (which include at least one crosswalk). For each area identified at the city area level, its compounding tiles are regrouped as follow:

1. The Street section level is initialised as the tile decomposition, but keeping track of the city areas borders (see figure 4.10.a).
2. All *pedestrian*, *building access* and *crosswalk* access tiles are merged into near-convex *pedestrian* sections if not separated by an area border (see figure 4.10.b).
3. All *road* and *crosswalk* tiles adjacent to the same *pedestrian* sections are merged into a *traversable road* section if a *crosswalk* tile is included or into a *road* section otherwise (see figure 4.10.c).

This level refines the city areas section by partitioning them depending on the nature of navigation zones. The generated decomposition provides knowledge on the presence of pedestrian zones in streets and on the crossing opportunities between them. This information is useful as it enables the evaluation of the long-term impact of the agent's local decisions.

The proposed partitioning process generates three semantically consistent hierarchical levels of decomposition of the outdoor navigable zones. The third level regroups the *street* sections into city areas (streets, crossroads, pedestrian areas...). It enables the selection of a set of streets and crossroads to travel to reach the agent's goal. The choice of where to cross a street or of a side of the road on which the pedestrian should walk is not made. This selects which streets the agents should travel while delaying more precise decisions. Those decisions can be taken during navigation, when relevant information is perceived. The second level regroups the tiles into street sections, considering pedestrian zones and the crossing opportunities between them. At this level, a path identifies the street sections and crossings the pedestrian should travel. As the sidewalk tiles are merged into near-convex zones, the agent is able to plan coarse

paths that do not consider punctual obstacles. Indeed, at a higher level of decision, there is no need to decide on which side to bypass an obstacle: this decision can be delayed to the moment the agent perceives the obstacle. Furthermore, this level provides knowledge on the future crossing possibilities. The agent can use this knowledge to decide whether it should cross the street in the current area or if it should delay this decision to one of the next areas along its path. This enables more freedom during navigation: the pedestrian can choose on which sidewalk he should navigate and where he should cross a street. He can also anticipate the lack of crosswalk and choose to use one earlier along its path. The first level subdivides the environment into navigation tiles. At this level, a precise path can be computed with consideration to meaningful elements like the access to building or to crosswalks.

1.4 Hierarchical partition of buildings

Buildings have their own specific structure. In order to properly represent this structure, we propose a hierarchical partitioning process adapted to buildings. It relies on commonly accepted architectural concepts such as rooms, doors, floors or stairs. This method decomposes buildings into three intermediate levels of abstraction and a navigation tiles level. The third intermediate level of abstraction distinguishes the exterior, covered exterior and indoor sections of the building. The second intermediate level of abstraction identifies floors separated by stairs in indoor sections. The first intermediate level of abstraction identifies rooms in each floor. The navigation tiles level of abstraction identifies near-convex semantically homogeneous zones. We illustrate the building partitioning process in figure 4.11, using the example house presented in figure 4.4. The partitioning process is performed in four steps:

1. Exterior *building sections*, such as gardens or courtyards for example, are often associated with buildings. These sections are identified by the fact they are not covered. Therefore, the first step of our building partition process identifies covered and uncovered sections of the building (see figure 4.11.a). This step uses the information provided by the prismatic spatial subdivision to tag cells as *covered* or *uncovered*. Based on this process, two sets of building sections are extracted: the set of covered sections and the set of uncovered sections. This information is useful to the partitioning process and can impact the agents' behaviour. For example, an agent may avoid getting out when traveling between two points of a building, even if it means traveling a longer distance.

2. Building sections are decomposed into floors separated by stairs. This decomposition makes sense, as people usually try to reach the right floor before considering their goal in this floor. Floors and stair steps are identified by cutting building sections at any *step* edge. Zones which borders are mainly composed of *step* edges are labelled as *steps*. The figure 4.11.b depicts the result of this process on our example environment.

3. Covered zone identified as a *floor* are assumed to be interior zones belonging to a construction (house, block of flat, public building...). Such building floors are usually composed of rooms separated by doorsteps. To achieve room decomposition, identifying doorsteps is required. Our method first decomposes each covered or uncovered regions into navigation tiles: near-convex zones separated by bottleneck (See figure 4.11. C). Door being bottlenecks by nature, the obtained zones are either doorsteps or parts of rooms. To identify which of these zones are doorsteps, we define a "door likelihood" function that is computed for each extracted navigation tile. Let $S(nt)$ be the surface of navigation the tile nt, $H(nt)$ be the average ceiling height of nt, $N(nt)$ be the set of neighbouring tiles, B(nt) be the set of *free edges* belonging to the borders of nt and $L(e)$ be the length of edge e. The 'door likelihood' function is computed thanks to three criteria:

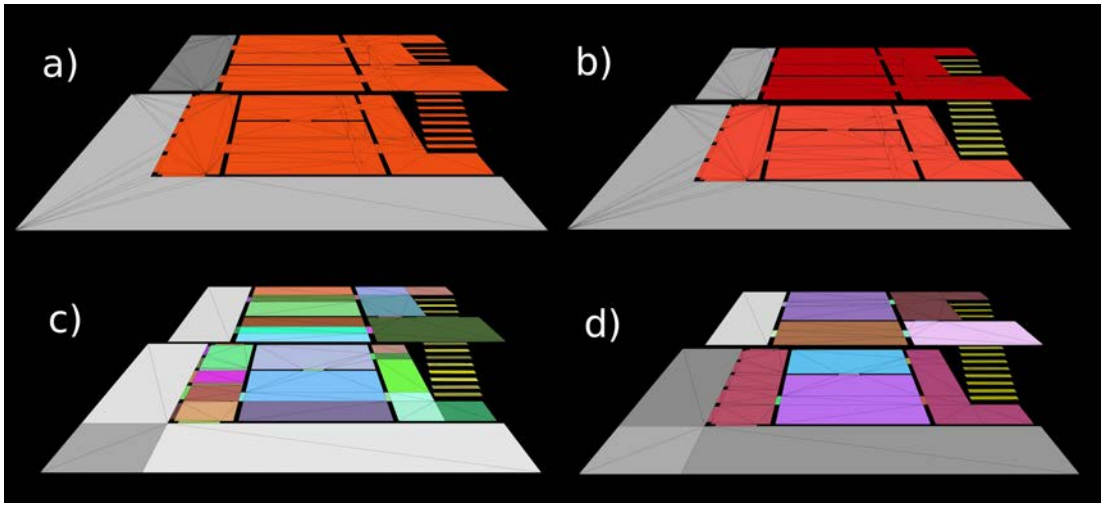


Figure 4.11 – Decomposition steps: a) buildings decomposition, b) floors decomposition, c) convex cells decomposition d) rooms decomposition

- $C_1(nt) = \frac{\sum_{nt' \in N(nt)}(S(nt'))}{S(nt)}$. A door is a small zone between bigger ones.
- $C_2(nt) = \sum_{nt' \in N(nt)}(\parallel H(nt') - H(nt) \parallel)$. A door's ceiling is often lower than the one of surrounding rooms.
- $C_3(nt) = \frac{1}{\sum_{b \in B(nt)}(L(e))}$. A door is bordered with narrow bottlenecks.

On the basis of those three criteria, the 'door likelihood' function (DL) is defined as follow:

$$DL(nt) = C_1(nt) * (1 + C_2(nt)) * C_3(nt)$$

This function tends to return low values for cells belonging to rooms and high values for cells defining doorsteps. To separate doors and rooms, we compute the mean value of the DL function applied to each cell of the topological map. Cells having a DL value greater than the mean value are tagged *door*, other ones are tagged *room*. The figure 4.12 shows the values computed for the two floors of our house example and the resulting environment decomposition is depicted in figure 4.11.d. This method is able to efficiently identify rooms and doorsteps. However, it may have trouble to identify uncommon separations such as archways or really large doors.

4. Finally, the detection of covered exteriors such as archways or covered alley is important for path-planning purpose. Indeed, such zones are covered but they are not considered as buildings. For example, people navigating from a building to another may use covered zones but not enter other buildings on their way. Rooms are retagged *covered exterior* if its borders are mainly composed of *free* or *step* edges and if it is mainly connected to uncovered zones.

Through these steps, multiple typologies of zones are identified at different levels of abstraction. Relying on these extracted zones, we create an informed hierarchical representation of buildings. This hierarchical representation is composed of five levels of abstraction: the city area level, three intermediate abstraction levels and the navigation tiles level. The city area level considers buildings as entities in the city, combining the actual constructions with their associated exterior areas. This representation is useful when considering a coarse path

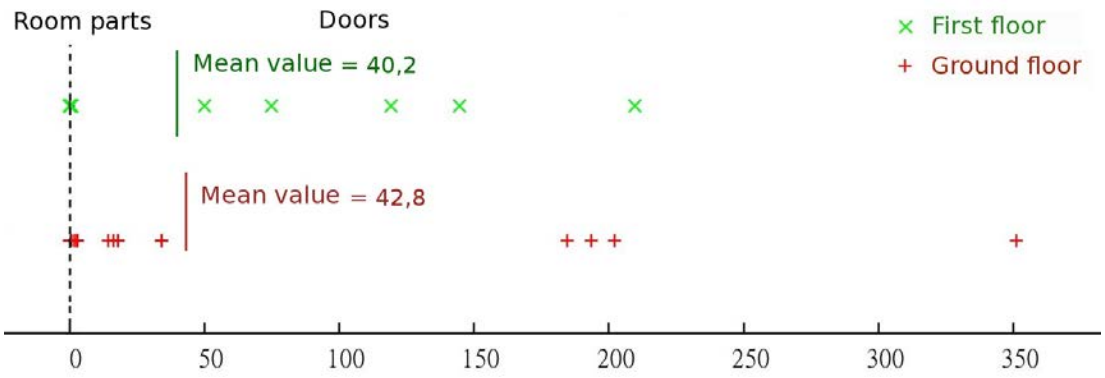


Figure 4.12 – Repartition of door likelihood values for covered convex zones

at the scale of the city: the only goal is to reach the building. The three intermediate level of abstraction respectively decompose the buildings into building sections, being either interior, exterior or covered exteriors, into floors connected by stairs and into rooms separated by doors. These levels of abstraction offer increasingly precise representation of the structure of the building. This enables the agent to consider first to reach the right building section, then the right floor and finally the right room. The navigation tiles level decompose the building into small near-convex zones similar to the one used in the decomposition of exterior zones. These navigation tiles are either doors, parts or rooms, steps, exterior tiles or covered exterior tiles. The characteristics of these tiles make this abstraction level suitable for local planning purpose.

Summary

Our environment abstraction process aims at organising the raw geometry of the environment into a semantically coherent hierarchical representation. The initial geometry is an unorganised set of 3D faces. The faces belonging to the navigable surfaces are labelled with information on the nature of the navigation zones, on the typology of locations and on the access allowance of zones. We use two different processes in order to extract meaningful zones, inspired by city planning and architectural concepts. The hierarchical representations of buildings and outdoors extracted by these partition processes are different, yet coherent. Indeed, the city areas level and navigation tiles level are common to both partitions. Due to the structural difference of building and outdoor environments, intermediate levels of abstraction differ. Yet, these intermediate levels pursue the same goal: guiding the local path planning of the agent by providing more precise information on the structure of the environment. The different levels of abstraction of the environment provided by these representations are adapted to different levels of decision making. The City area level decomposition is suitable for planning coarse path at the city level. Navigation tiles are adapted for local path selection. Intermediate levels of abstraction provide useful information on the structure of city areas, which can be used to guide the local decision making.

2 Hierarchical path planning process

When people plan a path in their environment, they do not consider all the details of the path at once but delay local decisions until suitable local information is available. In the previous section, we presented the generation of a hierarchical representation of urban environments. It

provides the agent with different levels of environment abstraction, each being adapted to a different level of decision making. We propose a hierarchical path planning process that takes advantage of this representation to generate more consistent paths. This process begins with the extraction of a hierarchical path planning graph from the environment partition. Then, graph search methods are used over the different levels of graph hierarchy. First, a path is planned in the higher level path planning graph, from the position of the agent to its goal area. Second, the intermediate levels of the path planning graph are explored in order to compute a long-term estimation of the effort required to reach the goal from intermediate sections. Finally, during navigation, sets of path options are computed inside the city areas, labelled with estimated costs required to reach the goal. This way, the agent is able to make local decisions during navigation relying on this estimation. We also propose adaptation mechanisms reacting to unexpected event by modifying the computed path options. This way, the agent can react to dynamic event happening in its vicinity. Each of these steps takes more details into account than the previous step, but, as they also consider a smaller subset of the environment, their computational cost remains low. In this section, we first explain how the hierarchical path planning graph is extracted. Then, we discuss the three steps of our path planning. Finally, we present the systems we use to adapt the planned path options to unexpected events.

2.1 Hierarchical path planning graph

Path planning methods are graph-search algorithm used to compute a path from an origin position to a goal position. They select a set of edges linking two nodes of the graph while optimising a cost function. These methods thus require the environment to be represented as a path planning graph. Nodes in this graph represent locations in the environment while edges represent opportunities to travel between locations. A cost function is also required to estimate the effort required to travel between locations. It uses many factors, such as the travelled distance or the nature of the environment to estimate a cost for every edge in the graph. This way, an optimal path in the graph is the one that links the origin and goal node while minimizing the combined cost of the used edges.

Our path planning method relies on the hierarchical partition of the environment described in the previous sections. Our environment partition process ensures that any zone belonging to a decomposition level is a union of zones identified in the lower level. We use this property to create a hierarchy of path planning graphs: the navigation tile graph, the intermediate sections graphs and the city areas graph. Note that more intermediate path planning graph levels are generated in buildings, as they are represented through more environment abstraction levels. For each level of the hierarchical partition, a node is created for every free border (i.e. set of common free edges) between two zones. As every zone of a level is the exact union of zones from the lower level, the border between two zones is also the exact union of borders between zones of the lower level. This property is used to hierarchically organise the planning graphs by linking a node in a given graph to the corresponding nodes in the lower level graph. Edges are created between nodes belonging to the same zone. In figure 4.13, we illustrate the path planning graph levels used to represent an example environment. Note that, in figure4.13.b, in buildings, we arbitrarily show the intermediate path-planning graph level associated with building sections. When planning a path between two locations, any of these paths planning level can be used, depending on the desired level of abstraction. In our path-planning method, the city areas level is used for high-level path planning, the intermediate sections level for long-term effort estimation and the navigation tiles level for local path options planning.

Path planning methods require a cost function to estimate the cost of navigating edges of the graph. In order to plan credible paths, we design a cost function providing an estimation of the effort required by humans to travel the corresponding paths. Pedestrians chose their path taking

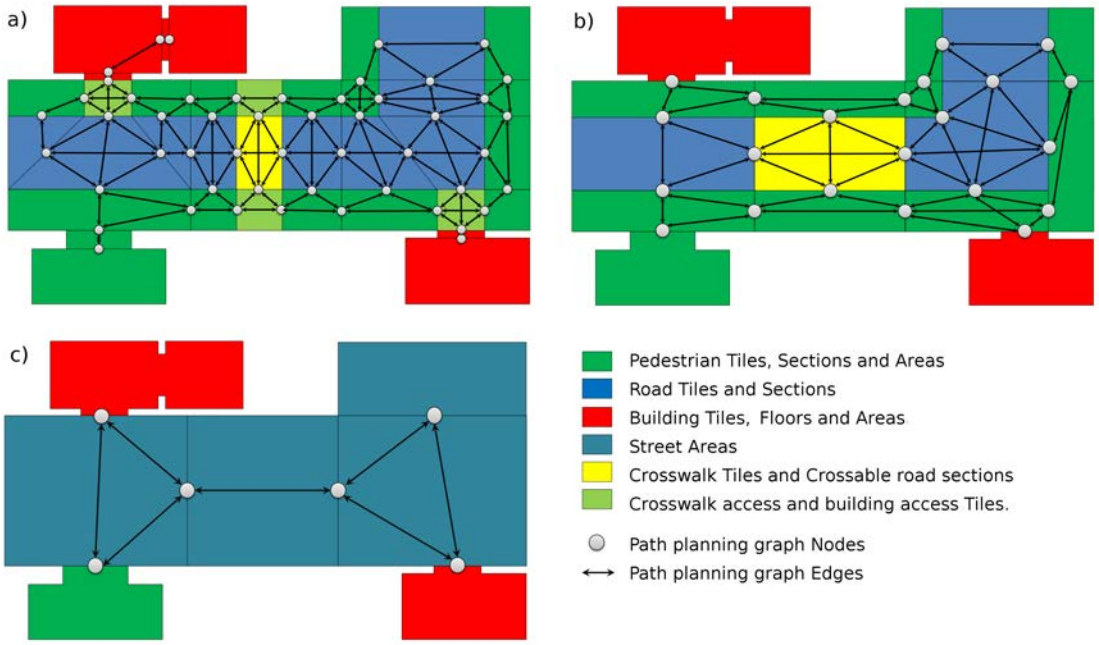


Figure 4.13 – Hierarchical path planning graphs extracted from the hierarchical levels of decomposition: the navigation tiles level (a) the street sections and building sections level (b) and the city areas level (c).

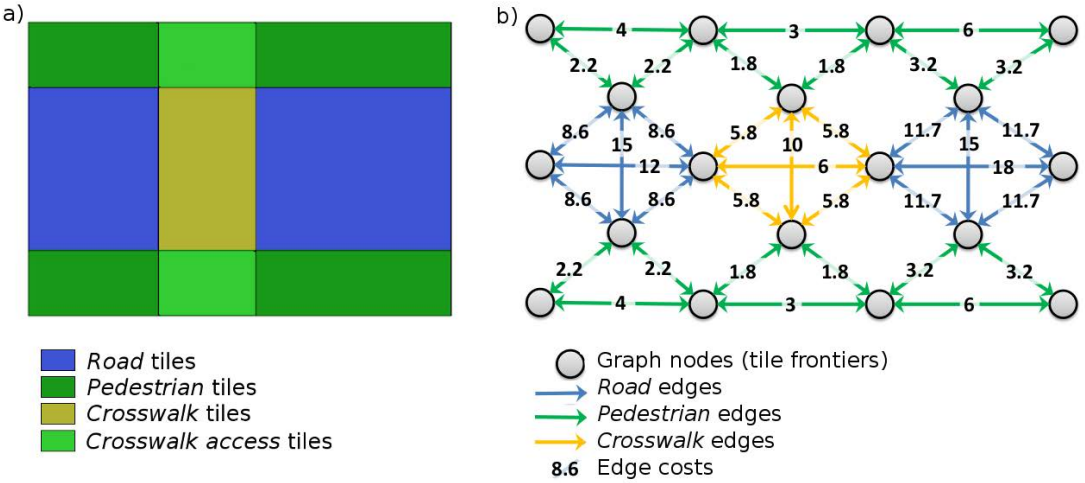


Figure 4.14 – The central street area from figure 4.13) (a) and the cost associated with edges in the associated path planning graph(b).

the distance to travel into account, as well as the nature of navigation zones [Jaklin et al., 2013]. In order to provide access to this knowledge, edges e in the path planning graphs are labelled with their length $e.length$ and with the nature of the crossed zones $e.nature$. In the navigation tiles graph and the intermediate sections graph, these typologies are either *pedestrian*, *road*, *crosswalk*, *interior* or *covered exterior*. In the city areas level, areas are composed of mixed cells typologies. We define high-levels zone natures: *streets* or *pedestrian areas* to be associated with the corresponding edges. Each agent a is given a preference factor $a.nature\_weight(nat)$ over each typology of edge nature nat. The estimated cost $cost(e)$ of travelling an edge is then

computed as follows:

$$cost(e) = e.length \times a.nature\_weight(e.nature)$$

Furthermore, information on the privacy status of zones is also added to the edges: a *private* label is added to edges in private zones, allowing access only to the pedestrians entitled to enter these zones (house owner ...).

2.2 Path planning algorithm

We propose three graph search algorithms, managing different levels of path planning: the high-level path planning, the long-term effort estimation and the local path options planning. The two firsts of them are executed before the agents leaves its initial position, the third one is executed during navigation, each time the agent is about to reach a new area on its path. These algorithms respectively rely on increasingly precise levels of environment abstraction. However, the more details are taken into account, the smaller is the considered zone. For this reason, the computational cost of these algorithms remains low.

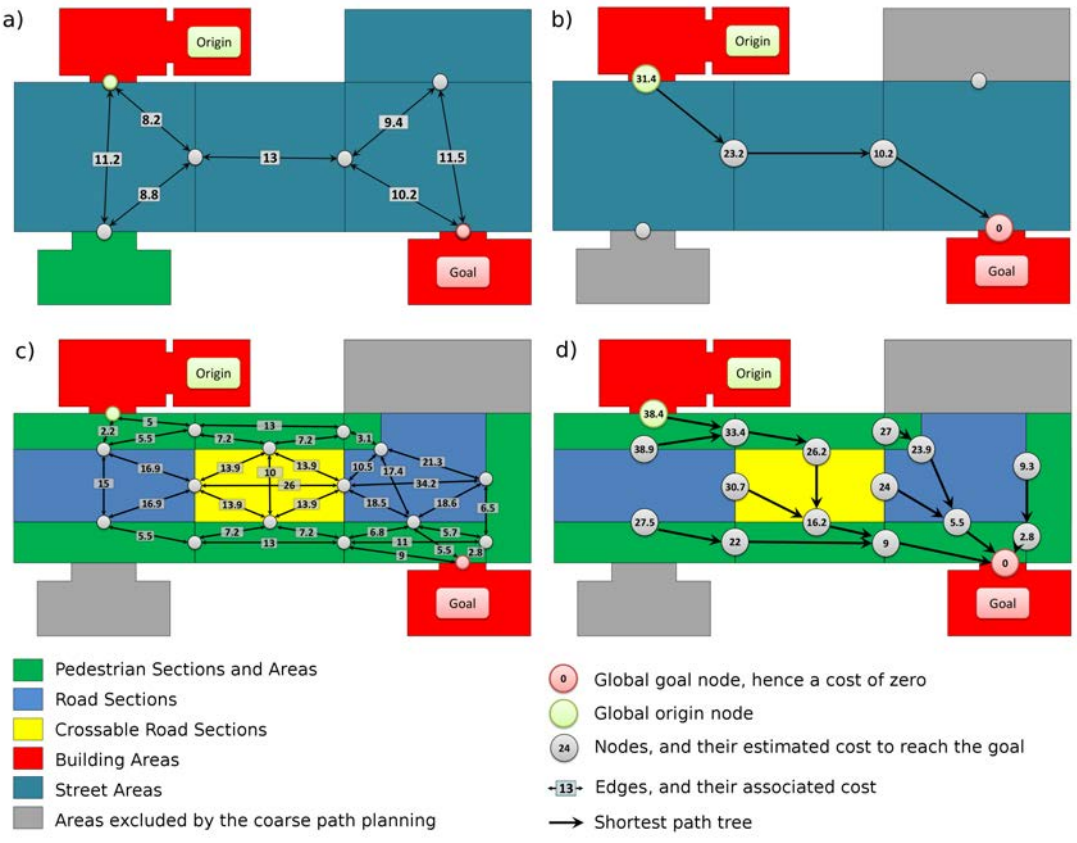


Figure 4.15 – The steps of our high-level path-planning algorithm (a and b) and long-term effort estimation (c and d).

High-level path planning In our model, the agent first considers a coarse path through the city. The high-level path planning selects a sequence of city areas that should be travelled

in order to reach the area containing the final goal. This is achieved by planning an optimal path in the city areas graph (see figure 4.15.a). This path is computed with an A* algorithm [Hart et al., 1968] using the Euclidean distance to the goal as a heuristic. At this step, the information on private areas is used if available. This ensures that an agent will not cross its neighbours' place or a private square as a shortcut. This high-level path planning aims at reduces the global complexity of the path planning process, as it limit the number of zones considered by the lower-level path planning algorithms (see figure 4.15.b.).

Long-term effort estimation When making local path-planning choices, some knowledge concerning the remaining travel is useful. For example, the future crossing opportunities along the path are taken into consideration when deciding whether to cross a street or not (one is more likely to avoid crossing a busy street if he knows there is a crosswalk in the next street on its path). In order to provide the agent with such knowledge, we compute long-term estimations of the remaining effort required to reach the goal along the computed high-level path. This long-term estimation algorithm relies on the intermediate levels of environment abstraction, as they offer a more precise representation of the connectivity of pedestrian navigation zones, being either sidewalks and crossing opportunities in streets or a hierarchy of building sections, floors and rooms in buildings. In building, this algorithm first explore the building sections level, a more precise effort estimation being computed through floors and rooms when the agent respectively reaches the right building section/floor.

In order to limit the complexity of the search in this more precise level of abstraction, the long-term effort estimation process only considers the city areas selected by the high-level path planning (see figure 4.15.b). The street sections and building sections path planning graph associated with the selected city areas are extracted (see figure 4.15.c). A Dijkstra algorithm is executed from the goal node. It computes the shortest path tree leading to this node, estimating for each node the minimal cost required to reach the goal (see figure 4.15.d). The obtained graph provides a set of local goals usable when planning a low-level path inside city areas. Each of these local goals is labelled with the estimated minimal cost required to reach the global goal from this local goal. This knowledge is used by the low level path planning to evaluate the long-term impact of a detour or to better choose a location where a street could be crossed for example. As this algorithm only considers the city sections selected by the high-level path planning and their decomposition into intermediate level sections, the cost of the Dijkstra algorithm remains low.

Local path options planning When navigating in a city, people delay some decisions. For example, a pedestrian will not decide his exact path through a street long before reaching this street. Instead, people usually wait until they perceive the details of the crossed areas to take more locals decisions (for example, crossing opportunities). Some decisions are even more delayed, such as the choice on which side to bypass a city light, which is made only when reaching it, depending on other pedestrians and obstacles. Classical path planning techniques tend to select a unique path that should be followed by the pedestrian. This can lead to situations in which multiple pedestrians struggle to pass on the same side of a pole while none pass on the other side. To avoid such issues and allow the agent to take a detour, we use an algorithm that identifies a set of path options inside an area, linking a local origin to one or multiple local goals.

In order to let the agent easily change its path depending on the dynamic events that take place in cities, we provide it with a set of options on how to reach the next city area along its path. These path options are planned by using the navigation tiles path planning graph and take the long-term effort estimation into account. This local path options planning process, depicted in figure 4.16, is performed in three steps:

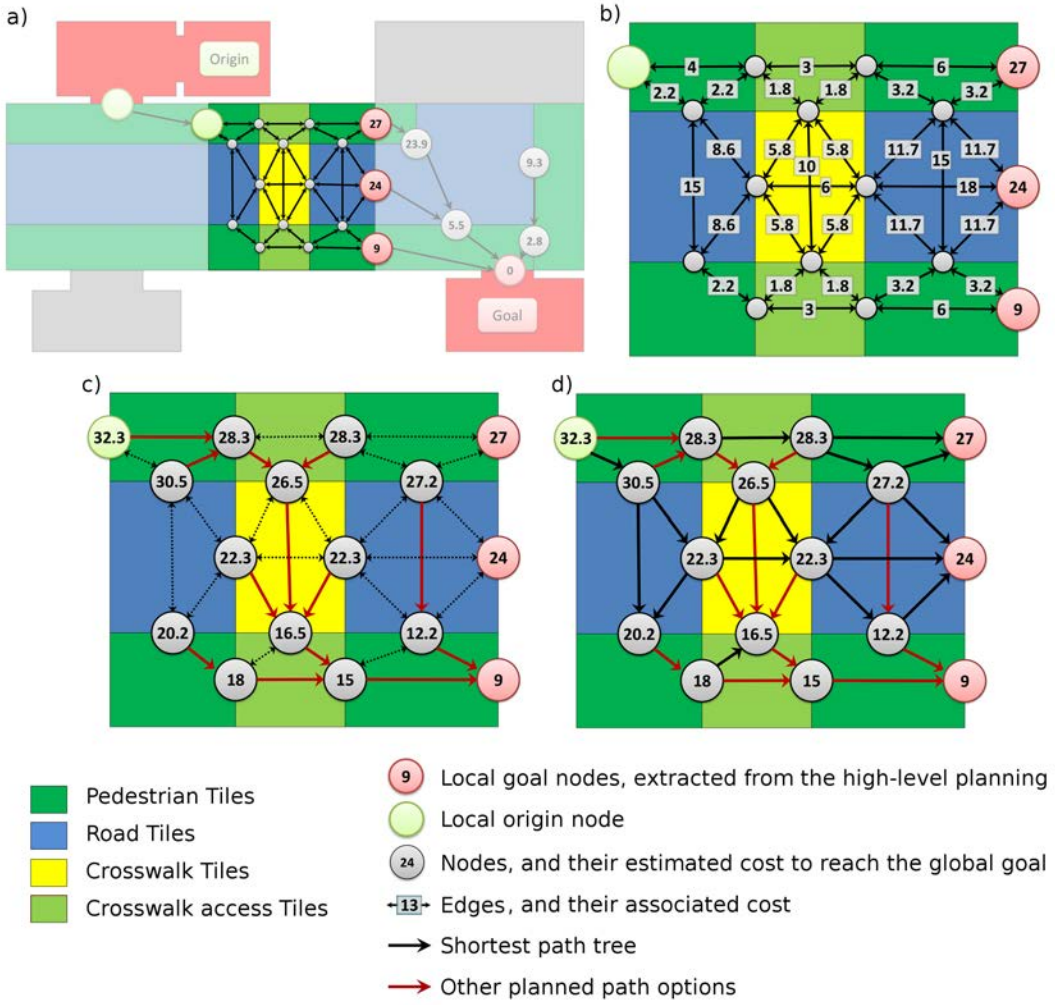


Figure 4.16 – The steps of our low-level path option planning algorithm.

1. When the agent is about to reach the border of a city area, the navigation tiles graph associated with this area is extracted (see figure 4.16.a). The node associated with the tile border the agent is about to reach is selected as the local origin node of the local path-options planning. The tiles borders that belong to the area border the agent needs to cross to follow its path are selected as local goals. The estimated cost associated with these local goals by the long-term effort estimation is retrieved (see figure 4.16.b). These costs indicate which border is preferable to cross to exit the area in order to avoid a future costly path.

2. A Dijkstra algorithm initialised with multiple goals is executed, computing the shortest path tree to any of the local goals (see figure 4.16.c). This algorithm estimates the minimum effort required to reach the global goal from any node of the area. The edges belonging to the computed minimal path tree are oriented toward the goal.

3. Undirected edges that link two branches of the shortest path tree are oriented in order to maximize the number of proposed detours while avoiding the creation of cycles or dead-ends. (see figure 4.16.d).

This process produces a path options graph in which all nodes are labelled with the cost associated with the estimated minimum cost path reaching one of the goals. This path is a subset of the original graph with no cycle or dead end. It ensures that using any of the edges exiting a node leads toward one of the goals. The resulting graph identifies the optimal path as well as possible detours. It also enables to estimate the impact of a detour on the final cost of the path. Finally, as this algorithm only considers the navigation tiles inside an area, the computation of the local path options remains efficient.

2.3 Path options selection and adaptation

Each time an agent enters a city area along its path, it is offered a set of local path options leading to the next area it should reach, with associated long-term impact estimations. The agent must then choose which of these local path options to follow, depending on the dynamic events it perceives along these paths.

2.3.1 Path options selection

Many local decisions such as dynamic obstacles avoidance for example, are taken during navigation. Reactive navigation systems deal with this kind of decisions, following the planned path while avoiding dynamic obstacles. However, these systems do not question the path that was planned in the first place. Yet, sometimes, people decide to change their whole path after encountering a local obstacle (for example, changing from side of a street to avoid a puddle of water). This kind of decision is in the domain of rational reasoning as it requires more look-ahead capabilities than reactive navigation systems offer.

In our system, every time the agent reaches a navigation tile border, multiple path options are offered to him, leading to other borders of this tile. The agent is provided with information of the estimated remaining cost to reach the goal from these tile borders. An estimation of the cost of traveling these path options is also provided by the navigation tiles path planning graph. If nothing unusual is detected concerning these path options, the agent carries on following the optimal path (see figure 4.17.a). If a local event is detected on one of the paths, for example a puddle of water or an incoming pedestrian, the agent increases the cost to the associated edge. Either this edge remains the estimated best option, and the agent continues following the optimal path, or another path option is evaluated as less costly, and the agent will change its course (see figure 4.17.b). This process enables the agent to take local rational decisions that makes him change a part of its path without having to launch another path planning. Behaviours are generated such as agents making detours to avoid static or dynamic obstacles on their paths. This local decision computation is performed regularly along the agent's path. However, its impact on the method's performances is low, as it only considers the path options associated with a single navigation tile.

2.3.2 Adaptation to unexpected events

Cities are by nature dynamic environments involving numbers of unexpected events impeding pedestrians' navigation, such as cars or groups of people obstructing a path. Pedestrians take these events into account when navigating and sometimes decide to change the path they intended to follow. As explained in the previous section, our method offers multiple path options to the agent, allowing it to react to obstacles placed on its immediate path. However, it does not take into account dynamic events happening further away on the path. For example, if an obstacle is placed on the optimal path of the agents few tiles away from its position, the path selection mechanism will only detect it when the tile containing this obstacle is reached. Yet, this kind of unexpected event has to be considered when making local decisions. However,

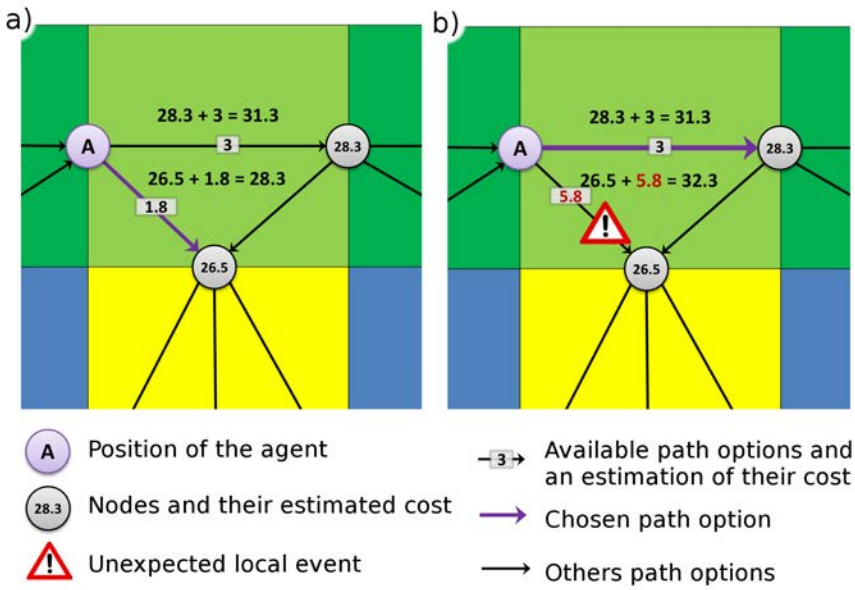


Figure 4.17 – Selection of a path option with (b) or without (a) unexpected local event.

replanning the whole path options graph every time such an event is detected is costly. In order to take unexpected events into account while avoiding the cost of a path options planning, our method relies on a process updating the path option graph during navigation. Three kind of unexpected events are considered: partially obstructed navigation tiles, completely obstructed navigation tiles and completely obstructed areas:

1. Some unexpected events sometimes do not completely block a path, but partially obstruct it, complexifying the access to a tile. For example, a crowd of people walking on a sidewalk or an unexpected density of cars on the road can increase the difficulty of crossing the zone. In that case, the edges associated with the partially obstructed tiles have their cost increased (see figure 4.19.a). The portions of the optimal path tree that are affected by this invalidation are re-evaluated. The cost of nodes in these portions is updated. Finally, the optimal path option graph is updated accordingly to these new costs (see figure 4.19.b).

2. When navigating in an area, the agent is able to perceive if an unexpected event completely obstruct one or multiple navigation tiles (for example, a car stopping on a crossroad as shown on figure 4.18.a).In that case, the nodes associated with the obstructed tile are invalidated, as well as the connected edges. The portions of the optimal path tree that are affected by this invalidation are re-evaluated. The cost of nodes in these portions is updated. The optimal path option graph is updated accordingly to these new costs (see figure 4.18.b). If no path is found to the local goals, it means that the area is completely obstructed.

3. When reaching a new area, the agent is able to detect if this area is completely obstructed (by roadworks, for example). This is the only situation requiring a complete replanning of the high-level path. The obstructed area is invalidated and not taken into account by the path replanning process. Given the hierarchical nature of the graph, this replanning is not as costly as a complete low-level path planning.

Using these methods, not only the agent is able to react to local events by choosing between multiple path options, but these path options are updated to reflect the perceived activity in

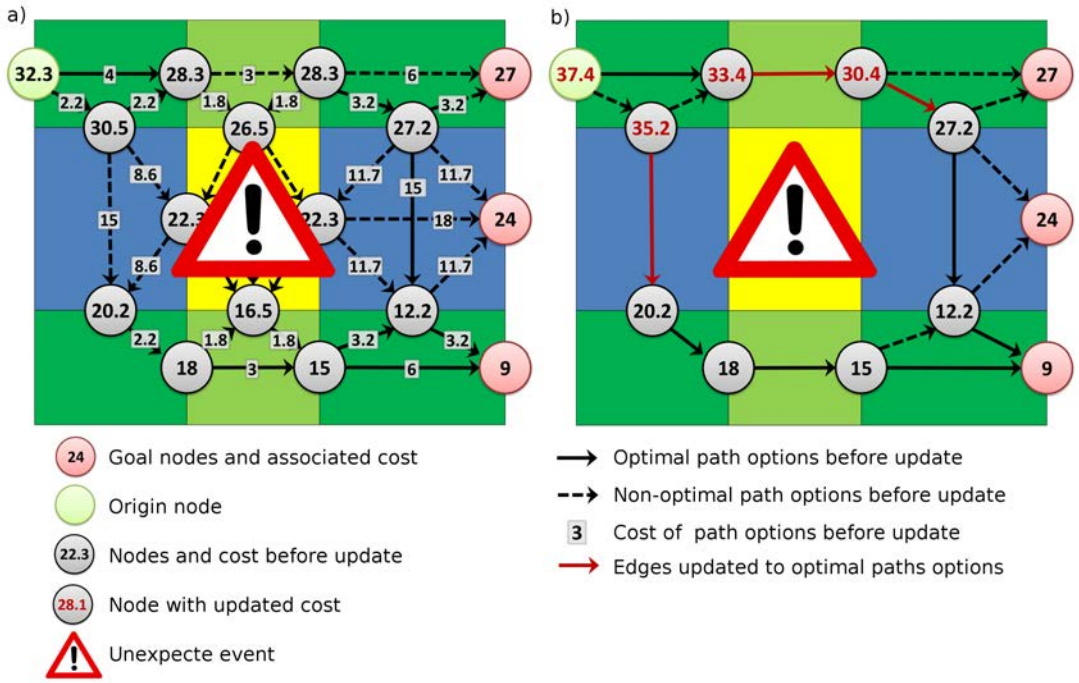


Figure 4.18 – An example of unexpected event completely obstructing a navigation tile (a) and the resulting updated path option graph (b).

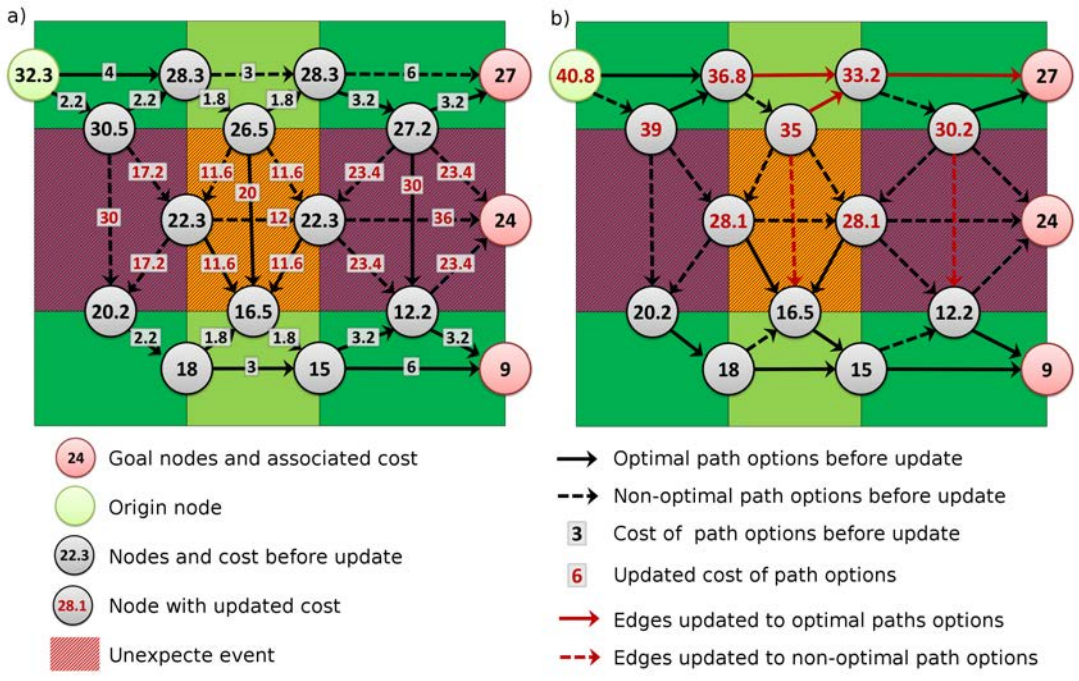


Figure 4.19 – An example of unexpected event partially obstructing a group navigation tiles, increasing their cost (a) and the resulting updated path option graph (b).

Environment abstraction level	Number of extracted zones
Navigation mesh	27 668
Navigation tiles	5300
Street sections	906
City areas	255

Table 4.1 – Number of zones extracted from the navigation mesh for each level of environment abstraction, for environment example shown in Figure 4.20.

the area. With no need of replanning the full path but in the most extreme cases, the agent takes local decision by taking into consideration its perception of this activity. This makes our method adapted to real-time agent path planning applications in dynamic environment such as urban areas.

3 Results

In this chapter, we presented a hierarchical representation of urban environments and a hierarchical path planning processes offering good properties for the simulation of pedestrian behaviours in virtual cities. In order to demonstrate these good properties, we applied our partition processes to example environments. In this section, we present and discuss the obtained results.

3.1 Partition of outdoor urban environments

In order to test the effectiveness of our urban environments partition process, we used a 3D model of a district of the city of Paris. This model was automatically generated using the Open Street Map database: the shape of streets and buildings is consistent with the actual district. We choose this specific district for the diversity of situations it features: small intricate alleys, large streets with central traffic islands, complex crossroads and pedestrian areas. Figure 4.20.a shows the extracted city areas. We see that the city area partitioning process efficiently identify streets and crossroads in this large range of configurations. The use of the near-convexity factor enables the correct identification of streets, even when their shape is not exactly convex. Crossroads are well identified, even when the crossing streets are far from perpendicular or when more than two streets are crossing. Figures 4.20.b shows the street sections partition of parts of the district. The partitioning process correctly identifies sections of sidewalks and crossing possibilities between them, even in complex configurations involving traffic islands and complex sidewalk shapes including punctual obstacles. Figure 4.20.c shows the extracted navigation tiles. Crossroads are well identified, as well as building and crossroad accesses. The remaining navigation surfaces are partitioned in near-convex tiles adapted to local path planning. Using such a complex real urban environment demonstrates the robustness of our method.

In order to estimate the reduction of path planning complexity, we compared, in the district of Paris example, the number of zones extracted at each abstraction level to the number of cells in the initial navigation mesh (see Table 4.1). We see that from 27 668 cells, our process extracts 255 city areas, thus reducing the complexity of a path-planning through the whole city by a factor 100. In these city areas, we see that our process extracts 906 street sections and 5300 navigation tiles. This means that there is a mean of around 21 navigation tiles in each street sections. Therefore, we can expect the complexity of the local path-planning process to stay low.

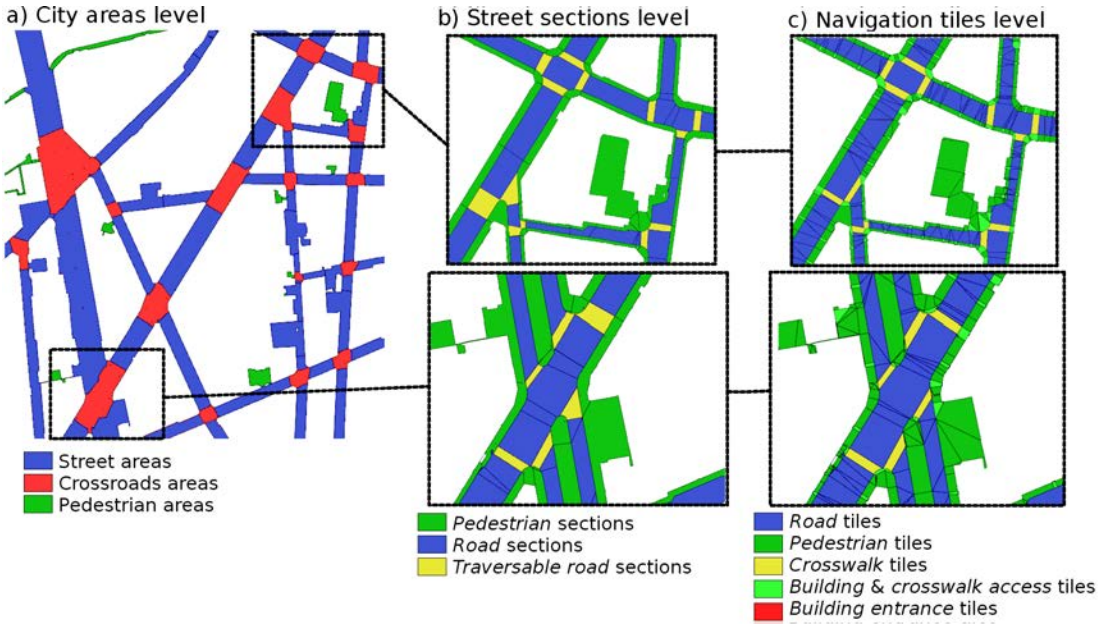


Figure 4.20 – Hierarchical decomposition of a real city map.

3.2 Partition of indoor environments

We tested the effectiveness of our building partition process on a complex example depicted Fig. 4.21.a. The environment contains a church and two houses. The church has heterogeneous rooms size and doorstep dimensions. It also contains pillars in the body of the church and unusual stairs leading to the pulpit. The building on the right of the figure contains a long and narrow corridor exhibiting numerous bottlenecks. The house on the left contains doorsteps of different width and height as well as a step roof and obstacles in some rooms. Figure4.21.b shows that covered sections are correctly identified and that the church body level and the pulpit level are distinguished, as well as the stairs between these levels. Figure 4.21.c shows the partition of buildings into rooms. We see that rooms are well identified, even the ones with complex structures due to furniture (in the house) or to numerous bottlenecks (the long corridor). In the church, we see that the side rooms were correctly identified as well, independently of their size or of the size of their doors. The identification of the church's body as a single room is not impaired by the presence of columns. This demonstrates the robustness of our partition process which has been able to identify all relevant information despite potential interferences induced by obstacles, pillars or irregular doorsteps width and height.

3.3 Path planning

In order to demonstrate the good properties of our path planning method in term of adaptation to unexpected event, we designed an example scenario. In this scenario, an agent has to travel the urban environment depicted figure 4.5. Multiple unexpected obstacles force this agent to modify its path during navigation. Figure 4.22 shows how the agent reacts to four of these unexpected events.

The figure 4.22.a.1 shows the initially selected city areas and the optimal path tree computed by the long-term effort estimation. When reaching the first crossroad, the agent perceives that the next street is obstructed by roadworks (see figure 4.22.a.2). In that case, all the path

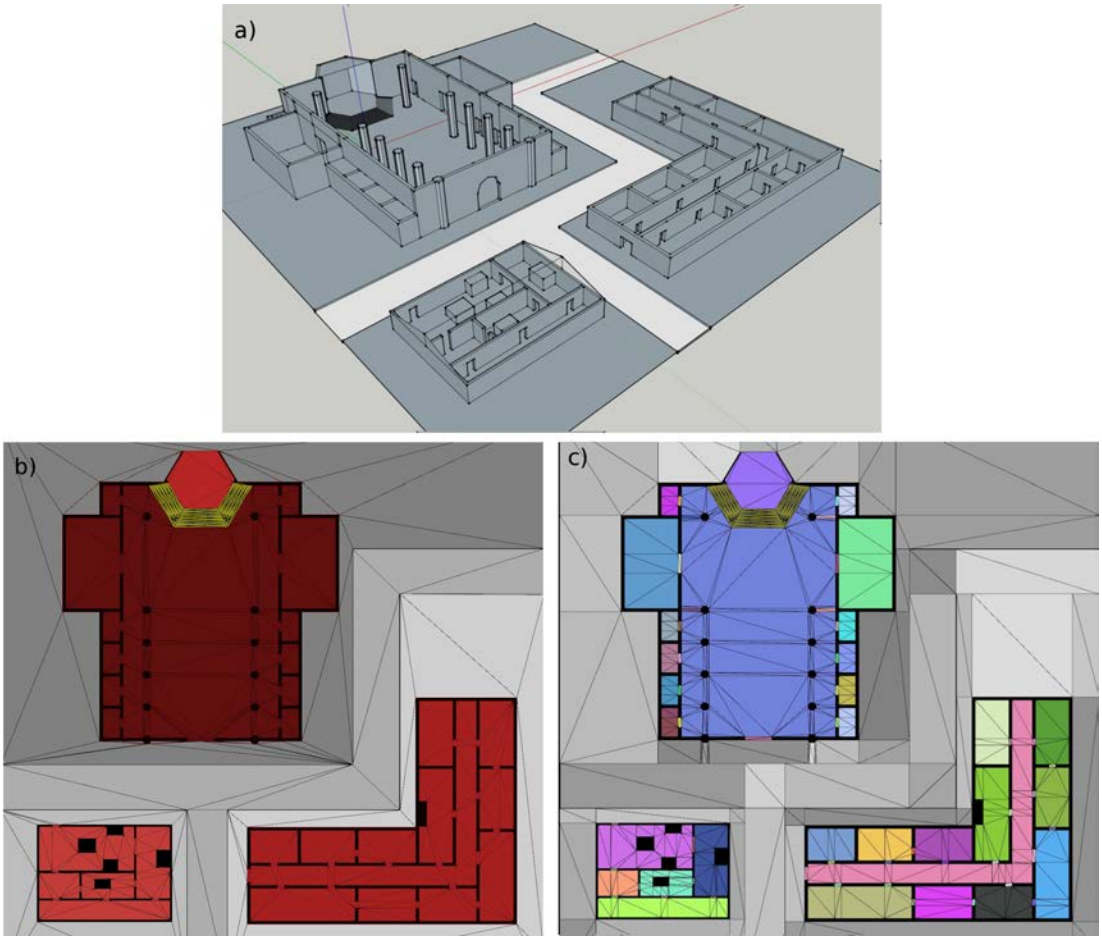


Figure 4.21 – a) Building with various and complex structures (roofs masked for better visibility) b) Its computed floor decomposition c) its computed room decomposition

options being invalidated, a new high level path is computed, as well as new a long-term effort estimation (see figure 4.22.a.3). As a result, the agent totally changes its path through the city.

After a short walk, the agent reaches a street featuring city lights in the middle of the sidewalk. When reaching the node prior to the middle city light, the agent is offered a choice between three path options, one passing by the left side of the city light, one passing by its right side, and one leading to the border of the road. The first on these path options is the one with the lowest estimated long-term effort (see figure 4.22.b.1). However, the pedestrian perceives another agent using the same path (see figure 4.22.b.2). This increases the cost associated with this path and the second best option, passing by the right side of the city light is selected (see figure 4.22.b.3). This way, the agent automatically selects a local path that minimizes its long-term effort, taking into account the dynamic obstacles in its vicinity.

A bit further on its way, the agent reaches an empty street. In this street, the optimal local path reaches the lowest-cost local goal by staying on the same sidewalk (see figure 4.22.c.1. However, a car parks on this sidewalk, invalidating this path option (see figure 4.22.c.2). This results in the selection of a new optimal path that crosses the street in order to reach the other sidewalk (see figure 4.22.c.3). With this path adaptation, the agent now navigates toward a local goal with a higher cost. The agent could have chosen to cross the street a second time in order to reach the lower cost local goal as initially planned. However, the long-term effort

estimation indicates to the agent that a crossing opportunity exists in the next area, making the option of returning on the initial sidewalk sub-optimal. This illustrates how the agent uses the long-term estimation to guide its local decision.

When reaching the crosswalk, the agent computes that the most optimal path option is the one going through the crosswalk on its right (see figure 4.22.d.1). However, a group of people exit a house on the opposite sidewalk and start chatting (see figure 4.22.d.2). Even if these people do not completely obstruct the sidewalk, the agent estimate that traveling the tile they stand on requires a higher effort as it implies passing through the middle of the small group. The supplementary effort is propagated through the path options graph. As a result, the optimal path becomes the one going through the crossroads on the left side of the agent, as it avoids the small group of chatters (see figure 4.22.d.3). This shows how the agent is able to update the local path option graph without replanning all of it, seamlessly selecting a path that minimizes the required effort.

These examples demonstrate the good properties of our path planning process in term of adaptation to unexpected events. These good properties result from the fact that this process takes advantage of the hierarchical representation of the environment to delay local decision making and compute multiple path options. Thanks to these properties, the agent is able to seamlessly react to unexpected events with a minimum of replanning. This enables the production of more credible pedestrian paths through urban environments, while greatly reducing the path planning computation cost.

Conclusion

In this chapter, we presented a decomposition process which generates a semantically consistent hierarchical decomposition of an urban environment. It relies on two partition processes, focussing respectively on outdoor urban environments and buildings. The outdoor environment partition process identifies urban entities such as streets, crossroads, sidewalks or buildings, for example. The building partition process identifies architectural concepts such as covered exteriors, floors, stairs and rooms, for example. A hierarchical path planning graph is extracted from these partitions. Each level of this path planning graph is adapted to a different level of decision making. We also proposed a hierarchical path planning process that takes advantage of the good properties of the hierarchical decomposition. Unlike most of existing hierarchical path planning methods that focus on reducing path planning complexity, our method focuses on generating smarter navigation behaviours. Local decisions are delayed until relevant information is available and path options are planned through the environment. This enables an efficient adaptation to a wide range of unexpected events without requiring a full path replanning. Finally, in our path planning process, the more precise representation of the environment is used, the smaller is the considered area. This way, the computational cost of the different steps of the process remains low, even in large environments. These properties make this method a good solution for planning consistent paths in large dynamic environments.

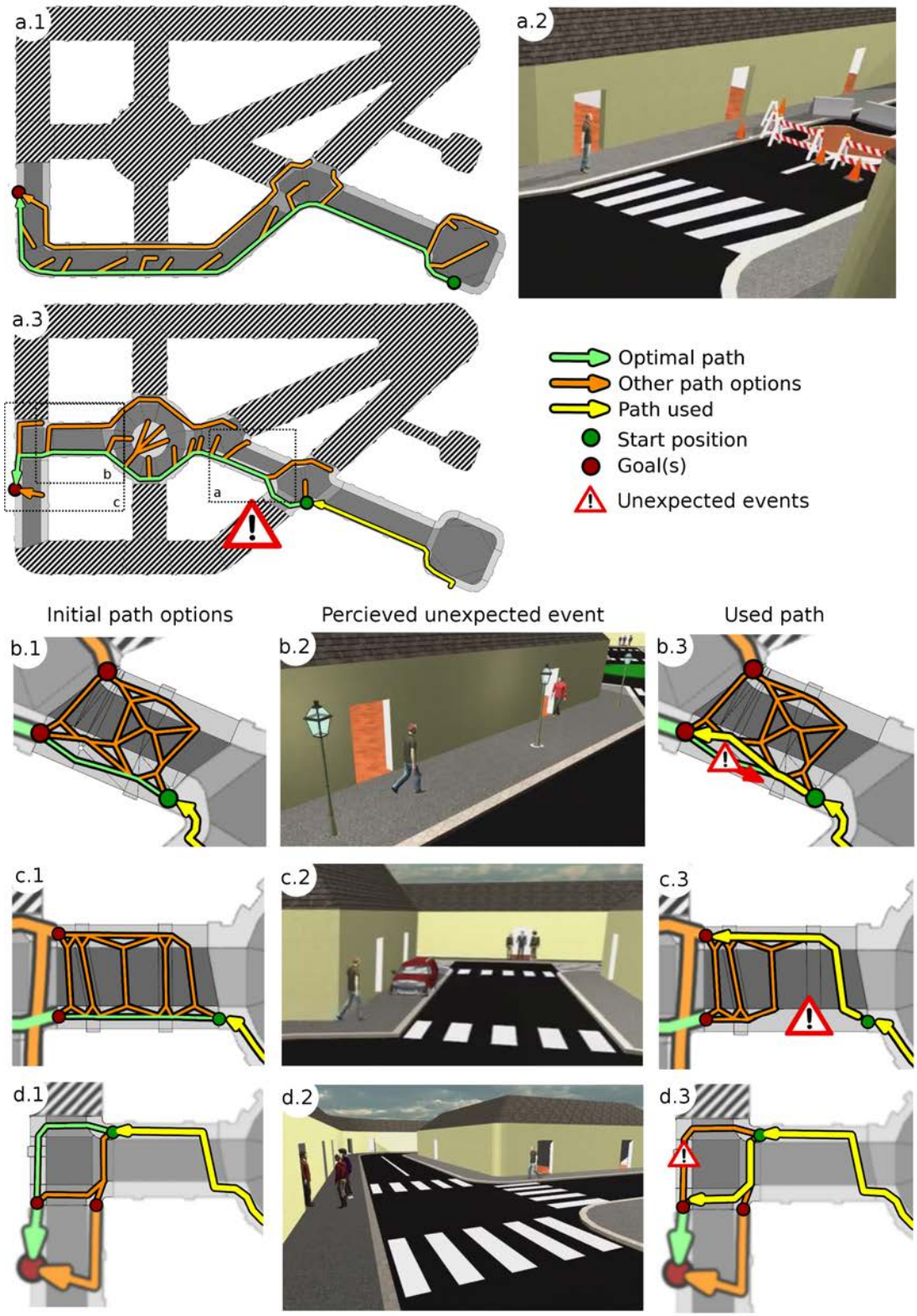


Figure 4.22 – Demonstration of our path planning ability to react to unexpected events (for clarity reasons, only a representative sample of the computed path option is displayed)

Activity scheduling under temporal and spatial constraint

5

When observing crowds in cities, global phenomena are noticeable at specific locations and times. For example, observable increases of flows densities emerge in front of schools around their end time and shopping streets get more crowded on Saturday afternoons than on Sunday mornings. These phenomena emerge from the sum of all pedestrians performing their daily activities. We believe that endowing agents with representative individual activity schedules enables the emergence of more credible pedestrian flows in virtual cities. By "representative", we mean that the produced activity schedules are statistically consistent with the ones humans would choose in the same situation. Such consistent activity schedules are complex to produce, as people rely on multiple interacting parameters when scheduling activities: the structure of their environment, spatial and temporal constraints associated with their activity and personal preferences. Indeed, when scheduling their daily activity, people take the configuration of their environment into account in order to choose a route that tends to reduce the navigation distance and energy consumption while maximizing its utility and preference [Kitazawa, 2004, Hoogendoorn and Bovy, 2004]. It implies that people do not simply go from nearest to nearest locations but tend to maximize the long term efficiency of their itinerary. This itinerary is spatially constrained. These spatial constraints can be a location typology (one can go to any bakery) or a specific location (one do not go to any workplace, but the one where he works). People are also subject to strong temporal constraints such as work hours, appointments times or shop closing times. Their activity heavily depends on these temporal constraints. For example, if a person has to take a train among multiple other tasks, his whole activity is affected by this strong constraint. Furthermore, given a similar situation, different people do not behave the same way. This is due to personal characteristics such as navigation speed or preferences over tasks and locations. Classical approaches used in behavioural animation do not strongly focus on the relation between the agent's activity, spatial and temporal constraints applied to this activity, and agent's personal preferences.

We propose a model that has been designed to endow virtual agents with representative long-term activity scheduling capabilities. Given an environment and an intended activity descriptions, this model computes a task sequence compatible with temporal and spatial constraints associated with the activity. Locations where these tasks should be performed are selected as well as a relaxed time interval identifying when they should be performed. The produced task schedule minimizes an effort function that combines navigation speed, distances, waiting times and personal preferences. This output is used to drive a navigation model and aims at generating more coherent pedestrian behaviours. The main benefit of our model is that agents take more consistent decisions as they better handle the fundamental relationship which exists between the environment, the agent and its constraints in activity scheduling. For instance, some non-trivial behaviour such as interlacing daily activities with one or several appointments can be easily described and efficiently carried out by the agents. Our activity scheduling process possesses look-ahead capabilities, enabling the generation of behaviours that

could not be obtained otherwise. For example, behaviours such as shopping before work in order to avoid missing a train in the afternoon or making a detour to drop shopping bags at home in order not to carry them over long distances require considering the long-term impact of decisions. Thanks to these good properties, our model can be used to easily populate a city with crowds of several thousands of agents that individually exhibit representative long-term task scheduling abilities. Our task scheduling model has been validated through an experiment, by comparing the obtained schedules to human-determined ones.

This chapter is organized as follows. We first present the modelling of the inputs of the proposed algorithm, namely the environment description, the agent characteristics and the intended activity description. In a second time, we describe the proposed algorithm that schedules tasks under spatial and temporal constraints. Finally, the result section discusses the interesting properties of our system and describes our validation experiment.

1 The agent, its environment and intended activity

Scheduling activities under spatial and temporal constraints requires a representation of the environment's structure, of the agent and of it activity. In this section, we propose such representations. The environment description depicts its structure, its nature and the opportunities it offers in term of activity realization. The agent's description describes the personal characteristics and preferences that may impact its decision making in term of activity scheduling. The activity description lists the different possible sequences of atomic tasks to perform in order to realise the activity. It expresses the temporal and spatial constraints that may impact the activity scheduling, as well as dependencies between tasks.

1.1 Environment representation

When people schedule their activity, they take into account the spatial organization of their environment. The locations where activities can be performed and the distance between them are considered to decide which of these locations to visit and in which order. Usually, people try to order their tasks in a way that minimizes their effort, as long as it is compatible with their constraints. This means that people do not go from closer to closer locations to perform their tasks, but can, for example, go to a distant area where all these tasks can be performed in clustered locations. People also take the nature of their environment into account when scheduling their activity, for example favouring locations they can reach by going through pedestrian streets and squares and avoiding private areas.

In order to be able to simulate such behaviours, the agent must have access to a representation of the environment indicating the locations where tasks can be performed, the accessibility and distance between these tasks, and the nature of navigation zones. In our model, this information is provided by a topological graph, illustrated in figure 5.1.b. This topological graph relies on the city areas graph, which represents the city as a network of interconnected streets, crossroads, buildings and pedestrian areas (see figure 5.1.a). Indeed, this graph already provides the position of buildings and an approximation of the distance between them, as well as the nature of the navigation zones. Information on the locations where tasks can be performed is added by assigning a unique identifier to areas that share the same functionalities from the simulation point of view. As explained is section 4, automatically identifying the nature and function of locations only through their geometry is almost impossible. This information is provided in the information of the geometry, either labelled by hand or extracted from existing representations (Informed maps such as Open Street Map, for example). A database describes the properties associated with the identifiers of areas: the location type and opening hours. The location type refers to a set of tasks that can be performed at the associated location. For

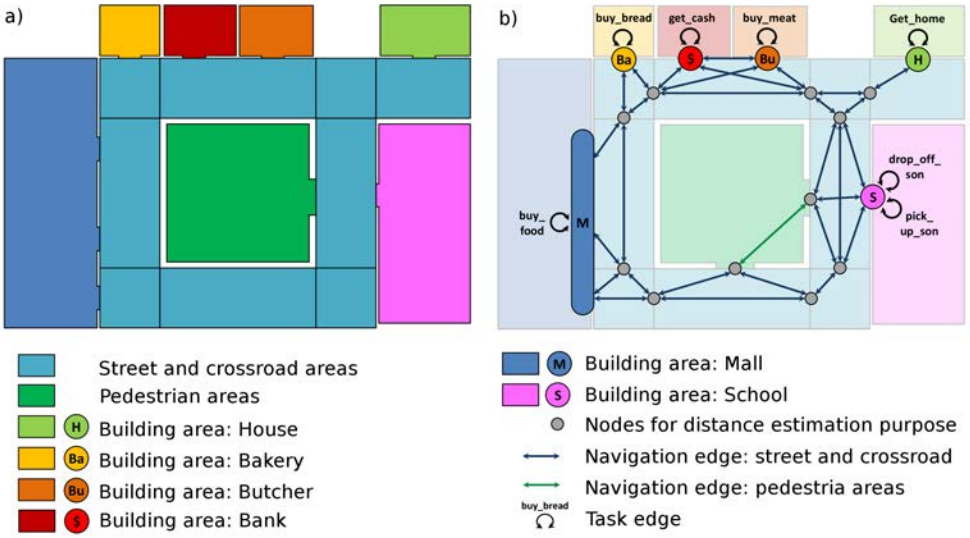


Figure 5.1 – An example city areas decomposition (a) and the extracted topological graph (b) (note that the two oriented navigation edges between each pair of locations are represented as a bidirectional edge).

instance, the location type can be a bakery in which tasks "buy bread" and "buy dessert" can be performed. The opening hours characterize time intervals during which tasks can be performed at the given location.

Important locations in the environment are represented as a set L of nodes l. Two kinds of edges exist in the graph: navigation edges $e \in E_n$ and task edges $e \in E_t$. Navigation edges link an origin location $e.l_o$ to an end location $e.l_e$ if this travel is possible without crossing another location. They are informed with the distance $e.length$ to travel between these locations and the nature of the crossed navigation zone $e.nature$. As most city areas can be travelled both ways, two locations are usually linked with two navigation edges going in opposite directions ($e_1.l_o = e_2.l_e \ \wedge \ e2.l_o = e1.l_e$). Task edges indicate the possibility of performing a task at a given locations. These edges are labelled with the identifier of a task $e.tk$ that can be performed at this location. As most task realisation usually do not involve travelling, these edges usually loop on the considered location ($e.l_o = e.l_e$).

Knowing the minimum remaining distance to travel to reach a location where a task may be performed is important to be able to evaluate the pertinence of this task. In order to provide a simple access to this information, a table $l.min\_dist(tk)$ is generated for each location l, listing the minimum distances to locations where each task tk may be performed. These tables are computed by a process based on the Dijksta algorithm, using the information provided by the edges. Each location can also be constrained by a valid time interval $l.valid\_interval$ to represent the time period tasks can be performed at this location. This is especially useful to represent shops or public building opening hours.

The topological graph aims at combining the characterization of accessibilities and the identification of locations where tasks can be performed. It offers an abstraction of the environment that only contains useful information: nodes model either a route choice or locations where tasks can be performed. It also provides information on the nature of navigation zones. This topological graph is used for two different purposes: locating tasks in the environment and estimating the effort required to travel between locations where tasks can be performed.

1.2 Agent characteristics

Two persons do not necessarily schedule similar activities the same way. For example, some people may prefer shopping in small shops than in malls and some others may prefer to travel more agreeable zones, even if it means taking a detour. This is due to people's personal characteristics and preferences. Taking these personal preferences into account generates variability in the behaviour of people, altering the flows in the crowd [Li and Allbeck, 2011]. Such variability is observed in real crowds. As our model aims at credibly simulate crowd behaviours, we take personal parameters into account when scheduling agent's activity. We associate the following characteristics and preferences to each agent:

- A set P of paces p (walking, hurrying and running for instance). A pace is characterized by a navigation speed $p.speed$ and a effort factor $p.effort$. This factor symbolizes the effort implied by using the corresponding navigation speed. The higher its value, the less likely is the agent to use the pace in a task schedule. Variations in these parameters enable the representation of agents with different navigation capabilities. For example, a child's "running" pace would be characterised by a high speed and low effort factor while an old person's running pace would feature a lower speed and would require an higher effort.

- A set of agent specific locations L_a. The purpose of these locations is to constrain the intended activity description in order to follow some logic related to the agent "life". For instance, an agent is especially associated with one home: the one it lives in. This way, it is possible to restrict some tasks of the agent to its home, and not to any house. These specific locations can also be used to grant access to private locations. For example, an agent is able to access a private workplace if the description states that it works there.

- A weighting coefficient $wait\_weight$. It evaluates if the agent is prone to wait (low value) or on the contrary is prone to do other things instead of waiting, even if it implies that it will have to hurry to be on time (high value). This coefficient enables the generation of agents with different relations to waiting, some preferring to hurry to perform some other tasks instead of waiting.

- Optional weighting coefficients over tasks $task\_weight(tk)$. They represent how much the agents likes to perform the task tk. They enable the generation of agents that prefer shopping in small shops than in malls, for example.

- Optional weighting coefficients over zones' natures $nature\_weight(nat)$. It represents the preference of the agent in term of choice of navigation zones as proposed in [Jaklin et al., 2013]. It enables the generation of agent favouring travels through parks or pedestrian streets, for example.

- An optional weighting coefficients over tasks efforts penalties $penalty\_weight$. It represents how much the agent is affected by an additional effort resulting from the realization of a task. For example, it can differentiate an agent able to carry heavy bag over long distances from another prone to drop them a soon as possible.

A wide variety of agents' archetypes, representing categories of population, can be described with those parameters. The values of these parameters are picked at random in intervals specific to the agent's archetype. For example, older people's range of walking speeds is in average lower to the one of businessmen. Specific locations such as homes and workplaces are randomly picked for each agent. Statistical data can be used to increase the probability of associating some categories of agents to specific locations. This enables to represent the fact that some categories

of people are more prone to inhabit given dwelling areas or to do given jobs, for example. This diversity in agents' descriptions enables the generation of multiple different ways of performing a same activity. This automatically generates diversity in the city population, improving its credibility.

1.3 Modelling agent intended activity

The aim of our model is to credibly schedule the activity an agent intends to achieve. We define an activity as a set of tasks to perform. Dependency relations may exist between those tasks (for example, an agent need to obtain cash before shopping, but can visit the bakery and butcher in any order). Equivalence relations may also exist between tasks or subsets of tasks (for example, an agent can either go to small shops or go to the mall to purchase groceries). We express these constraints through an activity graph describing all the possible ways of achieving the activity.

1.3.1 Tasks description

In our model, tasks are the building blocks of the agent's activity. Each task describes an atomic action that must be performed at a single location. Note that a task is atomic at a given precision level. For example, the task consisting in shopping in a mall is valid at a general city level, but could be divided in more precise tasks if considering the individual shops the mall is composed of. A database links each task tk to a description of its characteristics:

- An estimated duration $tk.duration(t)$ models the common knowledge about the usual time required to perform the task tk depending of the time. For example, the agent is aware that shops are usually more crowded in the afternoon than in the morning, and can take the resulting delay into account when planning his day.
- An effort factor $tk.effort$ expresses the effort spent in performing the task per unit of time. It guides the choice between equivalent tasks. For example, shopping in a small shop could be considered as less tiresome than shopping in a mall.
- An effort penalty $tk.penalty$. The effort penalty acts as a modifier of the effort related to navigating and performing the remaining tasks. For example, if somebody buys groceries, he has to carry some bags, which implies a greater effort during navigation as well as a discomfort when performing other tasks. This penalty can be removed by other tasks. For example, the task consisting of dropping shopping bags at home removes the penalty associated with shopping for groceries. This way, complex behaviours such as taking a detour to drops shopping bags at home before resuming the activity can automatically emerge without being explicitly described.

An example task description is shown in appendix A.3.

1.3.2 Activity description

The agent intended activity is modelled by using a hierarchical description based on the notions of tasks, activities and constructors (operators that describe how tasks and activities can be combined). It focusses on the tasks the agent intends to perform and on the possibly associated constraints (dependencies, locations and times). Activities are combinations of tasks or activities. An activity description aims at describing all possible realization variants in terms of valid tasks sequences. To ease the description of all those variants, we use the following constructors:

- The *either* constructor indicates a choice of one sub-activity in a set. It describes equivalence relations between these activities.

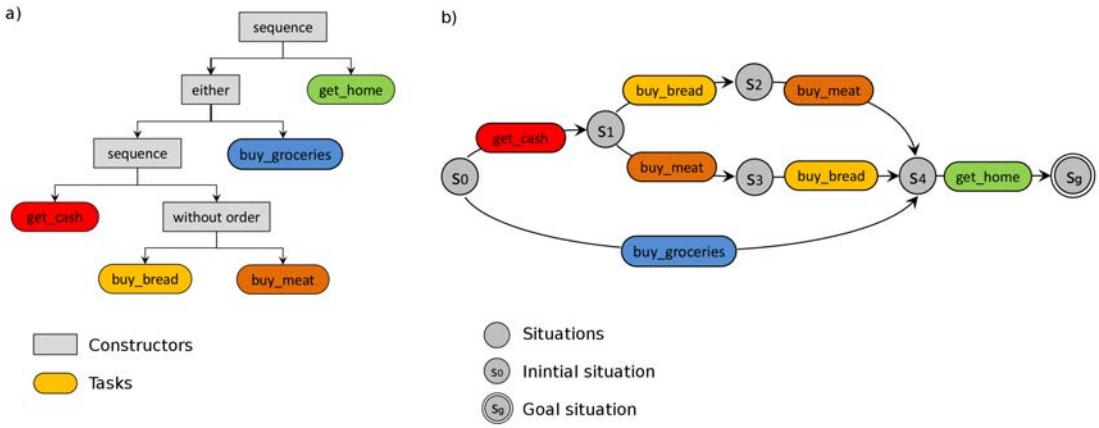


Figure 5.2 – A hierarchical description of an intended activity (a) and the equivalent minimized activity graph (b)

- The *sequence* constructor indicates that the sub-activities should be performed in a specific order. It enables the description dependency relations between tasks.
- The *without order* constructor indicates that the sub-activities should be performed in no specific order: no dependency relation exists between these sub-activities.
- The *interlace* constructor combines activities by interlacing them at the task level. It indicates that the tasks constituting the sub-activities may be performed in any order.

Those constructors provide the user with an expressive tool that can be used to describe very complex activities. For instance, *without order* and *interlace* operators enable to describe an activity that can be achieved in many possible ways. An example of such description is provided in figure 5.2.a. It describes the activity of an agent shopping for groceries on the way home. In this description, the agent can either buy all its groceries in a mall or in several small shops (butcher, bakery). In this last case, the agent must first withdraw some cash at an ATM. When used to describe an agent intended activity, tasks can be constrained with a location and a time interval. For example, the general task consisting of picking up its son a school has to be constrained to the school the child is attending. On the contrary, if no location constraint is provided, the location must be chosen among all appropriate ones. Bread can be bought in any available bakery, for example. The time interval constraint implies that the associated task must start within the given time interval. This can be used to model working hours or appointments for instance. If no temporal constraint is given, the task can start anytime during the opening hours of the chosen location. Temporal constraint can also be applied to an activity. This implies that all tasks compounding this activity must be performed within the associated temporal interval. An example activity description is shown in appendix A.4.

1.3.3 Activity graph

The representation of the agent's intended activity as a tree of activities is intuitive and easy to design by hand. However, this representation suffers from redundancy issues. Indeed, such a representation can be partially redundant if two concurrent activities share sub-activities (for example, the activities consisting of buying bread in a bakery or in a mall may share a task consisting of getting some cash at an ATM). This means that the tree representation in not optimal and that, for a same activity, several equivalent tree representations may exist. It also means that in case of a failure in one of these activity, the failure recovery process will have

trouble to determine that a common part of the concurrent activity may have been already satisfied. In order to provide the activity scheduling process with a more suitable description of the agent's intended activity, the tree representation is compiled into a minimal activity graph, as illustrated in figure 5.2.b. In our model, this activity graph is a state machine which aims at recognizing any sequence of tasks that can be used to perform an intended activity. This graph is composed of as set S of situations s. A situation is the execution state of an activity: it represents the fact that a set of task have been performed and some remain to be performed. For example a situation can express the fact that the agent retrieved cash but did not buy any food yet. Every path from the initial situation s_i to the final situation s_f of this graph is a valid sequence of tasks leading to the completion of the activity. Transitions tr model ways of progressing in the completion of the activity. Each transition carry a task $tr.tk$ which execution enables the transition from an origin situation $tr.s_o$ to an end situation $tr.s_e$. For example, in 5.2.b, the task *buy_groceries* enables the transition from the initial situation of the agent to a situation in which the agent has food and must go back home.

This activity graph is automatically built from the intended activity description. This building process is achieved through two passes: a pre-computation that only depends on the intended activity and a contextualization process that propagates environmental constraints in the activity graph.

1. **Pre-computation.** During the pre-computation pass, the activity graph is built by translating the intended activity description into a state machine. Each constructor used in the intended activity description has its equivalent in terms of state machine construction. For instance, a sequence operator concatenates two state machines; the interlace operator computes a product of several state machines. As the description of the intended activity may not produce an optimal state machine, a minimization algorithm [Hopcroft, 1971] is used. This algorithm guaranties that the computed state machine contains a minimal number of states and transitions linking the entry state to the end state. This also means that activities are properly factorised. This facilitates the recuperation from error, as a unique node correspond to the situation of the agent. A global task penalty $s.global\_penalty$ is also assigned to every situation $s \in S$. This global penalty is computed as the sum of the penalties $tk.penalty$ associated with the tasks in the sequence leading to this situation. This global task penalty represents the supplementary effort necessary to travel and perform tasks given all the tasks the agent performed. For example, moving around requires more effort if carrying shopping bags from multiple shops. Note that if different sequences of nodes lead to the same situation, their individual penalties should sum up to the same global penalty. Indeed, these sequences are considered as equivalent in the activity and thus should induce equal efforts. For example, in 5.2.b, this means that shopping in the mall is considered generating the same penalty as shopping in small shops, as the agent should get as burdened in each case.

2. **Contextualization of the activity graph.** This second pass aims at propagating constraints related to the environment in the activity graph. For each task tk labelling a transition tr , a contextualised valid time interval $tk.contextualised\_interval$ is computed. This new time interval represents all possible starting dates of the task. It is computed by intersecting the valid time interval $tk.valid\_interval$ assigned to tk and the union of the opening hours $l.valid\_interval$ of the locations where the task tk can be performed:

   $$tk.contextualised\_interval = \cup_{l,\forall e \in E_t, e.l_o = l, tk = e.tk}(l.valid\_interval \cap e.tk.valid\_interval)$$

   Finally, for each situation $s \in S$, a maximum admissible time $s.max\_time$ is computed. It describe the time before which this situation must be reached to enable the completion of

the global activity. If this time is already passed, it means that no task sequence using this situation can lead to the completion of the activity. This maximal admissible time is computed by considering the contextualised valid time interval $tr.tk.contextualised\_interval$ of the remaining tasks tk in the sub-graph derived from the situation s. To compute those maximal admissible times, a maximal time equal to the end time of the simulation is associated final situation s_f of the activity graph. Other situations are marked as not constrained. Then this maximal time is propagated backward in the graph, from its final situation s_f to its initial situation s_i. For each non constrained situation s which transitions lead to constrained situations, the associated maximal admissible time is computed. First, for each transition tr where $tr.s_o = s$, a time is computed as the minimum between the upper bound of $tr.tk.contextualised\_interval$ and $tr.s_e.max\_time$. Then, the maximal computed among all these transitions is assigned as $s.max\_time$ and the situation s is marked as constrained.

$$s.max\_time = max_{\forall tr, tr.s_o = s}(min(\overline{tr.tk.contextualised\_interval}, tr.s_e.max\_time))$$

During this contextualisation pass, the process checks if the task is realisable in at least one location in the environment. If not, the task is removed from the graph, as well as the other tasks in *sequence* with it. This removes unnecessary branches in the graph, further reducing its complexity.

Contrary to the initial activity description, the contextualized activity is minimal, with no redundancy. Therefore, two different descriptions of the same activity lead to the same minimal activity graph structure. This implies that our algorithm is not sensitive to the intended activity description but only to the intrinsic nature of this activity. This offers good properties in term of failure management: if a rescheduling is required, it can be performed starting from the current situation in the graph without risk of missing a valid solution. Time intervals associated with the situations of the activity graph characterize the possible feasibility of the remaining activity and take into account the intended activity constraints coupled with environmental constraints. Those intervals will be used to prune the solution search during scheduling.

1.3.4 Summary

These descriptions of the environment, of the agents and of their activities express the main parameters that can affect the realization of the agent's activity. Their spatial and temporal constraints are taken into account, as well as their personal characteristics and preferences. Such descriptions enable the computation of suitable activity scheduling with respect to the existing constraints. It also ensures variability in the activity scheduling: given the same activity graph, several agents with different characteristics may exhibit completely different behaviours. Furthermore, the initial descriptions of the environment, of the agent and of the activity are independent. The environment and activity are linked at runtime using a simple database describing the tasks that can be performed in different locations typologies. Specific agent's spatial constraints are also generated at runtime, linking the agent and environment descriptions. This means that new agents and activities can be seamlessly added to a simulation and that the agents are able to adapt their behaviour to new environments.

2 Activity scheduling algorithm

As discussed in the section's introduction, a statistically consistent activity scheduling process enables the generation of more credible pedestrian behaviours in virtual cities. In order to pro-

duce activity schedules that match the ones produced by people, we propose a task scheduling algorithm that takes into account the tight interaction between the time, the space, activities and personal characteristics. It combines the topology of the environment and the intended activity description in order to compute a sequence of instantiated tasks (tasks with associated location and starting time) that respects spatial and temporal constraints provided by the intended activity description. The produced tasks sequence minimizes an effort function that takes into account the realisation of tasks as well as the navigation between the locations where these tasks are performed. However, taking all these interacting parameters into account generates a high computation complexity. In our algorithm, a special effort is made to reduce this complexity through the use of filtering and pruning methods. In this section, we begin by a general description of the used algorithm. Then, we describe with more precision the different functions used in this algorithm. Finally, the relaxation of the time constraints associated with the computed task sequence is discussed.

2.1 Structure of the algorithm

Our scheduling algorithm is depicted in Algorithm 2. It is a variant of the A* algorithm [Hart et al., 1968]. It explores a search space ($L \times S \times P \times T$), which is the product of the nodes L in the topological graph, the situations S in the activity graph , the paces P and the time T. An exploration state st of this space contains a location $l \in L$, a situation $s \in S$, a pace $p \in P$ a time $t \in T$ and a cost $c \in \mathbb{R}$ representing the accumulated effort required to reach this state. The generation of successor states depends on the edges travelled to reach these states, being either *navigation edges* or *task edges*. The SUCCESSORS function, detailed in section 2.2, follows a different generation process in each of these cases. In the A* algorithm, the closed states structure contains all the already visited exploration states. Every time a new exploration state is reached, it is added to the opened states list if not already in the closed states list. However, in our problem, many exploration states can be excluded from the search. For example, a way of reaching a location in a given situation with a higher time and cost can be ignored as it is sub efficient in any case. For this reason, in our algorithm, we use a dedicated closed states structure (CLOSED) that enables the use of a filtering function FILTER. This filtering function drastically reduces the number of explored states. This structure and filtering function are detailed in the section 2.3. Furthermore, in our problem, strong temporal constraints can be defined in the activity description. Some states can thus be excluded from the exploration as they cannot lead to a solution that respects these constraints. In order to avoid exploring these states, our algorithm uses a PRUNE function, detailed in section 2.4, further reducing the complexity of the scheduling process. Finally, as in any A* search, our algorithm uses a heuristic to guide the exploration (see chapter 1, section 2.2). This heuristic estimates the remaining effort required to reach the goal from a given exploration state. It is used by the BEST function to pick the most promising state in the OPENED list. In the section 2.5, we describe the computation of this heuristic and how this heuristic is used by the BEST function to pick the most promising state from the OPENED list.

2.2 Successor state generation.

At every step of the task scheduling algorithm, the most promising exploration state $st = \{l, s, p, t, c\}$ is extracted from the list of opened nodes. The SUCCESSORS(st) function depicted in Algorithm 3 generates all successor states of the state st. A set of successor states $st' = \{l', s', p', t', c'\}$ is possibly generated for each edge e leaving the node l in the topological graph. The successor states generation depends on the nature of the edge e, being either a *navigation edge* or a *task edge*.

Algorithm 2 Task scheduling algorithm.

```
1: function SCHEDULING(st_i, s_g)
2:     CLOSED = ∅
3:     OPENED = st_i
4:     while OPENED ≠ ∅ do
5:         st = BEST(OPENED)
6:         if is_goal(st) then
7:             break
8:         OPENED = OPENED - st
9:         Sc = SUCCESSORS(st)
10:        for all st' ∈ Sc do
11:            if !FILTER(st',CLOSED)∧ !PRUNE(st') then
12:                UPDATE(st', CLOSED)
13:                OPENED = OPENED ∪ st'
14:    if OPENED == ∅ then
15:        FAILURE
16:    Construct schedule from best(OPENED)
17:    Relax constraints on schedule
```

During the successor state generation, the agent's pace does not change when navigating but only when performing a task. There are two main reasons for this choice. First, the length of navigation edges is not constant but varies from location to location. Therefore, among two locations at almost equal distance, one location could be favoured just because the agent could better adapt its average speed by using more navigation edges. Second, if a change of pace is allowed each time a navigation edge is used, the number of successor states increases drastically. This leads to higher memory requirements as well as a higher computation time. Moreover, as we later compute a relaxed time schedule, the pace of the agent is indicative during scheduling and only used for estimating the supplementary effort implied by hurrying between locations. The real pace of the agent is computed during navigation relying on the relaxed time schedule.

Navigation edge. If the edge e is a *navigation edge*, it represents a displacement between two inter-accessible locations. The successor state is generated with a location equal to the end location of the edge ($e.l_e$), meaning that the agent reached this new location. The successor state is associated with the same situation as its predecessor, as a displacement does not affect the situation in the activity graph. As we consider that the agent keeps the same pace all along the way between two task realisations, the successor state is given the same pace as its predecessor. The time associated with the successor state is equal to the time associated with its predecessor state updated by the duration of the displacement. This duration is estimated by taking the length of the path and the speed associated with the pace of the agent into account:

$$duration(e, st) = e.length/st.p.speed$$

Finally, the cost associated with the successor state is equal to the cost associated with its predecessor state updated by the effort induced by the displacement. This effort is estimated by taking into account the length of the edge, the supplementary effort associated with the pace of the agent and to the nature of the crossed zone as well as the penalty associated with the situation:

$$effort(e, st) = e.length \times st.p.effort \times a.nature\_weight(e.nature) \times (1 + st.s.global\_penality)$$

Algorithm 3 Successor states generation algorithm.

```
1: function SUCCESSORS(st)
2:     SUCCESSORS = ∅
3:     for all e ∈ En ∪ Et where e.l_o == st.l do
4:         if e ∈ En then [case of a displacement]
5:             l' = e.l_e
6:             s' = s
7:             p' = p
8:             t' = t + duration(e, st)
9:             c' = c + effort(e, st)
10:        else[case of a task realisation]
11:            transition = get_transition(s, e.tk)
12:            if transition ≠ NULL then
13:                for all pace ∈ P do
14:                    l' = l
15:                    s' = e.tk.s_f
16:                    p' = pace
17:                    t' = t + duration(e, st) + wait_duration(e, st)
18:                    c' = c + effort(e, st) + wait_effort(e, st)
19:        new st' = {l', s', p', t', c'}
20:        SUCCESSORS = SUCCESSORS ∪st'
21:    return SUCCESSORS
```

Task edge. If e is a *task edge*, it represents the possible realisation of a task tk at the location l. The agent only performs tasks if specified in its activity description. Therefore, a function $get\_transition(s, tk)$ checks that the task tk can be performed by the agent in situation s. In that case, the function retrieves the corresponding transition in the activity graph. In order to be able to respect the temporal constraints associated with its activity, the agent is allowed to change pace every time a task is performed. Thus, if the task tk can be performed, a successor state st' is generated for every available pace. Performing a task also implies a situation change in the activity graph. The successor states are associated with the situation the retrieved transition leads to. As task realisations do not affect the agent's location, the successor state is associated with the same location as its predecessor. The time associated with the successor state is equal to the time associated with its predecessor state updated by the duration of the task realisation and the potential waiting time. The duration of the task realisation is expressed in the description of the task as a function of the time.

$$duration(e, st) = e.tk.duration(t)$$

The waiting duration is equal to zero if the task realisation happens during the admissible time intervals associated with the location and with the task. If not, the waiting duration is equal to the remaining time until the next admissible time interval.

$$wait\_duration(e, st) = min_{t_{adm} \in (e.tk.valid\_interval \cap st.l.valid\_interval \wedge t_{adm} > t}(t_{adm}) - t$$

Finally, the cost associated with the successor state is equal to the cost associated with its predecessor state updated by the effort induced by the task realisation and the effort potentially induced by waiting. The effort caused by the task realisation is estimated by taking into account the duration of the task, the effort associated with this task and the supplementary effort associated by the agent to this task category:

$$effort(e, st) = duration(e, st) \times e.tk.effort \times a.task\_weight(tk) \times (1 + st.s.global\_penalty)$$

The waiting effort depends on the wait duration and the agent's wait effort weighting:

$$wait\_effort(e, st) = wait\_duration(e, st) \times a.wait\_weight$$

2.3 Closed states structure and filtering.

The SUCCESSOR(st) algorithm generates all possible successor states of the current state. Among these states, some cannot lead to an optimal solution. In order to avoid exploring unnecessary states, breadth-first search algorithms rely on a closed states list. New states are added to the opened states list only if their cost is the smallest found yet for the associated node. However, the exploration space (L× S ×P × T) of our algorithm is large. The time dimension T is especially problematic as it is, by nature, continuous and of unbound size. In this situation, storing the best cost found for any $\{l, s, p, t\}$ combination would be a problem. Indeed, two different states would almost always have a different t, which could lead to an unmanageable number of explored and stored states. The solution to this problem relies on filtering the states that are worst in any case than similar states (states sharing the same location, situation and pace). Given the definition of our problem, a state is potentially better than a similar state if its cost is lower. Indeed, having a lower cost means that the goal can be reached with a lower final cost. A state is also potentially better than a similar state if its time is lower. Indeed, a lower time value means that using this state may lead to the satisfaction of a future time constraint that might not be satisfied by using a state with a higher time value. This means that a state is always better than a similar state only if its cost and time are lower. In such a case, the second state cannot lead to a better solution and is thus pruned. Therefore, each time a successor state is generated, the closed states structure is interrogated to see if no other state sharing the same location, situation and pace was found with a better cost and time. If no better state is found, this proposition is verified. It means than the generated successor state can potentially lead to the optimal solution. In such a case, it is added to the opened list and the closed list is updated to take this new state into account.

The closed state structure $CLOSED$ is a three dimensional table (L × S × P). Each cell contains a sorted table storing couples $time, cost$.

In the algorithm, the FILTER(st) function, detailed in algorithm 4, prunes the state st if it possesses the same location, situation and pace as a closed state but a higher time and cost.

Algorithm 4 Filtering algorithm.

```
1: function FILTER(st)
2:     for all {t', c'} ∈ CLOSED[l][s][p] do
3:         if t' ≤ t ∧ c' < c then
4:             return true
5:     return false
```

If the state st is not pruned, the table $CLOSED$ is updated. The UPDATE (st,CLOSED) function, detailed in algorithm 5, removes all couples $\{t', c'\}$ from the $CLOSED$ table that are associated with the location, situation and pace of st and with higher time and pace.

These two functions ensure that the OPENED and CLOSED tables only contain a minimum number of states, thus enhancing the algorithm performances.

Algorithm 5 Update algorithm.

```
1: function UPDATE(st)
2:     for all {t', c'} ∈ CLOSED[l][s][p] do
3:         if t' ≥ t ∧ c' > c then
4:             CLOSED[l][s][p].remove({t', c'})
```

2.4 Exploration state pruning.

Activities are sometimes subject to strong temporal constraints. For example, shopping must be done before the shops closes. In that case, many generated exploration states that are not pruned by the FILTER(st) function cannot lead to a solution compatible with time constraints. For example, a state in which the agent has not bought bread yet and cannot reach any bakery before closing time can be considered invalid. We use the pruning function PRUNE(st) described in algorithm 6 to detect these invalid states and avoid adding them to the opened states list. This function is the equivalent to asking the questions "If I keep on with my current speed, do I have enough time to reach a location where to perform my next task?" and "Is the time remaining after I finish my next task sufficient to complete my activity?.

Algorithm 6 Pruning algorithm.

```
1: function PRUNING(st)
2:     for all tr ∈ Tr where tr.s_i = st.s do
3:         min_time = st.t + l.min_dist(tr.tk)/p.speed
4:         if min_time ∈ (tr.tk.contextualised_interval) then
5:             if min_time + min_{t∈T}(tr.tk.duration(t)) < tr.tk.s_e.max_time) then
6:                 return false.
7:     return true
```

The PRUNE(st) function used in the algorithm checks all the transitions tr leaving the situation $st.s$ in the activity graph and consider the task $tr.tk$ it carries. For each of these transitions, the minimum time required to reach the locations where to perform task $tr.tk$ is computed using the minimum distance to this task $l.min\_dist(tk)$ stored in location l as well as the speed associated with the current pace of the agent $p.speed$. This minimal time is added to the time associated with the state st in order to obtain the minimal time enabling the task $tr.tk$ to be performed. Two properties are checked for this computed time. First, time is compared to the contextualised valid time intervals $tk.contextualised\_interval$ associated with task tk. If the minimal time belongs to the valid interval, it means there might be enough time left to perform the task tk. Second, the minimal time at which the situation $tk.s_e$ can be reached is computed by adding the minimal time required to perform the task tk and the minimal duration of the task tk. If this time is inferior to the maximum suitable time $tk.s_e.max\_time$ associated with the situation $tk.s_e$, it means that enough time potentially remains to complete the activity after having performed the task tk. If both of these properties are verified for at least one of the transitions, it means that the agent might be able to complete his activity while fulfilling all temporal constraints using the state st. This state can thus be added to the opened states list.

This algorithm prunes two kinds of branches from the exploration tree. First, all branches in which the agent is not going fast enough to perform any task in time. Second, all branches in which the agent cannot complete his activity whichever pace used. This pruning greatly reduces the number of developed nodes in cases of time constrained scenarios.

2.5 Selection of the best exploration state

One of the major factors defining the scheduling algorithm performances is its ability to select the most promising state in the OPENED list at each iteration of the search. This selection relies on the cost of the state as well as on the estimation of the remaining effort required to reach the goal situation. The estimation of the remaining effort required to reach the goal is used as a heuristic function guiding the scheduling algorithm. This heuristic is computed by summing the effort associated with performing the remaining tasks is situation $\mathit{Eff}_{tasks}(s)$ and the effort associated with navigating between locations where these tasks should be performed $\mathit{Eff}_{nav}(l, s, p)$.

$$Heur(l, s, p) = \mathit{Eff}_{tasks}(s) + \mathit{Eff}_{nav}(l, s, p)$$

The **effort related to the execution of tasks** does not depend on the location where they are performed but only on the time. This means that an estimation of the minimal effort required to perform the remaining tasks can be precomputed using only the activity graph. It is possible to attach such an estimation to every situation in this activity graph. First, a minimal effort value $min\_eff(tk)$ is associated with each task labelling a transition of the activity graph. This minimal effort value is computed from the original task description as the product of the minimal duration of the task and of the cost of performing the task per time unit. Then the estimation of the remaining effort is computed for each situation as follows. The goal situation s_g of the activity graph is associated an effort value $\mathit{Eff}_{tasks}(s_g)$ equal to zero (there is no remaining task). Then, the activity graph is explored from this goal situation to the initial situation, computing the minimum remaining effort for each situation s:

$$\mathit{Eff}_{tasks}(s) = min_{tr, tr.s_o = s}(min\_eff(tr.tk) + \mathit{Eff}_{tasks}(tr.s_e))$$

The **estimated remaining effort related to navigating between different locations** where tasks can be performed depends on two factors: the location in the topological graph and the situation in the activity graph. For a given location $l \in L$ and a given situation $s \in S$, we define a function $D(l, s)$. This function returns a set of couples (d_1, d_2), one per transition tr such as $tr.s_o = s$. In a couple (d_1, d_2), d_1 is the minimal distance to a location where $tr.tk$ can be performed and d_2 is the shortest distance that must be travelled whichever the sequence of tasks performed after $tr.tk$ is. The estimated effort value is then computed as follows:

$$\mathit{Eff}_{nav}(l, s, p) = min_{(d_1, d_2) \in D(l,s)}(d_1 * p.effort + (d_2 - d_1) * min_{p \in P}(p.effort))$$

The distance d_2 is computed as follows. Let $Q(s)$ be the set of all possible remaining tasks sequences when in situation s, $Tk(q)$ be the set of all tasks in the sequence q given a situation s and location l, d_2 is computed as follows:

$$d_2(l, s) = min_{q \in Q(s)}(max_{tk \in T(q)}(l.min\_dist(tk)))$$

In order to lower the computation cost, $l.min\_dist(tk)$ is pre-computed for every location $l \in L$ and for any possible task tk. Furthermore, when evaluating d_2, we store intermediate results in a cache for later use. This cache associates a d_2 value to each couple (l, s) for which this value has been computed. This greatly enhances the performances of the heuristic computation.

The goal of the heuristic is to favour the exploration of the most promising states ie. the states which are the most likely to lead to the optimal solution. Every time a state st is added to the OPENED list, it is associated with an expected cost $c_{exp}(st)$. This expected cost is computed as: $c_{exp}(st) = st.c + Heur(st.l, st.s, st.p)$. The BEST($st$) function picks

the state in the list with the lowest expected cost, which is its most promising element. This process guides the search toward the optimal solution, reducing the number of explored states. It is especially efficient in the case of unconstrained activities, which makes this process complementary with the pruning function, which works better in constrained situations.

2.6 Task schedule and constraint relaxation.

From the sequence of states linking the final state to the initial state, the task schedule $\{tk_1, ..., tk_n\}$ is extracted, with associated starting times. However, these tasks are the earliest given the chosen navigation paces. As this schedule is intended to be performed by an agent navigating in a crowded environment, the dates are indicative but are not intended to be precisely respected. For example, if an agent intending to visit at 5pm. a shop closing at 7pm. gets delayed, it does not makes sense that it starts running to respect the scheduled time. Instead, the agent should just make sure to arrive soon enough to perform its shopping task before 7pm. To give more freedom to the agents, time constraints are relaxed by assigning starting time intervals to each task. Yet, the constraints associated to the future tasks the agent intends to perform may impact these relaxed time intervals. For example, if the previous agent intends to take a train at 6pm., it will perhaps have to hurry, even if the shop is not closing soon. In order to accurately estimate the impact of delays on the computed schedule's realisation, the relaxed time intervals computation consider the constraints associated with all the tasks remaining to perform.

Let $start(tk)$ be the start time of task tk, $end(tk)$ be the ending time of tk, $(start(tk_{i+1}) - end(tk_i))$ thus describing the intended travel duration between the locations selected to perform the tasks tk_i and tk_{i+1}. Let $OPEN(l)$ be the set of opening time intervals of the location l where tk is performed and $\mathcal{E}(I)$ be the envelope of a set of intervals I. We define the relaxed time constraint (ie. the interval during which the task can be started) $R(tk_i) = [\underline{R(tk_i)}; \overline{R(t_k)}]$ of a task tk_i in $\{t_0, ..., t_n\}$ as follows:

$$\begin{aligned} R(tk_n) &= \mathcal{E}(OPEN(tk_n) \cap [start(tk_n); +\infty]) \\ R(tk_i) &= \mathcal{E}(OPEN(tk_i) \cap [start(tk_i); \overline{R(tk_{i+1})} + start(tk_{i+1}) - end(tk_i) - duration(tk_i)]) \end{aligned}$$

If a task is started within its associated time interval, the agent should be able to perform the remaining tasks in time. In the contrary, it means that the schedule is compromised. In this case, either a slight increase of the agent's speed is sufficient to respect the relaxed constraints or an activity realisation error is detected, triggering a rescheduling of the remaining tasks.

Summary

Our task scheduling algorithm explores the $(L \times S \times P \times T)$ space in a similar way as a A* algorithm does. This means that all the different combinations of locations, situations and paces may be explored. This way, we can guarantee that no solution can be missed, and that the optimal solution is found. To overcome the complexity inherent too our problem, we propose a set of methods aiming at reducing the number of visited nodes. The filtering function takes advantage of the CLOSED structure to remove states that cannot lead to an optimal solution. The pruning function removes nodes that cannot lead to a solution satisfying temporal constraints. Finally, the evaluation of the remaining effort guides the search in order to limit the number of visited nodes. Furthermore, our algorithm uses the knowledge provided by the topological graph, the activity graph and the agent's description to generate a solution taking spatial and temporal constraints into account, as well as the personal characteristics and preferences of agents. This enables the generation of a relaxed task schedule, describing a

credible task sequence, labelled with locations and relaxed time interval. The algorithm also generates a high-level path I term of city areas to travel. This path takes the agent's preferences over the nature of navigation zones into account. It acts as the first step of our path planning process by selecting a set of city areas to travel between task realisations.

3 Results

To test our model properties and its consistency with humans' decisions, we created a 3D model of a city centre. This 3d model features an elementary school, two malls, two banks, three bakeries, two butchers as well as several houses and workplaces. We also designed an activity featuring a pedestrian having to get back home after work. On the way to home, some food must be bought, either by going to the mall or by getting some money at a bank and then going, in any order, to the bakery and the butcher. Moreover, children must be picked at school at 4:30pm. This pedestrian is allowed three different paces: *walking*, *hurrying* and *running*. At any time, it is also possible to pass by home to get rid of all the shopping bags. In the following, we discuss the properties of our scheduling algorithm and compare its results to tasks schedules produced by humans.

3.1 Activity scheduling properties

In order to demonstrate our algorithm properties, we first compare it to a model that does not possess look-ahead capabilities. We then compare the schedules produced with different initial setups. Figure 5.3 depicts the activity schedules obtained in these different situations.

We first compare the activity schedule computed by our algorithm (see figure 5.3.a) or by a reactive algorithm with no look-head capabilities (see figure 5.3.b). In this example, the agent leaves work at 4:15pm. In both cases, the agents choose to visit a mall before going to the school. However, our algorithm selects the mall closest to the school, even if the path is longer. Indeed, this way, the agent spends less time carrying groceries. This example demonstrates the impact of a long term decision and the use of effort penalties associated with tasks description.

In figure 5.3.c, compared to the first case, we only changed the initial location of the agent. This change does not only impact the choice of locations but also the performed tasks. In that case, the activity schedule has been optimized, enabling the agent to visit the bank and butcher before going to school and without being late. This results from the combination of spatial and temporal constraints with long term scheduling.

Finally, in figure 5.3.d, we show the impact of time constraints on scheduling. In that case, the agent leaves work at 4:25pm instead of 4:15pm. Our algorithm detects that there is not enough time to go to the mall. However, the agent, if hurrying, is able to pass by the bank in order to get some cash before picking up his children at school. Finally, it visits the bakery and butchery. Indeed, even if a faster pace implies a greater immediate cost, the agent avoids a later detour which would have increased the global cost of the task schedule.

3.2 Individual activity scheduling model evaluation

In order to validate the representativeness of our task scheduling model, we carried out an experiment aiming at comparing the output of our algorithm to human-generated schedules and to other scheduling methods. To do so, we provided 31 participants with ten maps describing ten activity scheduling setups with different starting time, workplace and home location. The provided environment and activity were the same as in the previously discussed examples. In each setup, when combining possible task sequences with locations, we obtain 438 potential

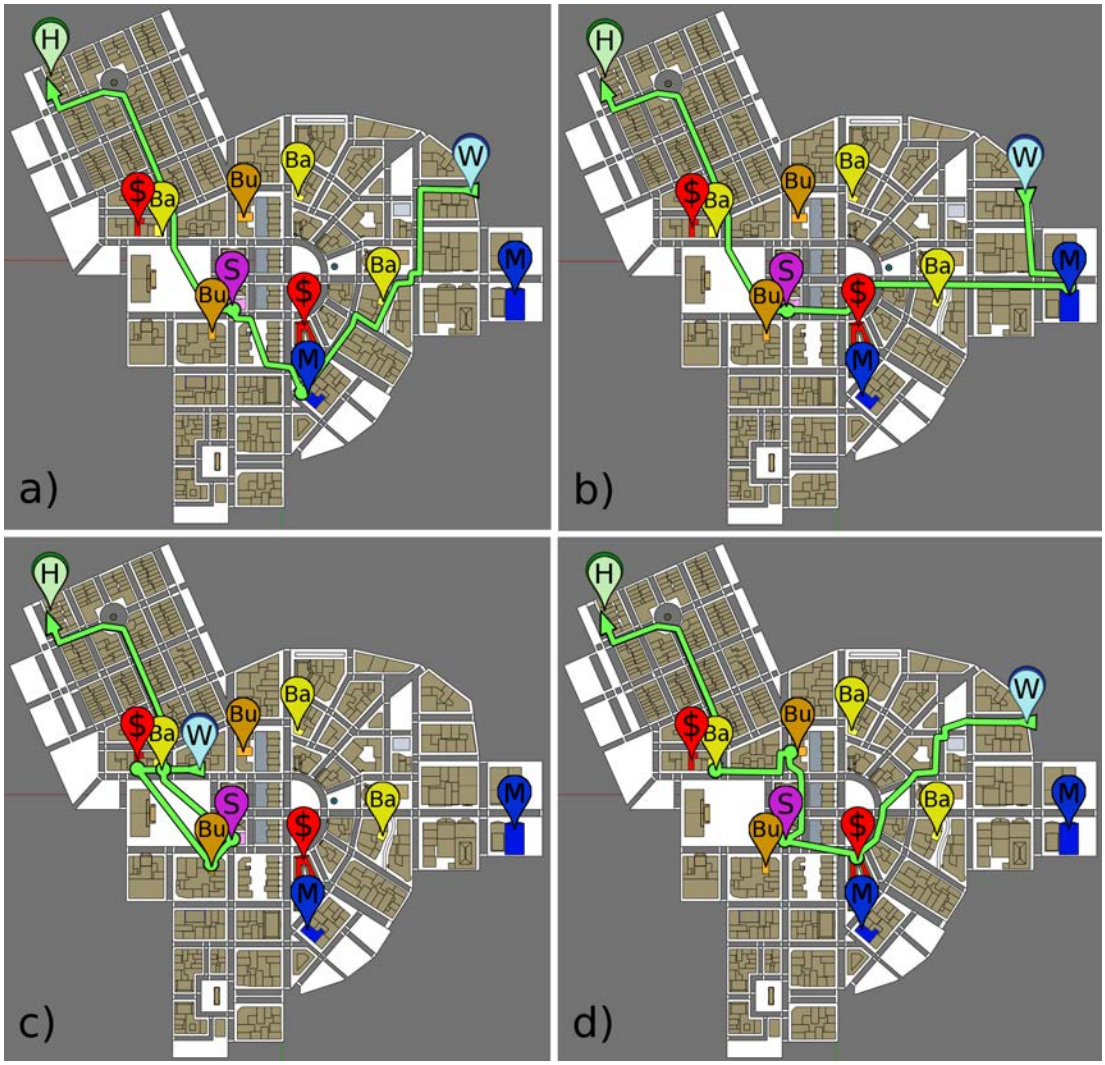


Figure 5.3 – Computed schedules for a parent worker leaving work at 4:15 pm, (a) with our model and (b) with a reactive model, (c) when changing home location or (d) leaving at 4:25pm.

solutions. We provided the participants with three rulers indicating the approximate time needed to travel given distances if walking, hurrying or running. Participants were asked to indicate the locations they would intend to visit in order to perform the activity if in a similar real-life situation. The explanatory sheet provided to the people participating in the experiment is shown in appendix B.refB1, as well as an example of scheduling setup in appendix B.refB2.

For each setup, the experiment results were grouped by intended task schedules. In figure 5.4, the "H" columns depict the statistical repartition of human task schedules for every setup. In the general case, one main solution could be identified, as well as few secondary ones. The remaining answers were scattered. Among our ten setups, we observe that from 34.6% to 91.7% of people select the same main solution, with a mean of 58.4%. The two principal solutions reach a mean of 75.6% of the answers, and this number goes up to 87.1% for the three principal ones. This demonstrates the fact that people generally agree on a small number of best solutions. However, in every case, some alternative solutions were proposed. These solutions represent a mean of 12.9% of participants' answers. This demonstrates the variability in humans' activity

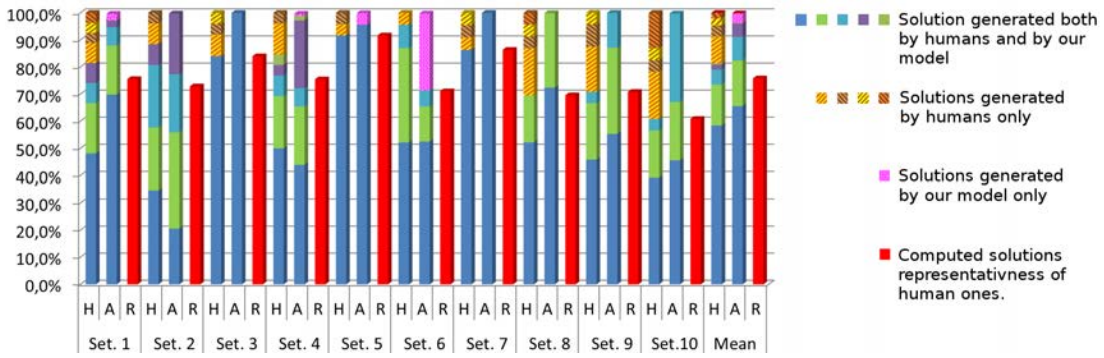


Figure 5.4 – Statistical repartition of different localised task sequences provided by the experiment participants (H) and automatically generated by our algorithm (A) for ten setups. (R) is the representativeness of the generated solution, computed as an histogram intersection between humans' and computed' solutions . (note that identical task sequences are represented by the same colour between columns H and A of one setup, not between different setups.)

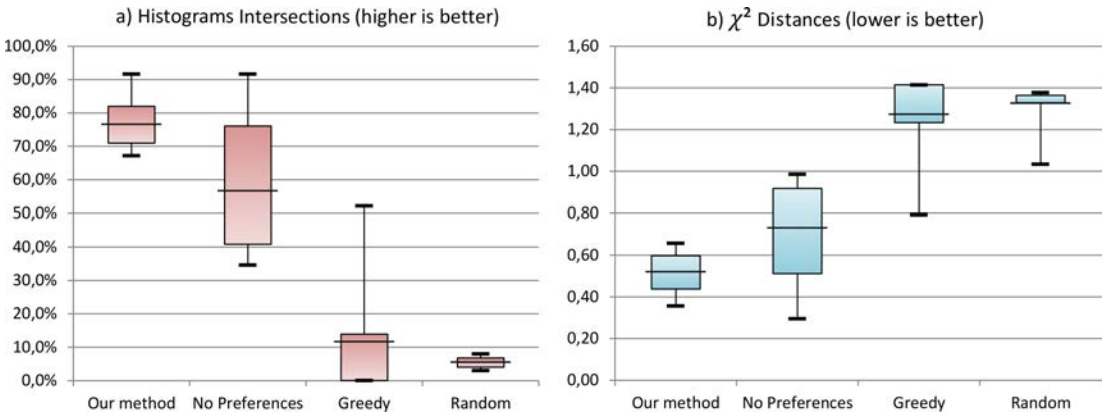


Figure 5.5 – Statistical repartition over the ten setups of the histogram intersections and the χ^2 distances between the human-computed and automatically computed schedules, with four different activity scheduling processes: our process, our process with no individual preferences, a greedy process and a random repartition.

scheduling.

After the experiment, we asked the participants to answer to some questions about their global preferences (Do you prefer shopping in malls or in small shops? How do you feel about waiting for ten minutes? ...). From the set of fulfilled forms, a statistical repartition of agents' parameters values (cost associated with tasks, pace cost, waiting time cost and effort penalty) has been computed (independently of human provided tasks schedules). We then created a set of 1000 agents parameterized by values randomly picked with a probability distribution matching those statistics. This set was used as an input to our scheduling algorithm in order to schedule the activity in the 10 setups provided to participants. The generated solutions were grouped and matched with the group of human-planned ones, see columns "A" in figure 5.4.

The statistical repartition of computed solutions is very similar to the observed one. In most of the setups (except setup 2), the main solution provided by our algorithm is the same as the main solution provided by the participants. We can also observe that most of secondary solutions were also generated: only a mean of 19.3 % of human-generated solutions were never generated by our algorithm. Those non-generated solutions imply a greater effort whatever the parameters of our algorithm are. This can be explained by the fact that people tend to use a

path globally directed to the goal, even if this path is longer. On the opposite, less than 4% of generated solutions corresponded to no human-planned solutions. In column "R" of Fig. 5.4, a representativeness value is computed. This value is defined by the intersection of histograms between human-generated and automatically generated schedules. This value represents the capability of our algorithm to generate populations with tasks schedules corresponding to humans ones. Over our 10 test setups, we obtained a representativeness between 61.2% and 91.4%, with a mean of 76.4%. This demonstrates that, given a coherent distribution of personal preferences, our algorithm is capable of generating representative tasks schedules that take into account human variability through parametrization.

Finally, in order to give an estimation of the benefice of our process, we computed the solutions to the same 10 setups with three other scheduling processes:

- **No preferences.** We use our scheduling process, except that all agents are given the same average characteristics and preferences. This means that all agents produce the same schedule.
- **Greedy.** An activity scheduling process with no look-ahead capabilities. It always performs the task requiring the minimum immediate effort to reach and perform.
- **Random.** A process that randomly selects one of the 438 possible schedules for each agent.

We compare the obtained schedules to the ones produced by humans through two indicators. The first is the representativeness factor, computed as the histogram intersection between the two sets of solutions. The second is the χ^2 (chi square) distance between the two sets of solution. The χ^2 distance is used in the field of correspondence analysis to estimate the proximity between two statistical repartitions. The lower is the computed χ^2 distance, the closer from each other are the statistical repartitions. Figure 5.5 shows the statistical repartition of the obtained indicator values over the ten setups, for the four scheduling processes. We observe that the scheduling process using average characteristics for all agents globally generates less consistent schedules, with a median representativeness of 58.1% instead of 77.2%. In the best cases, we see that this process reaches a representativeness similar to the one obtained with our process. This is due to the fact that, in some setups (3, 5 and 7), one of the solution is overwhelming compared to the others. The process that does not differentiate agents characteristics only produce this solution, thus being consistent with human schedules, sometimes even reaching a better χ^2 distance value. However, in the setups in which the humans selected more diverse solutions, this process gets far less consistent, as it only produce one of these solutions. In these cases, the representativeness goes down to a minimum of 34.7%, compared to the minimum of 61.2% obtained with our method. This demonstrate how the diversity of solutions produced when considering individual characteristics and preferences enables the production of statistically more consistent behaviours. The greedy process generally produces far less consistent solutions, with a median of 12.2% of representativeness and a far higher χ^2 distance. This model is sometimes able to reach a bit more than 50% of representativeness, when the greedy path is selected by more than half of the persons. However, we observe that, in the best case, this method did not make as good as our method did in the worst case. This demonstrates that people do not try to minimize their immediate effort but consider the long-term impact of their choices when scheduling an activity. Therefore, using a method able to take into account the strong interaction between space, time and activities enables the generation of statistically more consistent behaviours. Finally, as expected, the random process selected sets of solutions extremely inconsistent with the human-produced solutions. This demonstrate that producing a wide variety of solutions is not sufficient to match human

nbS	nbA	t_H	$t_{!H}$	$gain$
2	1	0.31 ms	0.88 ms	64.8%
6	3	6.98 ms	13.55 ms	48.5%
13	8	17.66 ms	28.20 ms	37.4%

Table 5.1 – Performance of our algorithm for three activity graphs consisting of nbS situations and nbA possible task sequences: t_H with estimated effort and $t_{!H}$ without; $gain$, mean gain for using the estimated effort.

activity, as people aim at minimizing their effort when performing their activity.

This experiment validates the consistency of the schedules produced by our model. It demonstrates how the ability to consider the relation between the space, the time, the activity and individual characteristics and preferences is important for producing credible schedules.

3.3 Performance evaluation.

As we intend to use this algorithm in real-time applications, it is essential to keep the computation times reasonable. Table 5.1 sums up the mean computation time averaged over 1000 planning calls in a 5754 nodes planning graph, using activity graphs of increasing complexity. Columns t_H and $t_{!H}$ show the algorithm performances with and without the proposed heuristic while the column $gain$ shows the percentage of gained computation time when using the heuristic. We see that the heuristic enables a gain of 64.8% for a very simple activity. However, this gain decreases with the complexity of the activity graph, as the heuristic has more difficulties to accurately estimate the remaining effort. We also observe that the performances are highly correlated with the number of possible arrangements nbA in the activity graph. These values are acceptable for real-time applications, but may become a problem with a more complex activity graph.

3.4 Crowd simulation

We used this model to populate our virtual urban environment with 10000 agents embedded with activities randomly selected from a small set of activity graphs. These agents were generated in real-time from 4 pm to 4.20 pm and animated along the planned path using a path optimization and a collision avoidance system inspired by the work in [Lamarche and Donikian, 2004][van den Berg et al., 2008][Ondrej et al., 2010]. By using the relaxed constraints on the computed activity schedules and possibly a partial rescheduling in case of detected potential failure, the agents were able to perform their activity. Some macroscopic crowd phenomenon emerged in the city. For instance, we could observe some flows going from the work areas to the housing areas, and higher densities of pedestrians appearing in front of the school around 4.30 pm. As an example, figure 5.6 shows the difference between obtained flows in front of the school at 4.10 pm and 4.20 pm. This shows how our model can be used to populate large environments over long periods of time. It also demonstrates that macroscopic phenomena such as realistic flows and densities automatically emerge from the activity of the agents.

Conclusion

In this chapter, we presented an original activity scheduling process. This process aims at selecting where, when and in which order several tasks, representing an intended activity, should

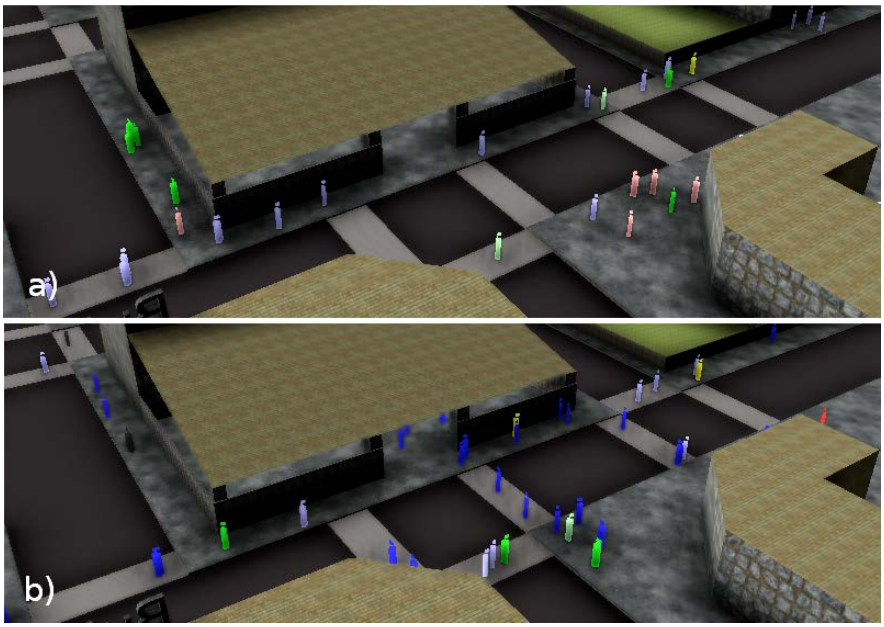


Figure 5.6 – Population in front of the school at 4.10 pm (a) and 4.20 pm (b). In b, a flow of people going to the school (in dark blue) appears, that did not exist in a.

be performed. The proposed model handles the tight relationship that exists between space, time, activities and personal preferences. The produced task schedules are optimized on the long term and exhibit adequate choices of locations and times with respect to the agent intended activity and its associated spatial and temporal constraints. Furthermore, our activity scheduling process relies on independent description of the environment, of the agents and of their intended activities. This way, new agents or activities can seamlessly be added to the simulation and an agent activity can be adapted to a-priori unknown environments, such as procedurally generated or user-generated ones. Archetypes of agents can be described, with statistical repartitions of preferences. This enables the generation of a statistically accurate population of a virtual city based on statistical studies.

We validated our scheduling algorithm results with an experiment. This experiment showed the variability in human-generated schedules. However, our scheduler was able to produce tasks schedules that are representative of humans' ones in 76.4% of our test cases. We also demonstrated how, thanks to the use of a heuristic, as well as filtering and pruning methods, our model is able to populate environments as large as city centres with thousands of agents performing consistent activities over time periods as long as a full day.

Cooperative tasks scheduling under temporal and spatial constraints

6

In the previous chapter, we presented our individual activity scheduling process. This process computes a coherent sequence of tasks labelled with locations where the agent should perform these tasks and relaxed temporal constraints. However, as presented in [Allbeck, 2010], real activities do not only feature agents individually going from one place to another, but usually include agents performing cooperative activities. These activities include, in addition to individual tasks, cooperative tasks requiring multiple agents to perform synchronised actions in compatible locations. For example, two virtual actors may have to meet to exchange an object or to be in two video-conference rooms at the same time to have a talk. In such scenarios, locations and times where and when tasks should be performed can be fixed, loosely constrained or even left unconstrained. If cooperative tasks' locations or times are not strongly constrained, the agents cannot schedule their activities independently from each other. Indeed, the scheduling of these tasks is greatly dependant on the activities and constraints of the other agents involved in the cooperative tasks. On the other way round, the scheduling of the cooperative tasks also impact the individual schedules of all the involved agents. Therefore, scheduling the activity of these agents, including individual and cooperative tasks, implies a time and space synchronization of cooperative tasks compatible with the fulfilment of individual tasks of each agent. To find a solution to this problem, coupled approaches tend to use a single process to compute the scheduling of all agents activities in parallel as a single problem, taking all their individual parameters into account. This is a complex problem as it requires considering the combination of all agents' possible tasks orderings and all possible locations and times where and when to perform these tasks. The search space induced by such a problem is huge, and increases exponentially with the number of agents involved in a cooperative task.

In this section, we present a coupled cooperative activity scheduling process relying on our individual activity scheduling process. It enables the scheduling of multiple agents' activities featuring cooperative tasks while reducing the associated computational cost. This process relies on the same kind of input data as the individual task scheduling. Identical descriptions of the environment and of the personal characteristics of the agents are used. The activity description is also similar, with the addition of cooperative tasks involving two or more agents. Our cooperative activity scheduling process produces an individual task schedule for each agent, with associated locations and relaxed time intervals. It also ensures that these individual schedules respect the synchronization constraints associated with cooperative tasks, as well as the temporal and spatial constraints associated with each agent's individual activities. A compromise is made, aiming at reducing the global efforts of all the implicated agents, taking their individual characteristics and preferences into account. This automatically induces variability in the computed solutions, depending on the agents involved in the cooperative activity.

In our work, we focus on activities featuring loosely constrained or unconstrained cooperative tasks, for two main reasons. First, scheduling such tasks is a more general problem, offering a greater challenge. Our solution to the general problem of cooperative activity scheduling is still valid in more constrained cases. Second, our system allows an activity description to be

used in a-priory unknown environments. In such environments, no precise spatial or temporal constraint can be determined a-priori for cooperative tasks, the choice of adequate locations and times being highly dependent on the structure of the environment. In these environments, scenario should rely on loose constraints such as typologies of locations and time intervals to have a chance of being realisable. Therefore, being able to schedule unconstrained cooperative tasks is critical when automatically adapting scenarios to a-priori unknown environments such as procedurally generated or user-generated environments.

After presenting an example scenario and the description of cooperative activities, we discuss the characteristics of the exploration space implied by the problem and define some useful concepts. Then, for clarity reason, we present our cooperative activity scheduling process in two steps. First, we present this process applied to activities sharing a unique cooperative task. Second we explain how to deal with activities featuring multiple cooperative tasks. Finally, we present some results and discuss the impact of the choices that have been made in order to reduce the algorithm complexity.

1 Specificities of the Process

Even if it shares many similarities with the individual activity scheduling process, our cooperative activity scheduling process possess specific requirements. It requires a way of representing cooperative activities and to combine them into a single search space. It also requires specific data structures to represent the state of the world, including the states of multiple agents. In this section, we first describe the example scenario that is used throughout the chapter as an illustration. We propose a way of describing activities that involve cooperative tasks. Then, we discuss the innate complexity of the search space inherent to the scheduling of cooperative tasks and we present our hint on how to reduce this complexity. Finally, we present the definitions of some concepts required to understand our process.

1.1 Example scenario

Throughout this chapter, we illustrate our cooperative task scheduling process using a scenario taking place in an office floor (see figure 6.1.a). It involves two agents: a student and his advisor. The environment features offices, meeting rooms, printer rooms, and a break room with coffee machines. The student has to work on a document, print it and have it signed by his advisor. He also wants to have a cup of coffee at any point of this activity (see figure 6.1.c). The advisor has to have a coffee, then, in any order, sign the document and do personal work, finally he has to get back to work (see figure 6.1.d). Each agent's work task has to take place in his own office. To get a coffee, they must go to a coffee machine. The student needs to be in a printing room to retrieve his document. The meeting between both agents is a cooperative task, not constrained to any place or time. However, it requires both agents to share the same location. We voluntarily kept this example simple, for clarity reasons. In order to demonstrate the performances of the process, we use more complex scenarios and environments in the result section.

1.2 Cooperative activity description

A virtual actor usually has multiple options for performing an activity. These options differ in the order he performs tasks, in the locations where he performs them and in how he navigates between these locations. In the previous chapter, we proposed a way of representing all the admissible task sequences as an activity graph. This graph is also informed with temporal constraints and effort penalties associated with tasks realisation. This representation is still

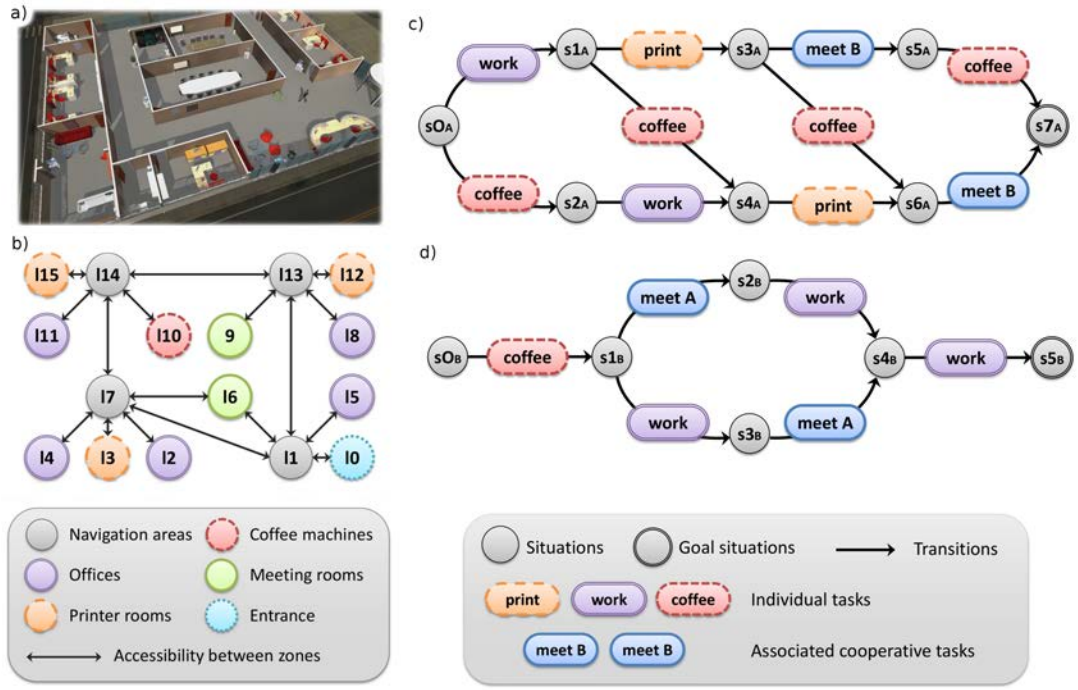


Figure 6.1 – 3D model of the office floor (a) and the associated topological graph (b). The activity graph of the student agent (c) and of its advisor (d)

valid for describing tasks sequences including cooperative tasks. However, in order to represent cooperative tasks, we add the possibility of labelling some tasks belonging to activity graphs of different agents as "cooperative". In our example scenario (see figure 6.1.c and d), the task *meet B* of agent a_A is associated with the task *meet A* of agent a_B. These tasks must be performed at a same time in compatible locations. The notion of "compatible locations" changes from a cooperative task to the other. For example, giving an object to another person requires sharing the same location, whichever it is. On the contrary, holding a video-conference with another person just requires that both these persons are in video-conference rooms. As individual tasks, cooperative tasks can be constrained to temporal intervals. It must be noted that agents may have different valid time intervals for the same set of cooperative tasks: our process is able to select a time which belong to the intersection of the valid intervals of all involved agents. Of course, if this intersection is null, no solution can be found.

Similarly to individual tasks scheduling, cooperative tasks scheduling is subject to be affected by the structure and nature of the environment as well as by the characteristics and preferences of involved agents. Therefore, the description of the environment and of the agent presented in the previous section is used without modifications by our cooperative activity scheduling process.

1.3 Search space

In a given environment, an intended activity can usually be performed in several ways. The behaviour of the agents can vary by the order of performed tasks, the locations where they are performed and the agents' navigation speed. The individual task scheduling explores, for each agent, a 4D space ($S \times L \times P \times T$), in order to determine an optimal individual

schedule. The cooperative task scheduling problem adds the constraint that some tasks have to be synchronized between multiple agents. A trivial solution to this problem would be to describe the activity of all the agents $a \in A_{coop}$ involved in a cooperative task tk_{coop} as a single combined activity. Then, a solution may be found by exploring the combined search space, i.e. the product of the search spaces of all involved agents. This method would ensure that the optimal location and time to hold the cooperative task is found. However, the search space, for n agents involved in a cooperative task, is $(L \times P \times T)^n \times \prod_{i=0}^{n} S_i$, in which S_i is set of situations associated with the activity graph of agent i. In others words, the dimension of the search space is exponential with the number of agents.

To avoid this complexity, our process relies on the idea that synchronization is not required for individual tasks, but only for cooperative tasks. This way, most of the agents' activities can be scheduled independently. The search spaces are combined only when a cooperative task has to be scheduled. By decoupling individual and cooperative task scheduling, our algorithm is able to greatly reduce the scheduling process complexity. The proposed algorithm is divided in five steps depicted in figure 6.2:

1. For each agent, the individual task scheduling algorithm computes all the potentially optimal ways an agent has to reach the situations in which the cooperative task can be performed. This step generates a filtered set of cooperation proposals, only containing proposals that can possibly lead to an optimal solution.

2. The cooperation proposals of all agents involved in the cooperative task are matched and synchronized. This creates a filtered set of possibly optimal cooperation configurations with associated locations and times.

3. The most promising of these cooperation configurations are selected.

4. For each selected cooperation configurations, each agent's individual task scheduling is finalized.

5. The outputs of the previous step are compared and the best solution is selected.

In the case of a scenario including agents involved in multiple cooperative tasks, steps 1 to 3 of the cooperative task scheduling algorithm are performed for each subsequent cooperative task, using only the agents involved in this task. Finally, when the most promising cooperation configurations are selected for the last cooperative task, steps 4 and 5 are performed.

For each agent, a suitable sequence of individual and cooperative tasks, with associated locations and times, is generated. The times associated with cooperative tasks are synchronized and fit the activity of all agents. The separation of individual task scheduling and cooperative tasks synchronization steps, coupled with the filtering techniques used in steps 1 and 2 of the algorithm as well as the most promising cooperation configurations selection, greatly reduce the complexity of the cooperative activity scheduling process.

1.4 Definitions.

In order to be able to describe our cooperative activity scheduling process, we first define a set of concepts it relies on:

- A **cooperation situation** s_{coopA} in S_{coopA} is a situation that allows the agent a_A to perform a cooperative task tk_{coop}. This means that at least a transition tr exists in the activity graph such as $tr.s_o = s_{coopA}$ and $tr.tk = tk_{coop}$. For example, $s3_A$ and $s6_A$ in the activity graph of agent A as described in figure.6.1.c are cooperation situations.

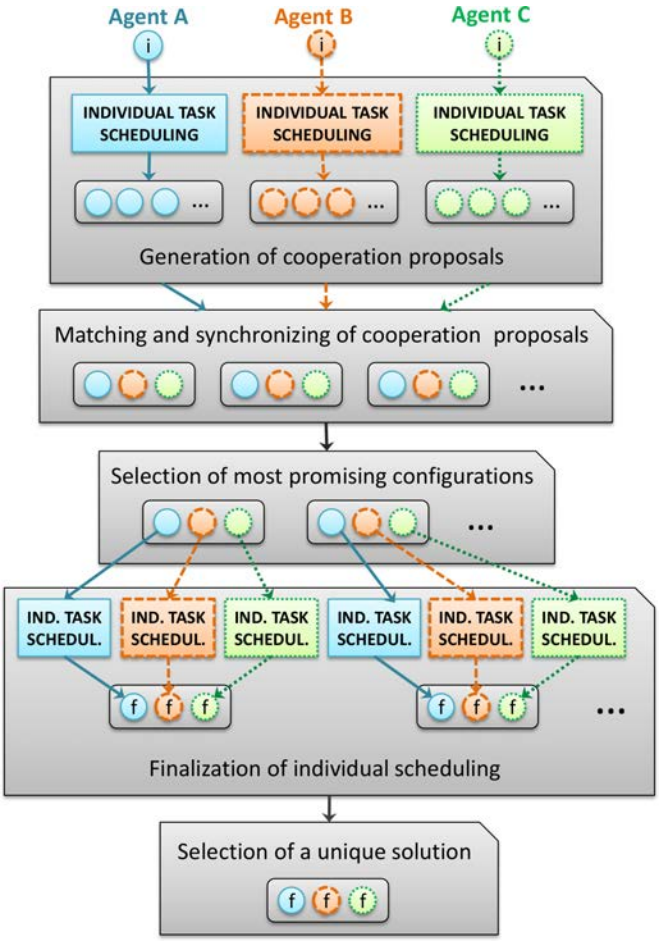


Figure 6.2 – The five steps of our scheduling algorithm, from the agents' initial states (i) to their final states (f)

- A **combination of cooperation situations** $Comb_{coop}$ is a tuple of n cooperation situations associated with a cooperative task tk_{coop}, one for each agent. It represents a global advancement of the scenario. For instance, the cooperation situation $\{s3_A, s1_B\}$, in figure.5.2 indicates that the agents are able to meet when the student has printed the document and when its advisor has had a coffee. The number of such combination cooperation situations $|Comb_{coop}|$ is equal to the product of the numbers of cooperation situation for each agent involved in the associated cooperative task. In our example, $|Comb_{coop}| = |S_{coopA}| \times |S_{coopB}| = 4$.

- A **cooperation state** is an exploration state including a cooperation situation. For more clarity, in our illustration, we will only represent a cooperation state with its location, situation and time. For instance, the state $\{l4, s3_A$,10:30$\}$ indicates that the agent a_A is able to go to meet his advisor in his office at 10:30 a.m. and that he did not have a coffee yet.

- For each cooperative task, a **compatibility function** $comp(l_A, l_B \dots)$ is defined. It indicates if the locations occupied by the agents are compatible with the task. For example, to exchange an object, the agents must share the same location, and if they want to have a video conference, they should be in adapted rooms. A **configuration** is a tuple of n

exploration states, one for each of the n agents $a \in A_{coop}$ involved in a cooperative task. A **cooperation configuration** is a configuration in which a cooperative task can be performed. It implies that all the states in this configuration are synchronised configurations states allowing for the same cooperative task to be held, in compatible locations. The configuration $\{\{a_A, l_A, s_A, p_A, t\}, \{a_B, l_B, s_B, p_B, t\}, \ldots\}$ is a cooperation configuration if:

- $a_A \in A_{coop},\ a_B \in A_{coop},\ \ldots$
- $l_A \in L,\ l_B \in L \ldots$ & $comp(l_A, l_B,\ \ldots)$
- $s_A \in S_{coopA},\ s_B \in S_{coopB},\ \ldots$
- $p_A \in P_A,\ p_B \in P_B\ \ldots$
- $t \in (tk_{coopA}.valid\_interval\ \cap\ tk_{coopB}.valid\_interval\ \cup\ \ldots)$

2 Cooperative task scheduling algorithm

Scheduling a cooperative task involving several agents requires to determine where and when each agent is able perform this task and to compare these opportunities in order to find a common solution. The individual task scheduling algorithm presented in the previous section is used to compute the cooperation proposals of each agent. These proposals are then matched and synchronized between all involved agents in order to find the best compromise in terms of location and time to perform the cooperative task. Finally, such synchronized proposals need to be validated in order to prove that each agent can finalize the remaining of its activity. Rather than exploring the combined search space of all agents involved in a cooperative task, our algorithm handles most of agents' activities independently. The combination of search spaces is only considered for cooperative tasks and only a subset of all possibilities is explored. In this section, we detail the five steps of our cooperative task scheduling algorithm, which were presented in the previous section.

2.1 Cooperation proposals generation

In our model, cooperative tasks are interlaced with the individual tasks the agent should perform. The locations and time selected to perform a cooperative task thus depends on the previously performed tasks, on the locations where they were performed and on the paces used by the agents. Therefore, the first step of our algorithm consists in developing, for each agent, all the optimal individual task schedules leading to a cooperation situation. These schedules are computed using the individual task scheduling process presented in the previous chapter.

For each agent, every time a cooperation state st is reached, this state is added to a set of cooperation states $PROP$ that we call cooperation proposals. This process ends when all possible cooperation proposals have been computed. A filtering function ensures that all the proposals $st \in PROP$ are such as no other state st' with $st.l = st'.l$ and $st.s = st'.s$ exist such as $st.t > st'.t$ and $st.c > st'.c$. This ensures that we store only the cooperation states that can potentially lead to an optimal cooperative task scheduling. In our example, the cooperative task is constrained only by the fact that agents must share the same location. Cooperation proposals are generated for every location in the environment, one for each task schedule that allows reaching this location in a cooperation situation.

2.2 Cooperation configurations generation

Once all cooperation proposals have been computed for all agents, they are matched and synchronized into an optimal set of cooperation configurations. By optimal, we mean that this

set contains exclusively the combinations of agents' cooperation proposals that can possibly lead to the optimal solution. First, a set of unsynchronized cooperation configurations is computed. To ensure the optimality property is respected, a cooperation proposal $\{l_i, s_i, p_i, t_i, c_i\}$ is matched with a cooperation proposal $\{l_k, s_k, p_k, t_k, c_k\}$ only if $comp(l_i, l_k)$ and if no other proposal $\{l_k, s_k, p_{k'}, t_{k'}, c_{k'}\}$ exists such as $t_k < t_{k'} < t_i$. For instance, in our example scenario, the student cooperation proposal $\{l4, s3_A$,10:30$\}$ will be matched with the advisor cooperation proposal $\{l4, s1_B$,10:15$\}$ if no other proposal from the advisor $\{l4, s1_B, t\}$ exists such that 10:15$< t <$10:30. This way, the number of obtained cooperation configurations is linear with the global number of configuration proposals. This process keeps, among multiple similar configurations, the one reducing the waiting duration. This drastically reduces the number of considered cooperation configurations while ensuring to keep all those potentially leading to the optimal solution. All generated cooperation configurations are then synchronized by synchronizing all the cooperation states times to the higher one found in the configuration, and by applying the waiting cost to the updated states. In the previous example, the obtained synchronised cooperation configuration is $\{\{l4, s3_A$,10:30$\}, \{l4, s1_B$,10:30$\}\}$.

2.3 Promising configurations selection

The previous steps of the process minimize the number of generated cooperation configurations while ensuring to keep the one leading to the optimal solution. However, the order of magnitude of this number is the product of the number of compatible combination of locations where the task can be performed by the number of cooperation situations combinations. To limit the algorithm complexity while trying to rapidly obtain a good quality solution, only a subset of promising cooperation configurations is selected. This selection relies on a globally estimated configuration cost. This cost is obtained by adding the costs of the states belonging to the configurations to an evaluation of the remaining effort required to reach the goal situation. This remaining effort is evaluated for each state with the heuristic defined in the chapter 5. By underestimating the real cost, this heuristic tends to favour the combinations of early cooperation situations in the activity graphs. To ensure more variety in the exploration and limit the bias implied by the heuristic, we pick the K lower cost configurations for every possible situation combination $Comb_{coop}$. This leads to the selection of $K \times |Comb_{coop}|$ cooperation configurations. For instance, in our example scenario, for $K = 4$ we would pick 16 configurations: the four with the lower estimated effort for the situation combinations $\{s3_A, s1_B\}$,$\{s3_A, s3_B\}$,$\{s5_A, s1_B\}$ and $\{s5_A, s3_B\}$. A low value for K implies a higher risk of missing the optimal solution but reduces the computation complexity.

2.4 Individual scheduling finalization

For each selected promising cooperation configuration, each agent runs an individual task scheduling process. This process starts from the state belonging to the cooperation configuration and related to the agent. It continues until the agent's goal situation is reached. For instance, in our example, the cooperation state $(l5, s3_A$,10:30) is developed until the goal state $s7_A$ of agent A is reached, with the state $(l10, s7_A$,10:43). If all agents succeed in computing their individual task scheduling, the given cooperation configuration is validated, otherwise it is discarded. This validation process is performed for each selected promising cooperation configuration. If at least one configuration has been validated, a solution has been found and the selected configuration is the one minimizing the sum of individual solutions costs. These solution costs can be weighted to give more importance to the effort optimisation of some of the agents over the others (for example, a student will usually adapt his schedule to his advisor's and not the contrary). If no configuration has been validated, a new set of promising configu-

rations is selected and the validation process is repeated. This ensures the completeness of our algorithm. Indeed, all possible cooperation configurations will be explored until one reaches a solution. If all the configurations are explored without any success, a scheduling failure is detected. This signals the non-existence of a solution to the given scheduling problem.

To further decrease the cost of the algorithm, we use a pruning technique to discard some cooperation configurations. Indeed, once a solution has been found, we obtain an upper bound C of the solution cost. A configuration is trivially discarded if its associated cost is greater than C. Moreover, each time an individual task scheduling associated with a state of the cooperation configuration succeeds, the estimated cost of the current configuration is refined. Once again, if during this cost refinement the current cost becomes greater than C, the configuration is discarded. The impact of this pruning technique varies with the nature of the problem. Yet, it is especially efficient in strongly constrained cases.

Our cooperative task scheduling process considers a subset of most promising cooperation configurations. A compromise has to be made between the quality of the computed schedules and the performances of the algorithm. If K is set equal to the infinite, the optimal solution is found, at the price of a high computation cost. If selecting only a small subset of the solutions, the optimality of the solution cannot be guaranteed. Yet, this method greatly reduces the complexity of the process by limiting the number of individual scheduling process to finalize. Moreover, we show in section 4 that this solution is often close to the optimal. Finally, regardless of the value of K the algorithm is complete, as it is able to select more configurations than the ones initially selected if they do not lead to a valid solution. This way, if a solution exists to the problem, it is found.

3 Multiple cooperative tasks

In the previous sections, we presented a solution to the cooperative task scheduling problem with n agents involved in a unique cooperative task. This process relies on the computation of cooperation proposals for each agent involved in the cooperative task, followed by the synchronization of these proposals. Some difficulties emerge in a scenario featuring agents involved in multiple cooperative tasks. For example, a scenario in which the student and his advisor also have to meet with the head of the lab at some point. Indeed, as described in the general overview (section 3.2), this problem is handled by iterating the three first steps of the cooperative task scheduling algorithm over each successive cooperative task. However, this process supposes that a precedence order of cooperative tasks is respected in the activity description of all agents.

For example, it requires the meeting with the head of the lab to be held either only before or only after the other meeting, for both agents.

Indeed, the scheduling process begins with developing all the individual schedules leading to the cooperation situations associated with the first cooperative task. These individual schedules cannot contain cooperative task. Therefore, if two cooperative tasks do not respect a global precedence order, it is impossible to develop all the cooperation proposals for any of these tasks, as some of these proposals require taking into account the other cooperative task. In this situation, a deadlock is reached in the computation and no solution can be found. A preprocessing of the activity graphs is required to resolve this problem. In this section, we propose a way of performing the cooperative activity scheduling in both situations.

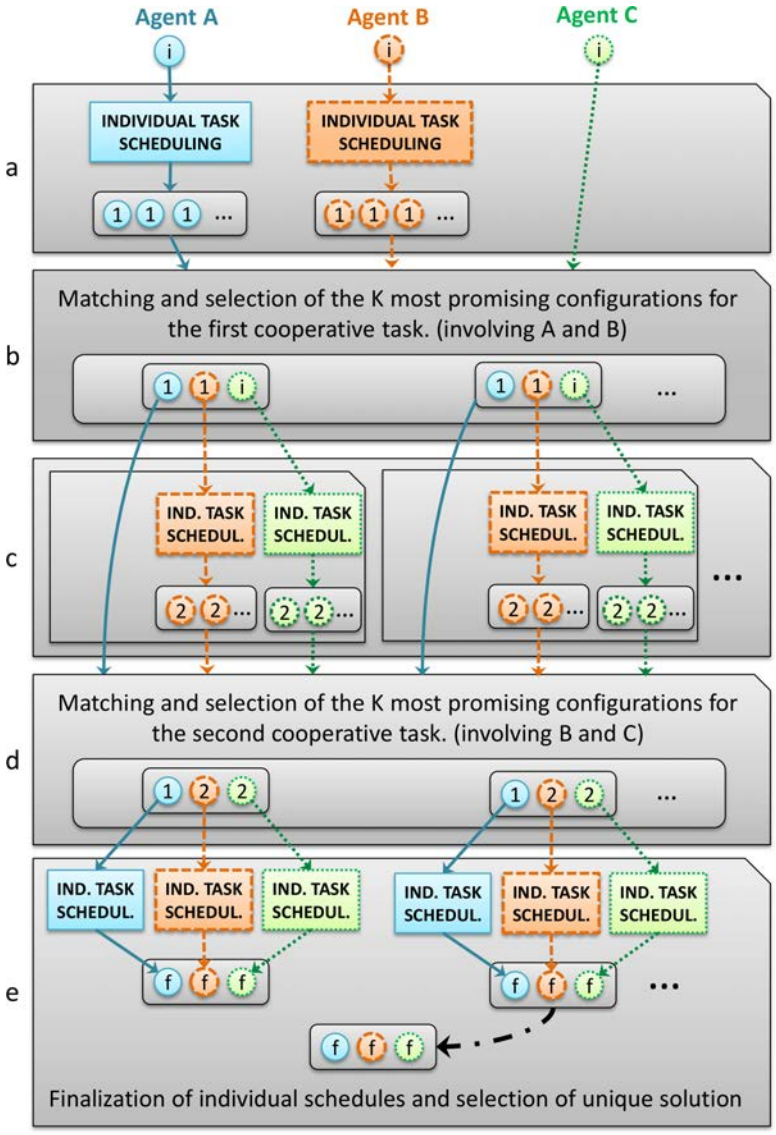


Figure 6.3 – Scheduling two successive cooperative tasks, the first one (a, b) between the agents A and B and the second one (c, d) between the agents B and C.

3.1 Cooperative activities respecting a global precedence order

If a strict precedence order of cooperative tasks is respected in the activity graphs of all agents, it means that these cooperative tasks should be performed in a defined order. In that case, the cooperative activity scheduling process iterates the three first steps of the algorithm over each successive cooperative task (see figure 6.3). For each of these tasks, only the agents involved in the task participate in its scheduling phase. For example, in figure 6.3, only the agents A and B are involved in the first cooperative task scheduling (a) and only the agents B and C are involved in the second cooperative task scheduling (c). The scheduling is performed as follow:

- The algorithm is bootstrapped by computing all the cooperation proposals for the agents involved in the first cooperative task tk_{coop} (see figure 6.3.a).

- These proposals are matched into cooperation configurations, and the K most promising are selected for each cooperative situation combination $Comb_{coop}$ (see figure 6.3.b). These configurations are completed with the initial states of the agents that were not involved in the meeting.
- For each of the selected cooperation configurations, a new step of cooperative activity scheduling is performed, using these configurations as initial states (see figure 6.3.c).
- When the $K \times |Comb'_{coop}|$ sets of cooperation configurations for the second cooperative task tk'_{coop} have been computed, the normal process would be to select the $K \times |Comb'_{coop}|$ most promising configurations from each of these sets, obtaining an increasing number of configurations to consider. Instead of that, in order to avoid an explosion of the number of configurations, we mix the cooperation configurations produced by all the cooperative activity scheduling executed in the previous step. We then select the $K \times |Comb'_{coop}|$ most promising among them. (see figure 6.3.d).
- This process is iterated over all the subsequent cooperative tasks. When the $K \times |Comb''_{coop}|$ most promising configurations are selected for the last cooperative task tk''_{coop}, the individual schedules are finalised as usual and the best final configuration is selected as the solution of the problem. If no solution is found, a backtracking method selects more promising configurations to explore, until a goal is found. This way, the completeness of the process is guaranteed.

This final solution provides an individual task schedule for each agent, including cooperative tasks which are synchronised with the other agents involved in these tasks. These individual schedules respect the constraints and preferences of all the agents that share a least one cooperative tasks. As we only select the $K \times |Comb^i_{coop}|$ most promising configurations for each subsequent cooperative task tk^i_{coop}, we reduce the overall complexity by lowering the global branching factor of the recursive iterations required to find a solution.

3.2 Cooperative activities respecting no global precedence order

In the general cooperative task scheduling case, the activity graphs do not always respect the precondition of global precedence order uniqueness. For example, in the case a cooperative task with the head of the lab can be held either before or after the meeting between the student and his adviser. In such a case, the scheduling process described in the previous section cannot be used, as it is unable to compute all the individual schedules leading to a cooperation situations allowing one of the cooperative tasks to be performed. In the previous example, we would need either to find all the ways of reaching the cooperation situation associated with the meeting between the student and his advisor or all the ways of the cooperation situation associated with the meeting with the head of the lab. However, in both these case, the individual scheduling is unable to compute all the possible tasks sequences, as some of them include the other cooperative task. In order to solve this problem, we divide it into multiple problems in which the cooperative tasks respect a global precedence order. For each activity graph that does not respect a global precedence order of cooperative tasks, we create one sub-activity graph per possible precedence order. The general problem is then solved by searching for the best sub-solution in all sub-activity graphs sharing a compatible cooperative task precedence order. Finally, the best of these sub-solutions is selected as the global problem solution.

Thanks to these methods, our cooperative task scheduling process is able to find a good solution to the problem even in complex scenarios including multiple cooperative tasks involving multiple agents, even if these tasks do not respect any global precedence order. In that case,

too, the process is complete: if a solution exists, it is found, even if it may require backtracking in the different steps of the process in order to select configurations that were not selected in the first place.

4 Results

In order to demonstrate the properties of our algorithm, we used it to solve the simple cooperative activity scheduling problem presented in the section 1 and illustrated in the figure 6.1. Both agents are allowed two different paces: walking and hurrying.

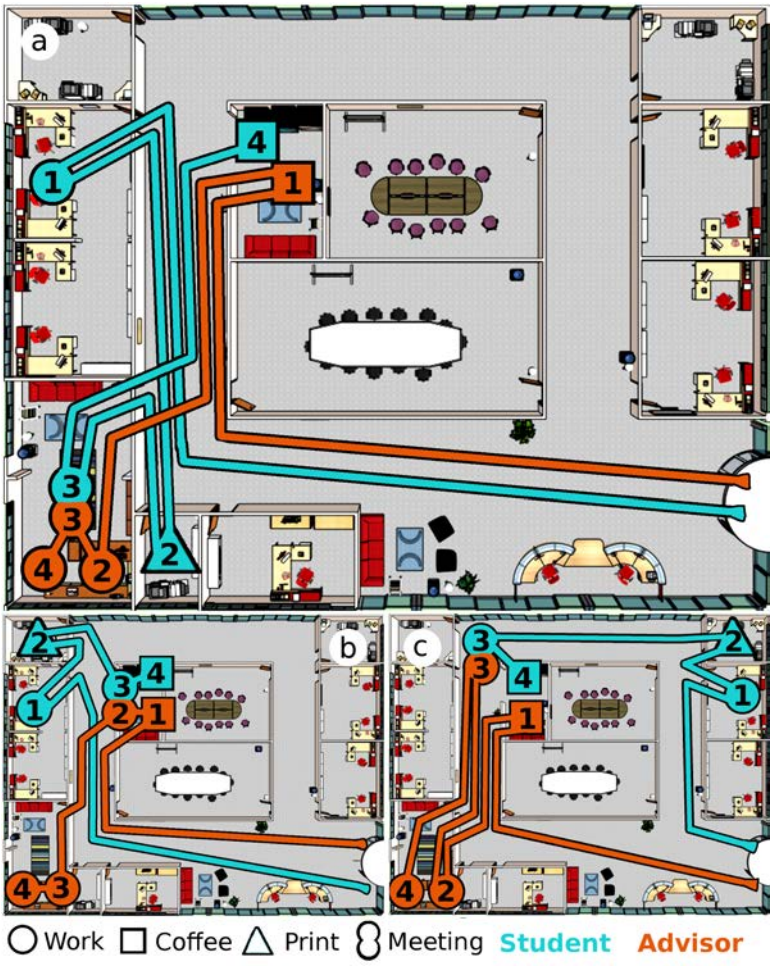


Figure 6.4 – Output of our algorithm when resolving our example scenario with slightly different setups.

4.1 Effect of constraints and initial setup on scheduling.

Using our algorithm, we scheduled the agents' individual tasks as well as the cooperative task. In the initial setup, both agents start their activities at 10:00 am from the entrance of the building ($l0$ in figure 6.1). Both agents are assigned a personal office ($l11$ for the student and $l4$ for the advisor) in which they can perform their *work* task.

Figure 6.4.a shows the output of our algorithm. The two paths indicate in which order the tasks are scheduled and the location where these tasks have to be performed. It is interesting to note that the student prints the document in the printer room next to his advisor's office and not in the one closer to his own. He also chooses to have his coffee after the meeting to avoid making his advisor wait for him. This forces him to take a longer path, but minimises the global effort required by the solution. This shows how our algorithm enables one agent to take the location and activity of other agents into account in their own task scheduling. The same scenario has been used with two slightly different setups. In the first case (see figure 6.4.b), the advisor's start time is moved forward at 10:15 am. In that case, the student has enough time to finish and print his document before his advisor finishes his coffee. In that case, the two agents meet in front of the coffee machine, avoiding extra travel for both of them. The student reacts to this fact by choosing the printing room next to his office and to the coffee machine. In the second case (see figure 6.4.c), the setup is the same as the initial one, but the personal office of the student is located in $l8$. In that case, even if the order in which the tasks are performed remains the same. As his work is done, the advisor leaves his office to meet with his student on his way to the coffee machine. It is also noteworthy that the student hurries on his way to this meeting to avoid making his advisor wait for too long. Both these examples show how our algorithm adapts a scenario to different temporal and spatial setups. The individual activities and characteristics of both agents are taken into account to select a good meeting compromise.

Figure 6.5 shows the result of the activity scheduling for four agents involved in a scenario including two cooperative tasks. Agent 1 has to finish and print a paper, meet agent 2 and then meet all the other agents in one of the meeting rooms. He also has to have a coffee. Agent 2 has to work in his office, have a coffee and meet agent 1 before meeting the other agents. Agent 3 has to finish and print a paper before meeting the other agents. Agent 4 just has to meet the other agents in one of the meeting rooms. We see that the choice of meeting room takes the activity of all involved agents into account. We also note that agent 1 prefer to delay his coffee break to avoid the other agents wait for him. This demonstrate the capability of our algorithm to find good compromises of locations and times that takes all the agents activities and constraints into account. These results demonstrate the good properties of our cooperative activity scheduling process. This process is able to schedule multiple cooperative activities involving different groups of agents. The computed individual schedules respect the activities and constraints of all the agents involved in the cooperative tasks. The locations where these cooperative tasks should be performed are selected in a way that minimizes the combined required effort. The process is able to seamlessly adapt to changes in the activity description as well as in spatial and temporal constraints. It makes it especially useful for adapting complex scenarios including loosely constrained cooperative actions.

4.2 Computation time.

The complexity of the cooperative activity scheduling problem depends on three main parameters: $|L_{coop}|$, the number of locations in the environment where the cooperative task can be held, $|Scoop|$, the number of occurrences of cooperation situations associated with this task in each agent's activity and $|A_{coop}|$, the number of agents involved in the task. To study the impact of these parameters, we used a large environment featuring 502 locations, in which we constrained the cooperative task to either to 20, 153 or all of these locations. We embedded the agents with an activity graph featuring a cooperative task that can be performed in 4, 8 or 16 cooperative situations for each agent. We solved the problem with 2, 4 or 6 agents involved in the cooperative task and a fixed value of $K = 4$. Table 6.1,column ct_p sums up the impact of these parameters on the mean computation time. We also solved these scheduling problems with filtering techniques disabled to evaluate their impact (see table 6.1, column ct_{np}). We observe

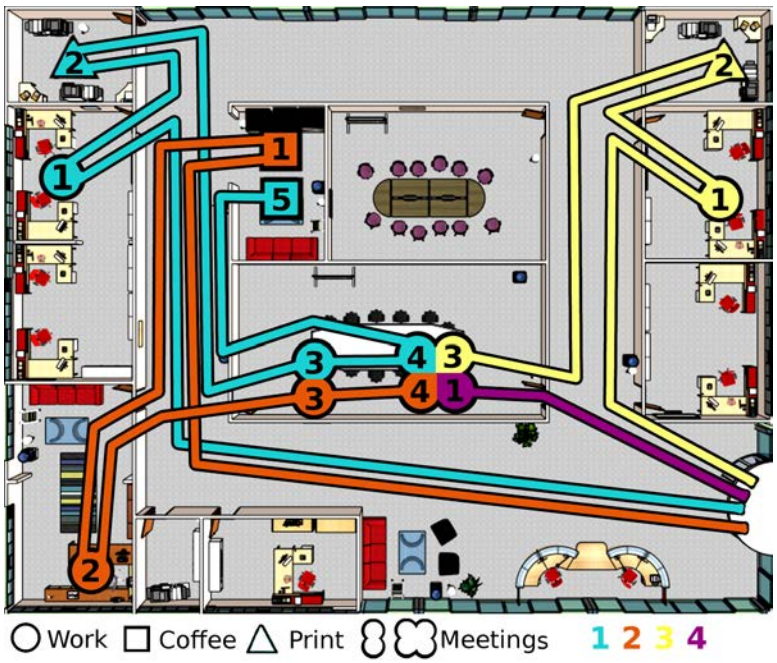


Figure 6.5 – A simple example featuring four agents involved in multiple cooperative tasks.

that the impact of the number of locations in which the meeting can be held is greatly amortized by our filtering techniques. This is due to the fact that the number of selected promising cooperation configurations remains the same, independently of this parameter. However, the number of selected cooperation configurations is directly correlated with the number of cooperation situations and with the number of involved agents, resulting in an exponential increase of the computation time when filtering techniques are disabled. Nevertheless, it is noticeable that our algorithm computation time does not evolve at such a rate. This is explained by the use of the pruning technique during the scheduling finalization, which removes a great number of the selected configurations without developing them. This demonstrates the efficiency of our filtering methods. A solution is found in a reasonable time even in the case of high dimensional spaces (24 dimensions in the case of 6 agents).

4.3 Effect of the number of selected promising configurations.

For every combination of cooperation situations, we select a subset of the K most promising cooperation configurations. The value of K is arbitrarily chosen when launching the scheduler. The higher this value, the closer to the optimal the final solution is, at the cost of a higher computation complexity. To evaluate the impact of K, we generated 100 different setups in our large environment, with $|L(coop)|$=502 and $|S(coop)|$=16. The impact of K is evaluated using different values: 1, 2, 4, 8 and max, which means that all cooperation configurations have been explored. The table provided in Fig. 6.2 sums up the obtained results. It details the values of K, the mean final costs obtained for the associated value of K and the percentage of optimal results obtained for 100 different runs. We observe that only exploring the best configuration for every combination of situations is sufficient to get a schedule with a cost value very close to the optimal, and even optimal in 94% of the runs. This proportion reaches 98% for K=4 and 100% for K=8. This shows that close-to optimal results can be obtained even when exploring a small subset of cooperation configurations.

$\|L_{coop}\|$	$\|S_{coop}\|$	$\|A_{coop}\|$	$\|C_g\|$	$\|C_s\|$	$\|C_p\|$	$\|C_p\|/\|C_s\|$	ct_p	ct_{np}
20	4	2	736	64	60	93.75 %	0.119 s	1.269 s
153	4	2	4 120	64	58	90.22 %	0.132 s	2.552 s
502	4	2	20 108	64	56	87.5 %	0.161 s	5.102 s
20	8	2	2 856	256	251	98.42 %	0.311 s	9.275 s
20	16	2	8 939	1 024	1 017	99.52 %	1.119 s	58.361 s
20	4	4	11 776	1 024	1 011	98.73 %	0.316 s	14.873 s
20	4	6	192 192	16 384	16 361	99.86 %	1.901 s	254.319 s
153	8	4	1 058 752	16 384	16 358	99.85 %	2.427 s	>20 min

Table 6.1 – Number of generated ($|C_g|$), selected ($|C_s|$) and pruned ($|C_p|$) cooperation configurations, percentage of pruned configurations ($|C_p|/|C_s|$) and computation times with filtering techniques (ct_p) or without (ct_{np}) relatively to the number of cooperation-allowed locations $|L_{coop}|$, to the number of cooperation situations $|S_{coop}|$ and to the number of agents $|A_{coop}|$. The value of K is fixed at 4 for all these setups.

K	*mean cost*	$p(opt)$
1	3328	94%
2	3319	95%
4	3313	98%
8	3310	100%
max	3310	100%

Table 6.2 – Mean path global cost and proportion of schedules with a global cost equal to the optimal $p(opt)$ for different values of K

5 Conclusion

In this chapter, we proposed a process that finds a close-to-optimal solution to the problem of scheduling activities involving cooperative tasks. Compared to the literature, our cooperative activity scheduling process addresses a more general problem: cooperative and individual tasks are scheduled concurrently, it handles task interdependencies and takes into account travels between locations where tasks are performed as well as agents' personal characteristics. Finally, optional temporal and spatial constraints are also handled. Our algorithm generates a tasks sequence for each agent that includes precisely synchronized cooperative tasks. As a consequence, virtual actors exhibit spatially and temporally consistent behaviours.

Because we schedule most of the agents' tasks independently and use filtering techniques to greatly reduce the search complexity, we are able to cope with high-dimensional problems in interactive time. However, some complexity issues still exist. Despite our filtering functions, the order of magnitude of the number of cooperation proposals remains exponential in the number of agents. Even if the computation is efficient, the exponential nature of the problem is still perceptible particularly if many cooperation configurations lead to a scheduling failure. We believe that such a process can help to more effectively schedule actors' activity in complex scenarios involving cooperative tasks between multiple agents. This is enforced by the fact that this process automatically adapts the realization of agents' activities to any environment topology. This makes it a good choice for dealing with virtual actors evolving in procedurally generated or user-generated virtual worlds.

Conclusion

In this thesis, we proposed a model that handles the activity scheduling and path planning of pedestrians in urban environments. This model focuses on the automatic generation of credible activity schedules and of adaptive paths in virtual cities. It has been designed to drive a low-level navigation and animation process by producing consistent agents' behaviours. This model relies on three main properties. First, it considers an environment representation relying on usual concepts, enabling the use of human-inspired behavioural models. Second, it relies on independent representations of environments, activities and agents, enabling a seamless adaptation of the simulation to new agents, activities or unknown environments. Finally, it takes into account the strong relationship between the space, the time, activities and individual characteristics in order to produce long-term activity schedules and paths that satisfy spatially and temporally constrained activities. Thanks to these properties, our model is able to produce a wide diversity of complex behaviours such as hurrying in order to avoid a latter detour, visiting a distant cluster of locations instead of closer scattered locations or making a detour home to drop groceries bags before resuming activity.

1 Main contributions

Our model consists of four original independent processes: an environment abstraction process: a hierarchical path planning process, an individual activity scheduling process and its adaptation to cooperative activities scheduling. A controller combines these processes at runtime, handling the navigation and activity realisation of agents and triggering the execution of the different components of the model as required.

Our environment abstraction process automatically extracts a semantically coherent hierarchical representation of an urban environment from an informed geometry of the environment inspired by city planning and architecture concepts. It only requires as an entry a raw geometry of the environment labelled with some basic information on the nature of navigation zones. A hierarchically informed roadmap is extracted, providing the agents with different levels environment abstraction, each being adapted to different levels of decision making. The fact that these levels of abstraction match human representations makes it suitable for scheduling and path planning methods inspired by human decision making.

Our hierarchical path-planning process takes advantage of the hierarchical nature of the environment representation to delay local decisions until relevant information is available. This process takes the structure and nature of the navigation surface into account. It produces a coarse path at the city level, which is refined as the agent navigates toward its goal. It provides the agent with local path options to choose from, associated with estimations of the long-term impact of these choices. These path options are updates during the simulation when unexpected events are detected, without requiring full replanning. This way, the agent is able to seamlessly select a path through the environment, taking local dynamic events as well as long-term consequences of related decisions into account.

Our individual activity scheduling process selects an optimal sequence of tasks to perform compatible with an intended activity description. Locations in the environment where to per-

form these tasks are selected, and a relaxed temporal schedule is produced. This process handles the tight relationship between the time, the space and the activity. It takes into account the structure and nature of the environment, as well as spatial and temporal constraints associated with the agents intended activities. It is also able to consider the agents personal characteristics and preferences, automatically producing variability in the execution of similar activities. We demonstrated through an experiment the validity of this model. Comparisons between activity schedules produced by humans, by our model and by other scheduling strategies shows that the different properties of our model allow the generation of agents' behaviours that are statistically consistent with the ones produced by humans in the same situations.

Our cooperative activity scheduling process focus on activities that include cooperative tasks requiring coordination between multiple agents. This process produces individual activity schedules for all the agents involved in cooperative activities, making sure that cooperative tasks are spatially and temporally coordinated between all the produced schedules. The produced solution is a good compromise, taking into account the situations, constraints and preferences of each agent involved in the task. We tackle the complexity inherent to the high dimensional nature of this problem by separating the scheduling of individual and cooperative tasks and by pruning sub-optimal solutions. Though the process is able to find an optimal solution, it is also able to find a close-to optimal solution in a fraction of the time. This compromise between optimality and efficiency can be tuned depending on the problem.

The model we propose combines these processes in order to handle the rational part of pedestrians' navigation behaviour in virtual cities, from environment analysis to activity scheduling and to hierarchical path planning. This model is able to produce consistent pedestrian behaviours, guiding the simulation of large crowds in which macroscopic phenomena automatically emerge. For example, higher densities of people may be noticeable front of schools around their end time and flows of people may emerge between housing and work areas at specific hours. This makes our model especially suitable for applications that require the generation of realistic pedestrian flows and densities in cities, such as city planning validation, for example. However, its use is not limited to city planning validation. Indeed, it is able to drive a wide range of applications, from the simulation of thousands of agents performing daily activities in a city to the automatic generation of a complex story including interactions between agents.

2 Future works

While we believe that our model is able to enhance agents' behaviours in a wide variety of applications, we think that it is possible to enhance it through extensions that could be the subject of future works. First, our model could be adapted to enable the description of multimodal transportation. Second, we think that the quality of the generated behaviour could be enhanced by taking into account the imperfect nature of human knowledge, by automatically tuning the parameters of the model or by coupling it with decisional models. Finally, we think that a more thorough validation of the generated crowd would be required.
Our model focuses on pedestrian behaviours, and, in the examples we used, we only considered people navigating by foot. However, in cities, people use numerous modes of transport, being either public transports or personal vehicles. Including multimodal transportation to our model would enhance the consistency of generated behaviours. On the first hand, the description of public transports such as buses or metros should be possible in our model. Indeed, it only requires the description of temporally constrained tasks displacing the agent to specified locations in a given time. However, with the addition of such tasks, our heuristic function would not remain admissible. Specific heuristic strategies could be proposed, such as taking public transportation into account only when the estimated travel time exceeds a given

value. On the other hand, the adjunction of individual vehicles such as cars or bikes would be more complicated. Indeed, in the case the agent uses his car and parks it at a given location, it must go to the same location to retrieve it. For this reason, the position of the vehicle would have to be considered as a kind of dynamic spatial constraint in the scheduling. This means that an extra dimension should be added to the search space for every vehicle available to the agent in order to keep track of all the possible locations the vehicle can be parked at. In order to avoid the complexity inherent to this problem, we believe that we could find use a method inspired by our cooperative activity scheduling process, selecting only a reduced number of promising locations to park vehicles and comparing the resulting schedules. However, the implications of such a method are still unclear and would require to be considered with more attention.

Our model does not take into account the incomplete nature of human knowledge. Indeed, it considers that the agents know the positions of all the locations and paths in the city as well as the temporal constraints associated with all these locations. In real situations, people have incomplete or even incorrect knowledge of their environment which greatly impacts their behaviour. For example, people miss some opportunities because they ignore the existence of a given location or, on the contrary, they go to locations that no longer exist. People also demonstrate exploration behaviours when looking for a kind of location in a little known environment. We would like to extend our model in order to be able to generate such behaviours. Associating confidence values to locations in the environment could represent the fact that people are unsure of the existence of these location as well as the supposition that such a location should exist in a given area. The activity scheduling process should be adapted to take into account these confidence values and the potential planning failures they induce. Specific strategies should also be proposed to handle exploration behaviours. Furthermore, even if our model is able to react to unexpected failures such as a closed location, it was not designed to react to unexpected opportunities such as the discovery of a more convenient location. Adding some opportunism to our model would add the possibility to improve the activity schedule depending on discovered opportunities. Specific process would be required, checking the impact of these new opportunities on the global schedule adapt this schedule if required.

Our model relies on a high number of parameters. Some of these parameters are objective values, such as the number of people living in a house, the time required to buy some bread or the average walking speed of a person. These parameters can either be arbitrarily defined or deduced from existing statistics. For example, statistics on the socio-professional categories of people in a city, their usual time constraints and the kind of housing they usually habit may be provided by the city administration. Such data should be used to automatically populate the city with more accurate repartitions of agents, enabling the emergence of more realistic global phenomena. However, some other parameters, such as the supplementary effort caused by walking with grocery bags, by walking on the road or by hurrying are far more subjective. Once again, these values can be arbitrarily defined, with no guarantee on the credibility of the produced behaviours. We believe that the values of these parameters could be tuned automatically by performing statistical studies on people preference or by comparing the generated flows in our crowds to real flows of people. The obtained parameters values would reflect more accurately the preferences of real agents, enabling the generation of more credible behaviours.

Our activity scheduling model relies on an intended activity descriptions that list all the possible ways of reaching a goal situation. However, it is not designed to handle conditional propositions such as "if the agents has cash, it can go shopping". In order to bring more flexibility to the model and expressiveness to the activity description, we would like to extend

our model so it can handle these conditional propositions. This would not only require the adjunction of these propositions to the activity description. It would also require taking into account the state of the world throughout the activity scheduling. However, this would impact the scheduling performances. At first glance, each considered proposition would require the adjunction of a dimension to the search space. Even if these dimensions were boolean, it would induce a critical increase of the scheduling complexity. This problem should be studied more in detail in order to find solutions with a lower impact on performances. Indeed, handling conditional propositions would be a first step to the coupling of our model with goal-oriented models that rely on the description of such propositions, such as STRIPS planner, HTNs or BDI models. We believe that such a coupling would enable to benefit of the good properties of both models by combining task planning and scheduling. For example, we could use of a SHOP model [Nau et al., 1999] to generate all the plans leading to the goal situation, which we could use to automatically produce conditional activity graphs adapted to the world. This would relieve the user of the manual description of activities, and would enable the generation of new activities in case of a scheduling failure. Furthermore, coupling our model with other kinds of decisional models could also bring a part of reactivity to the model, for example enabling the agent to perform optional tasks (have a drink ...) if he has some available time before performing its next task.

This kind of model would be a really powerful tool for the high-level authoring of complex scenarios in the field of virtual storytelling. It would enable virtual actors to mix a "normal" daily activity with specific events depicted in a scenario. That way, complex scenarios could be automatically adapted to a-priori unknown environments or agents. We believe that such a system could be especially useful to automatically integrate procedural stories and quests in video games or serious games that make use of procedurally-generated or user-generated environments.

The main issue of our work remains the global validation of the produced crowds. Indeed, if the validation of the visual credibility of a crowd requires a perceptive study, the validation of the consistency of the produced flows is far more complex. This is due to the fact that our model focusses on the generation of long-term behaviours in large environments, involving high numbers of agents. The validation of such a large-scale model would require a tedious comparison with real population data over full days in whole city blocks. This data rarely exists, and if it does, is not publicly available. The production of a large-scale study would be required, monitoring the real daily behaviours of high number of people. This would require substantial human and material resources and would face numerous privacy issues. And, even with such a study, the analysis of the obtained data would be extremely complex, as people are not neutral toward their environment. Indeed, in real situations, people's behaviours are biased by their habits or by the incomplete nature of their knowledge. The validation of our model is, in itself, a non-trivial subject of study.

Examples of input data

A

1 Example informed geometry

Figure A.1 – Example of an informed geometry of the environment. Navigation surfaces are labelled through the use of textures. For clarity purpose, only the navigation surfaces are shown here.

2 Example information database description

```
<!-- Mapping between informed zones of the environment
and avalaible tasks in those locations -->

<zoneTaskMapping>

  <location type='buildingHouse'>
    <task name='house'/>
  </location>

  <location type='buildingCommercial'>
    <task name='work'/>
  </location>

  <location type='buildingSchool'>
    <task name='school'/>
  </location>

  <location type='amenityBank'>
    <task name='bank'/>
  </location>

  <location type='shopBakery'>
    <task name='bakery'/>
  </location>

  <location type='shopButcher'>
    <task name='butcher'/>
  </location>

  <location type='shopChemist'>
    <task name='skip'/>
  </location>

  <location type='shopMall'>
    <task name='mall'/>
  </location>

</zoneTaskMapping>
```

.

3 Example task description

```
<!-- Description of a task consisting in buying bread
   in a bakery. This bakery is open from 8am. to 6pm.
   Buying bread takes 4 minutes, except between 4pm.
   and 6pm. A t this time, 6 minutes are required.
   Carrying bread increase the cost of subsequent
   actions by 0.5
-->

<taskDescription>

  <task name='bakery' effort_weight='0.5'>
    <!-- description of opened hours -->
    <opened>
      <timeInterval>
        <time day='0' hour='8' minutes='0' seconds='0'/>
        <time day='0' hour='18' minutes='0' seconds='0'/>
      </timeInterval>
    </opened>
    <!-- description of cost  -->
    <cost value='1'>
      <timeInterval>
        <time day='0' hour='8' minutes='0' seconds='0'/>
        <time day='0' hour='18' minutes='0' seconds='0'/>
      </timeInterval>
    </cost>
    <!-- description of task duration, depending on hours -->
    <duration day='0' hour='0' minutes='4' seconds='0'>
      <timeInterval>
        <time day='0' hour='9' minutes='0' seconds='0'/>
        <time day='0' hour='19' minutes='0' seconds='0'/>
      </timeInterval>
    </duration>
  </task>

</taskDescription>
.
```

4 Example activity description

```
<activityDescription>

  <!-- Description of an activity consisting in buying food by
     either taking money, going to the bakery and the butcher
     or going to the mall.
     param0 : the home location
  -->

  <activityGraph name='buyFood' parameters='1'>
    <either>
      <sequence>
        <task name='bank' />
        <interlaced>
          <task name='bakery' />
          <task name='butcher' />
        </interlaced>
      </sequence>
    <task name='mall' />
    </either>
  </activityGraph>

  <!-- Description of an activity consisting in taking children
     at school, buying some food and going back home.
     param0 : the home location
     param1 : the school location
  -->

  <activityGraph name='takeChilrenAndBuyFood' parameters='2'>
    <sequence>
      <interlaced>
        <!--<graph name='takeChildren' param0='1'/>-->
        <task name='school' location='1' />
        <graph name='buyFood' param0='0' />
      </interlaced>
      <task name='house' location='0' />
    </sequence>
  </activityGraph>

</activityDescription>
```

.

Activity scheduling process validation

B

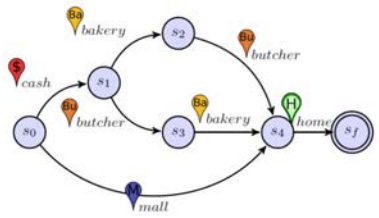
1 Experiment explanatory sheet

Experiment:

Plan a path in a city allowing performing a set of tasks while minimizing the global effort

Activity to perform:

- Buy food, either by going to the mall (M) or by getting some money at a bank ($) then going, in any order, to the Bakery (Ba) and Butcher (Bu).
- At any point of this activity, pick your child at school (S).
- Note that navigating with shopping bags is harder, but that you can drop them at home at any time.



Experimental setup:

Navigation speed: you can use three different paces: walk, hurry and run. You will be given a ruler showing, for each of these paces, the approximate time required to travel a given distance.

Note that hurrying and running are factors of effort.

Start: Work (W)

Goal: Home (H)

Tasks:

H: dropping bags at Home, 1 min duration to drop bags

M: buy food at the Mall, 8 min duration, heavy load (4 kg)

Ba: buy bread at the Bakery, 4 min duration, medium load (2 kg)

Bu: buy meat at the Butcher, 4 min duration, medium load (2 kg)

$: get some money at the Bank, 2 min duration

S: pick child at school at 4.30 pm, 5 min duration, don't be late!

Figure B.1 – The explanatory sheet provided to people participating to the experiment.

2 Setup example

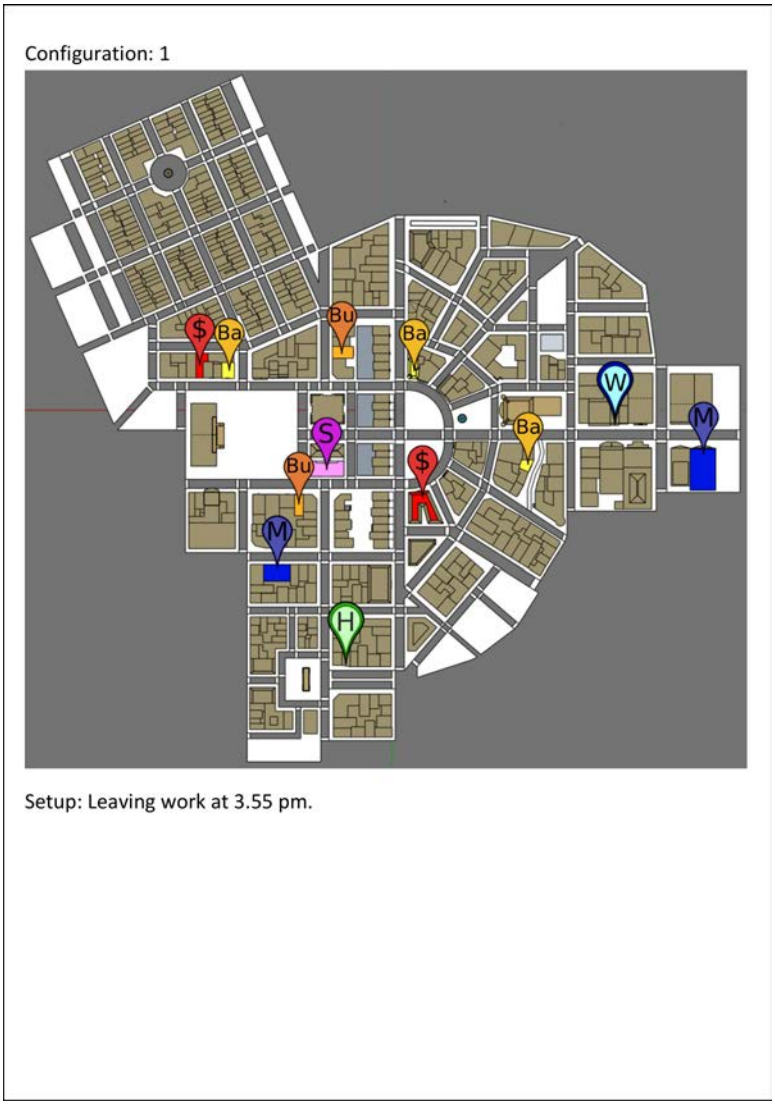


Figure B.2 – Example of scheduling setup provided to people participating to the experiment.

Publications

1. Jørgensen, C.J. and Lamarche, F., *From Geometry to Spatial Reasoning: Automatic Structuring of 3d Virtual Environments*, Motion in Games, 2011.
2. Jørgensen, C.J. and Lamarche, F., *Combining Activity Scheduling and Path Planning to Populate Virtual Cities*, International conference on Autonomous agents and multi-agent systems, 2013, pages 1129-1130.
3. Jørgensen, C.J. and Lamarche, F., *Semantically Consistent Hierarchical Decomposition of Virtual Urban Environments*, Smart Graphics, 2014, pages 133-145.
4. Jørgensen, C.J. and Lamarche, F., *Space and Time Constrained Task Scheduling for Crowd Simulation*, Technical Report, IRISA, 2014.

Bibliography

[Abaci et al., 2004] Abaci, T., Ciger, J., and Thalmann, D. (2004). Speculative planning with delegation. In *Cyberworlds, 2004 International Conference on*, pages 100–106. IEEE. 21

[Abaci et al., 2005] Abaci, T., Ciger, J., and Thalmann, D. (2005). Planning with smart objects. *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. 21

[Allbeck, 2010] Allbeck, J. M. (2010). Carosa: A tool for authoring npcs. In *Motion in Games*, pages 182–193. Springer. 107

[Arikan et al., 2001] Arikan, O., Chenney, S., and Forsyth, D. A. (2001). Efficient multi-agent path planning. In *Computer Animation and Simulation '01*, pages 151 – 162. 11

[Ayan et al., 2007] Ayan, N. F., Kuter, U., Yaman, F., and Goldman, R. P. (2007). Hotride: Hierarchical ordered task replanning in dynamic environments. In *Planning and Plan Execution for Real-World Systems–Principles and Practices for Planning in Execution: Papers from the ICAPS Workshop. Providence, RI*. 38

[Badawi and Donikian, 2004] Badawi, M. and Donikian, S. (2004). Autonomous agents interacting with their virtual environment through synoptic objects. *CASA 2004*, pages 179–187. 21

[Badler et al., 2000] Badler, N., Bindiganavale, R., Allbeck, J., Schuler, W., Zhao, L., Lee, S., Shin, H., and Palmer, M. (2000). Parameterized action representation and natural language instructions for dynamic behavior modification of embodied agents. In *AAAI Spring Symposium*, volume 147. 20

[Badler et al., 1997] Badler, N. I., Reich, B. D., and Webber, B. L. (1997). Towards personalities for animated agents with reactive and planning behaviors. In *Creating Personalities for Synthetic Actors*, pages 43–57. Springer. 20

[Badler et al., 1995] Badler, N. I., Webber, B. L., Becket, W., Geib, C. W., Moore, M. B., Pelachaud, C., Reich, B. D., and Stone, M. (1995). Planning and parallel transition networks: Animation's new frontiers. *Center for Human Modeling and Simulation*, page 91. 31

[Barraquand and Latombe, 1991] Barraquand, J. and Latombe, J.-C. (1991). Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649. 8

[Bayazit et al., 2002] Bayazit, O. B., Lien, J.-M., and Amato, N. M. (2002). Roadmap-based flocking for complex environments. In *Computer Graphics and Applications*, pages 104 – 113. 9

[Beer and Gallagher, 1992] Beer, R. D. and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive behavior*, 1(1):91–122. 30

[Bhattacharya et al., 2010] Bhattacharya, S., Likhachev, M., and Kumar, V. (2010). Multi-agent path planning with multiple tasks and distance constraints. In *ICRA*, pages 953–959. IEEE. 40

[Blum and Furst, 1997] Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial intelligence*, 90(1):281–300. 34

[Blumberg, 1996] Blumberg, B. M. (1996). *Old tricks, new dogs: ethology and interactive creatures*. PhD thesis, Massachusetts Institute of Technology. vi, 31

[Boissonnat et al., 1998] Boissonnat, J.-D., Yvinec, M., and Brönnimann, H. (1998). *Algorithmic geometry*, volume 5. Cambridge university press Cambridge. 15

[Bonet and Geffner, 2000] Bonet, B. and Geffner, H. (2000). Planning as heuristic search: New results. In *Recent Advances in AI Planning*, pages 360–372. Springer. 35

[Botea et al., 2004] Botea, A., Müller, M., and Schaeffer, J. (2004). Near optimal hierarchical path-finding. *Journal of game development*, 1(1):7–28. 23, 55

[Bowring et al., 2005] Bowring, E., Tambe, M., and Yokoo, M. (2005). Optimize my schedule but keep it flexible: Distributed multi-criteria coordination for personal assistants. In *Persistant Assistants: Living and working with AI, workshop at the AAAI Spring Symposium*. 40

[Brand and Bidarra, 2011] Brand, S. and Bidarra, R. (2011). Parallel ripple search–scalable and efficient pathfinding for multi-core architectures. In *Motion in Games*, pages 290–303. Springer. xvi, 23, 55

[Bratman, 1987] Bratman, M. (1987). *Intention, plans, and practical reason*. CSLI Publications. 35

[Brooks, 1995] Brooks, R. A. (1995). Intelligence without reason. *The artificial life route to artificial intelligence: Building embodied, situated agents*, pages 25–81. 30

[Charles et al., 2003] Charles, F., Lozano, M., Mead, S. J., Bisquerra, A. F., and Cavazza, M. (2003). Planning formalisms and authoring in interactive storytelling. In *Proceedings of TIDSE*, volume 3. 36

[Chew, 1989] Chew, L. P. (1989). Constrained delaunay triangulations. *Algorithmica*, 4(1-4):97–108. 15

[Choi et al., 2003] Choi, M. G., Lee, J., and Shin, S. Y. (2003). Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics (TOG)*, 22(2):182–203. 9

[Choset, 2005] Choset, H. M. (2005). *Principles of robot motion: theory, algorithms, and implementation*. MIT press. 8

[Coderre, 1987] Coderre, B. (1987). Modeling behavior in petworld. In *ALIFE*, pages 407–420. 31

[Coles et al., 2009] Coles, A., Fox, M., Halsey, K., Long, D., and Smith, A. (2009). Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 173(1):1–44. 41

[Cremer et al., 1995] Cremer, J., Kearney, J., and Papelis, Y. (1995). Hcsm: a framework for behavior and scenario control in virtual environments. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 5(3):242–267. 32

[Cushing et al., 2007] Cushing, W., Kambhampati, S., Weld, D. S., et al. (2007). When is temporal planning really temporal? In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1852–1859. Morgan Kaufmann Publishers Inc. 39

[Delaunay, 1934] Delaunay, B. (1934). Sur la sphere vide. a la memoire de george voronoi. 15

[Demyen and buro, 2006] Demyen, D. and buro, M. (2006). Efficient triangulation-based pathfinding. In *AAAI*, pages 942 – 947. 15

[Do and Kambhampati, 2003] Do, M. B. and Kambhampati, S. (2003). Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res.(JAIR)*, 20:155–194. 38

[Donikian, 2001] Donikian, S. (2001). Hpts: a behaviour modelling language for autonomous agents. In *Proceedings of the fifth international conference on Autonomous agents*, pages 401–408. ACM. 32

[Durupinar et al., 2008] Durupinar, F., Allbeck, J., Pelechano, N., and Badler, N. (2008). Creating crowd variation with the ocean personality model. In *AAMAS*, pages 1217–1220. 42

[Egges et al., 2004] Egges, A., Kshirsagar, S., and Magnenat-Thalmann, N. (2004). Generic personality and emotion simulation for conversational agents. *Computer animation and virtual worlds*, 15(1):1–13. 42

[Erdmann and Lozano-Perez, 1987] Erdmann, M. and Lozano-Perez, T. (1987). On multiple moving objects. *Algorithmica*, 2(1-4):477–521. 40

[Erol et al., 1994] Erol, K., Hendler, J. A., and Nau, D. S. (1994). Umcp: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, volume 94, pages 249–254. 36

[Farenc et al., 1999] Farenc, N., Boulic, R., and D.Thalmann (1999). Informed environement dedicated to the simulation of virtual humans in urban context. *Computer Graphics Forum (Proc. of Eurographics)*, 18:309–318. xvi, 22, 24, 56

[Fikes and Nilsson, 1972] Fikes, R. E. and Nilsson, N. J. (1972). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208. 34

[Funge et al., 1999] Funge, J., Tu, X., and Terzopoulos, D. (1999). Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 29–38. ACM Press/Addison-Wesley Publishing Co. 34

[Funge, 1998] Funge, J. D. (1998). *Making them behave: cognitive models for computer animation*. PhD thesis, University of Toronto. 33

[Garrido and Sycara, 1996] Garrido, L. and Sycara, K. (1996). Multi-agent meeting scheduling: Preliminary experimental results. In *Int. Conf. on Multiagent Systems*, pages 95–102. 40

[Gateau et al., 2013] Gateau, T., Lesire, C., and Barbier, M. (2013). Hidden: Cooperative plan execution and repair for heterogeneous robots in dynamic environments. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4790–4795. IEEE. 38

[Geraerts and Overmars, 2007] Geraerts, R. and Overmars, M. (2007). The corridor map method: Real-time high-quality path planning. In *ICRA*, pages 1023–1028. v, 11, 12, 13

[Geraerts and Overmars, 2006] Geraerts, R. and Overmars, M. H. (2006). Creating high-quality roadmaps for motion planning in virtual environments. In *IROS*, pages 4355 – 4361. 11

[Gibson, 1979] Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*, chapter The Theory of Affordances, page 127 143. Lawrence Erlbaum Associates. 19, 43

[Gloor et al., 2004] Gloor, C., Stucki, P., and Nagel, K. (2004). Hybrid techniques for pedestrian simulations. In *Cellular Automata*, pages 581–590. Springer. 8

[Granieri et al., 1995] Granieri, J. P., Becket, W., Reich, B. D., Crabtree, J., and Badler, N. I. (1995). Behavioral control for real-time simulated human agents. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 173–180. ACM. 30

[Grzeszczuk et al., 1998] Grzeszczuk, R., Terzopoulos, D., and Hinton, G. (1998). Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 9–20. ACM. 30

[Halsey et al., 2004] Halsey, K., Long, D., and Fox, M. (2004). Crikey-a temporal planner looking at the integration of scheduling and planning. In *Workshop on Integrating Planning into Scheduling, ICAPS*, pages 46–52. Citeseer. 39

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107. 18, 74, 93

[Hirtle and Jonides, 1985] Hirtle, S. C. and Jonides, J. (1985). Evidence of hierarchies in cognitive maps. *Memory & cognition*, 13(3):208–217. vi, 2, 24, 55

[Hoff III et al., 1999] Hoff III, K. E., Keyser, J., Lin, M., Manocha, D., and Culver, T. (1999). Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 277–286. ACM Press/Addison-Wesley Publishing Co. 11

[Hoffmann and Nebel, 2001] Hoffmann, J. and Nebel, B. (2001). The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, pages 253–302. 35

[Hoogendoorn and Bovy, 2004] Hoogendoorn, S. P. and Bovy, P. H. L. (2004). Pedestrian route-choice and activity scheduling theory and models. *Transportation Research Part B: Methodological*, 38(2):169–190. 85

[Hopcroft, 1971] Hopcroft, J. E. (1971). An n log n algorithm for minimizing states in a finite automaton. Technical report, Stanford University. 91

[Jaklin et al., 2013] Jaklin, N., Cook, A., and Geraerts, R. (2013). Real-time path planning in heterogeneous environments. *CAVW*, 24(3-4):285–295. vi, xiv, 22, 23, 57, 72, 88

[Jennings and Jackson, 1995] Jennings, N. R. and Jackson, A. J. (1995). Agent-based meeting scheduling: A design and implementation. *Electronics letters*, 31(5):350–352. 40

[Jiang et al., 2009] Jiang, H., Xu, W., Mao, T., Li, C., Xia, S., and Wang, Z. (2009). A semantic environment model for crowd simulation in multilayered complex environment. In *VRST*, pages 191–198. 25

[Kallmann, 2010] Kallmann, M. (2010). Navigation queries from triangular meshes. In *Motion in Games*, pages 230–241. 15, 56

[Kallmann et al., 2003] Kallmann, M., Bieri, H., and Thalmann, D. (2003). Fully dynamic constrained delaunay triangulations. *Geometric Modelling for Scientific Visualization*, pages 241–257. 15

[Kallmann and Thalmann, 1999] Kallmann, M. and Thalmann, D. (1999). *Modeling objects for interaction tasks*. Springer. 20

[Kallmann and Thalmann, 2002] Kallmann, M. and Thalmann, D. (2002). Modeling behaviors of interactive objects for real-time virtual environments. *Journal of Visual Languages & Computing*, 13(2):177–195. 20

[Kamphuis and Overmars, 2004] Kamphuis, A. and Overmars, M. H. (2004). Finding paths for coherent groups using clearance. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28. Eurographics Association. 11

[Kapadia et al., 2011] Kapadia, M., Singh, S., Reinman, G., and Faloutsos, P. (2011). A behavior-authoring framework for multiactor simulations. *Computer Graphics and Applications*, 31(6):45–55. 40, 43

[Kitazawa, 2004] Kitazawa, K.and Batty, M. (2004). Pedestrian behaviour modelling. In *Proc. of International Conference on Design and Descision Support Systems in Architecture and Urban Planning*. 85

[Korf, 1985] Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109. 19

[Kruskal, 1956] Kruskal, J. B. J. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. In *American Mathematical Society*. 38

[Kuffner and LaValle, 2000] Kuffner, J. and LaValle, S. (2000). Rrt-connect: An efficient approach to single-query path planning. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 2(Icra):995–1001. 10

[Laird et al., 1987] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64. 31

[Lamarche, 2009] Lamarche, F. (2009). Topoplan: a topological path planner for real time human navigation under floor and ceiling constraints. In *CGF*, volume 28, pages 649–658. vi, xvi, 15, 48, 60, 61, 62

[Lamarche and Donikian, 2002] Lamarche, F. and Donikian, S. (2002). Automatic orchestration of behaviours through the management of resources and priority levels. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, pages 1309–1316. ACM. 32

[Lamarche and Donikian, 2004] Lamarche, F. and Donikian, S. (2004). Crowd of virtual humans : a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum (Proc. of Eurographics)*, 23(3). 15, 23, 25, 58, 59, 62, 104

[Latombe, 1991] Latombe, J. (1991). Robot motion planning. *Kluwer Academic Publishers, Boston, MA*. 8, 15

[LaValle, 1998] LaValle, S. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Iowa State University. 9

[LaValle, 2006] LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press. 8

[Levesque et al., 1997] Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. B. (1997). Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1):59–83. 33

[Li, 2002] Li, T.-y. (2002). An incremental learning approach to motion planning with roadmap management. *Proceedings 2002 IEEE International Conference on Robotics and Automation*, pages 3411–3416. 10

[Li and Allbeck, 2011] Li, W. and Allbeck, J. (2011). Populations with purpose. *Motion in Games*, pages 132–143. 42, 88

[Li et al., 2012] Li, W., Di, Z., and Allbeck, J. (2012). Crowd distribution and location preference. *Computer Animation and Virtual Worlds*. 42

[Li et al., 2005] Li, Y., Gupta, K., and Payandeh, S. (2005). Motion planning of multiple agents in virtual environments using coordination graphs. In *ICRA*, pages 378–383. IEEE. 40

[Lien and Amato, 2004] Lien, J.-M. and Amato, N. M. (2004). Approximate convex decomposition of polygons. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 17–26. ACM. 58

[Loscos et al., 2003] Loscos, C., Marchal, D., and Meyer, A. (2003). Intuitive crowd behavior in dense urban environments using local laws. In *Theory and Practice of Computer Graphics, 2003. Proceedings*, pages 122–129. IEEE. 13

[Lozano-Perez, 1983] Lozano-Perez, T. (1983). Spatial planning: A configuration space approach. *Computers, IEEE Transactions on*, 100(2):108–120. 7

[Macho et al., 2000] Macho, S., Torrens, M., and Faltings, B. (2000). A multi-agent recommender system for planning meetings. In *Int. Conf. on Autonomous Agents, Workshop on Agent-based Recommender Systems*. 40

[Mallot, 1997] Mallot, H. A. (1997). Behavior-oriented approaches to cognition: Theoretical perspectives. *Theory in biosciences*, 116:196–220. 27

[Mattern and Sturm, 1989] Mattern, F. and Sturm, P. (1989). An automatic distributed calendar and appointment system. *Microprocessing and Microprogramming*, 27:455–462. 40

[McCarthy and Hayes, 1968] McCarthy, J. and Hayes, P. (1968). *Some philosophical problems from the standpoint of artificial intelligence*. Stanford University USA. 33

[Mekni, 2010] Mekni, M. (2010). Hierarchical path planning for situated agents in informed virtual geographic environments. In *3rd International Conference on simulation Tools and Techniques (SIMUTools)*. 15, 25

[Moreau, 1998] Moreau, G. (1998). *Modélisation du comportement pour la simulation interactive: application au trafic routier multimodal*. PhD thesis, Université Rennes 1. vi, 31, 32

[Moulin et al., 2003] Moulin, B., Chaker, W., Perron, J., Pelletier, P., Hogan, J., and Gbei, E. (2003). Multi-agent geosimulation and crowd simulation. In *Conf. On Spatial Information Theory*. 43

[Moulin and Larochelle, 2010] Moulin, B. and Larochelle, B. (2010). *Modelling, Simulation and Identification*, chapter CrowdMAGS: Multi-agent geo-simulation of the interactions of a crowd and control forces, pages 213–237. InTech. 43

[Nau et al., 1999] Nau, D., Cao, Y., Lotem, A., and Muñoz-Avila, H. (1999). Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, pages 968–973. Morgan Kaufmann Publishers Inc. 124

[Newell, 1994] Newell, A. (1994). *Unified theories of cognition*. Harvard University Press. 28

[Noser and Thalmann, 1998] Noser, H. and Thalmann, D. (1998). Sensor-based synthetic actors in a tennis game simulation. *The Visual Computer*, 14(4):193–205. 31

[Ondrej et al., 2010] Ondrej, J., Pettré, J., Olivier, A.-H., and Donikian, S. (2010). A synthetic-vision-based steering approach for crowd simulation. In *SIGGRAPH '10*. 104

[Overmars, 2002] Overmars, M. H. (2002). Recent developments in motion planning. In *Computational Science-ICCS 2002*, pages 3–13. Springer. 9

[Paris et al., 2006] Paris, S., Donikian, S., and Bonvalet, N. (2006). Environmental abstraction and path planning techniques for realistic crowd simulation. In *Computer Agents and Social Agents*. 22, 24, 59

[Paris et al., 2009] Paris, S., Mekni, M., and Moulin, B. (2009). Informed virtual geographic environements : an accurate topological approach. In *Int. Conf. on Advanced Geographic Information Systems & Web Services*, pages 1 – 6. 25, 43

[Pelechano et al., 2008] Pelechano, N., Allbeck, J. M., and Badler, N. I. (2008). Virtual crowds: Methods, simulation, and control. *Synthesis Lectures on Computer Graphics and Animation*, 3(1):1–176. 40

[Pettre et al., 2006] Pettre, J., Ciechomski, P. d. H., Maim, J., Yersin, B., Laumond, J.-P., and Thalmann, D. (2006). Real-time navigating crowds: scalable simulation and rendering. *Computer Animation and Virtual Worlds*, 17(3-4):445–455. 14

[Pettre et al., 2003] Pettre, J., Laumond, J.-P., and Simeon, T. (2003). A 2-stages locomotion planner for digital actors. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 258–264. Eurographics Association. 9

[Pettre et al., 2005] Pettre, J., Laumond, J.-P., and Thalmann, D. (2005). A navigation graph for real-time crowd animation on multilayered and uneven terrain. In *First International Workshop on Crowd Simulation*, volume 43, page 194. Citeseer. 13

[Porteous et al., 2011] Porteous, J., Teutenberg, J., Charles, F., and Cavazza, M. (2011). Controlling narrative time in interactive storytelling. In *AAMAS*, pages 449–456. 41

[Rannou et al., 2012] Rannou, P., Lamarche, F., and Cordier, M.-O. (2012). Enhancing the behavior of virtual characters with long term planning, failure anticipation and opportunism. In *Motion In Games*, pages 338–349. Springer. 37

[Rao et al., 1995] Rao, A. S., Georgeff, M. P., et al. (1995). Bdi agents: From theory to practice. In *ICMAS*, volume 95, pages 312–319. 36

[Reich et al., 1994] Reich, B. D., Ko, H., Becket, W., and Badler, N. I. (1994). Terrain reasoning for human locomotion. In *Computer Animation'94., Proceedings of*, pages 76–82. IEEE. 8

[Ren et al., 2011] Ren, Z., Yuan, J., Li, C., and Liu, W. (2011). Minimum near-convex decomposition for robust shape representation. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 303–310. IEEE. 57

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34. 30

[Saha and Isto, 2006] Saha, M. and Isto, P. (2006). Multi-robot motion planning by incremental coordination. In *Intelligent Robots and Systems*, pages 5960–5963. IEEE. 40

[Schank et al., 1972] Schank, R. C., Goldman, N., Rieger, C. J., and Riesbeck, C. (1972). Primitive concepts underlying verbs of thought. Technical report, DTIC Document. 21

[Shao and Terzopoulos, 2005a] Shao, W. and Terzopoulos, D. (2005a). Autonomous pedestrians. In *SCA*, pages 19–28. 13, 22, 27

[Shao and Terzopoulos, 2005b] Shao, W. and Terzopoulos, D. (2005b). Environmental modeling for autonomous virtual pedestrians. In *Digital Human Modeling for Design and Engineering Symposium*, pages 735–742. 13

[Shoulson et al., 2013] Shoulson, A., Gilbert, M., Kapadia, M., and Badler, N. (2013). An event-centric planning approach for dynamic real-time narrative. In *Proceedings of the Motion on Games*, pages 99–108. ACM. vi, 41, 43

[Siméon et al., 2000] Siméon, T., Laumond, J.-P., and Nissoux, C. (2000). Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493. 9

[Smith et al., 2011] Smith, S., Tumova, J., Belta, C., and Rus, D. (2011). Optimal path planning for surveillance with temporal-logic constraints. *Int. Journal of Robotics Research*, 30(14):1695–1708. 39

[Tecchia and Chrysanthou, 2000] Tecchia, F. and Chrysanthou, Y. (2000). *Real-time rendering of densely populated urban environments*. Springer. 14

[Tecchia et al., 2002] Tecchia, F., Loscos, C., and Chrysanthou, Y. (2002). Visualizing crowds in real-time. In *Computer Graphics Forum*, volume 21, pages 753–765. Wiley Online Library. 14

[Tecchia et al., 2001] Tecchia, F., Loscos, C., Dalton, R., and Chrysanthou, Y. (2001). Agent behaviour simulator (abs): A platform for urban behaviour development. *The First International Game Technology Conference and Idea Expo*, 1:18–21. 43

[Thomas and Donikian, 2000] Thomas, G. and Donikian, S. (2000). Modeling virtual cities dedicated to behavioral animation. *Computer Graphics Forum (Proc. of Eurographics)*, 19. vi, 22, 56

[Thrun and Bücken, 1996] Thrun, S. and Bücken, A. (1996). Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 944–951. 13

[Treuille et al., 2006] Treuille, A., Cooper, S., and Popović, Z. (2006). Continuum crowds. *ACM TOG*, 25(3):1160–1168. 8

[van den Berg et al., 2008] van den Berg, J., Lin, M. C., and Manocha, D. (2008). Reciprocal velocity obstacles for real-time multi-agent navigation. In *ICRA*. 104

[van Toll et al., 2011] van Toll, W. G., IV, A. F. C., and Geraerts, R. (2011). Navigation meshes for realistic multi-layered environments. In *ICRA*. 25

[Wiener and Mallot, 2003] Wiener, J. M. and Mallot, H. A. (2003). 'fine-to-coarse'route planning and navigation in regionalized environments. *Spatial cognition and computation*, 3(4):331–358. 24

[Wiggins, 1996] Wiggins, J. S. (1996). *The five-factor model of personality: Theoretical perspectives*. Guilford Press. 42

[Williams et al., 2001] Williams, B., Kim, P., Hofbaur, M., How, J., Kennell, J., Loy, J., Ragno, R., Stedl, J., and Walcott, A. (2001). Model-based reactive programming of cooperative vehicles for mars exploration. In *Int. Symp. Artif. Intell., Robotics, Automation Space*. 40

[Wong and Loscos, 2008] Wong, K. Y. and Loscos, C. (2008). Hierarchical path planning for virtual crowds. In *Motion in Games*, pages 43–50. Springer. 23

[Yahja et al., 1998] Yahja, A., Stentz, A., Singh, S., and Brumitt, B. L. (1998). Framed-quadtree path planning for mobile robots operating in sparse environments. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 1, pages 650–655. IEEE. 13

[Zunino and Campo, 2009] Zunino, A. and Campo, M. (2009). Chronos: A multi-agent system for distributed automatic meeting scheduling. *Expert Systems with Applications*, 36(3):7011–7018. 40

Abstract

Crowd simulation models usually aim at producing visually credible crowds with the intent of giving life to virtual environments. Our work focusses on generating statistically consistent behaviours that can be used to pilot crowd simulation models over long periods of time, up to multiple days. In real crowds, people's behaviours mainly depend on the activities they intend to perform. The way this activity is scheduled rely on the close interaction between the environment, space and time constraints associated with the activity and personal characteristics of individuals. Compared to the state of the art, our model better handle this interaction.

Our main contributions lie in the domain of activity scheduling and path planning. First, we propose an individual activity scheduling process and its extension to cooperative activity scheduling. Based on descriptions of the environment, of intended activities and of agents' characteristics, these processes generate a task schedule for each agent. Locations where the tasks should be performed are selected and a relaxed agenda is produced. This task schedule is compatible with spatial and temporal constraints associated with the environment and with the intended activity of the agent and of other cooperating agents. It also takes into account the agents personal characteristics, inducing diversity in produced schedules. We show that our model produces schedules statistically coherent with the ones produced by humans in the same situations. Second, we propose a hierarchical path-planning process. It relies on an automatic environment analysis process that produces a semantically coherent hierarchical representation of virtual cities. The hierarchical nature of this representation is used to model different levels of decision making related to path planning. A coarse path is first computed, then refined during navigation when relevant information is available. It enable the agent to seamlessly adapt its path to unexpected events.

The proposed model handles long term rational decisions driving the navigation of agents in virtual cities. It considers the strong relationship between time, space and activity to produce more credible agents' behaviours. It can be used to easily populate virtual cities in which observable crowd phenomena emerge from individual activities.

Résumé

Les modèles de simulation de foules visent généralement à produire des foules visuellement crédibles avec l'intention d'insuffler de la vie à des environnements virtuels. Notre travail se concentre sur la génération de comportements statistiquement cohérents qui peuvent être utilisés pour piloter des modèles de simulation de foules sur de longues périodes de temps, jusqu'à plusieurs jours. Dans les foules réelles, les comportements des individus dépendent principalement de l'activité qu'ils ont l'intention d'effectuer. La façon d'ordonnancer cette activité repose sur l'interaction étroite qui existe entre l'environnement, les contraintes spatiales et temporelles associées à l'activité et les caractéristiques personnelles des individus. Par rapport à l'état de l'art, notre modèle gérer mieux cette interaction.

Nos principales contributions se situent dans le domaine de l'ordonnancement d'activités et de la planification de chemin. Dans un premier temps, nous proposons un processus d'ordonnancement d'activités individuelles et son extension aux activités coopératives. Basé sur les descriptions de l'environnement, des activités désirées et des caractéristiques des agents, ces processus génèrent une séquence de la tâche pour chaque agent. Des lieux où ces tâches doivent être effectuées sont sélectionnés et un timing relâché est produit. Cet ordonnancement est compatible avec les contraintes spatiales et temporelles liées à l'environnement et à l'activité prévue par l'agent et par d'autres agents en coopération. Il prend également en compte les caractéristiques personnelles des agents, induisant de la diversité dans les ordonnancements produits. Nous montrons que notre modèle produit des comportements statistiquement cohérents avec ceux produits par des personnes dans les mêmes situations. Dans un second temps, nous proposons un processus de planification de chemins hiérarchique. Il repose sur un processus d'analyse de l'environnement automatique qui produit une représentation hiérarchique sémantiquement cohérente des villes virtuelles. La nature hiérarchique de cette représentation est utilisée pour modéliser différents niveaux de prise de décisions. Un chemin grossier est d'abord calculé, puis raffiné pendant la navigation lorsque de l'information pertinente est disponible, permettant ainsi à l'agent d'adapter son chemin à des événements inattendus.

Le modèle proposé gère des décisions rationnelles à long terme guidant la navigation des agents dans les villes virtuelles. Il prend en compte la forte relation entre le temps, l'espace et l'activité pour produire les comportements des agents plus crédibles de. Il peut être utilisé pour peupler facilement des villes virtuelles avec des foules au sein desquelles des phénomènes observables émergent de l'activité individuelle.