Conception et réalisation d'un environnement informatique sur la manipulation directe d'objets mathématiques, l'exemple de Cabri-graphes

Yves Carbonneaux

THESE

présentée par

Yves Carbonneaux

Pour obtenir le titre de

Docteur de l'Université Joseph Fourier - Grenoble 1

(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

Spécialité : **Informatique**

Formation Doctorale : **Recherche Opérationnelle**

Conception et réalisation d'un environnement informatique sur la manipulation directe d'objets mathématiques, l'exemple de *Cabri-graphes*.

Soutenue le 05 février 1998

Composition du jury :

Président :	**Nguyen Huy Xuong**, Professeur des Universités, Joseph Fourier
Rapporteurs :	**Michel Habib**, Professeur des Universités, Montpellier **Abdelkader Khelladi**, Professeur des Universités, USTHB, Alger
Examinateurs :	**Jean-Marie Laborde**, Directeur de Recherche CNRS, *directeur de thèse* **Claude Gomez**, Directeur de Recherche INRIA-Rocquencourt **Michel Mollard**, Chargé de Recherche CNRS, Laboratoire Leibniz
Invité :	**Charles Payan**, Directeur de Recherche CNRS, , Laboratoire Leibniz

Thèse préparée au sein du Laboratoire LEIBNIZ - IMAG

Je tiens ici à remercier l'ensemble des personnes qui, par leurs conseils, leurs encouragements et leurs remarques, ont contribué à l'aboutissement de ce travail.

A Jean-Marie Laborde, j'exprime toute ma gratitude et ma reconnaissance pour sa patiente, son aide et ses conseils avec lesquels il a su guider mes recherches.

Je remercie Abdelkader Khelladi et Michel Habib qui ont bien voulu rapporter sur mon manuscrit et pour tout le soin qu'ils ont apporté à cette tâche.

Merci à Claude Gomez, Michel Mollard et Nguyen Huy Xuong, et Charles Payan de bien vouloir participer à ce jury.

Je tiens à remercier Mourad Rafaï Madani, ami et collègue, pour son aide et ses conseils dans mes travaux de recherche.

Je souhaite ici remercier deux personnes, Safwan Qasem et Valérie Bellynck, pour les nombreuses discussions, autant scientifiques que philosophique, que nous avons partagées au cours de mes années de thèse.

Enfin, j'ai une pensée toute particulière à mes amis, du Leibniz ou d'ailleurs, pour m'avoir soutenu pendant les moments difficiles et avoir supporté mes sauts d'humeurs.

à mes grand-parents,
Maurice et Pierrette
Jean et Raymonde

à mes amis

Résumé

Notre travail s'intègre dans la conception et la réalisation d'un environnement informatique sur la manipulation directe d'objets mathématiques, en l'occurrence l'extension de *Cabri-graphes* dans le cadre du projet *Cabri*.

Dans la première partie, nous exposons le concept d'interface de manipulation directe. Nous présentons une évolution des principes associés à la manipulation directe. Nous montrons l'importance accordée à ce mode d'interaction.

Dans une deuxième partie, nous exposons des motivations associées à la mise en œuvre d'environnements informatiques sur la théorie des graphes. Nous analysons plusieurs environnements en fonction de trois critères : la visualisation, les fonctionnalités, et le mode d'interaction.

Dans une troisième partie, nous construisons un éditeur de graphes en fonction des principes de manipulation directe énoncés dans la première partie. Nous établissons les limites d'une telle interface en fonction des outils à notre disposition.

Dans la quatrième partie, plus théorique, nous présentons une étude sur le produit fibré de graphes, et sur la contraction de graphes comme un outil de visualisation de graphes.

Mots-clés :
Manipulation directe, interface de manipulation directe, conception d'interfaces, théorie des graphes, structures de données, algorithme, produit fibré, contraction de graphes.

Sommaire

Introduction

Sujet

Notre domaine d'étude est la conception et la réalisation d'outils informatiques sur la manipulation de concepts mathématiques abstraits, dans le cadre du projet *Cabri-graphes*. Nous abordons deux thèmes théoriques principaux : les interfaces de manipulation directe et la théorie des graphes. Nous présentons des environnements informatiques de manipulation de graphes. Nous construisons un éditeur de graphes. Nous illustrons l'étude de quelques problèmes de graphes (le produit fibré de graphe et la contraction/décontraction d'un ensemble de sommets) à travers notre éditeur.

Les interfaces de manipulation directe

Pour l'utilisateur, l'interface est le lieu où s'opère la communication avec le système informatique utilisé pour l'aider dans son activité. Comment se réalise cette communication ? Plusieurs modèles, formels ou non, ont tenté de répondre à cette question. Nous nous sommes intéressés aux interfaces de manipulation directe.

Dans les années 70-90, l'émergence de nouveaux concepts a permis la mise en œuvre d'interfaces résolument tournées vers l'utilisateur et fondées sur le concept de manipulation directe. Les contraintes techniques ont été des facteurs important dans le retard de la généralisation de ce type d'interface. Les chercheurs du centre de recherche de Rank Xerox à Palo Alto (Californie, Etats-Unis) ont développé de nouveaux concepts et ont construit la "Star Machine" [Smith et al. 1982], premier système d'exploitation basé sur la notion de métaphore du bureau. Malgré l'idée innovatrice de ce type d'interface, ce n'est qu'avec l'apparition du Macintosh et de son système de *bureau électronique* que ce type d'interface est devenu célèbre.

Aujourd'hui, le concept d'interface de manipulation directe est exploité dans les environnements informatiques, comme les éditeurs de texte, les éditeurs graphiques, ou les jeux vidéo et non plus uniquement dans les systèmes d'exploitation des machines. Il est relativement facile de concevoir des interfaces à désignation directe, où l'on propose à l'utilisateur d'appuyer sur une icône au lieu d'invoquer une commande via une entrée au clavier. Par contre, il est beaucoup plus difficile de réaliser une application véritablement intégrée fonctionnant en

manipulation directe. En effet, une difficulté majeure du concepteur d'interface de manipulation directe est l'identification des métaphores sur lesquelles les actions directes de l'utilisateur vont pouvoir se bâtir.

De plus, le problème de la conception d'interfaces, réellement adaptées aux utilisateurs, ne peut être résolu par l'informatique seule, ni par la psychologie seule ou l'ergonomie seule. Il ne peut être abordé qu'en intégrant les résultats et les connaissances de ces disciplines.

Intérêt des graphes et applications

Le problème des ponts de Königsberg, résolu par Euler, est le premier résultat formel de la théorie des graphes (18ème siècle). Elle s'est développée depuis la deuxième moitié du 19ème siècle, et connaît un essor depuis les années 30. Elle présente de nombreux liens avec l'algèbre, la topologie, et d'autres domaines de la combinatoire. Des applications de la théorie des graphes, et souvent aussi la motivation de nouveaux problèmes, se trouvent en informatique, recherche opérationnelle, théorie des jeux, théorie de la décision.

Les graphes représentent un instrument puissant pour la modélisation de nombreux problèmes combinatoires, qui seraient sans cela difficilement abordables par des techniques classiques comme l'analyse mathématique. En plus de son existence en tant qu'objet mathématique, le graphe peut représenter également une structure de données puissante pour l'informatique.

Les graphes sont actuellement l'outil privilégié pour modéliser des structures complexes. Ils sont indispensables pour représenter et étudier des relations entre des objets. Leurs applications sont très nombreuses dans plusieurs domaines. Les graphes interviennent dans la modélisation d'un système, en économie ou en automatique, et dans son évolution dans le temps. Les divers réseaux électriques, routiers, ou d'adduction d'eau sont des exemples de graphes. La décomposition en tâches d'un projet en informatique, dans le bâtiment ou dans les travaux publics illustre la puissance des outils des graphes. Les liens entre les informations dans les bases de données sont aussi représentés par des graphes.

Réalisation d'un environnement interactif sur les graphes

Il existe de nombreux environnements informatiques sur les graphes. La plupart ne sont que des outils pour représenter graphiquement les graphes, avec quelques fonctionnalités, pour les manipuler. Rares sont les éditeurs de graphes, permettant aussi bien la création interactive des graphes que leurs manipulations, et possédant un large éventail de fonctionnalités utiles dans les transformations, les opérations, et les calculs d'invariants sur les graphes. Une première version de *Cabri-graphes* [Habib & Laborde 83] a ainsi été élaborée.

Depuis, *Cabri-graphes* a beaucoup évolué ([Benzaken 86] et [Laborde 87]). Il est constitué d'un environnement d'édition de graphes associé à une boîte d'outils.Les outils permettent

d'effectuer différents calculs, tels que l'évaluation d'invariants de graphes (le nombre de sommets, la longueur du plus petit cycle, …). D'autres outils sont proposés pour réaliser des opérations classiques sur la structure du graphe (le graphe complémentaire, des produits de graphes, …) ou des transformations sur la représentation graphique du graphe (le miroir horizontal, le miroir vertical, la rotation, ou la visualisation d'une propriété du graphe).

Cabri-graphes est utilisé comme un outil pour la recherche, en permettant de créer rapidement des contre-exemples à des conjectures, ou de conforter le chercheur dans son approche de démonstration de théorèmes. *Cabri-graphes* favorise aussi l'apprentissage de la théorie des graphes en aidant à comprendre des propriétés sur les graphes.

Organisation de la thèse

L'étude présentée se décompose en quatre parties : une étude du concept de manipulation directe dans les interfaces homme-machine, une analyse des environnements informatiques sur la théorie des graphes, la modélisation d'une nouvelle interface sur les graphes et un approfondissement de quelques problèmes sur les graphes.

Dans le chapitre 1, nous abordons le domaine des interfaces de manipulation directe. L'objectif est de clarifier les notions associées à ce mode d'interaction. Dans un premier temps, nous examinons les principes fondamentaux associés à ces interfaces. Nous étudions ensuite les critères essentiels mis en œuvre dans leur conception et l'évolution des principes associés à la manipulation directe, exemple de WYSIWIG. Nous regardons aussi les problèmes inhérents à la manipulation directe et les solutions proposées. Dans la dernière partie, nous proposons une discussion sur l'avenir de la manipulation directe avec l'apparition des systèmes de réalité virtuelle et des progrès techniques réalisés dans la représentation et la manipulation d'objets virtuels.

Dans le chapitre 2, nous nous consacrons à l'étude des éditeurs de graphes. Nous suivons deux axes d'études. Dans le premier axe, nous exposons les motivations ayant conduit à la réalisation d'environnements informatiques sur la théorie des graphes. Les motivations sont l'utilisation de l'informatique comme une facilité d'utilisation, la visualisation de données abstraites, et aujourd'hui l'apprentissage des mathématiques discrètes. Dans le deuxième axe, nous classons les environnements informatiques existant sur les graphes en trois catégories : les bibliothèques d'algorithmes, les "*visualiseurs*" de graphes et les éditeurs de graphes. Nous analysons ces environnements selon trois critères : la visualisation, les fonctionnalités, et le mode d'interaction.

Dans le chapitre 3, nous présentons la conception et la réalisation des enrichissements apportés à un éditeur de graphes : *Cabri-graphs.* La version précédente ne modélisait que des graphes simples non orientés. Nous étudions les structures de données abstraites. Nous avons ainsi conçu une unique structure de données de type "*graphe*" afin de pouvoir manipuler tous les objets du domaine (la théorie des graphes) et des objets graphiques associés (par exemple, la flèche représentant l'orientation d'un arc). En nous aidant du modèle théorique élaboré au chapitre 1, nous avons créé une interface de manipulation directe sur une famille plus grande de graphes. Cet élargissement nous a conduits à concevoir des outils adaptés pour manipuler ces nouveaux graphes. Par exemple, de multiples représentations de l'arête par un segment, un segment brisé ou une courbe.

Dans le chapitre 4, nous traitons de quelques problèmes sur les graphes. Après une analyse de la notion d'homomorphisme sur les graphes, nous étudions le produit fibré de graphes. Pour terminer, nous agrandissons le concept de graphes, en permettant de contracter des sommets en un seul sommet. Ce nouveau sommet est associé à un sous-graphe du graphe initial. Dans ces deux études particulières sur les graphes, nous étudions la stabilité de quelques propriétés par les opérations. Nous donnons un algorithme théorique de ces opérations, qui ont été programmées dans notre environnement.

Chapitre 1

Les interfaces de manipulation directe

Depuis les années 70, l'émergence de nouveaux concepts a permis la mise en œuvre d'interfaces résolument tournées vers l'utilisateur et fondées sur le concept de manipulation directe. La littérature [Hutchins & al. 85] donne le travail de [Sutherland 63] comme la première création basée sur la notion de métaphore, la métaphore du cahier d'essai ou du cahier de brouillon. Les contraintes techniques ont été des facteurs importants dans le retard de la généralisation de ce type d'interface. Dans les années 80, des études fondamentales, menées au centre de recherche de Rank Xerox à Palo Alto (Californie, Etats-Unis), ont conduit au développement de la "*Star Machine*" [Smith et al. 82]. L'idée essentielle des chercheurs de Xerox Parc, qui ont travaillé sur la conception des interfaces homme-machine, est basée sur la notion de métaphore du bureau. Cette métaphore consiste dans la présentation sur un écran des symboles "familiers" d'objets sur un bureau. Ces objets pouvaient être sélectionnés et manipulés en les désignant avec un curseur contrôlé par une souris électronique. Malgré l'idée innovatrice de cette interface, il faut attendre l'apparition du *Macintosh* avec son système de "bureau électronique" et le développement de programmes comme *LisaDraw* et *MacDraw* pour rendre célèbre ce type d'interface.

La réalisation d'une interface centrée sur l'utilisateur constitue une véritable problématique. Au cœur de cette problématique, nous trouvons la notion de manipulation directe et le type de métaphore sur lequel elle se fonde et s'appuie [Shneiderman 82]. Shneiderman introduit les termes qui caractérisent ce type d'interface [Shneiderman 82 & 83]. Hutchins [Hutchins & al. 85] et Nanard [Nanard 90] dépassent l'aspect informatique qui caractérise ce type d'interface, et considèrent l'utilisateur comme un sujet cognitif et comme un élément d'appréciation. Le nouveau concept introduit sera celui de l'engagement direct.

L'objectif de ce chapitre est de clarifier la notion de manipulation directe en s'intéressant plus particulièrement aux interfaces appelées les interfaces de manipulation directe. Dans un

premier temps, nous examinons les principes fondamentaux associés à ces interfaces. Nous étudions ensuite les critères essentiels mis en œuvre dans leur conception. Dans la dernière partie, nous proposons une discussion sur l'avenir de la manipulation directe avec l'apparition des systèmes de réalité virtuelle et les progrès techniques atteints dans la représentation et la manipulation d'objets virtuels.

1. Les principes de manipulation directe

Nous dégageons dans ce paragraphe des conditions nécessaires pour qu'une interface soit reconnue comme interface de manipulation directe. Ces principes fournissent un cadre constructif pour nous permettre d'aborder le problème de la conception d'une interface de manipulation directe au paragraphe suivant.

1.1 La manipulation directe

1.1.1 Définition

Le terme de manipulation directe a été introduit dans la littérature par Ben Shneiderman [Shneiderman 82, 83]. Dans la définition proposée, de nombreuses idées déjà mises en application avec la *Star Machine* [Smith et al. 82] se retrouvent.

Pour Shneiderman, une interface est dite de manipulation directe si elle possède les propriétés suivantes :

1. Les objets concernés ont une représentation permanente,
2. Des actions physiques directes ou l'appui sur des boutons étiquetés sont utilisés au lieu de commandes textuelles de syntaxe compliquée,
3. Le résultat des actions sur les objets est immédiatement visible,
4. Les opérations sont rapides, incrémentales et réversibles.

Quelques précisions nous semblent nécessaires.

Au lieu d'une syntaxe compliquée, l'utilisateur utilise des actions physiques directes ou appuie sur des boutons étiquetés. Ce procédé a été traduit par le principe suivant : "voir et montrer au lieu de se rappeler et taper". Ce principe était déjà présent dans la métaphore du bureau électronique de la *Star Machine*. Sur le bureau, l'utilisateur réalise des opérations en choisissant dans une sélection proposée à l'écran. Il interagit directement au niveau de l'écran en choisissant des objets et en exécutant des opérations au moyen d'un dispositif de pointage qui

sert à désigner des éléments disposés sur le bureau. Toutes les actions sur les objets de l'écran sont effectuées par l'intermédiaire d'un outil de désignation, la souris en général, et non pas par des commandes cachées. «Les objets peuvent être compris purement au point de vue de leurs caractéristiques visibles. Les actions peuvent être comprises au point de vue de leurs effets sur l'écran.» [Smith & al. 82, pp. 260].

Les première et les troisièmes règles constituent un principe des interfaces de manipulation directe qui est le fameux WYSIWYG, de l'anglais "What you see is what you get", ou "Tel écran, tel écrit". Ce principe est attribué principalement à Don Hatfield d'IBM (Etats-Unis) pour décrire l'approche générale de ce type d'interface. Il n'y a pas de différence *significative* entre ce que l'utilisateur voit à l'écran et ce qu'il obtient après impression.

Les transformations sont dites incrémentales, c'est-à-dire qu'une légère modification de la configuration ne doit pas provoquer de changement trop violent dans l'allure des objets concernés. Cette règle souffre évidemment d'une exception pour le cas d'un modèle présentant, par nature, des discontinuités.

Les transformations doivent être réversibles, c'est-à-dire que si l'utilisateur revient en arrière en annulant physiquement une transformation (en utilisant, par exemple, l'outil classique "Annuler/Répéter"), les objets doivent revenir dans leur configuration ou leur disposition précédente.

1.1.2 Métaphores d'interaction

Dans le domaine de l'interaction homme-machine, comme dans celui de la linguistique, la métaphore est l'emploi d'un terme concret pour exprimer une notion abstraite par substitution analogique. Il existe essentiellement deux classes possibles dans le cadre d'une métaphore d'interaction [Hutchins & al. 85, pp. 317-318] :

La métaphore de la description

Dans un système traitant d'un monde abstrait, l'interaction est réalisée sous la forme d'un échange de descriptions entre l'utilisateur et le système. L'usager décrit les objets et les actions, mais il n'applique pas directement les actions sur les objets. L'interface est alors un intermédiaire entre l'utilisateur et le monde abstrait. Aucune action directe n'est possible sur l'objet. L'utilisateur n'effectue pas l'action sur un objet visible du domaine, il décrit une action sur l'objet supposé par l'intermédiaire d'un substitut qui peut être un langage de commande spécialisé ou le langage naturel, via le clavier ou la parole.

La métaphore du monde réel

L'interface s'efforce d'être la reproduction informatique du domaine de modélisation. On retrouve la métaphore du bureau électronique, rendue célèbre par la Star Machine de Rank Xerox. Dans ce type d'interfaces, les concepts du domaine, qui rappellent les propriétés du monde réel, sont présentés par des imitations ou des instances des objets abstraits du domaine modélisé. L'utilisateur agit directement sur les éléments du domaine, et ces objets abstraits se comportent comme les objets réels. Les actions s'effectuent généralement avec la souris ou tout autre dispositif de désignation. L'interface devient elle-même un monde explicitement représenté où aucun intermédiaire n'existe entre l'utilisateur et le domaine ainsi modélisé.

Depuis les deux publications de Shneiderman [Shneiderman 82, 83], plusieurs autres travaux ont été publiés pour discuter des mérites relatifs sur les interactions basées sur le langage ou l'action. Les auteurs sont tous d'accord pour assimiler les systèmes de manipulation directe avec la représentation du monde réel comme modèle.

1.2 Explications de la manipulation directe

L'équipe de Rank Xerox s'est principalement attachée à l'étude des facteurs technologiques favorisant cette nouvelle interface. Par contre, il existe d'autres facteurs que les facteurs technologiques qui favorisent la manipulation directe et qui interviennent pour créer ou empêcher la sensation du "direct".

Shneiderman [Shneiderman 87] propose un modèle psychologique du comportement de l'utilisateur. Hutchins et ses collègues [Hutchins & al. 85] se sont basés sur l'hypothèse que l'impression du "direct" est liée à des phénomènes psychologiques et cognitifs. Après Hutchins, Nanard [Nanard 90] considère l'utilisateur comme un sujet cognitif et comme un élément d'appréciation. Elle développe alors le concept d'engagement direct comme un outil de compréhension.

1.2.1 L'approche informatique

Les concepteurs de la *Star Machine* ont centré leur recherche sur l'utilisateur et la conception d'interfaces Homme-Machine en s'appuyant sur la notion de métaphore, celle d'un bureau avec ses dossiers, ses fichiers et ses outils. Ils ont proposé une nouvelle génération d'interfaces dont les principes ont été :

- une familiarité avec le modèle conceptuel de l'utilisateur,
- voir et montrer plutôt que se souvenir et frapper au clavier,
- sortie (d'impression) conforme à l'écran (WYSIWIG),
- jeu de commandes universel et générique, consistant et simple,

- interactions non modales
- adaptabilité à l'utilisateur.

C'est l'émergence du concept de métaphore du monde réel.

Ils furent sensibles à l'aspect trompeur du concept de manipulation directe, qu'ils résument de cette façon : «Quand tout devient visible, l'écran se confond subtilement avec la réalité, le modèle utilisateur s'identifie avec ce qui est à l'écran.» [Smith & al. 82, pp. 260]. Shneiderman décrit cet aspect trompeur dans des propos similaires : «La ruse, en créant un système de manipulation directe, est de proposer un modèle physique approprié à la réalité.» [Shneiderman 82, pp. 253].

1.2.2 Les approches psychologiques

Le modèle SSOA

Shneiderman [Shneiderman 83, pp. 64-65] constate que les applications, caractérisées par les propriétés citées, possèdent des attributs bénéfiques pour la manipulation :

- facilité d'usage et d'apprentissage,
- réduction et la facilité de correction d'erreurs,
- réduction de l'anxiété de l'utilisateur,
- meilleure compréhension du système.

Pour Shneiderman, la compréhension de la manipulation directe est possible en considérant le contexte du modèle SSOA ("syntactic-semantic objet-action" ou le modèle objet-action syntaxique-sémantique) [Shneiderman 87, pp. 206].

La connaissance syntaxique est une information sur le système. L'utilisateur doit se souvenir du nom et de la forme des commandes, telle que la commande "cd" sur station de travail pour accéder à un dossier. Ces commandes dépendant des types de machines, de systèmes et des applications. Cette connaissance syntaxique peut apparaître arbitraire pour un utilisateur débutant, difficile à apprendre et facile à oublier.

La connaissance sémantique est une connaissance sur le travail d'un système. Cette connaissance est composée des concepts du système dans lesquels sont inclus les objets et les actions. Les objets sont des items pouvant être manipulés et les actions sont des buts ou des méthodes et des opérations pouvant être utilisées pour manipuler les objets. Puisque la connaissance sémantique est générale, elle peut être transférée sur d'autres types de système. Etant fortement organisée, elle est facile à se souvenir et à mémoriser.

Dans ce contexte, Shneiderman explique le succès de la manipulation directe et des interfaces de manipulation directe. Les objets sont directement visualisés à l'écran et manipulables. De plus, chaque action produit un effet immédiatement visible.

Manipuler les représentations des objets peut être beaucoup plus naturel et proche pour la majorité des personnes. Les relations spatiales et les actions sont compréhensibles plus rapidement avec un aspect visuel que par une représentation linguistique. Les intentions ou les explications de problèmes amènent souvent les personnes à des représentations visuelles appropriées des objets mathématiques abstraits.

L'engagement direct

Partant de la définition de la manipulation directe définie par Shneiderman, Hutchins et ses collègues cherchent à expliquer les bases psychologiques de l'attirance des utilisateurs pour ce type d'interface. Ils examinent la notion de *direct* en détail en proposant qu'elle soit mise en relation à la fois avec les *distances* psychologiques entre les buts et les actions de l'utilisateur sur l'interface, et avec l'impression d'engagement psychologique, l'impression de contrôler soi-même directement l'ordinateur plutôt qu'à travers un intermédiaire (un langage de commande par exemple).

Distance sémantique

La distance sémantique exprime les relations entre les buts que l'utilisateur s'est fixés et les significations des expressions du langage d'interface. Elle traduit l'effort de mise en correspondance entre la connaissance sémantique que possède l'utilisateur du domaine et la connaissance sémantique qu'il possède sur le système.

Distance articulatoire

La distance articulatoire reflète les relations entre la signification d'une expression du langage et sa forme. Elle traduit l'effort de traduction entre la connaissance sémantique que l'utilisateur possède d'un système et sa connaissance syntaxique des éléments de présentation du système.

La notion de distance est fortement associée à l'application de Norman, [Norman & Draper 86], sur la théorie de l'action en sept phases, dans le domaine des interactions homme-machine. La distance s'applique sur la différence entre la manière dont l'utilisateur réfléchit sur un problème et la manière dont ce problème est représenté par l'environnement informatique. Un système qui réduit ces distances contribue à un sentiment d'engagement de la part de l'utilisateur dans ses actions. Cet engagement s'applique à un style particulier de la représentation des objets basé plutôt sur la métaphore du monde réel que sur la métaphore de la description. Les systèmes peuvent favoriser un tel sentiment d'engagement dans les actions, en

visualisant tous les objets du domaine, et en permettant à l'utilisateur de les manipuler physiquement.

Hutchins, Hollan et Norman décrivent la manipulation directe comme un concept complexe qui ne peut pas être capturé par une liste de caractéristiques superficielles, et qui est difficile à définir : *«Pour nous, la notion de "manipulation directe" n'est pas un concept unitaire, ni même quelque chose qui peut être quantifié en lui-même. C'est une notion orientée.»* [Hutchins & al 85, pp. 317]. Ils caractérisent la manipulation directe comme une façon de ressentir l'interface.

Ils concluent que les systèmes à manipulation directe vont vraisemblablement être bénéfiques dans la mesure où ils simplifient le passage entre les buts et les actions de l'utilisateur, mais avec un coût de perte de puissance de l'abstraction dans l'interaction : *«Fondamentalement, les systèmes seront bons et puissants pour des objectifs, pauvres et faibles pour d'autres.»* [Hutchins et al. 85, pp. 337].

1.2.3 L'approche cognitive

Pour Nanard [Nanard 90], la notion fondamentale intervenant dans la définition de la manipulation directe est celle d'impression d'engagement direct. Elle en donne une définition : «L'impression d'engagement direct correspond à ce que l'usager ressent lorsqu'il peut agir directement et librement sur les représentations des objets de son propre monde et percevoir de façon immédiate leurs réactions.» [Nanard 90, pp. 42-43].

Cette notion d'impression est par nature très personnelle. Elle est donc subjective, et par conséquent elle est subtile à définir.

Le terme agir s'oppose au terme décrire. La manipulation directe des objets est un procédé opposé à l'utilisation d'un langage de commande pour manipuler les objets même si la frontière entre les verbes agir et décrire est assez délicate à définir. Il faut limiter ou éliminer les intermédiaires, sans pour autant réduire la liberté de l'utilisateur. Ceci suppose la désignation directe des objets. La construction des objets devrait être l'action la plus directe. Cette construction d'un objet du domaine ne doit pas obliger l'utilisateur à nommer explicitement cet objet dans le cadre d'une utilisation future. Evidemment, l'objet construit doit avoir un nom interne permettant au noyau fonctionnel de le manipuler. La liberté d'action est parfois contradictoire avec la philosophie finale de l'environnement. Par exemple, pour un environnement destiné à l'enseignement, l'apprentissage de concepts et de méthodes impose naturellement une certaine discipline : toute liberté ne peut pas être laissée à un élève. Les réactions immédiates du système sont une caractéristique fondamentale des interfaces de manipulation directe. L'acception ou le refus d'une action doit être perceptible. Dans le cas d'une construction, l'objet se dessine. Dans le cas contraire, l'utilisateur doit être averti de ce

refus par toute sorte de retour d'information (les retours d'information sont étudiés au paragraphe 2.3).

Pour Bellemain [Bellemain 92, pp. 101], deux remarques relativement importantes s'imposent. Premièrement, Nanard ne semble pas inclure dans les actions sur les objets l'élaboration et la création de leur représentation. Il s'agit uniquement de manipuler des matérialisations d'objets, mais pas de les élaborer. Deuxièmement, le fait d'agir directement sur les objets semble plus facile à réaliser dans le cas de systèmes simples comme les systèmes d'exploitation ou plus généralement pour les manipulations de notions informatiques que dans le cas de modèles mathématiques. Agir directement est beaucoup plus délicat dans le cadre d'un environnement informatique destiné à l'apprentissage de connaissance abstraite comme les mathématiques. Dans ce dernier cas, les relations et les objets utilisés sont plus abstraits et la construction de connaissances sur ces objets et relations par la manipulation de leur matérialisation nécessite une abstraction plus importante.

Les principes de la manipulation directe ont été jusqu'à présent plutôt mis en œuvre dans la conception d'interfaces graphiques permettant la manipulation, souvent nécessaire, d'objets informatiques à l'intérieur d'environnements informatiques. Ces objets ne sont pas représentés mais plutôt symbolisés par des images, et la manipulation du symbole représente pour l'utilisateur la manipulation de l'objet lui-même. Ces manipulations sont destinées plutôt à simplifier la mise en œuvre de notions informatiques intervenant dans l'utilisation d'un système qu'à permettre la construction de connaissances relatives à ces notions. Dans le cas d'environnements destinés à l'apprentissage, la manipulation directe s'applique à des représentations d'objets mathématiques et non pas aux objets eux-mêmes.

1.3 Les évolutions de la manipulation directe

Il est important de comprendre que les caractéristiques des interfaces utilisateurs, et en particulier des interfaces de manipulation directe, sont déterminées par les possibilités technologiques. Les critères et les définitions évoluent ou se précisent dans des conditions similaires.

Dans ce paragraphe, nous allons discuter des évolutions et apporter des précisions sur le terme de manipulation directe. Ces dernières permettent d'apporter des solutions aux inconvénients de la manipulation directe. Nous commencerons par affiner les critères des interfaces de manipulation directe énoncés par Shneiderman.

1.3.1 De nouveaux principes

Les principes énoncés par Shneiderman sont aujourd'hui mis en œuvre par la plupart des environnements informatiques sans avoir nécessairement une interface de manipulation directe. Ces systèmes sont composés en général des boîtes d'outils pour la création des objets. Ils proposent des boutons sur lesquels l'utilisateur appuie afin de mettre en œuvre des fonctionnalités. Ils disposent d'une multiplicité de boîtes de dialogue afin d'échanger des informations avec l'utilisateur. Ils possèdent ainsi un emballage de manipulation directe, sans avoir les principes de la manipulation directe. C'est pourquoi quelques précisions et des règles supplémentaires paraissent nécessaires. La numérotation poursuit la liste déjà commencée au paragraphe 1.1.1 :

5. Les transformations sont contrôlables en temps réel [Laborde 95, pp. 36].
6. Le déplacement des objets concernés doit être continu.
7. Tous les objets représentés doivent être manipulables.
8. La création des objets s'effectue naturellement et directement.
9. Aucun ordre sur les éléments des procédures.

Explicitons ces propriétés.

5. Les transformations doivent être contrôlables par l'utilisateur en temps réel sans l'intervention des interactions de type modal, c'est-à-dire sans le recours systématique à une boîte de dialogue pour modifier la moindre configuration adoptée. Evidemment dans certains cas, il est difficile, voire impossible, d'éviter des interactions modales, comme dans la sauvegarde de données dans un fichier.

6. La règle 6 précise la règle 4 (énoncée par Schneiderman) pour l'opération particulière du déplacement. Les transformations appliquées aux objets consécutivement au déplacement du curseur doivent être continues. Le déplacement des objets n'est en aucune manière une animation ou une succession d'images calculées au préalable comme le font certaines applications. En effet, ces applications repèrent les objets initiaux ou l'image de départ et calculent, en fonction d'un déplacement de la souris, sans aucun déplacement visible, l'emplacement terminal des objets ou le lieu d'arrivée. Elles évaluent la distance séparant ces deux lieux et proposent une succession d'images pour simuler un déplacement [Chatty 93].

Une autre façon d'opérer pour ces applications, surtout pour les déplacements délicats, est d'effectuer un déplacement par saccade. Par exemple, pour une rotation dans le plan, un pas de déplacement, préalablement défini de 5° par exemple, permet d'imiter cette transformation. Un faux-semblant de manipulation directe est présent en proposant à l'utilisateur de modifier en temps réel ce pas de déplacement, en l'augmentant ou en le diminuant, en pressant

successivement les touches du clavier (qui sont en majorité les flèches du clavier). Ces applications offrent un simulacre de déplacement, une animation du déplacement, comme un langage de commande, et non un véritable déplacement d'objets. Une représentation permanente des objets doit être visible pour l'utilisateur lors d'un déplacement d'un ensemble d'objets.

7. Tous les objets représentés peuvent être manipulés à l'aide d'outils de désignation, en général la souris. Aucun objet, dans une interface de manipulation directe, n'est fixé a priori, sauf si cela représente une propriété de cet élément. Cette règle avec la règle 1 sont deux postulats fondamentaux dans les interfaces utilisateur : l'utilisateur voit sur l'écran ce qu'il est en train de faire et il peut pointer sur ce qu'il voit.

Un intérêt pratique se trouve dans un test de planarité ou dans la représentation planaire d'un graphe planaire avec comme critère d'esthétisme de cette visualisation la minimisation des longueurs des arêtes. Ces techniques peuvent nécessiter, dans leur algorithme, de fixer des sommets. Un autre exemple est la construction d'un objet à partir des éléments déjà existant dans la figure. Le nouvel objet possède alors des contraintes de construction. Une option consiste à interdire son déplacement indépendamment des objets qui lui sont liés. Ainsi, l'objet n'est manipulable pour le déplacement que par l'intermédiaire de ses objets de base.

8. La règle 8 précise la règle 5 pour une opération particulière, la création des objets. Les objets sont créés sans utilisation de boîte de dialogue, soit par la sélection des éléments déjà existants ou en les créant à la volée. Dans *Cabri-graphes*, la création d'arêtes s'effectue de plusieurs façons possibles. Une arête est un lien entre deux sommets. Si ses extrémités existent alors, le dessin de l'arête s'opère, sinon l'extrémité de l'arête est créée directement, à l'endroit du clic de la souris. Dans *Cabri-géomètre II*, il est possible de créer des points à la volée, c'est-à-dire en étant à proximité d'un point d'intersection, en choisissant un point quelconque d'un objet géométrique ou par simple clic de la souris dans la figure.

9. Toute procédure comporte un certain nombre de paramètres. Pour les commandes nécessitant la sélection d'objets, il faut éviter au maximum un ordre dans cette sélection. Evidemment, cette remarque admet directement une exception dans le cas précis où l'ordre est une nécessité de la procédure. La construction d'une perpendiculaire à une droite passant par un point s'effectue soit en choisissant d'abord le point et ensuite la droite, soit dans le sens inverse, la droite en premier et ensuite le point.

Dans *Cabri-géomètre II*, il est possible de construire les points nécessaires à une construction, au fur et à mesure des besoins, qu'il s'agisse de points libres, de points sur des lignes droites ou sur des courbes, ou encore de points d'intersection. De plus dans la création d'objets, aucun choix n'a été imposé sur l'ordre des paramètres de construction ; un exemple parmi d'autres est la construction d'une droite parallèle à une autre.

Ces procédés sont des exemples de non-obligation pour l'utilisateur, favorisant ainsi sa liberté d'action et sa libre exploration du logiciel.

1.3.2 Evolution du principe WYSIWIG

Ce principe décrit un style dans la conception des systèmes interactifs, et en particulier pour les interfaces de manipulation directe. Il n'y a pas de différence significative entre ce que l'utilisateur voit à l'écran et ce qu'il obtient après impression.

Des actions

Pour Dix et ses collègues, [Dix & al. 93, pp. 126], une interface WYSIWIG implique que la différence entre la représentation et le résultat final soit minimum. L'utilisateur est alors capable de visualiser facilement le document final à partir de la représentation donnée par le système.

Malgré la puissance de ce principe, les auteurs soulignent que «But WYSIWIG is not a panacea for usability. What you see is all you get !» [opus cité], pouvant être traduit par «Mais WYSIWIG n'est pas la panacé. Il n'y a rien d'autre que ce que vous voyez !» ou «… Vous voyez tout ce que vous avez !» soit "Telle action, tel écran". Ils présentent l'exemple du traitement de texte et d'une figure à placer en haut de la page. Comme la pagination et le format de la feuille d'impression sont des variables, la figure ne se trouve donc pas forcément en début de page après des modifications de ces paramètres. Par contre, dans un éditeur graphique, ce nouveau sens "Telle action, tel écran." exprime parfaitement les principes de la manipulation directe, parce que toutes les actions de l'utilisateur induisent des conséquences immédiatement visibles sur la représentation de la figure.

Thimblely, [Thimblely 90, pp. 249], analyse le principe de WYSIWIG dans le temps. La formulation habituelle est plus d'ordre matériel qu'un style d'interaction. Le principe exprime dans un temps présent des implications futures : «What you see is what you will get.», soit "Tel écran, telle impression à venir.". L'utilisateur voit sur l'écran le document tel qui est s'il était imprimé. Pour l'auteur, cela suggère de meilleurs matériels pour améliorer les interfaces utilisateurs. Les écrans de grande taille, la couleur, de nouveaux outils de désignations, des entrées ou sorties, sonores ou vocales, contribuent fortement aux progrès des interfaces.

Ce progrès technique ne doit pas faire oublier le principe d'interaction. Thimblely propose une autre signification du principe dans un temps passé : «What you see is what you have got.», soit "Tel fait, tel écran.". L'utilisateur peut voir ce qu'il a fait à l'intérieur du système. Toute action de sa part produit une réaction immédiatement visible de la part du système.

Des attentes

Les objets manipulés ne sont plus uniquement un texte, mais aussi des figures ou des objets mathématiques abstraits dans un éditeur graphique. L'évolution des objets manipulés entraîne

une évolution dans le concept. Pour Laborde, [Laborde 95, pp. 34], au delà de ce principe WYSIWIG, on tente de satisfaire plutôt à un principe de WYSIWYE : «What you see is what you expect.», soit "Tel écran, telle attente.". L'aspect graphique des objets correspond à ce que l'utilisateur s'attend à obtenir après ses actions ; il représente l'état du système à un moment donné, ce qui lui donne le statut de référence.

Ce statut est primordial, par exemple, dans le cadre de la géométrie et de son apprentissage. Comme la machine a répondu aux manipulations de l'utilisateur, si une erreur intervient c'est l'utilisateur qui s'est trompé dans sa démarche. Ceci entraîne une contrainte logicielle très importante. Un exemple est la construction manuelle de droites perpendiculaires dans *Cabri-géomètre* II. En bougeant une des droites, cette propriété disparaît. Par contre, en utilisant une fonctionnalité de base, construction d'une droite perpendiculaire à une autre, la manipulation de la droite de base entraîne aussi le déplacement de la droite perpendiculaire.

Gritzman et ses collègues, [Gritzman & al. 95], proposent une autre interprétation : «What you see is what you can do right now.» ou "Tel écran, telle action.", c'est-à-dire que l'utilisateur voit uniquement ce qu'il peut effectuer maintenant. L'utilisateur est occupé à une tâche particulière et il souhaite n'avoir que les actions possibles "attachées" à cette tâche. Ceci implique au système d'interdire les actions non associées à la tâche. Nous nous trouvons devant une gestion préventive des actions de l'utilisateur.

Le comportement comme la représentation externe des objets

Les interfaces de manipulation directe permettent d'agir directement sur les objets représentés à l'interface d'un environnement. La réaction des objets à une action peut-être différente suivant les relations "géométriques" qui les lient à d'autres objets, mais aussi suivant les choix didactiques effectués par les concepteurs du système. Il est aussi possible qu'un objet refuse de réagir à une action contradictoire avec ses propres propriétés. Dans l'exemple cité plus haut sur des droites perpendiculaires, la droite perpendiculaire est construite automatiquement ; elle est par nature associée à la première droite et elle ne peut donc pas être déplacée. Cette contrainte n'est pas définitive, car elle peut être supprimée à tout moment.

Le comportement des objets du système reflète donc aussi leurs caractéristiques géométriques. Il représente ainsi certaines des propriétés de l'objet qui ne sont pas accessibles à partir de la représentation seule. Il constitue une partie de la représentation externe des objets perceptibles à la surface du système.

L'interface passe du type WYSIWIG à un principe plus complexe. Ackerman [Ackerman 95] la décrit par le principe suivant «What you see is how I behave.» ; ceci peut se traduire par "Ce que vous voyez est la façon dont je me comporte" (le 'je' étant pour les objets du domaine), soit "Tel comportement, tel écran" traduisant les propriétés de l'objet du domaine lors de la

manipulation de l'objet. Un exemple simple est le déplacement d'un triangle ; suivant les propriétés du triangle (isocèle, équilatérale, rectangle ou quelconque) son déplacement diffère. Le comportement de l'objet est le résultat d'une action de l'utilisateur, immédiatement observable sur l'écran.

1.3.3 La manipulation directe dans les environnements d'apprentissage

Dans le cas d'un modèle mathématique complexe, la manipulation directe ne suffit pas à appliquer toutes les actions possibles pour la construction ou la manipulation d'objets. Dans certains cas, les manipulations peuvent être précédées de déclarations de la part de l'utilisateur, afin de préciser la nature de son action, afin que le système dispose des informations en quantité suffisante permettant de transformer cette action. Cependant, dans cette transformation, une part importante à l'interprétation concerne ce qui n'est pas déclaré par l'utilisateur. Cette interprétation peut encore être traitée par le système afin que les actions de l'utilisateur aient un sens par rapport au modèle mathématique. Elles se traduisent par la mise en œuvre des intentions de ce dernier.

Le principe de la manipulation directe énoncé par Nanard insiste toutefois sur le fait que les articulations entre les intentions de l'utilisateur et les actions qu'il peut entreprendre doivent être réduites. Dans le cas de l'apprentissage, l'utilisateur doit pouvoir exprimer ses intentions suffisamment librement pour avoir l'impression d'engagement direct, c'est-à-dire d'agir librement sur les objets. Les contraintes liées aux manipulations des objets et les relations représentées dans l'environnement doivent apparaître, dans les rétroactions de cet environnement vis-à-vis des actions de l'utilisateur, uniquement comme des contraintes qui déterminent les actions de l'utilisateur [Bellemain 92, pp. 102].

Ainsi, pour parvenir à ce fonctionnement, le système doit disposer, en plus des caractéristiques du domaine mathématique, des connaissances sur l'utilisateur afin de pouvoir déterminer ses intentions. Le système doit avoir la capacité d'interpréter le plus d'actions possibles, qu'elles soient conformes à des opérations permises sur les objets ou en marge de ces opérations. Cette capacité d'interprétation doit remplacer ce qui n'est pas explicitement déclaré par l'utilisateur pour préciser la nature de ces actions.

De la même façon que la formulation de commandes, le système a aussi, avec la manipulation directe, le rôle complexe d'interpréter, en fonction de multiples critères, les actions de l'utilisateur. Cependant, même si cette interprétation du système reste pauvre et qu'elle ne permet pas une réelle manipulation directe, elle n'a pas les mêmes exigences ni les mêmes manifestations lorsqu'elle gère la manipulation physique de représentations que lorsqu'elle gère des commandes. La manipulation directe offre en particulier la possibilité à l'utilisateur d'effectuer des actions souvent accompagnées d'implicites, sans qu'il ne soit forcé par une

formulation spécifique à expliciter ces implicites. Par ailleurs, les réponses du système aux formulations de l'utilisateur peuvent porter sur des problèmes syntaxiques spécifiques du langage ou des aspects formels n'ayant pas toujours un sens pour l'utilisateur.

1.3.4. Les inconvénients de la manipulation directe

Le premier des problèmes posés par la manipulation directe est la représentation à l'écran des entités abstraites. Selon les outils dont on dispose, cette représentation est plus ou moins facile. Le problème est de trouver un moyen de représenter des données, pour permettre leur compréhension et leur manipulation.

La représentation spatiale des objets et leurs symboliques, la sélection directe des objets, et les commandes complexes ou répétitives sont d'autres problèmes à résoudre pour obtenir une interface de manipulation.

La représentation spatiale des objets

[Shneiderman 83, 87] recense trois problèmes importants relatifs à la manipulation directe. Les représentations spatiales et visuelles ne sont pas nécessairement en progrès sur le texte, parce qu'elles peuvent être trop étendues, amenant des ascenseurs sur la fenêtre. Les informations pertinentes ne sont pas forcément visibles à l'écran. Des ascenseurs et des actions multiples sont souvent nécessaires pour obtenir l'information désirée. L'utilisateur n'aperçoit généralement qu'une vue partielle de l'image globale. Plusieurs outils, permettant de mieux visualiser les données, ont été développés dans ce sens. Un exemple est la vue en œil de poisson (voir l'application *VCG tool* de la figure 2.11). L'augmentation de la taille des écrans d'ordinateur facilite la visualisation d'un nombre toujours grandissant d'informations.

Un autre problème est que l'utilisateur doit apprendre la signification des composants de la représentation visuelle. Les icônes graphiques peuvent être propres à chaque concepteur, mais elles peuvent requérir plus ou moins de temps d'apprentissage qu'un mot. Des standards sont apparus comme une icône pour évoquer un mot ou une action, même si certains concepts n'ont pas la même métaphore suivant les cultures.

Le dernier problème est que la représentation graphique peut être égarée. Les utilisateurs peuvent saisir rapidement les représentations analogiques, mais ensuite tirer des conclusions incorrectes sur des actions permises. De nombreux tests doivent être effectués pour affiner les objets et les actions à visualiser et minimiser les effets négatifs.

La représentation graphique

Le problème, que nous abordons maintenant, est le niveau d'abstraction du système graphique. Certains, comme la boîte d'outils du Macintosh, ont un modèle de dessin direct. Cela signifie que les fonctions disponibles permettent d'afficher des formes géométriques dans une fenêtre, mais que celle-ci ne mémorise pas leurs contenus, si ce n'est en terme de pixels. Le dessin direct

complique singulièrement la tâche qui incombe à l'application, et donc à son programmeur. En effet, lorsque des objets graphiques représentent des données, on a souvent besoin de les modifier, les déplacer ou les effacer. Comme le système graphique ne garde pas mémoire de ces objets, cela signifie que l'application doit effectuer elle-même les opérations. Cela suppose qu'elle gère une structure représentant l'organisation des objets graphiques. Cette structure est aussi indispensable pour afficher de nouveau des parties de fenêtres lorsque le système de fenêtrage ne gère pas lui-même le réaffichage.

D'autres systèmes, comme Xtv [Beaudon-Lafon & al., 90], utilisent un modèle de dessin qui consiste non plus à dessiner sur un écran, mais à visualiser des objets que l'on a disposés dans une structure d'affichage. Pour le concepteur, les objets présents dans la structure sont considérés comme présent à l'écran ; c'est le système graphique qui gère le réaffichage. Les fonctions offertes ne sont plus des fonctions de dessins, mais des fonctions de manipulation de la structure. Cette approche simplifie considérablement la tâche du concepteur, parce qu'un objet graphique correspond à chaque objet abstrait.

La sélection graphique des objets

Les interfaces de manipulation directe ne sont pas adaptées pour toutes les applications, en particulier pour les traitements de textes, la conception assistée par ordinateur, et les bases de données [Buxton 93, pp. 22]. Les deux problèmes principaux sont la sélection directe des objets visibles à l'écran et la répétition d'une action.

Pour la sélection d'objets graphiques, il existe plusieurs solutions. Une solution est de sélectionner un objet par l'intermédiaire de son étiquette (dans le cas où les objets admettent un étiquetage) comme dans le cas de *Cabri-géomètre II*. Comme l'étiquette est associée à un objet et représente une caractéristique de l'objet, cette solution est relativement simple à mettre en place. Une autre solution est une gestion beaucoup plus fine des objets graphiques (voir le paragraphe *2.3.4 La sélection d'objets*).

Les commandes complexes et répétitives

Pour la répétition de commandes simples, plusieurs idées sont apparues. Pour Frölich [Frölich, 93, pp. 322], il faut séparer les concepts d'immédiateté et de manipulation, et plutôt parler d'interaction directe. Il interprète le concept d'immédiateté comme la notion de distance minimum[1]. Il est donc possible de l'appliquer aux systèmes de manipulation directe et aux systèmes conventionnels. Frölich entrevoit même la conception d'interfaces acceptant les deux types d'interactions.

Une autre solution est l'adoption d'un langage de haut niveau permettant à l'utilisateur de créer ses propres commandes, les macrocommandes. Mais cette solution ne résout que le

[1] La définition de distance est celle donnée par Hutchins & al. (voir paragraphe 1.2.2).

problème des tâches compliquées en construisant une nouvelle commande, qui est une succession de plusieurs commandes simples afin de réaliser cette tâche complexe avec l'ajout d'un article de menus. Un apport est l'ajout de commandes textes, comme dans l'exemple AppleScript. Ce langage se doit d'être simple, facile d'utilisation, et surtout ne permettre que des actions qu'il est aussi possible d'effectuer par manipulation directe, sinon il serait inutile, puisqu'on y retrouverait les inconvénients des langages de commandes.

Brad Myers [Myers 92] construit un nouveau type d'interface, appelé les interfaces par démonstration (Demonstrational Interfaces). Par manipulation directe, l'utilisateur effectue des opérations. Si deux opérations identiques se succèdent, le système crée alors un programme et informe l'utilisateur s'il veut recommencer sa commande à tous les objets de même type. Le terme de démonstration est utilisé parce que l'utilisateur démontre le résultat désiré en utilisant des exemples.

Conclusion

La définition de la manipulation directe, présentée dans ce paragraphe, permet d'évaluer un type d'interface. Elle met en évidence l'objectif vers lequel doit s'efforcer d'atteindre une interface de manipulation directe. Cette définition doit cependant être omniprésente, dans l'esprit de chaque concepteur, à chaque étape de la conception et de la réalisation d'une interface de qualité, même si le concepteur est restreint dans sa conception.

L'engagement direct réfère au style particulier de représentation basée sur la métaphore du monde réel plutôt que sur la métaphore de description (ou du monde abstrait) pour l'interaction. Les systèmes peuvent encourager un sentiment d'engagement par la représentation graphique des objets du domaine et aussi permettre à l'utilisateur de les manipuler physiquement plutôt que par l'intermédiaire d'un langage de description. «*Les interfaces de manipulation directe sont celles qui minimisent la distance et maximisent l'engagement.*» [Hutchins & al. 85, pp. 335] et «[…] *et elles présentent le type d'interface le plus "direct" pour l'utilisateur*.» [Frölich 93, pp. 316].

2. La conception d'une interface de manipulation directe

Concevoir une interface consiste à définir son apparence, mais surtout à déterminer la manière dont cette apparence va évoluer au cours du temps, c'est-à-dire ses réponses face aux sollicitations de son environnement, venant du noyau fonctionnel ou de l'utilisateur. Cette évolution est nommée dynamique de l'interface. Aujourd'hui, les outils disponibles, pour la construction des interfaces, permettent de décrire des évolutions "macroscopiques" : l'affichage

ou la disparition de boîtes de dialogue ou le passage d'un formulaire à un autre pour les dialogues homme-machine. De manière complémentaire, il existe aussi une dynamique microscopique, intervenant sur des échelles de temps beaucoup plus fines. L'exemple le plus courant est l'icône sur laquelle on a cliqué et qui se déplace avec le curseur de la souris : chaque déplacement est une modification négligeable de l'aspect de l'interface. La description de cette dynamique microscopique est l'enjeu principal pour les outils de construction des interfaces et en particulier des interfaces de manipulation directe.

2.1 Modélisation de l'interface

La représentation informatique d'un domaine passe par une phase de modélisation pour laquelle il existe des outils de spécification. Au niveau de l'interface, il est aussi nécessaire de spécifier les objets et les fonctionnalités nécessaires à la manipulation directe. Il est fréquent de fusionner l'outil de spécification du domaine avec celui de l'interface.

Le développement d'applications a montré la nécessité d'une organisation méthodique des objets à gérer.

2.1.1 Spécification de l'interface

Dans la conception de l'interface d'un environnement interactif, et en particulier de manipulation directe, il est indispensable de spécifier les représentations externes des objets abstraits du domaine, et les actions nécessaires à leurs manipulations. L'utilisateur se servira de ces outils et des objets pour communiquer avec le noyau fonctionnel de l'application. L'interface peut être modélisée en plusieurs niveaux ou composantes :

- le niveau conceptuel, dialogue, et entrée/sortie pour [Ziegler & al. 88],
- le niveau de présentation, d'abstraction, de contrôle dans le modèle PAC [Coutaz 90],
- le niveau du domaine, conceptuel, de présentation, et visuel pour [Bernat 94].

Cette présentation par niveau met en correspondance la représentation externe d'un objet et sa cohérence abstraite ou interne. Une unité de contrôle est chargée de maintenir la cohérence entre les deux représentations. Elle met à jour la représentation externe si les valeurs de la représentation abstraite changent et elle modifie la représentation abstraite si l'utilisateur agit par manipulation directe sur la représentation graphique.

La représentation abstraite d'un objet n'est pas toujours suffisante pour en déterminer la représentation graphique. Cette représentation graphique peut dépendre aussi d'autres objets de la figure qui peuvent n'avoir aucune dépendance fonctionnelle avec l'objet considéré. Un

exemple simple est la représentation dans l'espace d'un segment qui se décompose en plusieurs étapes :

- il est fonction de la représentation abstraite du segment et celle des objets dont il dépend, ici les coordonnées des extrémités, avec en prime plusieurs systèmes de coordonnées possibles,
- ses attributs graphiques sont attribués à défaut par le système, ou ils sont donnés par l'utilisateur (couleur, épaisseur et représentation du trait, …),
- elle est fonction des autres objets pour ses relations de dépendance (une des extrémités du segment appartient à d'autres objets) et par la représentation de ces objets qui peuvent partiellement cacher celle du segment en construction.

Cette dernière étape, dans l'élaboration de la représentation graphique, effectue une traduction sémantique de l'objet en propriétés graphiques. La sémantique de la représentation d'un objet géométrique sur le dessin ne se limite pas à sa seule représentation abstraite, mais doit aussi tenir compte de sa position relative par rapport aux autres objets de la figure (et donc générer des parties cachées de l'objet), de la position relative de l'utilisateur ou la caméra regardant l'espace considéré, et des sources de lumières éventuelles.

Il faut concevoir un modèle tenant compte d'objets graphiques manipulables directement et comprenant toutes les remarques citées.

2.1.2 Modélisation de l'interface

Nous proposons ici une modélisation tenant compte des besoins d'identification les objets du domaine et mettant en évidence quatre aspects de l'interface. Ils sont appelés niveau du domaine, niveau conceptuel, niveau de présentation et niveau visuel.

Le niveau du domaine

Le niveau du domaine est un niveau théorique, indépendant de tout concept informatique, que le concepteur de l'environnement informatique souhaite modéliser. Le concepteur détermine aussi ses objectifs, comme l'objectif final du logiciel : un environnement pour l'apprentissage (des mathématiques), un outil de visualisation ou un prototype destiné aux chercheurs, etc. Le type d'interface et le mode d'action seront aussi choisis. En somme, un cahier des charges est élaboré.

Le niveau conceptuel

Choix des objets du domaine

Dans le niveau conceptuel, le concepteur modélise les objets du domaine. Il utilise les choix de représentation des connaissances. Il est lui-même composé de deux parties : un niveau interne ou attribut informatique et un niveau externe ou attribut graphique.

L'attribut informatique est la formulation ou la traduction en terme d'objets informatiques des objets du domaine. Un ensemble sera modélisé par une pile, une file ou une liste suivant les besoins de l'application à modéliser. Elle représente les structures de données abstraites du domaine pour des objets abstraits.

L'attribut graphique regroupe toutes les données relatives à la visualisation des objets du domaine (taille, couleur, forme, etc.) en relation avec les objets du domaine. La représentation d'un objet abstrait dépend de sa représentation graphique : une couleur pour identifier une texture (marron pour le bois, grise pour le métal, jaune pour l'or etc. …). Dans la modélisation d'architecture, une couleur est associée pour représenter la texture du matériau utilisé : les nuances du marron pour les variétés de bois et le gris pour le métal par exemple.

Le niveau relationnel

Choix des métaphores

Le niveau relationnel est le niveau des relations entre les objets du domaine. Il se décompose, lui aussi, en deux parties : un attribut de dépendance et un attribut visuel.

L'attribut de dépendance correspond aux relations abstraites associant deux objets. L'exemple le plus classique est celui de la construction d'un objet à partir d'autres objets déjà existants, comme celle d'un sommet contracté ou celle d'un segment à partir de points existants.

L'attribut visuel correspond à la modélisation d'une caractéristique sémantique, telle que l'aspect sélectionné, l'état visible ou invisible.

Remarquons que le niveau de conception et le niveau relationnel associent les quatre attributs à un seul objet. Nous vérifions bien ainsi une règle communément utilisée voulant faire correspondre un objet du modèle pour tout objet du domaine. Joëlle Coutaz parle d'objets sémantiques [Coutaz 90].

Le niveau informatique

Choix de la programmation

Le niveau informatique est décomposé en deux parties.

La première partie est le niveau de l'interface graphique utilisée par le concepteur selon les plates-formes disponibles avec les objets spécifiques d'une interface. Il dépend fortement de la plate-forme et du langage de développement et traduit en terme physique tous les attributs sémantiques et relationnels d'un objet. Il permet l'expression de la manipulation directe. Ainsi, l'état sélectionné peut être traduit par des poignées apparentes, une vidéo inversée, un clignotement, une surbrillance, etc.

La deuxième partie est la programmation proprement dite des objets abstraits. En reprenant l'exemple précédent, l'implémentation du type abstrait ensemble (une pile, une file ou une liste) s'effectue par un tableau, par une liste chaînée ou par un mixage d'un tableau et d'une liste

chaînée, etc. C'est le choix visuel pour l'interface et le choix d'implémentation des types de données abstraites.

2.2 Le mode opératoire

Un problème posé par la manipulation directe des objets est celui de l'ordre dans lequel les informations sont données au système. L'utilisateur désire exécuter une procédure portant sur tels ou tels objets, doit-il indiquer l'action à effectuer puis les éléments sur lesquels portera l'action ou bien sélectionner un ensemble d'objets avant de choisir l'action à exécuter ?

2.2.1 Description de l'action

Il faut distinguer deux notions dans une action : l'aspect syntaxique correspondant à la description physique de l'action et l'aspect sémantique.

Aspect syntaxique

Les actions *physiques* autorisées sont en nombre relativement limité : le clavier et des outils de désignation, exemple de la souris qui possède un ou plusieurs boutons ou un joystick. Les actions possibles sont, par exemple, le clic, le double-clic, le drag (déplacement d'un objet consécutif au déplacement de la souris ayant un bouton enfoncé). Elles peuvent être combinées avec des touches spécifiques du clavier : Ctrl-clic, Shift-clic, …

Aspect sémantique

Au niveau graphique, les actions ont un verbe comme nom, par exemple les verbes créer, désigner, déplacer, associer, ou informer. A chaque action correspond une combinaison de touches clavier et/ou une manipulation de la souris. La sémantique de ces actions n'est pas commune à toutes les plates-formes ; il n'existe pas de normes. Sous Windows, *associer* se fait par un Ctrl-clic alors que sur Macintosh on utilise Shift-clic, sans oublier la touche supplémentaire, la touche "pomme ou " existant uniquement sur le Lisa d'Apple et la touche commande "" sur Macintosh.

2.2.2 Définitions des modèles opératoires

La manipulation directe d'objets graphiques consiste en des combinaisons de verbes et d'objets, le verbe étant une action de l'utilisateur, et l'objet une entité abstraite dont on manipule la représentation. Elle s'effectue selon l'un des deux modes opératoires suivants : nom/verbe ou verbe/nom.

Le modèle nom/verbe

Dans ce premier cas, l'utilisateur sélectionne un objet ou un ensemble d'objets, puis il détermine l'action qu'il veut leur appliquer. Ce modèle simplifie la programmation, parce qu'il limite le nombre d'actions sémantiques. Il est souvent associé aux deux actions suivantes : la sélection et l'association de plusieurs objets, puis le déplacement de ces derniers. On peut associer des objets sans se préoccuper de leur type. La vérification de la cohérence entre le type des objets sélectionnés et l'action demandée est faite a posteriori ; une action est valide uniquement si les types des objets sont cohérents avec l'action.

Le modèle verbe/nom

Dans ce deuxième cas, l'utilisateur sélectionne une fonction, et ensuite les objets auxquels elle doit être appliquée. Ce type d'action limite l'accès aux seuls objets qui lui correspondent. Les objets accessibles sont donc déterminés par le contexte. Par contre, une action syntaxique peut avoir plusieurs correspondants sémantiques. Ainsi, dans un environnement informatique sur la géométrie, si la commande choisie est "créer un segment", l'action attendue est "sélectionner un point". Si la commande est "intersection de droites", l'action attendue est "sélectionner une droite". Ces actions ont des sémantiques voisines mais néanmoins différentes. Elles correspondent à la même action syntaxique : un clic. De plus, ce modèle facilite la mise en place d'une aide contextuelle [Bernat 94]. En effet, le choix d'une commande permet de renseigner l'utilisateur sur les objets à utiliser pour valider l'action correspondante.

2.2.3 Le choix du mode opératoire

Un paradigme fondamental structurant les interactions homme-machine, selon la présentation propre d'Apple [Apple 87] et un standard des interfaces utilisateur chez IBM [IBM 88], est le principe du modèle d'interaction nom/verbe. La philosophie, cachée derrière ce paradigme nom/verbe, est de mettre l'utilisateur au centre du contrôle des actions, c'est-à-dire que l'utilisateur est libre de choisir quelles actions seront appliquées aux objets de son choix. Apple en fait explicitement un principe général de conception : *« L'utilisateur, non l'ordinateur, initie et contrôle toutes les actions.»* [Apple 87].

Ce choix du modèle d'interaction nom/verbe n'est pas vraiment établi. Pour Ziegler *«Le plus fréquemment, la manipulation directe utilise la séquence syntaxique 1) sélectionner des objets, 2) appeler une fonction.»* [Ziegler & al. 88, pp. 126].

Buxton est plus catégorique : *«Dans les systèmes de manipulation directe, le modus operandi (mode opératoire) normal est de sélectionner et ensuite d'opérer …. L'immédiateté de cette indication est une composante clé de la manipulation directe.»* [Buxton 93, pp. 21].

Par contre pour Bernat, le choix initial d'une action limite l'accès aux seuls objets qui lui correspondent et il facilite la mise en place d'une aide contextuelle. Ils considèrent ainsi que : *«… le modèle action/objet nous semble plus adapté à un environnement d'apprentissage. Il favorise l'exploration libre du logiciel.»* [Bernat & al. 95, pp. 217].

Nom/verbe et verbe/nom : notions duales

Le choix initial a été nom/verbe, parce que les premiers logiciels utilisateurs étaient les traitements de texte, et qu'il fallait choisir un mode pour effectuer une action. Un exemple type est celui de l'éditeur de texte "vi" sur UNIX, où un mode particulier correspond à une série d'actions, comme l'insertion ou la suppression. Pour la manipulation d'objets graphiques, le choix initial a donc aussi été nom/verbe. De plus, dans la métaphore du bureau électronique de la Star Machine ou du Macintosh, deux commandes ne peuvent s'effectuer que par ce type de mode opératoire. La première commande est de sélectionner, puis de déplacer des icônes ou des objets. La seconde commande est le service couper-copier-coller s'effectuant après une sélection d'objets. En outre, le principe de cohérence a conduit l'adoption de ce choix. Ce principe de consistance d'un logiciel peut se définir par cette expression : une même formulation pour tous. Or pour les deux familles d'action, le déplacement et le service couper-copier-coller, il faut avoir sélectionné au préalable les objets avant de pouvoir effectuer l'action.

L'idée de mettre l'utilisateur au centre des actions (initier, puis contrôler les actions) est aussi vraie dans les deux modes opératoires. Les vérifications de cohérence de types s'effectuent a posteriori dans le mode nom/verbe et a priori pour le mode verbe/nom. Si tous les objets définissant une action sont sélectionnés alors l'action est valide ; par contre, dans le mode verbe/nom, il faut vérifier à chaque sélection la validation de chaque objet. C'est un avantage si aucun ordre n'est précisé sur la liste des objets à considérer dans l'action.

Le contrôle de l'action pour la gestion ou la prévention des erreurs est identique dans les deux modes. Une différence notable est dans le type d'aide que le type de mode opératoire fournit. Elle est contextuelle pour le mode verbe/nom (par exemple, une liste d'objets peut être affichée dans une barre d'état, pour avertir l'utilisateur des objets à sélectionner pour activer la commande), tandis que l'aide est une aide en ligne pour le mode nom/verbe.

Le mode verbe/nom favorise donc une exploration plus libre du logiciel que le mode nom/verbe, mais il se peut qu'il ne réponde pas entièrement au principe de cohérence (voir le paragraphe 2.4.1 Cohérence).

Le choix dans le type d'interaction dépend fortement du domaine de modélisation. Pour la théorie des graphes, il n'existe que deux objets de construction : le sommet représenté par un cercle ou un point, et l'arête pouvant avoir plusieurs types de représentation, comme un segment, une ligne brisée ou une courbe. En géométrie, la construction des objets est une

opération de base. Il est naturel de positionner l'action en priorité, sans aucun ordre préétabli sur les paramètres de construction, afin de mieux appréhender le monde de la géométrie.

Le mode choisi dépend aussi du domaine à modéliser. Dans le cas de la géométrie, le nombre de paramètres pour une action est limité à un petit nombre pour la construction d'un objet, ou les vérifications d'une propriété. Dans notre étude, la théorie des graphes, les actions possibles s'effectuent sur des parties graphes, c'est-à-dire des éléments qui sont encore des graphes. Ce nombre de composantes augmente avec la taille du graphe et il devient vite très grand. Par conséquent, dans un mode verbe/nom, il faudrait pouvoir signaler d'une façon quelconque au système le fait d'avoir terminé la sélection d'objet, comme par exemple un bouton marqué "fini" (voir le système *Link* dans la figure 2.17). Ce bouton n'est pas concevable, voire même contraire aux principes de manipulation directe.

Les deux notions, verbe/nom et nom/verbe, sont donc relativement duales l'une de l'autre. Le domaine à modéliser et les choix didactiques voulus déterminent le mode opératoire le mieux adapté à la situation. Le mode opératoire n'est donc pas un élément déterminant pour les interfaces de manipulation directe.

2.3 Le retour d'informations

Le retour d'informations est une réaction du système aux actions de l'utilisateur. Ces réactions se manifestent sous forme d'expressions de sorties dont l'interprétation permet d'évaluer la situation, et de la comparer au but recherché. Selon les réponses, l'utilisateur peut poursuivre ou modifier son plan d'action.

2.3.1 Informer l'utilisateur

L'attente ou les intentions de l'utilisateur peuvent être satisfaites de plusieurs manières, par exemple :

- il informe pour rassurer lors d'une opération relativement longue,
- il informe pour réduire la charge cognitive,
- il informe pour remédier aux erreurs et aussi pour demander de plus amples informations dans des situations critiques.

Informer pour rassurer est agréable à l'utilisateur, dans la mesure où ce dernier sait alors si l'opération est en cours d'exécution ou si elle a échoué. Dans le cas d'une copie d'un fichier ou la récupération d'un fichier via le réseau internet, un message indique le temps restant pour la récupération et le pourcentage de données déjà ramenées (voir la figure 1.2).

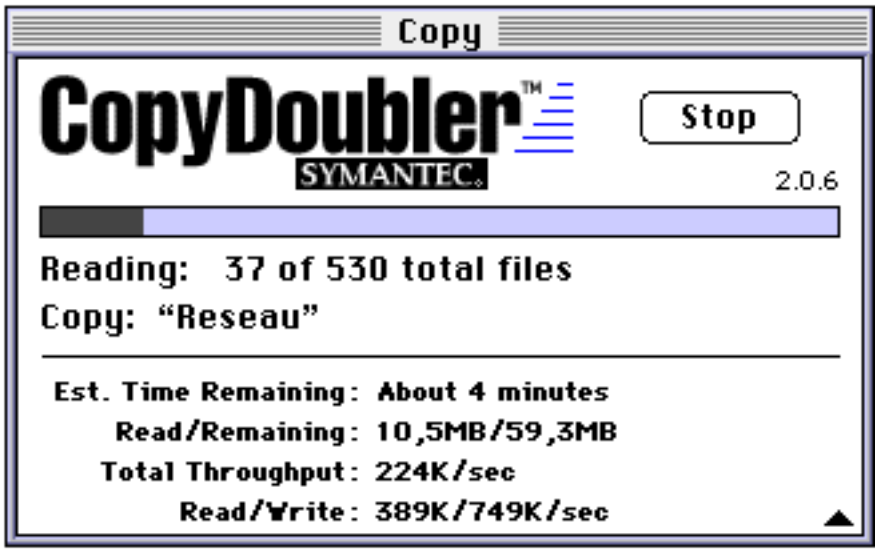


Figure 1.2 : Un utilisataire donnant des informations relatives à une copie de fichiers dans l'environnement du Macintosh. Un outil similaire a été intégré dans le système 8.

Réduire la charge cognitive signifie donner des informations constantes sur le logiciel. Dans l'environnement de *Cabri-graphes*, la nature des graphes visualisés est constamment affichée (voir la figure 3.24). Dans Word6, la barre horizontale, en bas de l'écran indique à tout moment la position courante du curseur de la souris à l'intérieur d'un document, ainsi que le nombre de pages et de sections, et l'heure (voir la figure 1.3).

Figure 1.3 : La barre d'informations relative dans un fichier de Word6.

Il faut réduire au minimum les boîtes de dialogue dans la communication entre le système et l'utilisateur. Evidemment, l'introduction d'une boîte de dialogue est parfois inévitable et nécessaire, comme dans le cas de la sauvegarde d'un fichier. Lors de l'affichage de ces dialogues, il faut renseigner au mieux l'utilisateur en donnant des précisions sur l'opération demandée ou sur les informations que le système retourne.

Un domaine de recherche intéressant est celui du problème de l'introduction de plus de retour d'action de la part du système. La solution riche s'apparenterait à celle mise en œuvre dans les gants cybernétiques à retour d'efforts médiatisés en même temps que les réalités virtuelles. A défaut, des solutions "temporisées" ou sonores sont envisageables. Une solution "temporisée" consiste à afficher un message à l'utilisateur pendant une durée déterminée (une ou deux minutes), afin de ne pas bloquer le système. L'ajout complémentaire d'un effet sonore est aussi appréciable.

(a) Le message demande la confirmation d'une commande entraînant de profondes modifications.

(b) Le message propose à l'utilisateur dans le premier dessin de définir un rectangle afin de pouvoir coller le graphe copié précédemment. En cas d'échec (rectangle trop petit), le système demande à l'utilisateur de définir un rectangle plus grand.

Figure 1.4 : Des exemples de messages dans l'environnement de *Cabri-graphes*.

Une autre manière d'informer l'utilisateur est de réserver un espace où le système pourrait lui donner des informations, une sorte d'information du système mais sans être une aide en ligne, juste une information supplémentaire pour une action similaire à la situation donnée. Dans *Cabri-graphes*, des messages de ce type sont possibles. Les représentations dans la figure 1.4 proposent des exemples.

L'introduction de la notion de message vocal représentant l'état du système ou l'attente du système pour une action précise, avec évidemment toujours la possibilité d'interrompre l'action en cours, est un autre axe de recherche. Dans une interface de manipulation directe, l'utilisateur, surtout s'il est novice ou inexpérimenté, n'est pas à l'abri de mauvaises manipulations. Il est judicieux d'avoir des garde-fous pour éviter des aberrations de construction ou la perte du contrôle de l'opération en cours [Baulac 90], avec par exemple :

- la possibilité d'abandonner une opération en cours ou bien annuler l'opération qui précède et revenir à l'état antérieur ; les actions sont réversibles,
- seules les opérations ayant une signification dans l'état courant du domaine sont disponibles ; c'est la notion de prévention des erreurs,
- le système intervient en affichant des messages pour signifier des impossibilités ou pour demander la confirmation d'une opération ayant des conséquences importantes ; il faut un retour d'informations pertinentes et bien adaptées,
- le système offre la possibilité de revenir à la figure initiale, si cette dernière a été dessinée à partir d'un fichier.

Des effets visuels ou sonores faciles d'utilisations sont aussi présents afin de donner des indications sur l'opération en cours ; en voici quelques exemples :

- les objets désignés clignotent jusqu'à la fin de l'opération (dans le cas verbe/nom),

- le curseur change de forme selon la nature de l'opération,
- le type d'un objet sélectionnable est indiqué à côté du curseur,
- le nom de l'opération en cours est indiqué pendant toute l'opération,
- lors d'un déplacement d'objet, la position initiale de l'objet est indiquée en pointillé,
- un message vocal, par exemple un bip sonore, est alors émis lors d'une erreur de manipulation.

L'absence de retour d'information amène l'utilisateur à spécifier des commandes illégales, à oublier le mode d'interprétation des entrées ou à avoir des difficultés à identifier la cause d'une erreur. Cette absence de retour d'information peut aussi conduire à inférer des faits inexacts que l'image renforce par absence de contradiction. Ceci est d'autant plus vrai si l'environnement informatique est un outil d'apprentissage de connaissance.

2.3.2 Le rôle du curseur

Le curseur reprend le comportement familier des outils de dessin rencontrés dans un environnement papier-crayon. Il les étend en permettant d'élargir leurs champs d'actions initiaux par la programmation de nouvelles fonctionnalités adaptées. La représentation du stylo informatique s'adapte à la création d'objet, la main voulant attraper un objet pour la manipulation de l'objet, la barre ou I d'insertion (aussi appelée poutrelle) pour l'édition d'un texte, la loupe afin d'indiquer une multitude d'objets présents à l'emplacement du curseur, ou la sélection par un rectangle des objets avec les deux petites flèches (voir la figure 1.5).

Figure 1.5 : Quelques-uns des curseurs pour indiquer l'état du système, la prochaine action attendue ou le type d'action possible. Ils ont été aggrandis (la taille est multipliée par 2) afin de mieux les voir.

Le curseur est l'outil par excellence pour se repérer sur l'écran d'un ordinateur. Il possède plusieurs formes possibles, mais de nombre borné par une reconnaissance possible du symbole. Un groupe d'actions similaires ont ainsi une représentation identique pour le curseur, comme le déplacement d'un objet ou un groupe d'objet. Le rôle fondamental de la forme d'un curseur actif est de rappeler en permanence l'état du système, la prochaine action attendue ou de suggérer un type d'action possible.

Pour l'équipe de Xerox Parc «*Star* modifie aussi la représentation du curseur comme une indication supplémentaire. » [Smith & al, 82, pp. 278]. Ce changement d'état du système est un rappel constant pour l'utilisateur.

Le curseur dépasse les possibilités du crayon dans un environnement papier-crayon. La représentation du stylo informatique se modifie pour s'adapter à la création d'un objet, à la manipulation de l'objet ou la sélection des objets. La représentation du curseur est donc synonyme à la fois de l'état du système et de la prochaine action à effectuer. Cette représentation symbolise l'action avec un message supplémentaire indiquant un choix possible pour l'utilisateur.

2.3.3 Détection et prévention des erreurs

Les erreurs sont inévitables, mais le système peut en faciliter la détection, la détermination et la réparation. La détection d'une erreur déclenche un retour d'information pour avertir l'utilisateur. La détermination de l'erreur fournit des indications utiles pour le remède. Elle signale à l'attention de l'utilisateur pourquoi l'action a échoué en affichant un message compréhensible à l'écran, et non pas un numéro de code ou un texte seulement utilisable par le concepteur. Coutaz [Coutaz 90] associe le remède à l'erreur aux fonctions du système, et surtout à l'existence des commandes "Annuller" et "Répéter" pour l'utilisateur. Une amélioration sensible est la gestion préventive de l'erreur, comme le préconise Fekete [Fekete 96].

Figure 1.6 : L'interface au démarrage de *Cabri-graphes* (version 3.14). Les menus non autorisés sont grisés afin de prévenir toute erreur possible. Les deux seuls menus accessibles sont standards dans une application récente : le menu fichier ou "File" et le menu édition ou "Edit" (avec la présence éventuelle d'articles de menus indisponibles).

Le service de prévention d'erreur doit permettre de savoir si l'appel à une fonction du noyau sémantique provoque une erreur. Ceci est une des huit règles d'or de la conception du dialogue par Shneiderman [Schneiderman 83, pp. 59] :

«*Offrir une gestion simple des erreurs.* Autant que possible, concevoir le système de manière à ce que l'utilisateur ne puisse pas faire d'erreur sérieuse. Si une erreur est faite, le système doit la détecter et offrir un mécanisme simple et compréhensible pour la gérer […]. Une

commande erronée doit laisser le système inchangé, ou le système doit donner des instructions pour revenir à l'état précédent.»

Le service de prévention des erreurs permet le retour d'information nécessaire à la manipulation directe. Par exemple, lorsque des objets graphiques sont sélectionnés, les articles de menus agissant sur la sélection doivent être désactivés si leur activation produit une erreur (voir la figure 1.6). Le noyau sémantique devrait être capable à tout moment de tester la validité d'une commande. Dans *Cabri-graphes*, le noyau ajuste les articles des menus et les menus susceptibles d'être choisis en fonction des choix précédemment faits par l'utilisateur : la nature du graphe à l'écran et son existence, la présence éventuelle d'une sélection de sommets ou d'une sélection d'arêtes, les préférences sur les objets (étiquette visible ou cachée), etc.

Donnons un exemple dans l'environnement *Cabri-graphes*. Pour chaque session de travail, l'utilisateur choisit la nature du graphe (simple ou multiple, orienté ou non, avec ou sans boucles). Par défaut, le graphe est déclaré simple, non orienté et sans boucles. Le logiciel se comporte de manière intelligente en ce sens que si l'arête à dessiner est incompatible avec les propriétés du graphe (par exemple on veut dessiner une arête multiple pour un graphe simple), son dessin doit être ignoré.

Cette prévention des erreurs n'est pas en opposition avec la détection des erreurs et les remèdes. Elle la complète et facilite la gestion proprement dite des erreurs en effectuant un travail préventif. Ce travail préliminaire soulage non seulement cette gestion de l'erreur mais aussi la gestion des actions réversibles.

2.3.4 La sélection d'objets

La sélection d'objets graphiques pose souvent des problèmes. Une gestion plus fine sur les données graphiques est alors nécessaire pour une gestion des ambiguïtés relatives aux représentations d'objets superposés ou pour choisir un élément de l'espace modélisé sur un écran, soit dans un espace discrétisé.

Problème de la représentation des objets

Les transformations appliquées aux objets consécutivement au déplacement du pointeur doivent être continues. Cette représentation continue des objets déplacés (un sommet et ses arêtes incidentes dans le cas d'un graphe) ne doit aussi jamais altérer le reste du dessin comme un effacement partiel de la figure. Il peut être confortable pour l'utilisateur, pendant le temps du déplacement, de sauvegarder la position initiale des objets d'une quelconque manière non équivoque (en pointillé par exemple), mais sans être en position visuelle marquante, la surbrillance étant préférée et réservée pour les objets sélectionnés.

Problème de sélection d'un objet parmi plusieurs

A ces trois caractérisations de la manipulation directe, il faudrait ajouter *la gestion des ambiguïtés*. En effet, il arrive que la sélection d'un objet à la souris soit difficile. Il est alors souhaitable de présenter à l'utilisateur l'ensemble des objets entourant le point désigné par la souris ; une représentation de cet ensemble peut se faire, par exemple, par un menu déroulant à l'endroit marqué par le clic de la souris. L'utilisateur n'a plus qu'à choisir dans ce menu l'objet voulu. Cette gestion de l'ambiguïté peut être de deux types : instaurer un ordre de préférence sur les objets ou laisser une indépendance vis-à-vis des objets sélectionnés.

Ordre de préférence

Si le domaine modélisé possède relativement peu d'objets, il est peut-être préférable de choisir un ordre de préférence dans la liste des objets. Dans l'environnement sur les graphes, exemple de *Cabri-graphes*, un ordre de préférence a été adopté. Cet ordre est le suivant : le sommet, l'arête, la flèche, l'étiquette.

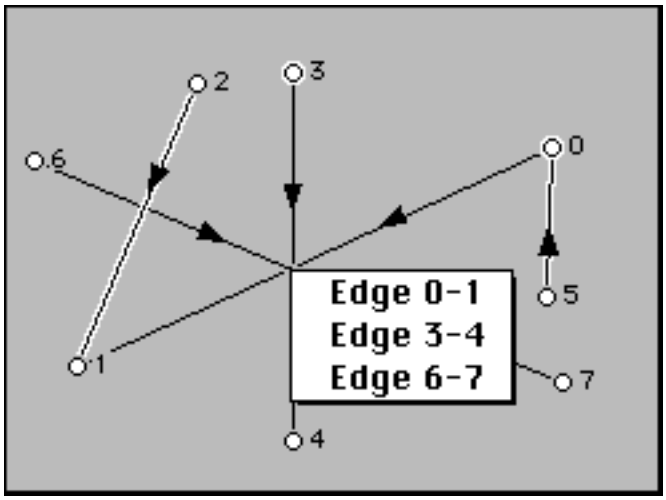


Figure 1.7 : Un graphe orienté dans l'environnement de *Cabri-graphes*. Les deux arêtes et le sommet numéro 4 sont en surbrillance, pour indiquer qu'ils sont sélectionnés.

Par contre, lorsque des objets de même nature sont sélectionnés, un choix est laissé à l'utilisateur. Dans la figure 1.7, le menu déroulant affiche à l'intersection des segments trois choix possibles d'arcs pour que l'utilisateur désigne explicitement l'arc sur lequel il voudra effectuer une opération.

Indépendance dans la sélection

Pour les environnements informatiques choisissant la solution d'afficher tous les objets sous le curseur de la souris, il faut que la gestion des informations soit précise pour permettre d'afficher la liste des objets présents.

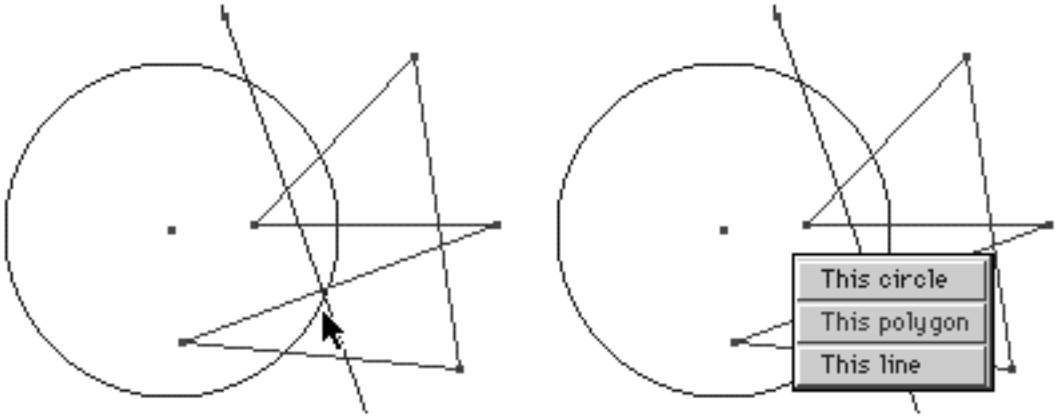


Figure 1.8 : Gestion de l'ambiguïté dans *Cabri-géomètre* II.

De plus, la liste complète des candidats possibles est affichée, par l'intermédiaire d'un menu déroulant à l'endroit du clic de la souris par exemple. Il ne faut pas effectuer le parcours de cette liste en présentant, par un clignotement du candidat éventuel, et attendre pour chacun une réponse de l'utilisateur (exemple dans l'environnement de *GraphEd* dans le chapitre 2).

Dans le cadre de la géométrie, la gestion de l'ambiguïté, indépendante des objets, facilite la manipulation des objets intrinsèques. Elle permet à l'utilisateur de mieux distinguer les objets les uns par rapport aux autres (voir figure 1.8).

Cette manipulation directe ne doit pas s'arrêter aux objets graphiques déjà dessinés, mais s'effectuer sur de nouveaux objets en offrant à l'utilisateur une liberté d'action dans ses mouvements.

Différencier sélection et déplacement pour un singleton

Remarquons que la notion de *"drag and drop* " ou *"glisser et déposer"* un objet unique s'accompagne généralement d'une sélection de cet objet, juste avant son déplacement ou après son déplacement. Ce choix peut sembler être une erreur, puisque le déplacement d'un objet unique ne s'accompagne pas de changement dans la structure interne, mais plutôt de modification dans la visualisation des objets, sauf dans des cas bien précis.

2.3.5 Absence de mode

Une interaction non modale est un des principes émis par les pionniers des interfaces à manipulation directe. Elle permet de mieux assurer l'universalité des commandes à l'intérieur de l'application.

Le fonctionnement est dit modal dès que l'on a choisi un mode. Un mode, pendant qu'il est activé, restreint alors les opérations qu'un utilisateur peut effectuer. Le mode, et donc le système, enferme l'utilisateur dans une opération unique et il ne l'autorise pas à effectuer

d'autres tâches tant que cette opération n'est pas achevée, ou d'autres manipulations propres à ce mode. C'est le cas par exemple de l'éditeur de texte "vi" d'UNIX qui prend en compte la frappe d'un caractère que si l'utilisateur s'est placé dans le mode écriture.

En revanche, l'absence de mode offre à l'utilisateur un contrôle plus important sur ce qu'il peut faire. Il a le choix d'effectuer plus d'une opération à la fois.

Il arrive parfois que le recours à un mode soit la meilleure façon de résoudre un problème. La plupart des modes acceptables entrent dans l'une des catégories suivantes :

- des modes à long terme, tel que le traitement de texte par rapport à l'édition graphique.
- des modes "instables" à court terme, dans lesquels l'utilisateur doit constamment agir pour maintenir le mode activé. Maintenir la bouton de la souris enfoncé pour faire défiler une image ou maintenir la touche *Majuscule* pour étendre une sélection sont des exemples.
- le mode *Alerte* dans lequel l'utilisateur rectifie une situation anormale avant de pouvoir poursuivre

D'autres modes sont acceptables s'ils émulent une situation courante du monde réel qui est lui-même modal. Réaliser la sélection d'un outil dans une application graphique est semblable à la sélection d'un véritable outil de dessin dans la vie réelle.

Si l'environnement informatique a recours à des modes, il doit le signaler par un message clair et il indique le mode courant. Ce signal doit se trouver à proximité de l'objet le plus concerné par ce mode. Le pointeur variable de nombreuses applications graphiques en est un bon exemple. Selon la fonction (ou le mode) sélectionnée par l'utilisateur, le pointeur a une allure différente (voir le paragraphe *2.3.2 Le rôle du curseur*). Il faut également que l'utilisateur puisse facilement entrer ou sortir d'un mode en cliquant, par exemple, sur un symbole différent d'une palette.

En conclusion, les retours d'informations doivent être immédiats. Le système doit réagir à toute action ou toute suite d'actions pertinentes pour l'utilisateur. Les retours d'informations doivent être visibles : le système doit indiquer d'une façon ou d'une autre sa réaction. Et les retours d'information sont également instructifs : l'image doit montrer l'état des variables physiques.

2.4 Des principes standards

Les principes, que nous allons maintenant aborder, sont incontournables pour toutes les interfaces homme-machine, et donc aussi indispensables pour les interfaces de manipulation directe.

2.4.1 La cohérence

Un premier principe est celui de la cohérence. La cohérence repose sur le caractère unitaire des constituants de l'interface. Elle assure le fait qu'une suite des commandes à exécuter est identique pour réaliser une tâche commune. Le choix de la métaphore d'interaction sert d'élément fédérateur à l'homogénéité de l'interface. La cohérence est devenue maintenant une règle fondamentale dans la conception des interfaces homme-machine.

Cohérence - aspect lexical

Une difficulté rencontrée dans la définition du lexique est le choix d'une nomenclature. Très fréquemment la terminologie du système ou celle de l'application est floue ou elle ne correspond pas au vocabulaire de l'utilisateur. L'imprécision des termes des articles de menus ou une mauvaise icône sur un bouton peuvent conduire l'utilisateur débutant à formuler une intention mal appropriée.

Contrairement aux apparences, cette activité de dénomination n'est ni marginale, ni facile. De plus, le choix d'un terme ou d'une icône pour un bouton est influencé par l'héritage culturel du concepteur et des futurs utilisateurs.

Il faut veiller et il faut vérifier qu'une fonction sémantique ou un concept soit toujours désigné par le même nom et ceci quel que soit le contexte rencontré. L'exemple de la fonction "terminer" dans l'environnement UNIX est particulièrement évocateur. Pour spécifier la fin d'une saisie d'un message électronique, la notation lexicale est ".", tandis qu'elle est désignée par la lettre "q" pour quitter la messagerie et le mot "logout" pour terminer la session.

Cohérence - aspect pragmatique

La cohérence pragmatique concerne essentiellement la localisation et l'organisation spatiale des informations. La cohérence spatiale aide l'utilisateur à acquérir une connaissance motrice qui permet d'anticiper les actions physiques. Ainsi, l'utilisateur sait par avance où diriger son regard, comment orienter les mouvements de la souris ou quel doigt actionner. Cette cohérence pragmatique aide à franchir la distance articulatoire.

Un exemple devenu classique dans les environnements informatiques est la barre principale des menus. Toutes les commandes relatives aux fichier dans le premier menu, nommé justement "Fichier", tandis que celles sur l'édition du document, avec les services "Annuler/Répéter" et "couper-copier-coller" se situent dans le second menu, nommé "Edition". Ces deux menus sont devenus des normes pour les applications ainsi que sur les systèmes Macintosh et les compatibles PC.

2.4.2 Les commandes universelles

Les applications de manipulation directe ont démontré qu'il était possible de réduire le nombre des différentes opérations. Des opérations identiques sont utilisées à travers différentes tâches et dans plusieurs domaines d'application, exemple des commandes similaires dans un texte ou dans une figure. Elles sont appliquées comme des commandes universelles ou génériques.

La création de commandes génériques a été un des objectifs dans la conception du système de la "Star Machine" [Smith & al. 83]. Les exemples typiques de telles commandes génériques sont maintenant des fonctionnalités classiques comme *déplacer*, *copier*, et *supprimer*. Elles peuvent être appliquées de la même manière sans connaître la tâche spécifique et les objets. Les interfaces de manipulation directe facilitent l'utilisation de peu de commandes génériques parce qu'une grande part des fonctionnalités exigées sont en général représentées par la richesse sémantique des objets.

L'existence des commandes universelles facilite l'utilisation des environnements informatiques et elle permet le réinvestissement de l'expérience acquise avec d'autres environnements. Un avantage des commandes universelles se trouve dans un plus petit nombre de règles d'interaction à apprendre par l'utilisateur. Ainsi, les menus *Fichier* et *Edition* sont devenus des standards et ils sont fournis par les boîtes à outils, dans la construction d'une application, mais il reste encore quelques différences syntaxiques dans d'autres commandes : l'exemple du copier/coller entre le système Macintosh utilisant sa touche particulière "Commande" et les compatibles PC (surtout Microsoft) avec la touche "Contrôle". Remarquons aussi que les abréviations dans les environnements informatiques correspondent aussi à des normes. La différence entre Word5 et Word6 en français pour la commande *imprimer* et le raccourci clavier correspondant est un exemple flagrant : " ⌘-i" pour imprimer dans Word5 et "⌘-p" dans Word6, le "p" pour le verbe anglais "print".

Implicitement, pour les logiciels et les systèmes au niveau international, des normes de facto se sont imposées, généralement en référence à la langue de Shakespeare.

L'universalité des commandes devrait *transcender* les environnements informatiques. Au sein d'un même système, la commande devrait s'appliquer à tous les objets manipulés : le son, le texte, l'image, … pour les fonctions communes comme le service copier-couper-coller, l'effacement ou le déplacement. Un inconvénient peut être une plus faible efficacité parce que les opérations complexes doivent être composées de plus d'opérations génériques simples. Cette universalité des commandes doit être aussi appliquée pour d'autres types de commandes ou actions suivant les domaines modélisés. L'exemple classique en géométrie est l'intersection entre deux représentations d'objets, quels que soient les deux types d'objets.

2.4.3 Les macrocommandes

Les macrocommandes sont aux langages d'interaction ce que la procédure est aux langages de programmations : un mécanisme d'abstraction et une technique d'extension. En tant que mécanisme d'abstraction, elle répond au phénomène cognitif de l'apprentissage qui, avec l'expérience, encapsule un ensemble organisé de connaissance plus élémentaires. Etant une technique d'extension, elle permet de concilier le particularisme et la généralité.

Fournir un langage de haut niveau, à travers les macrocommandes pour suppléer les commandes complexes, est difficile mais pas impossible à mettre en place dans un environnement de manipulation directe. Les concepteurs utilisent alors un langage simple, comme le langage AppleScript dans le système d'exploitation du Macintosh, pour résoudre ce problème.

On se souvient de la distance sémantique à parcourir entre la formation de l'intention et la spécification du plan de commandes. Une façon de raccourcir cette distance est de fournir à l'utilisateur un langage de haut niveau qui lui permet d'exprimer directement les structures de décomposition des problèmes les plus fréquents. Il utilise alors les fonctionnalités de l'environnement informatique pour façonner ses propres procédures. Il construit ainsi son propre monde à partir des briques de base du logiciel.

2.4.4 Annuler/Répéter

Shneiderman [Shneiderman 83] énonce la règle suivante :

«*Permettre une annulation facile des actions.* Autant que possible, les actions doivent être réversibles. Cette caractéristique diminue l'inquiétude, car l'utilisateur sait que les opérations erronées peuvent être annulées. Il est donc encouragé à explorer des opérations qui ne lui sont pas familières. L'annulation peut porter sur une seule action, la saisie de données ou un groupe entier d'actions.» Tout le monde connaît l'importance de cette commande dans les traitements de texte.

Les transformations doivent être réversibles. L'utilisateur peut donc revenir en arrière en annulant physiquement une opération sur la structure de données (par exemple l'identification de sommets dans un graphe) ou sur la représentation graphique des objets (un déplacement, une représentation de la figure comme dessiner une représentation planaire d'un graphe planaire, ou un effet d'optique). Les objets doivent revenir dans leurs configurations ou leurs dispositions précédentes.

Dans les interfaces homme-machine, il existe deux modèles : l'annulation à un niveau et l'annulation à plusieurs niveaux. Dans le premier cas, seule la dernière commande de l'utilisateur est annulée, tandis que, dans le second cas, l'utilisateur peut annuler et répéter plus d'une commande, en général un nombre limité par les ressources informatiques disponibles.

Signalons que les nouveaux systèmes autorisent une annulation, illimitées en nombre, de toutes les actions effectuées. Une question importante, que le concepteur doit se poser, est de savoir quel type d'actions est à annuler.

Pour des raisons techniques, plusieurs environnements proposent le modèle à un niveau. La mise en œuvre des fonctions *Annuller* et *Répéter* repose sur une sauvegarde permanente de l'état du système. Pour rendre ces commandes plus lisibles et compréhensibles pour l'utilisateur, il est souvent nécessaire d'associer à chaque opération un nom particulier lié à son type. Ainsi la fonction *Annuller* renseigne l'utilisateur sur le type d'annulation à restaurer, par l'intermédiaire d'un article dans le menu *Edition* , avec un message succinct et non pas le simple verbe *"Annuler"*.

Le fait de sauvegarder l'état modifié peut être coûteux en temps et en place. D'un point de vue de l'interface homme-machine, plus l'utilisateur peut revenir en arrière, meilleure est l'interface. Néanmoins, au-delà de quelques niveaux d'annulation, le retour relève de l'ordre du confort, au détriment de la sécurité. Un moyen d'éviter le gaspillage de trop de ressources par ce confort est de limiter de nombre maximum de retours, en éliminant au fur et à mesure les commandes les plus anciennes.

2.4.5 Les préférences utilisateurs

Personnaliser le comportement du système avec un fichier préférence est considéré comme une liberté supplémentaire accordée à l'utilisateur. Les exemples sont multiples : la possibilité d'altérer un lexique, comme le changement de langues dans *Cabri-géomètre* II, le total paramétrage de la barre d'icônes dans *Word6*, une barre d'icônes pour des tâches répétitives ou spécifiques à chaque utilisateur.

Ces types de paramétrages, des valeurs par défaut initialisées par l'environnement ou choisies ultérieurement par l'utilisateur, peuvent se révéler fort utiles. Un autre exemple, bien particulier, est la liberté de choix dans les articles des menus de *Cabri-géomètre* II. Une liste d'articles est proposée pour chaque menu. L'utilisateur sélectionne ceux qui seront temporairement supprimés de la liste. Ainsi, dans le cadre d'un apprentissage de la géométrie, le professeur peut cacher des articles de construction répondant directement à un problème donné, et laisser l'élève avec le minimum d'outils lui permettant d'appréhender une nouvelle connaissance.

Remarquons qu'un certain nombre de ces principes, principalement dans les environnements informatiques d'apprentissage en mathématiques, ne peuvent s'appliquer que si l'utilisateur comprend ce qui se passe.

2.4.6 Des outils pour l'interface

La construction d'outils pour la réalisation d'interfaces correspond au désir de réaliser des interfaces efficaces et agréables dont la nécessaire complexité interne est difficile à gérer sans aide. Les services offerts par les outils sont calqués sur un modèle d'organisation de l'interface ; ils sont relativement évolués selon les outils. Les progrès des outils et des modèles sont donc étroitement liés.

Les boutons et menus déroulants

Permettre plusieurs actions après un clic de la souris et un temps d'attente sur la position offre aussi de multiples possibilités à l'utilisateur suivant le contexte : exemple des navigateurs sur le réseau internet ou de la gestion des ambiguïtés dans une sélection d'objets.

Cette simplicité dans le travail évite le classique passage aux menus déroulants sur la barre des menus ou des boutons dans une boîte d'outils appropriés. C'est plus qu'un double des commandes, c'est aussi une réduction de la charge cognitive de l'utilisateur. En effet, l'utilisateur n'a pas à se souvenir de tous les attributs ou toutes les possibilités associées à un objet ou à une situation particulière, le système, et donc l'interface, s'en charge.

Les éditeurs graphiques

On trouve plusieurs catégories d'outils : les boîtes d'outils, les squelettes d'applications, les langages de description, et les éditeurs graphiques. Les éditeurs d'interface sont les outils les plus difficiles à réaliser. En effet, leur but est de permettre la création d'une interface par les moyens les plus visuels possibles, ce qui exclut toute forme de programmation complexe par leur utilisateur. La plupart des boîtes à outils actuellement disponibles fournissent des menus, des boîtes de dialogues, la gestion des fenêtres et de boutons, prêtes à être utilisées. Avec de tels outils, on peut facilement réaliser des panneaux de contrôles et des formulaires. En revanche, ils offrent peu de supports pour des interfaces représentant les données de manières spécifiques au domaine, et utilisant des styles d'interaction variés. Ainsi, on ne trouve pas d'outils pour décrire une interface iconique ou un outil de dessin. On peut facilement imaginer comment décrire des représentations graphiques : les outils de dessin le font très bien. Mais le problème majeur reste celui de l'interaction, et du comportement des représentations graphiques.

3. Manipulation directe et technologie

Les interfaces de manipulation directe sont maintenant devenues très répandues grâce aux disponibilités, à faible coût, de trois technologies : 1) balayage graphique, spécialement dans la

visualisation bitmap[2], pour la représentation d'images complexes, 2) des outils de désignations, spécialement la souris, pour ses entrées analogiques rapides, et 3) la puissance des ordinateurs dédiés aux réactions dynamiques et intentionnelles face aux actions de l'utilisateur.

3.1 Les interactions de base

Ce paragraphe résume brièvement les principaux développements des technologies des interactions homme-machine. L'article de Myers [Myers, 96] nous a été d'un grand recours.

3.1.1 La manipulation graphique des objets

L'interface de manipulation directe, dans laquelle les objets visibles sur l'écran sont directement manipulables avec un pointeur de désignation, a été montrée en 1963 par Ivan Sutherland dans Sketchpad [Sutherland 63]. Sketchpad supportait la manipulation d'objets en utilisant un stylo optique, incluant la saisie des objets, leur déplacement, le changement de taille et en utilisant des contraintes. Il contenait, comme nous pouvons le remarquer, les prémisses de multiples idées importantes pour les interfaces.

Un système, AMBIT/G, implémenté au laboratoire Lincol au MIT (Massachusetts Institute of Technology, Etats-Unis) en 1968, utilisait la représentation iconique, la reconnaissance gestuelle, les menus dynamiques avec des items sélectionnables en utilisant un pointeur de désignation, sélection des icônes par le pointeur, et styles modals et non modals d'interaction.

David Canfield Smith a introduit le terme icône dans sa thèse sur Pygmalion en 1975 [Smith 75]. Dans les années 70, les chercheurs du centre de recherche de Xerox PARC ont travaillé sur les techniques de manipulation directe. Un de leur chercheur, Alan Kay, a entrevu le concept d'interface de manipulation directe à propos de "Dynabook" [Kay 77]. Les premiers systèmes utilisant la manipulation directe ont été Star Machine [Smith & al. 82], le Lisa d'Apple [Williams 83] et le Macintosh [Williams 84]. Ben Shneiderman a introduit le terme de manipulation directe et donné ses fondements psychologiques [Shneiderman 82, 83].

3.1.2 La souris

La souris a été développée au laboratoire de recherche de Standford (Etats-Unis) en 1965 pour remplacer le crayon optique, utilisé depuis 1954 [Goldberg 88, pp. 68]. La plupart de ses utilisations courantes ont été démontrées par Doug Engelbart en 1968 [Engelbart & al. 68]. Depuis, le succès de la souris n'a pas été démenti.

---

2 Le pixel est le point élémentaire d'une image ; c'est également l'élément de une mémoire vidéo qui correspond à un point de l'écran graphique lorque la fenêtre de visée comprend ce point. Sur un écran noir et blanc, chaque pixel

Par contre, il a existé beaucoup de simplifications de la souris sur le nombre de boutons : de trois boutons à deux boutons, puis à un seul bouton. Le Lisa d'Apple a instauré la souris à un bouton, mais permettant d'associer une ou plusieurs touches spéciales du clavier pour simuler des boutons virtuels pour certaines actions.

Cette notion du nombre de boutons n'est pas bien définie ; elle est subjective. Le test avec les utilisateurs sur lesquels l'équipe de la Star Machine a effectué les tests sur la souris avec un seul bouton n'ont pas été concluants. Aujourd'hui encore, les trois principales machines ont chacune leur propre nombre de boutons. Il existe un seul bouton pour la souris dans la gamme des Macintosh, et ceci a été un choix délibéré. Il existe deux boutons pour les souris avec les compatibles P.C., et trois pour les stations de travail.

3.1.3 Les fenêtres

Les fenêtres multiples ont été proposées par Engelbart en 1968 [Engelbart & English 68]. Alan Kay proposa l'idée de fenêtres pouvant se chevaucher dans sa thèse en 1969 [Kay 69]. Ce type de fenêtrage apparu en 1974 dans le système Smalltalk [Stallman 79] à Xerox PARC, puis dans le systèmes InterLisp [Teitelman 77], plus tard Xerox Star (1981), le Lisa d'Apple (1982), le Macintosh d'Apple (1984) et enfin avec Microsoft avec Windows (1990-1995). Le système X-Window a été développé par le MIT en 1984 [Scheifler & Gettys 86]

Il faut différencier plusieurs types de gestion des fenêtres. Le Macintosh considère active la fenêtre se trouvant *physiquement* au-dessus des autres. Pour activer une autre fenêtre, il faut que l'utilisateur explique clairement son choix en la sélectionnant. Par contre, sur les stations de travail, la position de la souris détermine la fenêtre active. Ainsi, une fenêtre presque complètement dissimulée par d'autres pourra être la fenêtre active si la souris se trouve sur le dernier morceau visible. Ce qui entraîne, évidemment, un trouble et une source d'erreur chez l'utilisateur, qui pense travailler dans une fenêtre précise, mais qui, en fait, effectue des commandes dans une autre.

Les différents moyens de désignation (des capteurs sensibles à la position de la tête ou du regard, ou des entrées audio), de retours d'informations plus complets (les sorties audio ou la téléolfaction [This 97]), ou encore de rétroactions (dans le joystick ou la souris, ou pour les simulateurs de vol) sont d'autres améliorations dans les interactions homme-machine. Ils sont associés avec les outils des interfaces de la *réalité virtuelle*.

---

est soit noit, soit blanc et peut donc être représenté par un bit mémoire. Cet affichage est ainsi appelé bitmap (en mode point).

3.2 La réalité virtuelle ou la réalité augmentée

La réalité virtuelle est une simulation par ordinateur dans laquelle le graphisme est utilisé pour créer un monde qui semble réaliste. De plus, le monde synthétisé n'est pas statique mais répond aux ordres de l'utilisateur (les gestes, la parole ou d'autres outils d'entrée). Plus généralement, un système de réalité virtuelle est une interface qui implique de la simulation en temps réel et des interactions via de multiples canaux sensoriels. Ces canaux sensoriels sont ceux de l'homme : la vision, l'ouïe, le toucher, l'odorat, et le goût.

La métaphore du monde, virtuel ou artificiel, est un modèle à la mode depuis quelques années. Elle est utilisée dans des systèmes de réalité virtuelle comme les simulateurs de vol, par exemple, ou des jeux en trois dimensions utilisant un casque.

3.2.1 Métaphore du monde virtuel

Dans un système de réalité virtuelle, la métaphore n'est pas simplement capturée sur l'écran. Pour être plus précis, l'utilisateur est lui-même décrit à l'intérieur de la métaphore, créant littéralement une alternative ou une virtuelle réalité. Aucune action que l'utilisateur accomplit n'est supposée devenir plus naturelle, et ainsi les mouvements de l'utilisateur sont interprétés, au lieu de touches de clavier pressées, de clics de boutons ou de mouvement d'un outil de pointage externe comme la souris. Un système sur la réalité virtuelle a aussi besoin de la position et de l'orientation de l'utilisateur. Par conséquent, l'utilisateur est souvent équipé avec des systèmes de repérage afin que l'environnement informatique puisse les localiser et interpréter leur mouvement correctement [Burdea & Coiffet 93].

La réalité virtuelle ne doit pas être confondue avec la réalité augmentée. Avec cette dernière l'utilisateur interagit avec le monde réel par les voies naturelles, et simultanément il utilise le calculateur pour les informations synthétiques l'aidant à mieux appréhender son environnement.

3.2.2 Historique

La réalité virtuelle n'est pas la dernière invention à la mode puisqu'elle date de plus trente ans. Le brevet américain n° 3 050 870 a été accordé à Morton Heilig pour son invention du "Sensorama Simulator".

Cette préfiguration d'une station de travail en réalité virtuelle possédait un système vidéo, en trois dimensions, obtenu par des caméras 35 mm accolées. Elle fournissait du mouvement, de la couleur, du son stéréo, des odeurs, du vent à l'aide de petits ventilateurs situés près de la tête de l'utilisateur et un siège vibrant. Ainsi, il était possible de simuler un voyage à motocyclette à travers New-York, où le conducteur pouvait sentir le vent aussi bien que les nids de poule de la route grâce au siège vibrant (voir figure 1.9).

Figure 1.9 : Première simulation de la réalité.

3.3.3 Les outils de la réalité virtuelle

L'une des trois composantes, caractérisant la réalité virtuelle, est l'interactivité. Pour qu'il y ait interaction entre l'homme et la machine, il est nécessaire d'utiliser des outils spéciaux conçus à la fois pour permettre d'accéder au calculateur et pour en recevoir un retour d'informations. Aujourd'hui, les outils sont variés tant en fonctionnalité qu'en objectif, pouvant s'adresser à plusieurs canaux sensoriels de l'homme. Par exemple, le mouvement du corps est poursuivi par des capteurs de position tridimensionnels, les gestes de la main sont modélisés par des gants sensitifs, le retour visuel est envoyé sur des écrans stéréoscopiques, le son virtuel est calculé par des générateurs de sons, la perspective et l'orientation de la scène sont changées à l'aide de boules ou de poignées de commandes. Quelques-uns de ces systèmes d'entrée/sortie se trouvent sur le marché, d'autres sont encore des prototypes. Ceci montre que les champs de recherche sont encore actifs, nouveaux et prometteurs.

Les capteurs d'entrée

Les capteurs d'entrée présentés sont des outils actuellement utilisés.

la souris

Le déplacement dans un monde virtuel avec une souris nécessite un temps d'immersion important. En effet, une souris se déplaçant dans un espace à deux dimensions, il faut traduire ce déplacement en trois dimensions.

les gants sensitifs

Les gants permettent de détecter le mouvement des doigts, ainsi que les déplacements de la main. Il existe trois types de gants.

Les gants utilisant les fibres optiques, comme le DataGlove, inventé par Thomas Zimmerman. Ce gant a été le premier à être commercialisé dès 1987. Des fibres optiques sont posées au-dessus de chaque doigt. Dès qu'un doigt fait un mouvement, la fibre optique est déformée, on peut alors déterminer l'angle articulaire par mesure de l'intensité de la lumière réfléchie par la fibre.

Un autre type de gant utilise des capteurs à encre conductrice, comme le PowerGlove de Nintendo. Des capteurs sont placés au-dessus des doigts, lorsque l'on plie le doigt, la longueur du capteur augmente, ce qui accroît la résistivité qui est traduite en une valeur angulaire.

Enfin, une structure métallique exo-squelettique peut être mise sur un gant, comme le Dextrous Hand Master d'Exos Inc. 1990. Les angles articulatoires sont mesurés par des capteurs à effet Hall placés aux articulations de la structure mécanique. Les doigts doivent conserver une liaison étroite avec la structure.

Reconnaissance de la parole

La reconnaissance de la parole est un sujet d'étude depuis plusieurs années.

Suivi du regard

Une caméra suit l'œil de l'utilisateur pour déterminer ce qu'il regarde.

Les capteurs de sortie

Les capteurs de sortie correspondant aux sens de l'homme.

La vue

La perception de la profondeur peut se faire avec un œil (monoscopique) ou bien avec les deux yeux coopérant (stéréoscopie). Dans tous les cas, il faut tromper le cerveau dans la perception des images. Deux techniques sont utilisées : des casques ou des lunettes.

La sensation d'immersion donnée par les lunettes stéréo est différente de celle fournie par les casques, car ces deux techniques répondent à des paradigmes différents de la réalité virtuelle. Avec les lunettes, l'utilisateur n'est pas solidaire de l'écran de visualisation. Plutôt que d'être entouré par le monde virtuel, il voit ce dernier à travers une fenêtre.

L'ouïe

Un système audio-virtuel est une expérience sonore enregistrée qui contient une information psycho-acoustique significative permettant une altération de la perception humaine de telle sorte que la personne croit que l'expérience sonore enregistrée a lieu réellement.

Lorsqu'une balle rebondit devant nous, un son monophonique est suffisant. Si la balle rebondit hors du champ de vision, l'utilisateur ne peut plus dire où la balle se trouve. Le

système de simulation doit être doté d'un outil qui modifie le bruit pour permettre à l'opérateur de savoir d'où le bruit provient.

Il existe des difficultés de représentation. Les sons dans une pièce réelle rebondissent sur les murs, le plafond et le plancher. Ces sons réfléchis s'ajoutent à ceux reçus directement de la source sonore. Un modèle de calcul, pour un environnement contenant six murs et quatre sources sonores, nécessite environ un milliard d'opérations par seconde.

Le touché

Les sensations de toucher sont principalement rendues par les retours tactiles et le retour des forces. Le toucher est très important, notamment pour toutes les simulations où le champ de vision est occulté. Le retour tactile fournit une information sur la géométrie de la surface de contact, sur la rugosité, la température ou même le glissement d'un objet. Le retour d'effort donne une information sur la force totale de contact, l'élasticité, ou le poids.

Retours tactiles et retour de force

Le retour tactile pneumatique consiste à équiper un gant de micro-ballons, tandis que le retour vibro-tactile consiste à chauffer un fil à mémoire de forme. Lorsqu'un courant passe, le fil chauffe et donc se déforme et vient toucher un doigt, par exemple.
Le bras à retour d'effort contient des moteurs qui effectuent des pressions sur l'utilisateur.

Dans des situations particulières, où le champ de vision est occulté ou totalement noir, comme l'entraînement des chirurgiens sur des corps virtuels, nous avons besoin de notre sens du toucher virtuel tout comme dans la vie réelle.

La littérature technique mélange souvent retour tactile et retour d'effort. Considérons une main appuyant très légèrement sur un bureau. Les premiers capteurs à répondre sont ceux situés sur l'extrémité des doigts. Si la main appuie plus fort, les muscles de la main et de l'avant-bras commencent à se mobiliser. Les forces sont maintenant ressenties par les capteurs situés sur des muscles et les os (kinesthésie) et non simplement aux extrémités des doigts.

Cet exemple montre que les capteurs tactiles fournissent des informations sur la géométrie de la surface en contact. Est-ce une surface plane ? Est-ce le bord d'un bureau ? Les capteurs tactiles fournissent aussi des données sur la rugosité de la surface de contact, sa température ou même le glissement d'un objet entraîné par la pesanteur. De manière complémentaire, le retour d'effort donne une information sur la force totale de contact. Les capteurs peuvent aussi donner une information sur l'élasticité de l'objet, sur la surface de contact (flexible ou rigide), ou sur le poids de l'objet à saisir (lourd ou léger).

Conclusion

Certains principes de manipulation directe sont plus flagrants ou visibles pour certains domaines que pour d'autres. Les mathématiques en général, et la géométrie en particulier, sont pour ainsi dire le domaine par excellence où les propriétés et les principes de la manipulation directe peuvent être complètement exploités. Les interfaces à manipulation directe ont pour origine l'utilisation d'un mode d'interaction basée sur la métaphore du monde réel.

Au cœur de notre problématique, une interface centrée sur l'utilisateur, se situent la notion de manipulation directe et le type de métaphore sur lequel la manipulation directe se fonde et s'appuie [Shneiderman 82]. Pour [Laborde 85], la manipulation directe est souvent liée au principe de métaphore, mais ils ne vont pas forcément de pair, même s'ils constituent un couple fort.

Ce nouveau type d'interface utilisateur a été aussi transporté dans les domaines d'applications du système, comme les éditeurs de texte ou les éditeurs graphiques. Il faut remarquer que s'il est relativement facile de concevoir des interfaces à désignation directe, où l'on propose à l'utilisateur d'appuyer sur un bouton au lieu d'invoquer une commande via une entrée au clavier, il est beaucoup plus difficile de réaliser une application véritablement intégrée fonctionnant en manipulation directe. En effet, une difficulté majeure du concepteur d'interface de manipulation directe est l'identification des métaphores sur lesquelles les actions directes de l'utilisateur vont pouvoir se bâtir.

[Nanard 1990, pp. 41] *«En réalité, il n'est pas possible de caractériser la manipulation directe exclusivement par la forme externe des interfaces. Celle-ci n'est pas à elle seule la cause de leur puissance.»*

Chapitre 2

Etat de l'art sur les éditeurs des graphes

Depuis plusieurs années, de nombreux environnements informatiques ont été réalisés pour les mathématiques discrètes. Des domaines très variés comme la recherche pure, la recherche centrée sur les applications, ou les études pédagogiques et d'apprentissage, ont bénéficié de ces outils. Les applications plus spécifiques à la théorie des graphes ont aussi été développées. Elles sont de nature assez semblable, mais bien particulière dans leur approche de leur domaine d'application. Depuis le début des années 90, de nouveaux environnements informatiques sur la théorie des graphes sont apparus.

Dans ce chapitre, et dans un premier temps, nous donnons les principales définitions relatives à la théorie des graphes. Ensuite, nous nous attardons sur les différentes motivations accordées à ces environnements informatiques (professionnel, recherche ou enseignement). Nous définissons différents aspects relatifs à ces systèmes : des critères visuels (représentation et apparence du graphe), des fonctionnalités (structurelles ou graphiques) et le mode d'interaction proposé.Ensuite, suivant cette approche, nous analysons plusieurs environnements, que nous avons classés en trois catégories : les bibliothèques de procédure, les "*visualiseurs*" de graphes et les éditeurs de graphes.

1. Définitions sur les graphes

La théorie des graphes possède son propre langage. Il peut sembler difficile dans un premier temps, parce qu'il contient de nombreux termes spécialisés dont la signification n'a souvent aucun rapport avec le sens commun. Le but de ce paragraphe est de présenter vocabulaire utilisé sur les graphes afin de décrire les principaux modèles théoriques et informatiques sur les graphes et de présenter des algorithmes.

Les définitions sont pour la plupart tirées des livres de Harary [Harary 69], de Berge [Berge 73, 83] et de N'guyen Huy Xuong [Xuong 92].

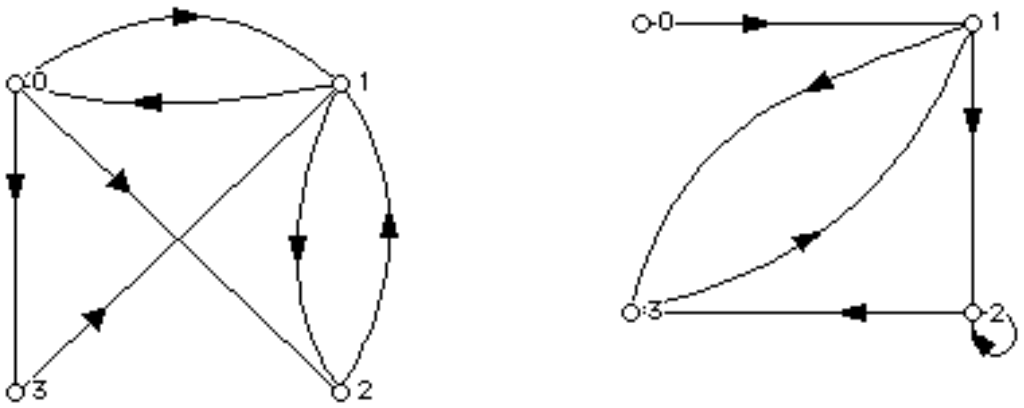
Définitions

Un graphe $G = (V, E)$ est un couple constitué par un ensemble $V = \{ v_1, v_2, \ldots, v_n ; n \geq 1 \}$ et par une famille $E = \{ e_1, e_2, \ldots, e_m ; m \geq 0 \}$ d'éléments du produit cartésien $V \times V = \{ (u, v) / u \in V, v \in V \}$. Un élément (u, v) de $V \times V$ peut apparaître plusieurs fois dans la famille E.

Les élements de V sont appelés les sommets de G, et le nombre de sommets de G est appelé *l'ordre* de G.

Les éléments de E sont appelés les arcs de G. Un arc de la forme (u, u) est appelée une *boucle*. Pour un arc $e = (u, v)$, l'élément u est son extrémité initiale, et v est son extrémité terminale.Si plusieurs éléments de la forme (u, v) de $V \times V$ apparaissent dans E, alors les arcs sont appelés *parallèles* ou *multiples*. Le graphe est alors appelé un *multi-graphe* ou *graphe multiple*.

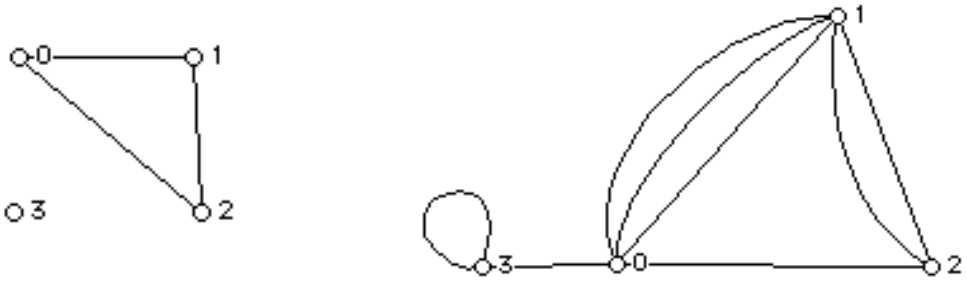
Un *p-graphe* est un graphe avec au plus p arêtes entre deux sommets.G est appelé graphe *multiple*.



(a) Un graphe simple orienté. (b) Un graphe orienté avec une boucle.

Figure 2.1 : Exemple de deux graphes orientés.

Si aucun ordre n'est précisé sur les extrémités des arcs, alors les élements de E sont appelées des arêtes et le graphe G est dit non orienté, sinon le graphe G est dit orienté.



(a) Graphe simple. (b) Graphe multiple avec boucle.

Figure 2.1 : Deux autres exemples de graphes.

Soit *u* un sommet du graphe *G*. Le sommet *v* est un *successeur* de *u*. s'il existe un arc ayant son extrémité initiale en *u* et son extrémité terminale en *v*. L'ensemble des successeurs de *u* se note : $\Gamma^+(u)$

De même, le sommet *v* est un *prédécesseur* de *u*. s'il existe un arc ayant son extrémité initiale en *v* et son extrémité terminale en *u*. L'ensemble des prédécesseurs de *u* se note $\Gamma^-(u)$.

L'ensemble des sommets voisins du sommet *u* se note

$\Gamma(u) = \Gamma^+(u) \cup \Gamma^-(u)$

Lorsque $p = 1$, le *p*-graphe devient un 1-*graphe*. Les éléments e_i étant distincts, *E* devient un ensemble de *m* éléments (au lieu d'une «famille»), et on peut écrire :

$E \subset V \times V$, et $|E| = m$

Deux arcs (ou deux arêtes) sont dits *adjacents* s'ils ont au moins une extrémité commune.

Si un sommet *u* est l'extrémité initiale d'un arc $e = (u, v)$, avec $u \neq v$, on dit que l'arc est *incident à u vers l'extérieur*. Dans un graphe *G*, le nombre d'arc de la forme (u, v) se note $d^+(u)$, et s'appelle le *demi-degré extérieur* de *u*.

On définit de même un arc *incident à u vers l'intérieur* et le *demi-degré intérieur* $d^-(u)$.

$d(u) = d^+(u) + d^-(u)$ est le nombre d'arcs ayant une extrémité en *u* (chaque boucle étant comptée deux fois), et s'appelle le degré de *u*.

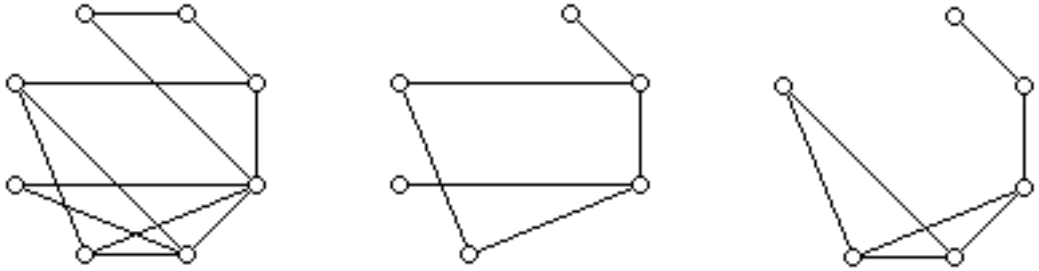
Un *graphe simple* est un graphe ne possédant ni arcs multiples ni boucles.

Un graphe *dense* est un graphe possédant beaucoup d'arêtes par rapport au nombre de sommets. Pour un graphe simple, le nombre des arêtes *m* est en fonction du nombre *n* de sommets du graphe. Il est maximum pour la valeur de $m' = n(n - 1)/2$. En général, un graphe

simple est dense pour une valeur $m \geq 2/3$ m'. Inversement, un graphe est dit *clairsemé* s'il possède peu d'arêtes.

Parties de graphes

Le *sous-graphe* de *G* engendré par $S \subset V$ est le graphe le graphe G_S dont les sommets sont les éléments de *S*, et dont les arcs sont les arcs de *G* ayant leurs deux extrémités dans S.



(a) Graphe initial (b) Sous-graphe (c) Graphe partiel

Figure 2.3 : Exemples de sous-graphes.

Le graphe partiel de *G* engendré par $F \subset E$ est le graphe le graphe $G_S = (V, F)$ dont les sommets sont les éléments de *S*, et dont les arcs sont les arcs de *F*. Autrement dit, on élimine de *G* les arcs de *E* - *F*.

Un *sous-graphe partiel* de *G* est un sous-graphe d'un graphe partiel de *G*. Si G est le graphe des routes de France, la carte routière des routes nationales est un graphe partiel, la carte routière de l'Isère est un sous-graphe ; la carte routière des routes nationales de l'Isère est un sous-graphe partiel.

Représentation graphique des graphes

L'étude d'un graphe s'associe généralement de la représentation "*graphique*" du graphe sur une feuille de papier ou sur l'écran de l'ordinateur. Cette représentation est elle-même devenue un sujet d'étude englobant la représentation graphique de données sur un écran d'ordinateur.

A chaque sommet d'un graphe, on associe généralement un point ou un petit cercle du plan comme sa représentation et à chaque arête un arc de Jordan reliant ses extrémités. Un arc de Jordan est un segment, une succession de segments appelés généralement un segment brisé ou une courbe régulière.

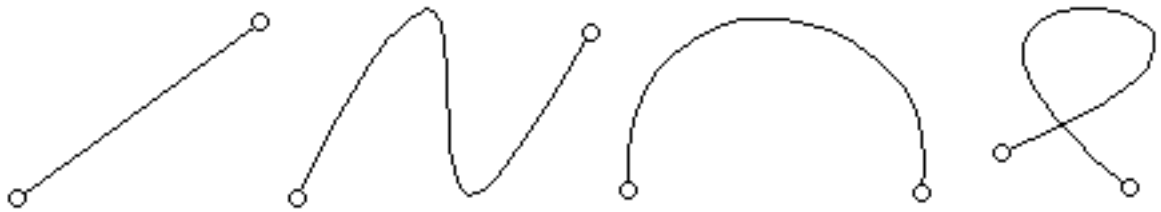


Figure 2.4 : les diverses représentations d'une arête.

La plupart des environnements informatiques sur les graphes utilisent le segment brisé pour représenter une arête.

2. Pourquoi un éditeur de graphes ?

La construction d'un éditeur de graphes ou plus simplement d'un "visualiseur"[1] de graphes est utile pour diverses raisons. La visualisation de données abstraites est un exemple. Des études sur la topologie des graphes, et en particulier des graphes planaires, ont conduit à l'élaboration de nouveaux algorithmes de reconnaissance des graphes planaires et de dessin automatique. La manipulation des données abstraites et l'étude des problèmes modélisés par des graphes ont contribué aux constructions d'éditeurs de graphes.

L'évolution des environnements sur les graphes s'est produite en plusieurs phases, chacune suivant ses propres motivations. Dans ce contexte, il serait malaisé de différencier les deux notions d'éditeur et de visualiseur, dans le sens où leurs développements ont été conjoints et presque similaires. Il faut signaler, malgré tout, des différences assez franches dans leurs conceptions. Nous exposons dans le prochain paragraphe les principaux environnements informatiques actuellement disponibles.

2.1 Les motivations

Tout développement d'environnements informatiques est initialement motivé par des manques ou des lacunes dans les systèmes déjà existants, ou par le souci d'effectuer le mieux possible des tâches répétitives ou complexes par l'ordinateur. Parallèlement, des préoccupations plus professionnelles sont venues renforcer les premières réalisations informatiques. Plus récemment, des études didactiques ont conduit à concevoir des systèmes plus modestes et mieux adaptés à l'enseignement et à l'apprentissage des concepts de bases de la théorie des graphes.

---

[1] Le mot "visualiseur" est une traduction du mot anglais "visualizer", construit récemment sur la racine du verbe anglais "to visualize". Il signifie que l'environnement informatique permet de représenter sur un écran d'ordinateur des données, mais qu'il n'autorise aucune modifications de ces dernières.

Aujourd'hui, avec le développement rapide du réseau mondial de communication, un nouvel élan a été donné dans le développement d'environnements informatiques pour analyser et comprendre cet immense réseau.

Dans sa thèse [Tallot 85], Didier Tallot fait remonter le premier éditeur de graphes à 1969 [Wolfberg 69][2]. Les progrès dans ce domaine ont été fortement liés aux avancées constatées dans les domaines du génie logiciel et des interfaces homme-machine. L'arrivée des interfaces graphiques et des pointeurs de désignation comme la souris ont ainsi ouvert de nouvelles perspectives de développement.

2.1.1 Motivation initiale

Les premières motivations avancées pour la réalisation d'un éditeur interactif de graphes sont généralement proches de celles exprimées dans le document de constitution du groupe Cabri dans le cadre du groupe de travail PRC[3] Mathématique et Informatiques [Habib & Laborde 83]

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

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

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

Cet objectif initial est aujourd'hui renforcé par l'expérience. En effet, on a montré que l'intérêt du développement d'un tel cahier de brouillon informatique ne résidait pas uniquement dans sa mise à disposition aux chercheurs et aux enseignants. L'utilisation par des étudiants d'un éditeur de graphes, pour l'expérimentation d'outils et de concepts informatiques, en théorie

---

[2] Le lecteur intéresse par une historique du problème pourra se référer à l'article de [Dao & al. 86].

[3] PRC : Programme de Recherche en Commun regroupe des chercheurs français dont l'objectif est la collaboration entre les laboratoires et les universités.

algorithmique des graphes, ou pour l'apprentissage de nouveaux concepts dans ces domaines, apporte une motivation supplémentaire. De plus, il est ainsi possible de proposer dans le cadre du développement de cet environnement informatique une très grande variété de projets de fin d'études.

Dans certains sujets de projets, le graphe est considéré comme un objet et on demande, par exemple, aux étudiants l'élaboration et la construction d'algorithmes offrant une représentation visuelle des graphes respectant des propriétés graphiques ou structurelles.

2.1.2 Visualisation de données abstraites

Le domaine des mathématiques discrètes a la particularité de modéliser rapidement des problèmes de la vie réelle et de les résoudre. Des contrats de recherche, entre les chercheurs et le monde industriel, ont favorisé le développement des environnements informatiques. La recherche opérationnelle et le voyageur de commerce, l'ordonnancement des outils sur une machine, ou la représentation des graphes et la réalisation de circuits imprimés, sont des exemples parmi d'autres. Un éditeur de graphes s'avère donc très utile à toute personne utilisant les graphes comme outil de modélisation s'appuyant sur une représentation graphique.

Dans ce dernier exemple, la représentation graphique des graphes sur un plan a contribué d'une part à des développements théoriques sur le test de planarité d'un graphe, et d'autre part à la construction de nouveaux algorithmiques et de nouvelles heuristiques pour dessiner automatiquement les graphes suivant des contraintes données. Elle a également permis, dans un cadre plus général, l'étude de la représentation de données abstraites, de leurs manipulations et de techniques permettant de mettre en évidence des parties pertinentes ou des détails de ces données par rapport à l'ensemble.

Dans la visualisation de graphes, le problème principal consiste à placer les sommets, les arêtes et les étiquettes du graphe dans le plan afin que le dessin puisse rendre visibles la structure et les propriétés des graphes. Les calculs de *"jolis"* dessins de graphes sont quantitativement difficiles, et les graphes sont souvent denses. Il peut exister des graphes pour lesquels certains critères sont contradictoires. Par exemple, pour éviter que des arêtes soient dessinées à travers un sommet, il peut être nécessaire que l'arête soit déformée (utilisation d'une représentation de l'arête par un segment brisé ou par une courbe).

Le fait qu'un graphe soit joli dépend aussi d'un sentiment personnel de l'utilisateur. Bien sûr, il existe des critères communs d'esthétique qui sont utilisés par la plupart des outils de dessins de graphes pour trouver de bons placements pour les sommets. Nous donnons une liste non exhaustive d'exemples : placer les sommets suivant une hiérarchie de niveaux, éviter les croisements des arêtes et des sommets, garder la représentation des arêtes par un segment et non par une courbe ou un segment brisé, minimiser la longueur des arêtes, favoriser les placements

des sommets (par exemple, placer les sommets sur une grille), placer des sommets de même nature proche les uns des autres, etc.

Un outil interactif de dessins de graphes devrait utiliser des heuristiques et des fonctionnalités pour réduire la quantité d'informations à visualiser lorsque la taille du graphe devient grande. En effet, pour des graphes denses, les objets (les sommets, les arêtes et les étiquettes) sont souvent superposés, alors que des sous-graphes sont jugés plus importants que le reste du graphe dans certaines situations. Il doit alors être possible d'assigner des priorités sur ces sous-graphes dont la représentation doit être le plus lisible possible.

2.1.3. L'apprentissage des mathématiques

L'application des graphes à des problèmes de décision ou d'optimisation, ou l'étude de quelques algorithmes de graphes n'est abordée, en France, que dans l'enseignement supérieur : les licences et les maîtrises de mathématiques ou d'informatiques, les écoles d'ingénieurs, les instituts universitaires de technologies, et les facultés de sciences économiques. On peut formuler plusieurs remarques.

Le sujet ne fait pas l'objet d'un cours complet. Une partie importante des concepts sous-jacents à la théorie des graphes est survolée. Seuls quelques problèmes classiques sont traités comme les problèmes de plus courts chemins, ou les méthodes d'ordonnancement.

Des méthodes générales et puissantes d'optimisation sont présentées, comme la méthode du recuit simulé ou la méthode *tabou* . Les algorithmes sont décrits trop souvent de manière littéraire et peu détaillée et uniquement pour une application manuelle. Ils exigent encore beaucoup de travail pour être programmés.

Le plus souvent, aucune application n'est mentionnée, ou alors le problème type et l'algorithmique sont introduits pour résoudre un seul problème typique de la discipline (en sciences économiques ou en automatique, par exemple). Il manque en fait des applications diverses d'un même modèle, tous domaines confondus. Cela fait que des personnes d'une autre discipline ne voient pas l'intérêt potentiel d'un modèle pour son travail.

Par contre, en Hongrie, l'enseignement des mathématiques discrètes et en particulier de la théorie des graphes commence dès le secondaire [Tankönyvkiadó 85]. Des notions classiques sur les graphes sont abordées par la résolution de problèmes simples (la notion de graphe eulérien ou de planarité d'un graphe) ou par l'approche de technique de résoltions (la méthode de Dirichlet appelée la cage à pigeons).

Au Canada, un enseignement des mathématiques discrètes (et des graphes) a été donné dans certaines régions de la province du Québec au niveau secondaire. Puis, cet enseignement a été abandonné au profit des mathématiques "pures". Prochainement en 1997 ou en 1998, des outils de modélisation et d'optimisation seront développés à l'aide de modèles tels que les graphes. L'objectif de ce module d'apprentissage des mathématiques [Bouchard 97] est double :

développer des connaissances afin de percevoir l'utilité des outils mathématiques dans la vie courante, et aborder des concepts de la théorie des graphes.

Aux Etats-Unis, NCTM (National Council of Teachers of Mathematics[4], 1989) a préconisé la mise en place de la théorie des graphes et des autres domaines des mathématiques discrètes dans les nouveaux programmes d'enseignement. Une tendance récente mais grandissante est d'enseigner plus tôt les mathématiques discrètes dans la scolarité (curriculum mathématique et les sections K-12 pour les américains). Les mathématiques discrètes deviennent un élément important des études supérieures dans les universités à partir de 1980. Depuis, il est apparu important pour NCTM d'encourager l'enseignement de cette discipline dans les programmes scolaires du secondaire, au collège et au lycée. En 1989, l'enseignement des mathématiques discrètes est devenu un des standards au niveau secondaire pour NCTM dans son article "The NCTM's Curriculum and Evaluation Standards for School Mathematics". NCTM a depuis publié un livre, "Discrete Mathematics Across the Curriculum K-12" [NCTM 89], contenant beaucoup de matériel sur les mathématiques discrètes, leurs enseignements et leurs utilisations dans le cursus scolaire, par exemple des références à des ouvrages, à des articles et surtout à des environnements informatiques.

Une orientation de l'enseignement des mathématiques est l'utilisation de nouveaux outils, comme l'ordinateur et des environnements d'apprentissage.

2.1.4. Les utilisations récentes

Depuis 1992, de nouveaux environnements informatiques de visualisation de graphes sont apparus. Ils correspondent à deux tendances. La première tendance est la mise à disposition d'outils informatiques à des fins pédagogiques et d'apprentissage des mathématiques discrètes, et en particulier des notions et des propriétés simples sur les graphes. La deuxième tendance est à la demande expresse des industriels et des informaticiens. Elle consiste en l'élaboration d'outils complexes et assez complets pour visualiser différents types de réseaux : les canalisations des villes, les réseaux de métro, les circuits électriques. Un autre fait marquant est l'expansion du réseau *internet* et par conséquent des réseaux informatiques locaux ou globaux. Ces sites internet utilisent les éditions de liens de type hypertexte, qui peuvent être représentés par des graphes.

Des environnements informatiques permettent d'observer en temps réel les activités d'un réseau en visualisant les connexions entre les machines. Malheureusement, ces environnements ne possèdent aucun outil pour modifier le graphe des connexions. D'autres outils, comme le

[4] Cet organisme indépendant est chargé de "planifier" les programmes de mathématiques dans l'enseignement américain.

système AOLpress [AOLpress 95], aident les utilisateurs à construire des sites en représentant l'architecture du site.

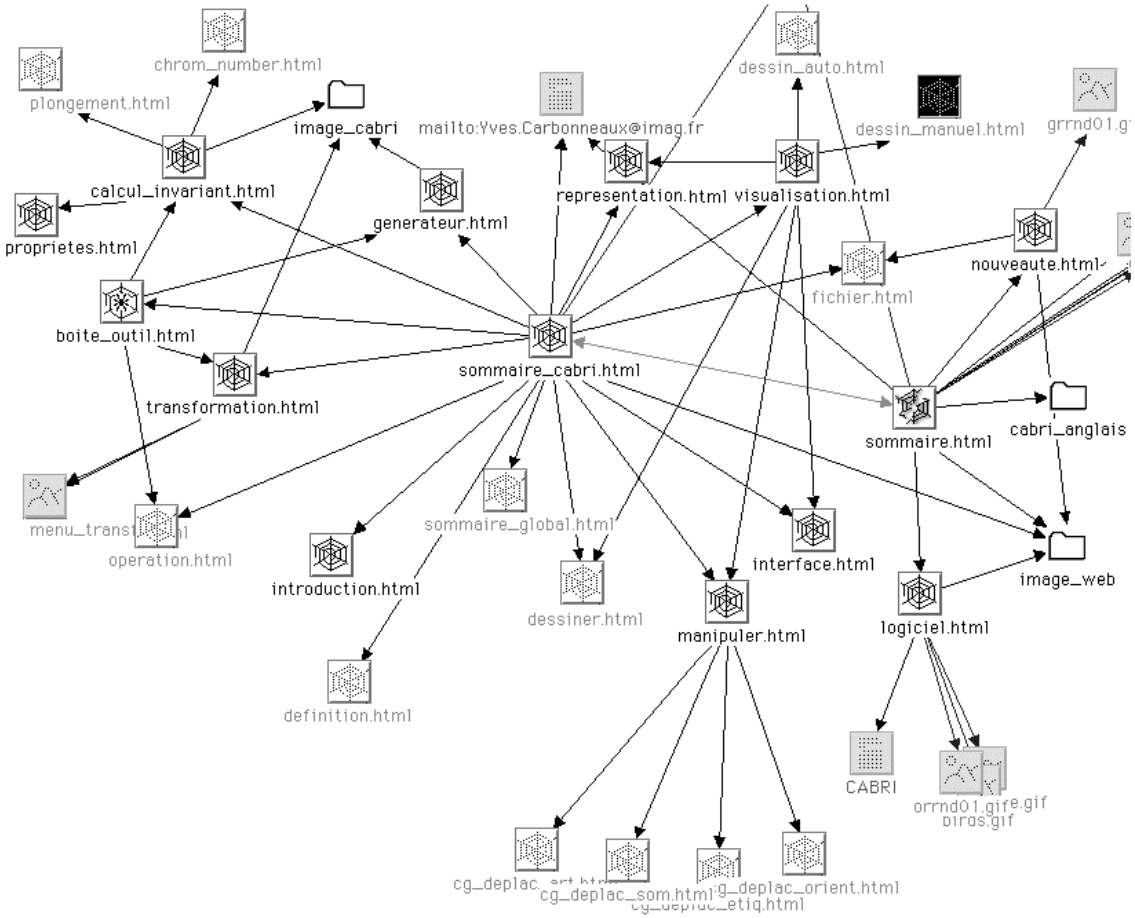


Figure 2.5 : Début de l'architecture du site internet relatif à *Cabri-graphes*.

La figure 2.5 décrit des fichiers composant le site internet[5] relatif à notre environnement actuel *Cabri-graphes* [Carbonneaux 97]. Les sommets de ce graphe sont les fichiers textes et les arcs sont les liens hypertextes entre les fichiers. Le logiciel représente une simple visualisation du graphe décrit avec des facilités de manipulation.

2.2. Critères d'analyse

L'objectif de ce paragraphe est de faire ressortir des critères d'évaluation des environnements informatiques sur les graphes, et donc dans un premier temps des critères d'analyse. Il est très difficile d'évaluer des environnements informatiques, et surtout d'entreprendre une comparaison des uns par rapport aux autres. Conçus spécifiquement pour un problème donné, ils ont chacun

[5] L'adresse éléctronique du site web sur *Cabri-graphs* est http://www-cabri.imag.fr/CabriGraphes/ à la date du 01 octobre 1997.

leur philosophie propre. Cependant de grandes lignes de comparaisons se dégagent et seront nécessaires pour procéder à une étude qualitative de ces environnements.

Plusieurs axes de recherche recoupent nos propres centres d'intérêt comme la visualisation des données, les fonctionnalités de l'environnement et les modes d'interaction, en s'attachant particulièrement à la manipulation directe.

2.2.1. Visualisation

Il existe deux aspects important dans la visualisation d'un graphe : les procédés de dessin des graphes et les représentations graphiques des graphes sur un écran.

Représentation du graphe

Les représentations graphiques du graphe s'effectuent de deux manières différentes : une méthode manuelle ou une méthode automatique. Le premier procédé laisse l'utilisateur libre de placer les sommets et les arêtes du graphe manuellement, par simple clic de la souris, par exemple. Or, avec des graphes denses, cette approche est souvent une source d'erreurs et elle devient vraiment longue à effectuer. Le second procédé dessine automatiquement les graphes. Il est préférable d'utiliser cette méthode pour des graphes denses, mais il ne devrait pas pour autant négliger ou supprimer l'aspect manuel. La représentation automatique permet aussi de mettre en évidence des propriétés intrinsèques au graphe ou des propriétés géométriques de sa représentation, comme la planarité du graphe.

Apparence du graphe

Le deuxième aspect important est l'apparence du graphe. En plus de l'esthétique du graphe, un atout supplémentaire est la possibilité de choisir les caractéristiques graphiques des sommets, des arêtes (la forme, la taille, la couleur, …) et des étiquettes (la police, le style, la taille, la couleur, la position, …). Une disponibilité supplémentaire est de pouvoir modifier ces caractéristiques aussi simplement que possible. Cependant, cet atout ne doit en aucun cas devenir prédominant au point d'obtenir un environnement de dessin amélioré et non plus un environnement sur les graphes.

2.2.2. Fonctionnalités

Pour tous les environnements informatiques, une image seule comme représentation graphique d'un graphe n'est pas suffisante. Diverses commandes sont utiles et nécessaires : des outils structurels pour modifier la structure du graphe, des outils graphiques pour modifier l'aspect du graphe, des outils visuels pour une meilleure visualisation et appréhension du graphe, sans oublier les procédures de sauvegarde du graphe sous un format simple à écrire et à lire. Le

format le plus utilisé est un simple fichier texte ou un format de type Postcript[6] pour sauvegarder uniquement la représentation du graphe.

Les fonctionnalités structurelles

Il est intéressant d'effectuer des calculs sur une partie ou sur tout l'ensemble du graphe pour connaître ses propriétés graphiques ou théoriques. Les invariants classiques de la théorie des graphes sont par exemple le nombre chromatique ou savoir si un graphe est hamiltonien ou encore planaire. Des transformations d'ordre structurelles (la modification de la structure des graphes) sont par exemple l'ajout ou la suppression de sommets ou d'arêtes. Elles s'avèrent très utiles pour des modifications du graphe. D'autres outils possibles sont un générateur aléatoire de graphe ou des opérations de construction sur le graphe afin de créer d'autres graphes. Un avantage apprécié est de pouvoir accomplir toutes ses modifications sur l'ensemble ou une partie du graphe, en choisissant directement les candidats potentiels.

Les fonctionnalités graphiques

Les algorithmes de dessin automatique sont utilisés pour placer les sommets et les arêtes en fonction d'un ou de plusieurs critères d'esthétique. Ils sont souvent employés pour la représentation des graphes denses. Ces critères décrits par des attributs sont importants pour produire un "*joli*" dessin. Par exemple, un des critères d'esthétique les plus courants minimise le nombre de croisement d'arêtes dans un dessin de graphes. Souvent, les critères d'esthétique sont choisis suivant une propriété structurelle telle que la planarité, une symétrie, ou une hiérarchie des sommets dans le graphe. Il est quelques fois possible de satisfaire plusieurs critères simultanément.

La persistance

Une fonctionnalité indispensable est la persistance, c'est-à-dire la préservation de la structure des graphes au-delà de l'exécution des applications qui les créent. Cette procédure autorise, pour un faible coût, la communication entre les applications. Evidemment, cette méthode n'est pas la plus élaborée, et il convient de l'améliorer suivant les outils disponibles sur les stations de travail et les environnements de développement.

Un environnement informatique sur les graphes doit être capable de sauvegarder les graphes sous format d'un fichier texte par exemple. Généralement, les éditeurs de graphes offrent seulement comme format externe consistant une liste de sommets et d'arêtes pour préserver le graphe. Ceci n'est concevable que pour des environnements ne représentant pas les graphes sur un écran d'ordinateur. Un éditeur de graphes doit être capable de sauvegarder non seulement la structure du graphe, mais aussi plusieurs autres attributs qui lui sont associés : des

---

[6] Postcript est un langage d'impression utilisé par les imprimantes.

informations graphiques, des relations entre les éléments du graphe, par exemple, ou le statut d'une opération de la session courante.

2.2.3. Mode d'interaction

Une portion significative du code dans une application interactive consiste en une interface utilisateur graphique [Bobrow & al. 86] et [Sutton & Sprague 78]. Les interfaces de manipulation directe, telles que celles utilisées dans les éditeurs de graphes, sont parmi les plus difficiles à implémenter [Williams 83]. Malheureusement, par construction, les divers éditeurs de graphes restreignent leur utilisation à des applications particulières.

Deux modes d'interaction sont envisagés : les langages de commandes, ou la manipulation directe. Chacune possède ses avantages et ses inconvénients. L'animation sur les graphes est plus commode à programmer avec le langage de commandes, alors que l'interaction directe et manuelle du graphe s'effectue dans le cadre de la manipulation directe.

3. Des environnements informatiques sur la théorie des graphes

Nous présentons dans ce paragraphe les principaux environnements informatiques traitant de la théorie des graphes. Nous les avons classés en trois catégories : les bibliothèques d'algorithmes, les visualiseurs de graphes et les éditeurs de graphes. Les environnements informatiques seront analysés suivant les critères énoncés précédemment.

3.1. Les bibliothèques d'algorithmes

On trouve deux sortes de bibliothèque d'algorithmes. La première catégorie est constituée d'une bibliothèque de programmes sur les graphes, comme le système *GraphBase* [Knuth 94]. Les seules actions disponibles sont alors des opérations et le calcul des invariants sur les graphes. Dans la deuxième catégorie se rangent les bibliothèques écrites au-dessus d'un logiciel d'intérêt plus général, comme *Combinatoria* [Skiena 90] et *VEGA* [Pisenski 95] utilisant *Mathematica* [Wolfram 91] comme interface, et Maple-V [Maple-V 91] en ajoutant un nouveau groupe de commande. Généralement peu adapté à la théorie des graphes, ce type de logiciel devient rapidement inefficace pour des graphes de grande taille. Le mode d'interaction associé est un langage de commande.

Combinatorica

Combinatorica [Skiena 90] est une bibliothèque de procédures pour le logiciel *Mathematica*. Cet ensemble de procédures propose des outils pour la théorie de la combinatoire (avec une permutation d'éléments dans un ensemble, une partition d'un ensemble donné, ou la composition d'ensembles et d'autres plus spécifiques pour la théorie des graphes.

Visualisation : La visualisation des graphes se fait par l'intermédiaire de l'interface de *Mathematica*. Les graphes représentés sont de diverses natures, même si la multiplicité des arêtes ou les boucles ne sont pas distinctement visibles lors de la représentation du graphe.

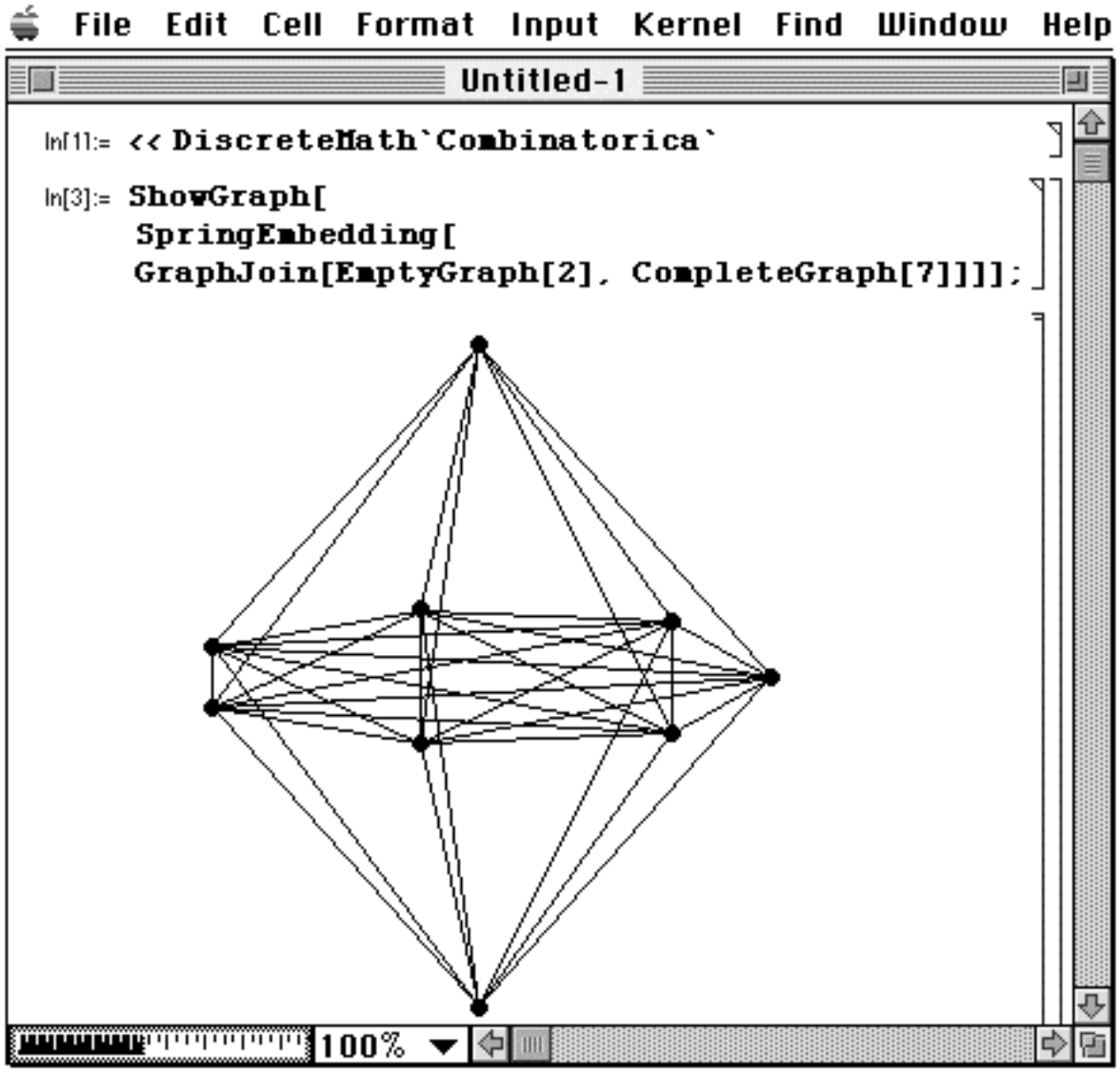


Figure 2.6 : Une fenêtre *Combinatorica* dans l'environnement de *Mathematica* sur Macintosh.

L'interface graphique de *Combinatorica* est restreinte à celle de *Mathematica*. Des outils sont disponibles pour représenter un graphe sous différents angles. Les sommets d'un graphe peuvent être mis sur un cercle ou sur une grille. Les graphes admettent des transformations géométriques simples : la rotation, la dilatation et la translation. Des représentations des différents parcours BFS et DFS[7] du graphe sont permises. Les algorithmes de dessin des arbres existent. Un outil supplémentaire, l'heuristique de Peter Eades [Eades 84] appelée le plongement par ressort, est disponible. *Combinatorica* utilise aussi les fonctionnalités graphiques de *Mathematica*..

Fonctionnalité : Plusieurs types de constructions de graphes sont présents : l'union et l'intersection de graphes, la somme ou la différence de graphes en utilisant les matrices d'adjacence associées à chaque graphe, les produits classiques de graphes[8], et un générateur aléatoire de graphes simples par séquence des degrés. Le calcul des principaux invariants de graphes et les vérifications de propriétés classiques du graphe existent également.

Les graphes sont sauvegardés sous la forme d'un fichier texte lisible par le système SPREMB [Eades & al. 88] et non sous format *Mathematica*. Le principal problème est que cet environnement est utilisable uniquement par *Mathematica* et *SPREMB*.

Mode d'interaction : Le système d'interaction sur les données est le langage de commandes de *Mathematica*, soit le type d'interaction le plus éloigné de la manipulation directe. De plus, aucune manipulation directe sur le graphe n'est donc disponible. Pour toute modification du graphe, de nature graphique ou structurelle, il faut utiliser les outils développés par *Combinatorica* ou les fonctionnalités de *Mathematica*.

Combinatorica a été développé comme une extension de *Mathematica*.

Stanford GraphBase

Le système *Stanford GraphBase* [Knuth 94] propose des outils de calcul pour la théorie de la combinatoire. Des outils plus spécifiques ont été créés pour la théorie des graphes et pour les réseaux. Il possède en outre une base de données qui peut être utilisée pour tester des

[7] La représentation BFS - Breath First Search ou exploration en largeur d'abord - choisit un sommet privilégié et dessine à partir de celui-ci ses voisins successifs par couches. Une représentation DFS - Deapth First Search ou exploration en longueur d'abord - est une partition des sommets par la plus courte distance ou par le plus court chemin d'un sommet donnée aux autres sommets de graphe.

[8] Le produit fibré est étudié dans le chapitre 4. Il est, à notre connaissance, programmé uniquement sur *Cabri-graphs.*

algorithmes combinatoires. Pour la théorie des graphes, cette base contient plusieurs graphes bien particuliers servant de contre-exemples à beaucoup de théorèmes.

Visualisation : Il n'existe pas actuellement d'interface graphique à cet environnement informatique.

Fonctionnalité : Les fonctionnalités de *GraphBase*, pour la théorie des graphes, sont surtout centrées sur deux thèmes : d'une part le calcul de propriétés, d'autre part les transformations et les opérations classiques. L'utilisateur peut ainsi programmer et tester ses propres algorithmes, conforter ou valider ses hypothèses, ou trouver dans la bibliothèque de graphes des contre-exemples à ses conjectures, s'ils sont dans la base.

Mode d'interaction : Les procédures sont écrites uniquement dans le langage *CWEB*[9]. Aucune interface graphique connue n'existe pour l'instant.

Stanford GraphBase est une collection de programmes *CWEB*. Il est développé à l'université de Stanford (Etats-Unis) avec comme responsable D. Knuth. Il est disponible sur les stations de travail.

Maple

Il existe dans le logiciel *Maple* [Maple-V 91] des bibliothèques de procédures utilisant les structures combinatoires et traitant des réseaux et des graphes. Les graphes construits sont simples ou multiples, orientés ou non et ils peuvent avoir les boucles.

Visualisation : Les graphes sont visualisés dans une fenêtre particulière pour X-Maple, l'équivalent de Maple pour les stations de travail, sinon aucune visualisation n'est possible.

Fonctionnalité : Les graphes sont créés de plusieurs manières avec des commandes appropriées. Dans la première méthode, la création de graphes s'effectue simplement avec l'ajout ou la suppression de sommets ou d'arêtes. Avec une autre méthode, des graphes particuliers sont construits directement par l'intermédiaire de plusieurs commandes : des graphes complets, des cycles ou des graphes de Petersen. Un générateur aléatoire de graphes simple est disponible ; il ne demande que le nombre de sommets et d'arêtes ou le nombre de sommets et la probabilité d'une occurrence d'arête.

---

[9] *CWEB* [Knuth 93] est un langage de programmation particulier : une combinaison des langages TeX et C (ou C++), avec quelques pages de règles additionnelles simples.

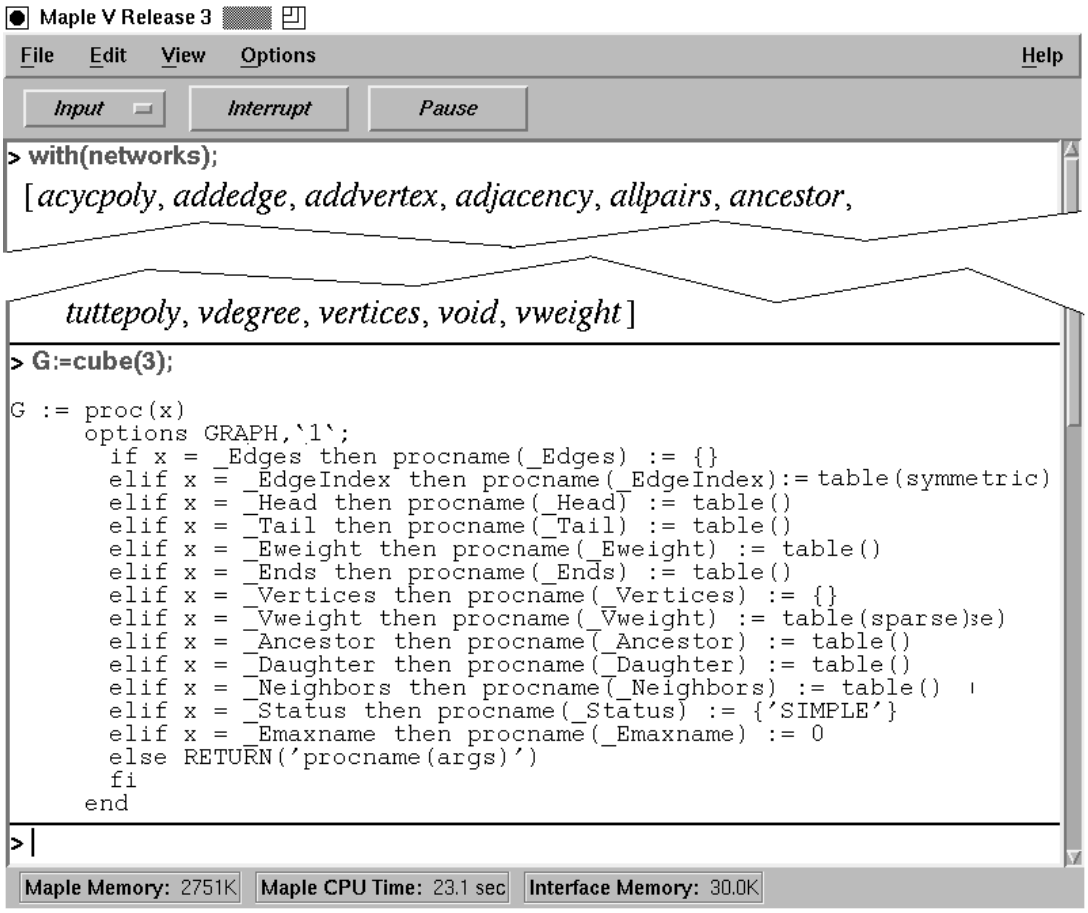


Figure 2.7 : La fenêtre textuelle de *X-Maple*, et la définition du graphe "Cube" dans *X-Maple*.

Les propriétés classiques sur les graphes et les réseaux sont disponibles : les calculs sur les flots, la connexité, les arbres couvrants disjoints ou de poids minimum, les plus courts chemins, le polynôme de Tutte et ses évaluations spéciales, et les polynômes caractéristiques.

Mode d'interaction : Le système de manipulation des données est le langage de commandes de *Maple*. Comme pour *Mathematica* , ce langage de commande est à l'opposé de la manipulation directe.

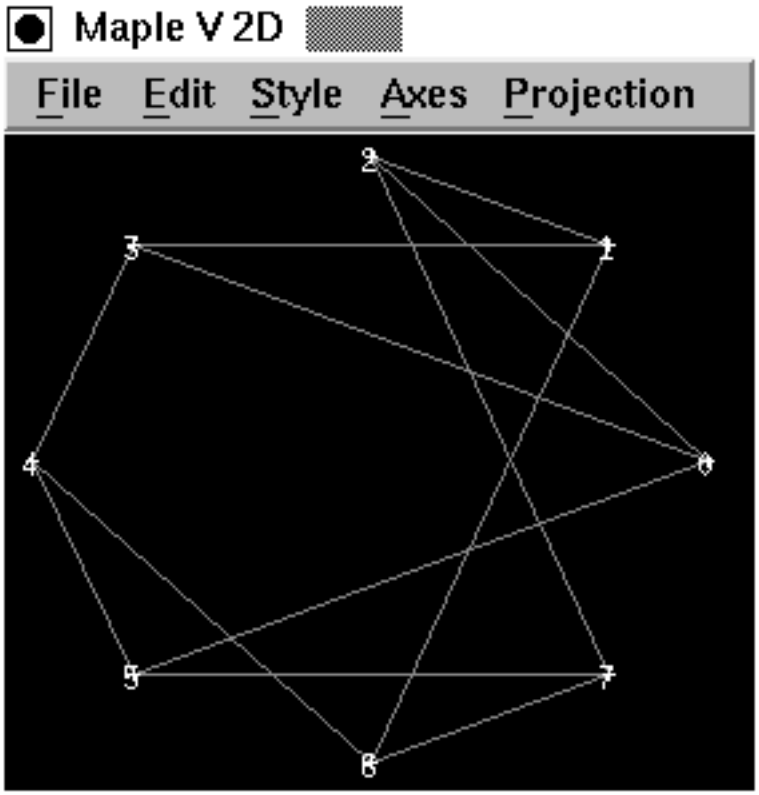


Figure 2.8 : La fenêtre graphique de *X-Maple* du graphe décrit dans la figure 2.7.

Maple est développé à l'université de Waterloo (Ontario, Canada). Il est disponible sur tous les types d'ordinateurs, les stations de travail et les ordinateurs personnels.

Vega

Le système *Vega* [Pisenski 95] est un environnement pour l'étude de structures mathématiques discrètes. Il est composé de plusieurs procédures servant d'intermédiaire entre son noyau fonctionnel et des applications comme *Mathematica* ou *MathSource*. Il possède aussi des procédures fournissant une interface avec d'autres systèmes informatiques simples sur les graphes comme *Nauty*[10] [MacKay 90] et *GAP*[11] [GAP 97].

---

[10] *Nauty* est un ensemble de procédures pour déterminer les isomorphismes et les automorphismes de graphes, avec en option un étiquetage automatique du graphe. *Dreadnaut* est une interface interactive simple pour *Nauty* incluant un générateur simple de graphes.

11 *GAP* - Groups, Algorithms and Programming est un système pour les calculs algébriques discrets, en particulier sur la théorie des groupes.

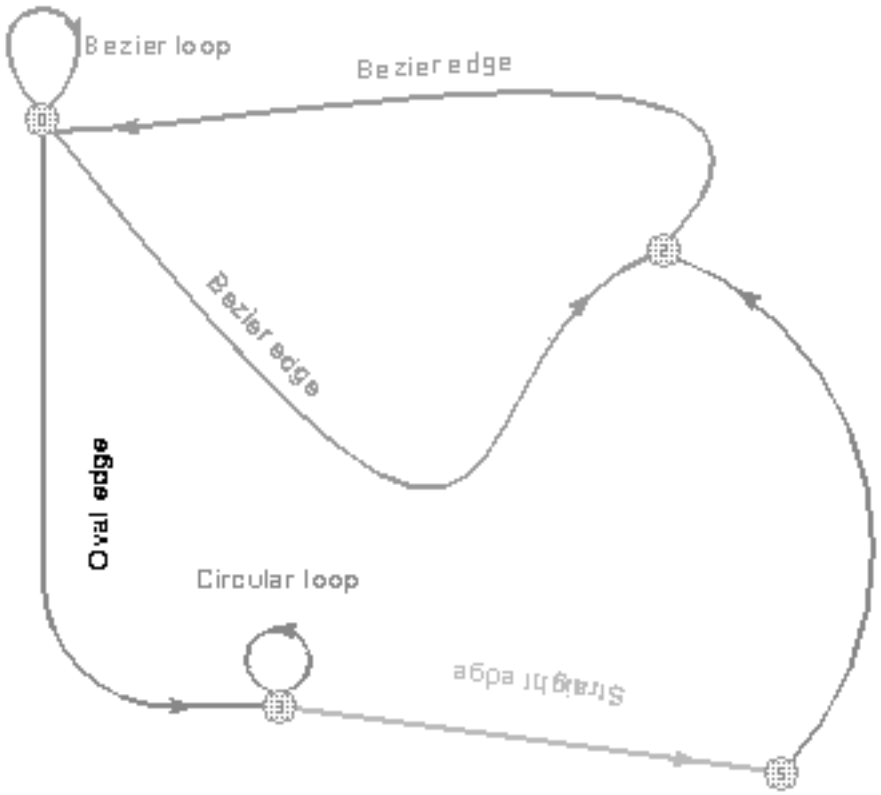


Figure 2.9 : Un graphe dans l'environnement de *Vega* avec comme outil d'interface, l'environnement de *Nauty*.

Visualisation : Par l'intermédiaire d'un visualiseur de graphes comme *Dreadnaut* ou *Mathematica*, *Vega* a la possibilité de dessiner des graphes en deux ou trois dimensions. Il accepte les graphes orientés avec des étiquettes et une coloration des arêtes et des sommets. L'étiquette des arêtes peut être un entier. Ces outils de dessin sont principalement ceux de *Mathematica*.
Plusieurs options sont proposées pour visualiser des graphes. *Vega* crée des fichiers, au format Postscript, pouvant être utilisés par le système *Postgraf* de Bor Plestenjak. Ces fichiers sont convertis sous un format image et ils sont disponibles sur le site internet "The Vega Graph Gallery". Il existe plusieurs visualiseurs capables de représenter les fichiers de données sur les graphes.

Fonctionnalité : Des programmes externes sont utilisés pour effectuer plusieurs tests sur les graphes : tester la planarité d'un graphe, trouver un cycle hamiltonien dans un graphe, ou factoriser un graphe régulier. Des algorithmes de dessin automatique de graphes existent également. *Nauty* est utilisé pour trouver le groupe d'automorphismes d'un graphe. *Vega* possède un générateur aléatoire de graphes. Il génère des graphes non forcément connexes, des graphes planaires ou des graphes planaires maximaux. Il vérifie des propriétés classiques, comme l'hamiltonicité ou la coloration, et il dispose d'outils relatifs aux réseaux.

Les fichiers de sauvegarde de *Vega* sont compatibles avec l'environnement de *Combinatorica*. Cette sauvegarde est standard ; on associe à chaque sommet ses coordonnées et son voisinage.

Mode d'interaction : Elle correspond à celle de *Combinatorica*, et donc de *Mathematica*.

Vega est développé à l'Institut de Mathématiques, de Physique et de Mécanique (IMPM) de Ljubljana (Slovénie), avec comme responsable Skiena. Il est disponible sur les stations de travail.

3.2. Les visualiseurs de graphes

Les systèmes comme *da Vinci* [Frölich & Werner 94], *Swan* [Yang & al. 96], ou encore *VCG tool* [Sander 94] sont essentiellement centrés sur la visualisation et sur l'édition de graphes. Ils ne permettent aucune modification structurelle sur les graphes, comme l'ajout ou la suppression de sommets, d'arêtes, des étiquettes associées, etc.

Principalement orientée vers les industriels, la fonction la plus importante est de présenter à l'utilisateur un *"beau"* dessin de graphe. Pour cela, des algorithmes de dessin automatique sont utilisés. Ils respectent certaines contraintes et des critères d'esthétique. Les logiciels mettent à la disposition de l'utilisateur de nombreux outils pour dessiner automatiquement les graphes et les visualiser.

Très efficaces dans leur domaine, ces systèmes se concentrent sur le seul problème de la représentation correcte et lisible des graphes. Ils n'apportent aucun outil pour la manipulation des éléments du graphe par interaction sur les objets du domaine, par exemple la manipulation directe pour le déplacement des objets.

da Vinci

da Vinci [Frölich & Werner 94] est un système de visualisation des graphes orientés et multiples sans altération directe de la structure des graphes. Chaque modification doit être faite par une application associée externe. Une interface de communication entre les applications est utilisée comme outil de communication avec cette application qui contrôle la structure des graphes. Par contre, la visualisation des graphes s'effectue dans *da Vinci*. *da Vinci* est donc un outil de visualisation générique des graphes, et un outil de communication avec les applications. Il possède une aide en ligne et via le réseau internet. Il dispose de plusieurs algorithmes de dessin de graphes.

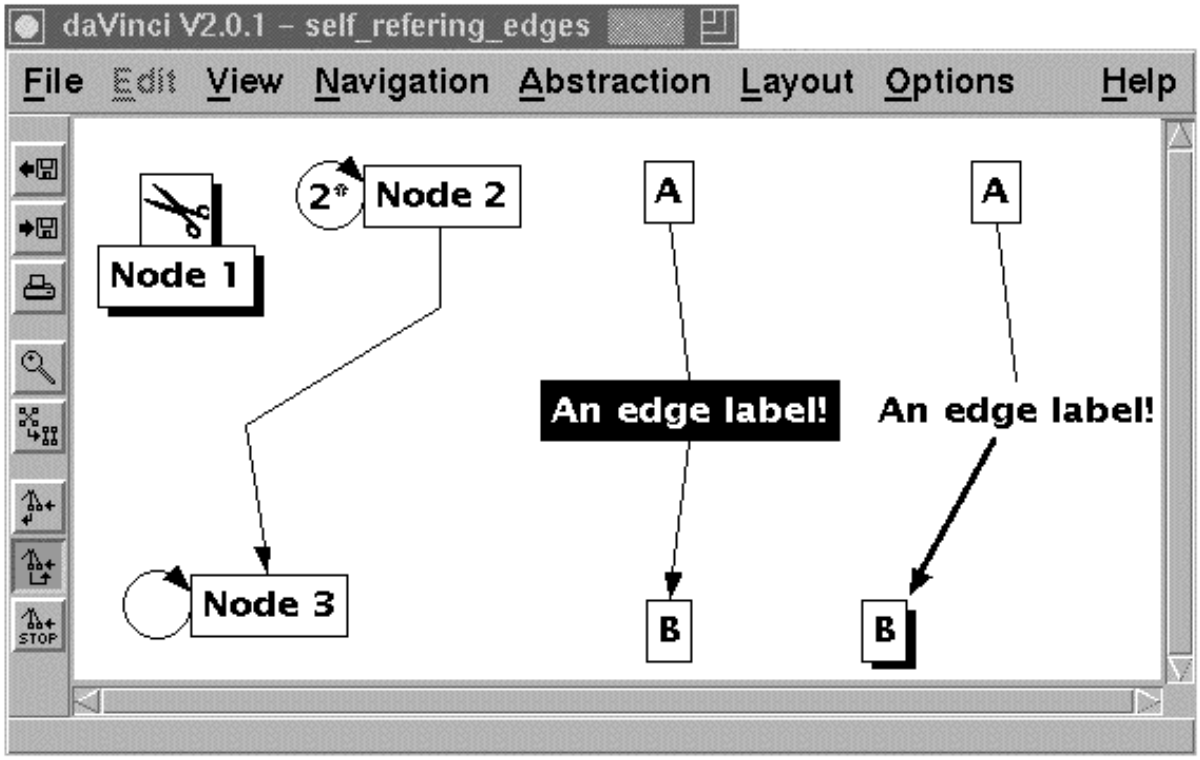


Figure 2.10 : Un arbre dans une fenêtre de *da Vinci*.

Visualisation : Les graphes sont dessinés sur une grille. Les arêtes autorisées sont des boucles ou des segments brisés, permettant les arêtes multiples. Les arêtes sont dessinées initialement par un segment. Les points de contrôle associés aux segments brisés n'apparaissent que lors du déplacement des sommets. Le changement des attributs pour les sommets et les arêtes s'effectue après une sélection à la souris. Les étiquettes associées aux sommets sont placées à l'intérieur de la représentation des sommets. Les étiquettes des arêtes ne sont pas véritablement une étiquette associée à l'arête. C'est l'étiquette d'un sommet n'ayant aucune représentation visuelle. Ceci est parfaitement visible dans la *figure 2.10* où l'arête, entre les sommets étiquetés respectivement A et B, a été sélectionnée. En fait, entre un sommet virtuel et l'étiquette de l'arête, un lien a été mis en évidence en l'épaississant.

Les arêtes, les arcs et les flèches sur les arcs sont fixes. Pour modifier la représentation des graphes, plusieurs algorithmes sont proposés avec des contraintes sur les objets ; par exemple, les sommets sont immobiles dans le plan. D'autres contraintes sont citées dans la partie **fonctionnalité**. *da Vinci* utilise des abstractions graphiques dans la visualisation pour rendre temporairement invisible une partie non intéressante du graphe (le menu "Abstraction"). Il permet de réduire ou d'agrandir l'échelle de représentation. Un outil original est le système de vues multiples : la possibilité d'avoir plusieurs fenêtres représentant un même graphe. Evidemment, toutes les modifications

des attributs ou le déplacement des objets dans une fenêtre sont reproduits dans les autres.

Fonctionnalité : Les principales fonctionnalités sont d'ordre graphique. *da Vinci* est responsable exclusivement du contrôle et de la manipulation des graphes. Il n'effectue aucune modification sur la structure des graphes. Des algorithmes de dessin automatique de graphes orientés ont été développés dans *da Vinci* , suivant différents critères : en utilisant la méthode de Sugitama, ou encore avec un nombre minimum de croisement d'arêtes ou un nombre minimum de coudes pour les segments. Des algorithmes spécifiques aux arbres ont été programmés. Une bibliothèque de dessin de graphes avec contraintes est présente dans le menu "Layout". Les trois opérations courantes sont représentées par les trois derniers boutons, à gauche de la fenêtre, et elles correspondent à : "Preserve Node Order" ou préserver l'ordre des sommets, "Modify Node Order" ou modifier l'ordre des sommets, et "Fixed Neighbour Nodes" pour fixer les sommets du voisinage.

Les opérations de sauvegarde couramment utilisées sont présentes dans le menu "Options". Il existe deux formats de sauvegarde des informations. Les fichiers de suffixe "*.daVinci* " utilise le format appelé représentation des termes et la sauvegarde de la structure des graphes s'effectue par un fichier texte. Les graphes peuvent donc être créés arbitrairement par un éditeur de texte. Les fichiers de suffixe "*.status"* conservent la structure des graphes et également des informations sur la représentation du graphe (les attributs et les positions des sommets et des arêtes avec les points de contrôle).

da Vinci communique avec les autres applications en utilisant une API, Application Programmer Interface ou l'Interface de Programmation d'Application. Cette méthode implique dans ce cas la responsabilité des transformations de la structure des graphes par une application externe.

Mode d'interaction : *da Vinci* autorise la sélection d'un ensemble de sommets (directement ou dans le menu "Navigation") ou la recherche d'un sommet particulier. Les seuls objets susceptibles d'être déplacés sont les sommets et les arêtes. Les étiquettes des sommets sont fixes et elles sont à l'intérieur de la représentation du sommet. Le symbole au-dessus d'un sommet est une paire de ciseaux (voir la figure 2.9), précisant que le sous-graphe, dont la racine est ce sommet, est invisible à l'écran, le sous-graphe étant l'ensemble des sommets descendant uniquement du sommet racine.

Le déplacement des sommets est en permanence associé à une contrainte (voir le paragraphe sur la ***Visualisation***). Il se déplace sur les nœuds de la grille sous-jacente. Il en résulte que les arêtes se déforment et peuvent alors être aussi *manipulées*. Il faut d'abord sélectionner une arête pour faire apparaître les points de contrôle du segment

brisé la représentant. Les points de contrôle sont déplacés uniquement sur des nœuds de la grille se trouvant sur une même horizontale. Si aucun point de contrôle n'existe à un moment donné pour le segment, l'arête associée ne peut pas être déplacée. Le déplacement vertical des sommets crée ou supprime des points de contrôle pour les arêtes incidentes.

Le mode d'interaction n'est pas celui de la manipulation directe, et il est incomplet, puisque seuls les sommets et les points de contrôle d'une arête peuvent être déplacés directement.

da Vinci est développé à l'université de Bremen (Allemagne) avec comme responsables Michael Fröhlich et Mattias Werner. Il est disponible sur les stations de travail.

Swan

Swan [Yang & al. 96] est un système de visualisation de structure de données : les tableaux, les listes, les arbres, les graphes.

Il est composé d'une bibliothèque de procédures et d'une interface facile d'utilisation. Il est utilisé comme une présentation intermédiaire pour les instructions dans les structures de données et les cours d'algorithmique. Dans la programmation graphique, il visualise les structures de données par une figure et par la structure interne de sauvegarde. Les étudiants s'en servent pour leurs cours d'algorithmes, et aussi comme une plate-forme pour expérimenter des algorithmes de dessins de graphes.

Swan est utilisé par deux types d'individu : le concepteur et l'utilisateur. Ils disposent d'outils, grâce à une bibliothèque graphique spécifique, pour visualiser les structures de données utilisées dans son programme. L'utilisateur, qui peut être aussi le programmeur, visualise les structures de données du programme. Une animation des algorithmes sur les structures de données visualisées contribue à comprendre les algorithmes implémentés.

Visualisation : Plusieurs algorithmes de dessin automatique sont programmés. Ils sont spécialisés pour les tableaux, les listes et les arbres et les graphes. Le logiciel *Swan* ne permet pas de créer des graphes, mais il autorise la modification des attributs du graphe, des sommets ou des arêtes (les composantes graphiques et l'étiquetage par l'intermédiaire du bouton "Attribut").

Il existe huit boutons sur la fenêtre principale, en haut et à droite. Les quatre premiers boutons sont utilisés pour déplacer la région de visualisation dans les quatre directions indiquées. Les deux boutons suivants sont utilisés pour le zoom. Les deux derniers boutons "Swap" et "Move" sont utilisés pour échanger la position ou déplacer les sommets dans un graphe.

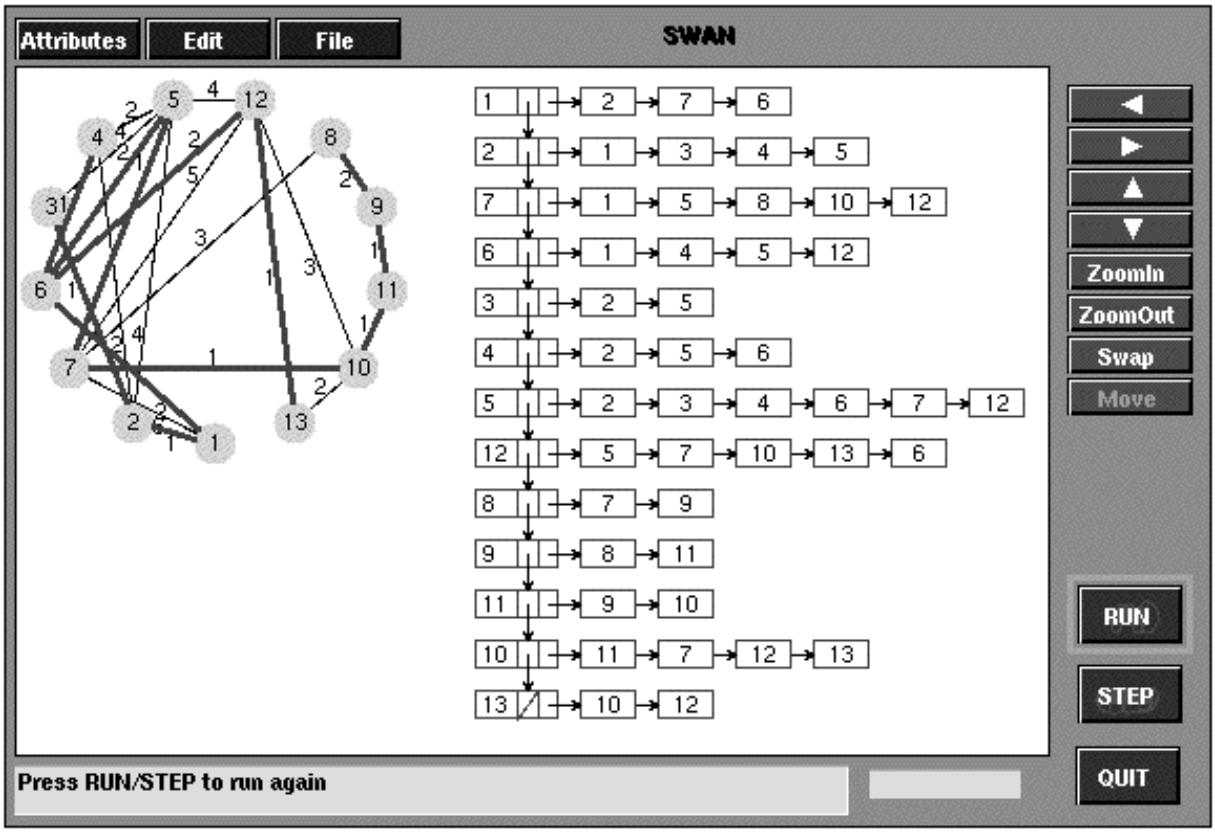


Figure 2.11 : Arbre de poids minimum - fenêtre du logiciel Swan.

Fonctionnalité : La structure du graphe est modifiable en choisissant les fonctions d'éditions de graphes dans le menu "Edit" : des sommets et des arêtes peuvent être ajoutés ou supprimés. Les commandes sont contrôlées par le concepteur qui choisit les fonctions disponibles. Les trois boutons restant sur la fenêtre principale, en bas et à droite, sont utilisés pour contrôler le processus d'un programme annoté.

Seul le dessin de graphes est sauvegardé dans un fichier, la structure de données associée au dessin étant liée au programme.

Mode d'interaction : Il permet de comprendre les algorithmes utilisant des structures de données et de vérifier si les algorithmes fonctionnent correctement.

Swan est un logiciel développé au département informatique de l'université de technologie de Virginie (Virginie, Etats-Unis). Il est écrit dans les langages C et C++, et il est disponible sur les stations de travail.

VCG tool

VCG tool [Sander 94] est un système de visualisation de très grands graphes orientés sans altération directe de la structure des graphes. Successeur d'*Edge tool* [Newberry 93], il combine les avantages de ce dernier avec une rapidité d'exécution et de nouveaux concepts pour

la visualisation des graphes, comme la vue en œil de poisson[12] (voir la figure 2.13). Plusieurs heuristiques de dessins de graphes ont été programmées, en particulier des améliorations sur l'algorithme de dessin dit de Manhattan.

Visualisation : Plusieurs formes sont disponibles pour les sommets et les arêtes (un segment simple, un segment brisé ou une courbe) avec des contraintes. L'étiquetage des sommets est à l'intérieur de la représentation du sommet. Les graphes sont uniquement des graphes orientés et simples.

VCG tool permet de visualiser des graphes à partir d'un fichier texte dans lequel des spécifications sur la représentation spatiale du graphe ont été données. Si la position des sommets n'est pas fixée, le logiciel dessine le graphe en utilisant plusieurs heuristiques. *VCG tool* cache ou montre des régions spécifiques du graphe, et il autorise les animations sur le dessin de graphes.

Fonctionnalité : Il contient plusieurs heuristiques de dessin automatique de graphes. Ces algorithmes minimisent le nombre de croisement d'arêtes. Ils placent les sommets dans une hiérarchie de niveaux, ou ils minimisent la longueur des arêtes ou encore ils centrent les sommets. Ces algorithmes admettent plusieurs types de version suivant les classes de graphes bien spécifiques (par exemple les arbres) ou ils sont programmés avec des options de contraintes multiples.

Il produit différents formats de données de sortie comme le Postscript, par exemple. Le langage GDL - "Graph Description Language" ou Langage de Description des Graphes - a été créé pour décrire un graphe et sa représentation graphique. Une description complète des sommets et des arêtes est sauvegardée dans un fichier par des mots réservés au langage. Le noyau de ce langage est compatible avec le langage de spécification des entrées d'*Edge tool*. Un échange de fichiers entre les deux environnements est possible. Les images du graphe (format Bitmap) ou les fichiers (format Postscript) sont exportables.

[12] La vue de poisson place la figure sur une sphère. Ce procédé permet de mettre en évidence des éléments du graphes.

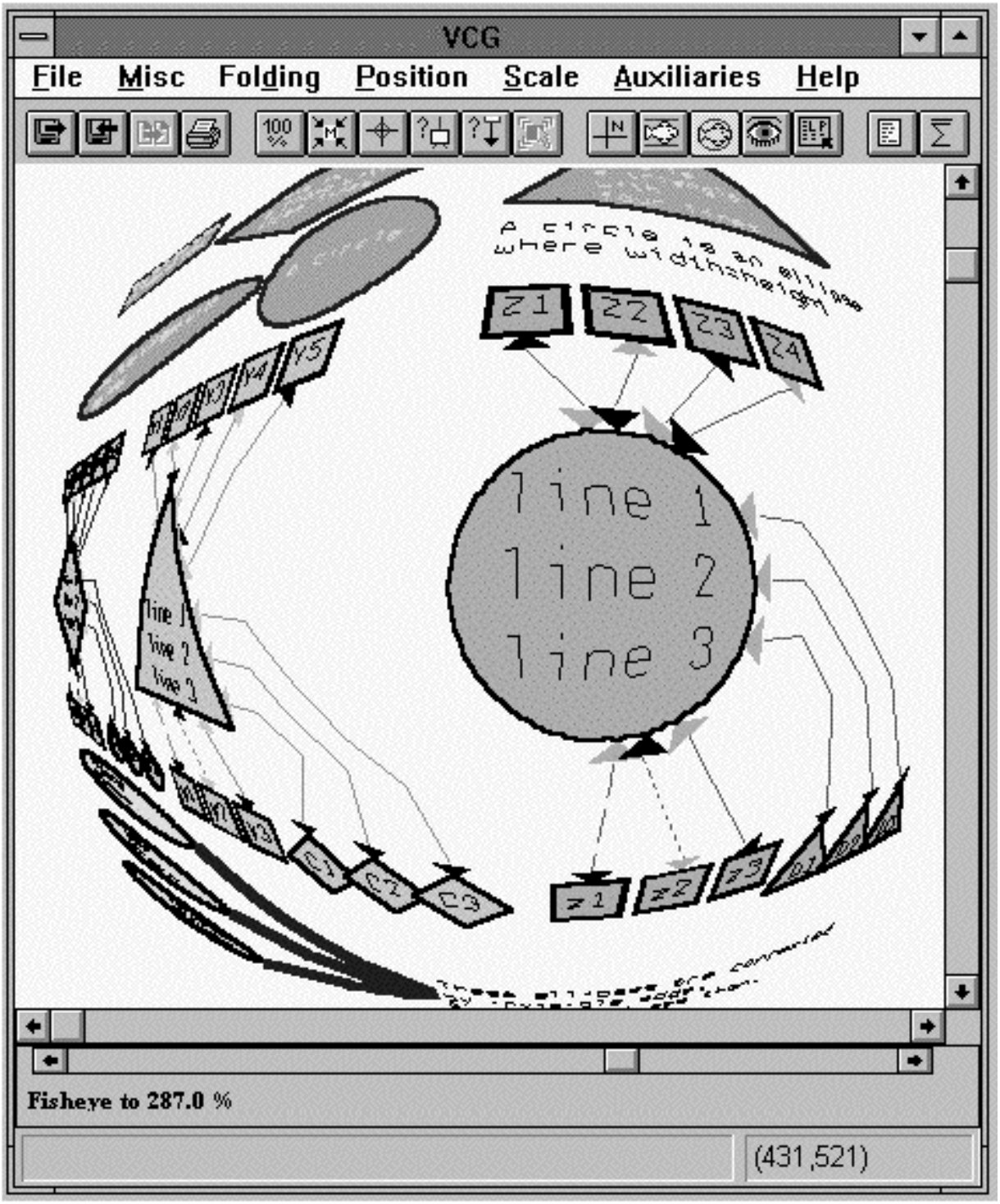


Figure 2.12 : La fenêtre de *VCG tool* pour MS-DOS avec une vue en œil de "poisson".

Mode d'interaction : Toutes les commandes s'opèrent sur les arêtes, sur les sommets, sur un ensemble de sommets. Elles s'effectuent par l'intermédiaire d'un menu général ou d'une boîte de dialogue. Aucune commande directe sur les objets n'est disponible, sauf la sélection d'un sommet. Le mode d'interaction se réalise par l'entremise des menus déroulants et de l'utilisation de quelques boîtes de dialogue pour préciser certaines commandes.

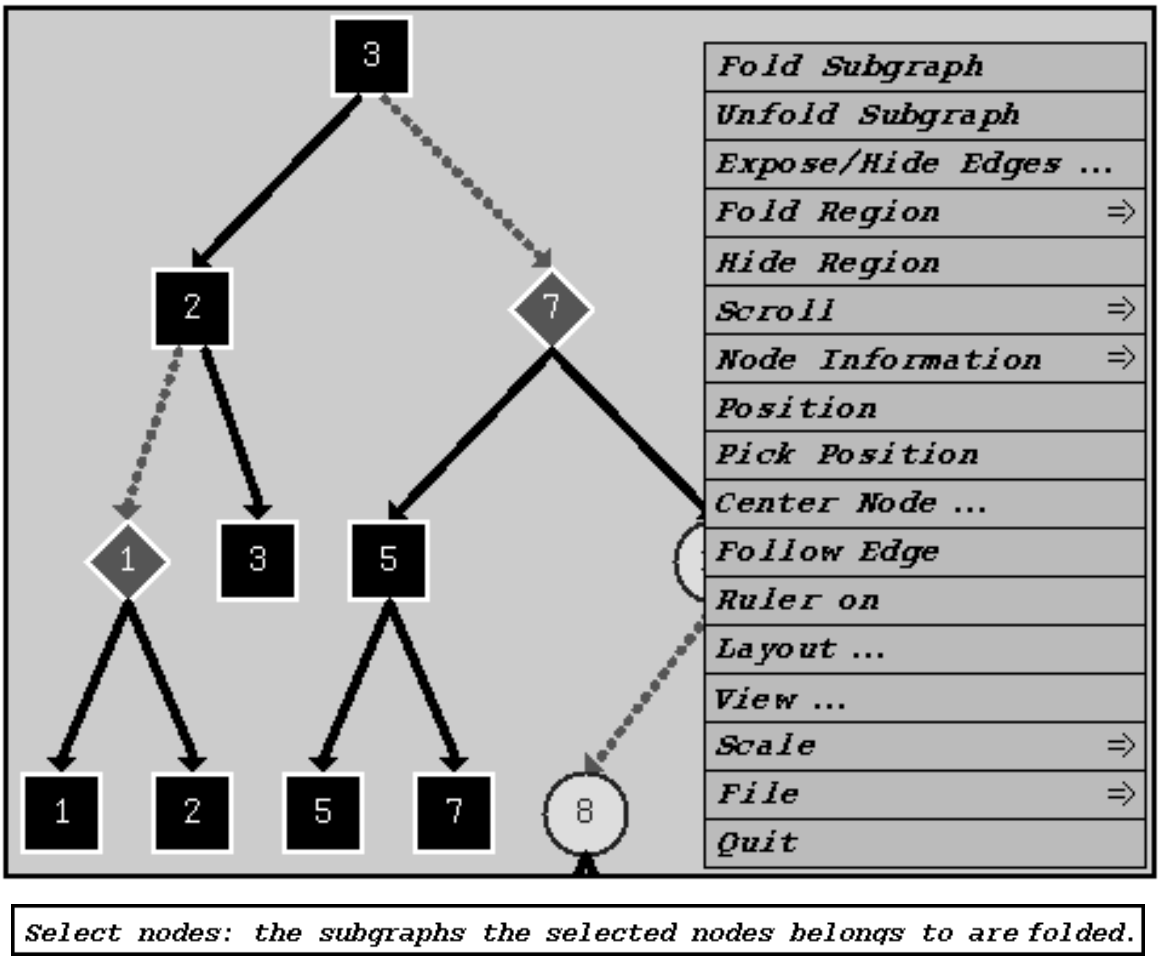


Figure 2.13 : Les menus de *VCG tool* sur une station de travail.

VCG tool est développé à l'université de Saare, Saarbrücken (Allemagne) avec comme responsable Georg Sander. Il est disponible sur les stations de travail et sur les compatibles PC.

3.3. Les éditeurs de graphes

Les systèmes sont par exemple *Cabri-graphes* (voir les travaux de [Habib & Laborde 83], [Benzaken 86], [Baudon 90], [Carbonneaux & al 95], et [Carbonneaux 97]), *GraphEd* [Himsolt 94], *Graph Layout Toolkit* [Graph Layout Toolkit 96], *Groups&Graphs* [Kocay 90], *Link* [Berry 96], ou *Metanet* [Gomez & Goursat 90].

Ils se décomposent en deux parties. La première partie s'applique à la visualisation des graphes avec son ensemble de fonctionnalités. La seconde partie traite de la modification des graphes avec un module pour les opérations internes à la structure des graphes et un autre pour leurs représentations à l'écran.

Cabri-graphes

Nous avons cité dans le paragraphe sur les motivations, le document de constitution du groupe Cabri. Ce groupe regroupait plusieurs équipes de recherche en combinatoire[13]. Plusieurs travaux,en particulier des thèses, y ont été consacrés, et trois environnements ont été élaborés. *Kabri* a été réalisé à l'Ecole des Mines de Saint-Etienne, dans l'équipe de Michel Habib (voir [Tallot 85] et [Dao & al., 86]). L'étude a été poursuivie au CNET. *Unicorn* [Fourneau 86] a été réalisé à l'université d'Orsay, au Laboratoire de Recherche en Informatique. *Cabri-graphes*, a été conçu et élaboré au Laboratoire de Structures Discrètes et de Didactique. Ces trois environnements ont été similaires dans leur conception, et proches dans leur réalisation.

Dans cette section, nous détaillons uniquement *Cabri-graphes*. Plusieurs prototypes ont été réalisés (voir [Benzaken 86], [Laborde 87], et [Baudon 90]). Nous avons poursuivi l'effort de développement en abordant différents types de problèmes : la manipulation de graphes orientés et multiples par exemple.

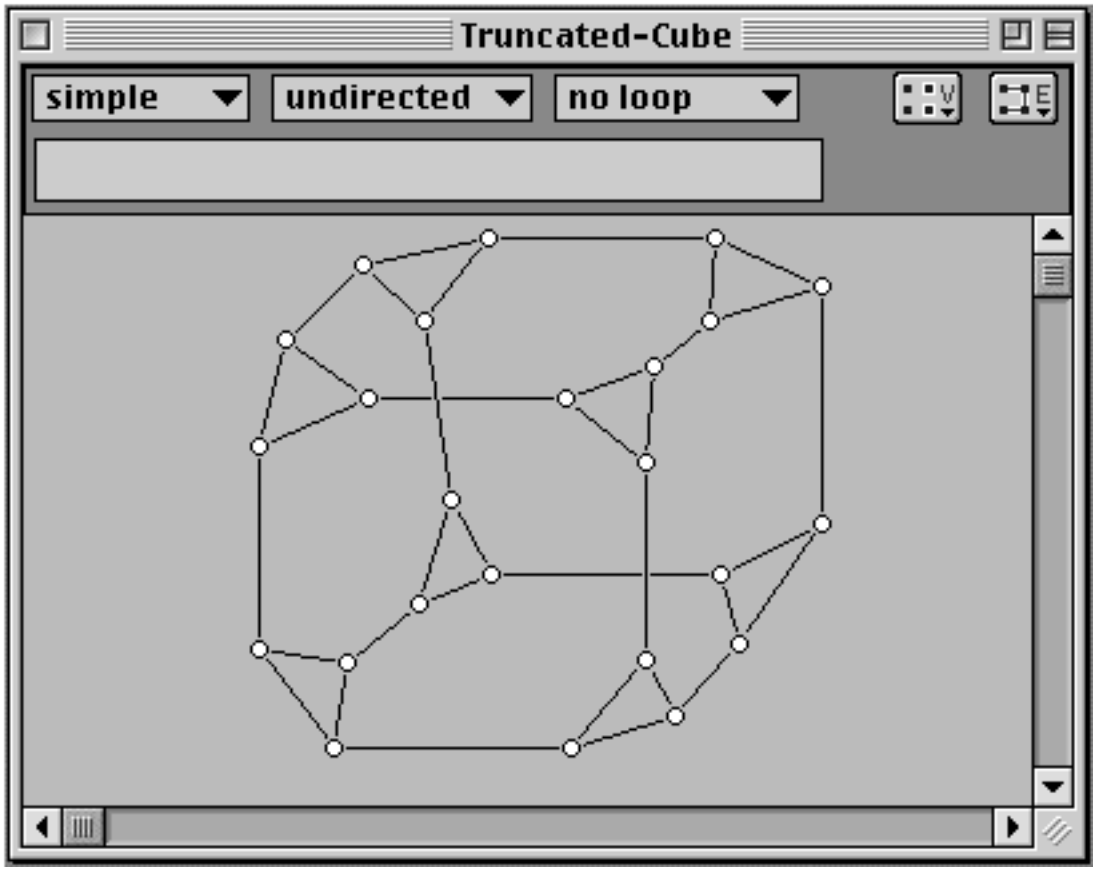


Figure 2.14 : Un graphe dans l'environnement de *Cabri-graphes.*

[13] L'idée de *CABRI* (CAhier de BRouillon Informatique) s'est élaborée au CIRM à Marseille en octobre 82 au cours d'une réunion du PRC "Ensembles ordonnés" par des combinatoriciens (Grenoble, Saint-Etienne, Paris-Orsay, Le Mans, Montpellier). Le groupe *Cabri* est créé lors de la réunion du PRC Mathématiques et Informatique en 1983 [Habib & Laborde 83].

Cabri-graphes est un éditeur de graphes. Les graphes dessinés sont simples ou multiples, orientés ou non, avec ou sans boucle. Les sommets et les arêtes peuvent être étiquetés. Les graphes sont créés directement, et tous les objets sont manipulables.

Visualisation : Les paramètres de l'aspect graphique des sommets et des arêtes sont la taille, la forme et la couleur. Le dessin d'arêtes multiples entre sommets est supporté ainsi que les boucles. L'utilisateur peut sélectionner un ensemble de sommets ou d'arêtes et effectuer des modifications sur l'apparence de cet ensemble. Une arête est représentée par un segment, un segment brisé ou une courbe de Bézier cubique.

Pour la création des sommets un simple clic à l'écran suffit. La création des arêtes provoque celle de ses extrémités si elles n'existent pas (pour plus de détails voir le paragraphe *4.3 Manipuler les objets*).

Fonctionnalité : L'utilisateur choisit entre le dessin manuel (création et manipulation des objets à l'aide de la souris) et le dessin automatique de graphes (visualisation de propriétés du graphe : cycle hamiltonien ou dessin d'un graphe planaire).

Il est possible de vérifier principaux invariants d'un graphe, d'effectuer des opérations sur les graphes (les produits de graphes, le graphes aux arêtes, le plongement de graphes, …) ou des transformations graphiques (les mirroirs ou la rotation sur un ensemble de sommets) et structurelles (le complémentaire, passer d'un graphe multiple à un graphe simple ou d'un graphe orienté à un graphe non orienté, …).

L'entrée minimum de l'éditeur est la structure du graphe : l'ensemble des sommets et le voisinage des sommets pour un graphe simple, l'ensemble des arêtes multiples et des boucles si nécessaires. Les autres informations sont optionnelles : par exemple, la taille, la couleur, ou la représentation des objets.

Le service couper-copier-coller est présent avec plusieurs option. Le service Annuler/Répéter existe, et il informe l'utilisateur du type d'annulation par un message explicite.

Mode d'interaction : Le mode d'interaction est la manipulation directe. Tous les objets du graphe sont manipulables : le sommet, l'arête, la flèche associée à un arc, l'étiquette associée à un sommet ou à une arête, la sélection courante de sommets ou d'arêtes, et le graphe.

La gestion des ambiguïtés opère déjà par un ordre de préférence sur les objets : sommet, arête, flèche étiquette. Par contre, une gestion plus précise est effectuée lors d'un clic à proximité d'un ensemble d'arêtes. Un menu déroulant s'affiche, offrant le choix à l'utilisateur de sélectionner une arête.

Cabri-graphes est développé à l'université de J. Fourier (Grenoble) au laboratoire Leibniz, avec pour responsable J-M. Laborde. Il est disponible sur Macintosh.

GraphEd

GraphEd [Himsolt 94] est éditeur de graphes centré sur la visualisation et l'animation de graphes. Un des nouveaux aspects de *GraphEd* est son support pour la manipulation interactive de grammaires de graphes [Ehrig & al. 87].

Visualisation : Les arêtes sont attachées aux sommets par le coin ou le centre de la représentation du sommet. Le dessin d'arêtes multiples entre sommets est supporté ainsi que les boucles. L'utilisateur peut sélectionner un ensemble de sommets et effectuer des modifications sur l'apparence de cet ensemble. Il peut définir un type de sommet et choisir son apparence en spécifiant l'icône associée au sommet. L'apparence des arêtes est spécifiée de la même manière.

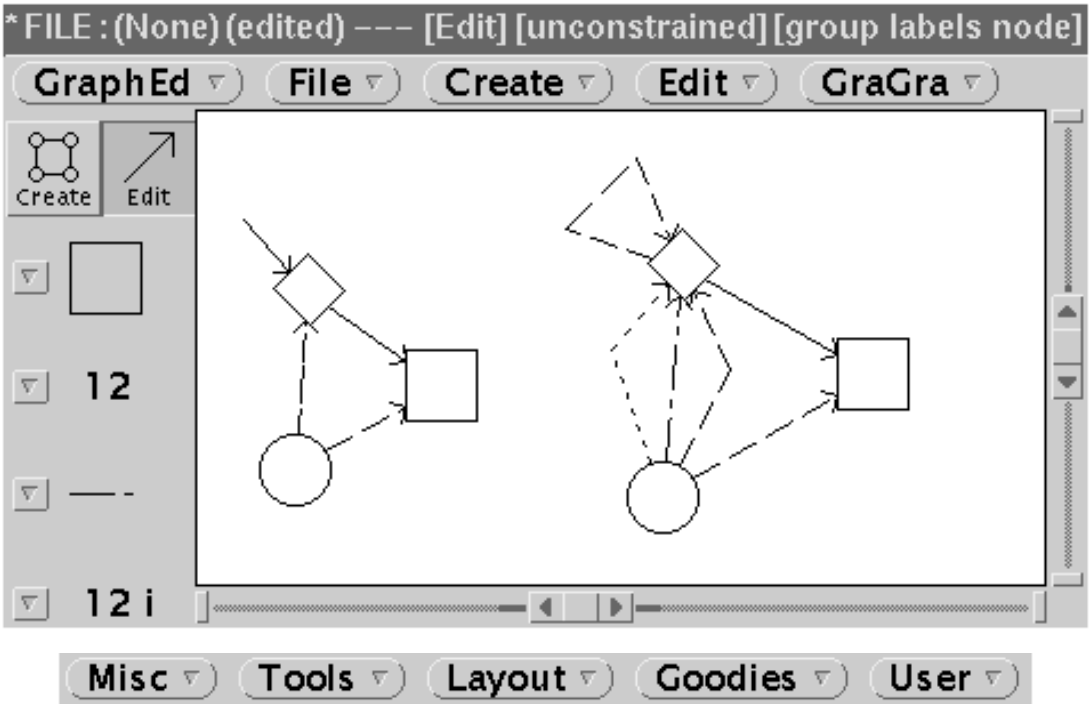


Figure 2.15 : Deux représentations d'un même graphe multiple dans l'environnement de *GraphEd*.

Pour la création des sommets un simple clic sur l'écran suffit. Pour la création des arêtes, il faut commencer à partir d'un sommet par un clic, ensuite tant qu'un clic ne s'effectue pas sur un sommet, un nouveau coude est ajouté au segment représentant l'arête. Si les deux sommets sont identiques, alors une boucle a été créée.

Fonctionnalité : L'utilisateur choisit entre le dessin manuel et le dessin automatique de graphes. Des algorithmes de dessin de Sugiyama [Sugiyama 81], sur les arbres et sur les graphes planaires non orientés sont disponibles.

L'entrée minimum de l'éditeur est une liste d'adjacence donnant la position des sommets. D'autres informations optionnelles spécifiées dans des champs d'entrée de données incluent des informations supplémentaires sur les sommets et les arêtes (la taille, la couleur, la représentation, ...), sur l'apparence des sommets et des arêtes types, et sur la position et la position de la fenêtre d'édition.

Le service couper-copier-coller est présent, mais il est restreint parce que la copie d'un graphe est automatiquement visualisée dans la fenêtre au-dessus du graphe sans intervention de l'utilisateur. Le service Annuler/Répéter n'existe pas. *GraphEd* peut être utilisé comme une bibliothèque par des applications.

Mode d'interaction : Le mode d'interaction est de type modal, puisque l'utilisateur choisit un mode dans lequel il va agir : ***Create*** pour modifier la structure du graphe (ajout ou suppression de sommets ou d'arêtes), ***Edit*** pour déplacer les objets du graphe et ***Text*** pour toutes les procédures liées aux étiquettes.

La création interactive d'une arête multiple ou d'une boucle est possible, mais sa représentation peut ne pas être initialement visible et compréhensible. La *figure 2.15* représente le même graphe. Sur la droite, nous avons le graphe créé initialement, et sur la gauche le graphe dont les arêtes multiples ont été déformées.

La gestion des ambiguïtés dans la sélection des objets est limitée au parcours d'une liste d'objets. La position du clic de la souris détermine la liste des objets. Elle est modulable avec des touches modificatrices pour ne présenter à l'utilisateur que les sommets ou les arêtes, sinon tous les objets sont proposés. Pour chaque objet, l'avis de l'utilisateur est demandé. Si la réponse est négative, alors la liste propose l'objet suivant ; sinon l'objet est sélectionné et l'utilisateur peut le modifier.

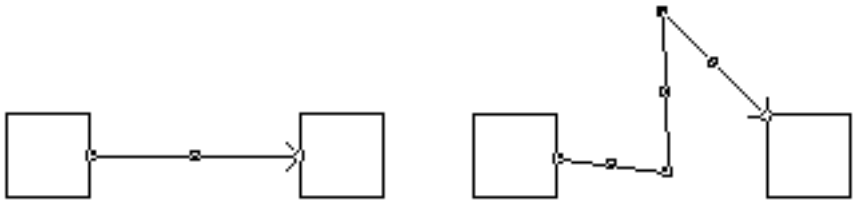


Figure 2.16 : Les points de contrôle d'une arête dans *GraphEd*.

Les objets manipulables de *GraphEd* sont les sommets et la sélection courante de sommets. Les arêtes sont aussi manipulables, mais uniquement par l'intermédiaire de points de contrôle qui sont utilisés pour sa représentation. Le concept d'ensemble d'arêtes semble inexistant. Les arêtes sont modifiables à partir des points de contrôle du

segment les représentant. Soient trois points de contrôle placés dans l'ordre suivant *p'*, *p* et *p"*. Déplacer le point *p* crée directement par dichotomie un premier point de contrôle entre *p'* et *p*, et un second entre *p* et *p"* (voir la figure 2.16).

GraphEd (dont le successeur est *GraphLet* [Himsolt 96]) est développé à l'université de Passau (Allemagne) avec pour responsable F. Brandebourg. Il est disponible sur les stations de travail.

Graph Layout Toolkit

Graph Layout Toolkit [Graph Layout Toolkit 96] est un environnement informatique commercial dont nous avons testé une version de démonstration.

Visualisation : Les graphes représentés sont orientés ou non, simples ou multiples, avec des boucles. Il permet de créer des graphes, d'ajouter des sommets et des arêtes entre des paires de sommets déjà existants, ou de détruire des sous-graphes. La forme des sommets est un rectangle, dont la hauteur et la largeur sont des paramètres, avec une représentation à choisir parmi le rectangle, le losange et le cercle. Les arêtes sont représentées par des segments brisés.

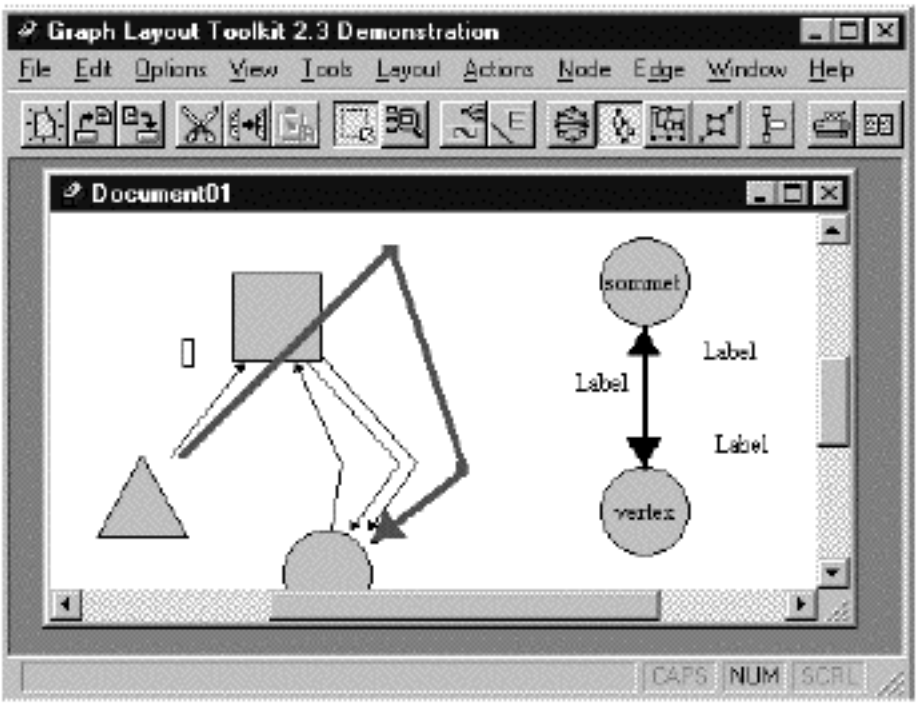


Figure 2.17 : Un graphe multiple dans *Graph Layout Toolkit*.

La création des sommets s'effectue par un simple clic sur l'écran. Pour la création des arêtes, il suffit de cliquer sur un sommet, ensuite de laisser le bouton de la souris

enfoncé, et finalement de le relâcher sur un autre sommet. L'arête est représentée uniquement par un segment lors de la création.

Graph Layout Toolkit positionne les étiquettes associées aux arêtes au point milieu du segment représentant l'arête. Les étiquettes des arêtes évitent les chevauchements avec les sommets, les arêtes, et les autres étiquettes.

Des fonctionnalités spécifiques pour des graphes denses sont proposées, par exemple dissimuler des sous-graphes suivant une représentation hiérarchique du graphe avec comme racine le sommet choisi. Le nombre de niveaux à cacher est sélectionné. Le procédé possède un inconvénient au niveau de la visualisation du nouveau graphe. Contrairement à *da Vinci* représentant un sommet contracté par une paire de ciseaux au-dessus du sommet, *Gaph Layout Toolkit* ne fait aucune différence graphique entre un sommet simple du graphe et un sommet contracté.

Fonctionnalité : Il existe plusieurs styles de représentation, par exemple le dessin hiérarchique, des options de représentation avec un parcours du graphe, et une récupération des coordonnées de chaque sommet. Le graphe peut aussi être dessiné avec un programme de l'utilisateur. Le tracé de nature incrémentale ("Incremental layout") permet de faire efficacement un dessin du graphe, en préservant des propriétés graphiques, comme des contrôles délicats sur le tracé des graphes non connexes. L'algorithme de tracé orthogonal ("Orthogonal layout") propose un seul coude par arête et un traitement particulier pour les boucles. Le mode de dessin "Upward drawings", ou le dessin suivant l'orientation pour les graphes orientés, sont disponibles. Le dessin hiérarchique du graphe avec une représentation orthogonale des arêtes est préféré pour des graphes denses, ajouté à un algorithme de dessin, ou à l'utilisation d'une représentation symétrique.

Les outils classiques de l'édition (couper, copier, coller, déplacer, dupliquer) sont disponibles. La commande "coller" modifie l'icône du curseur. L'utilisateur choisit l'emplacement pour coller le graphe à copier, contrairement à la commande "dupliquer" qui effectue la copie du graphe sélectionné et qui le dessine légèrement décalé par rapport à la sélection.

Graph Layout Toolkit permet des sorties Postscript en couleur. Les attributs de la page, du graphe, des sommets, des arêtes et des étiquettes peuvent être adaptés suivant les équipements de sortie informatique. Le système produit un document, image de la représentation du graphe, qui peut alors être importé dans des éditeurs de texte. Il supporte des nuances de gris, les couleurs, et la rotation de l'image. Un modèle de navigation élaboré permet de créer facilement des liens entre les graphes dans une application.

Mode d'interaction : Les objets pouvant être sélectionnés sont un ensemble de sommet, un ensemble d'arêtes et une étiquette. Par contre, les objets pouvant être déplacés sont uniquement une sélection de sommets, les points de contrôle de l'arête (ils apparaissent selon le type de représentation du graphe), et l'étiquette des arêtes. La position de l'étiquette d'un sommet ou d'une arête est définie par défaut. L'étiquette se déplace sur toute la fenêtre directement avec l'aide de la souris ou indirectement par le déplacement de l'objet associé. L' environnement *Graph Layout Toolkit* possède un traitement complet des étiquettes.

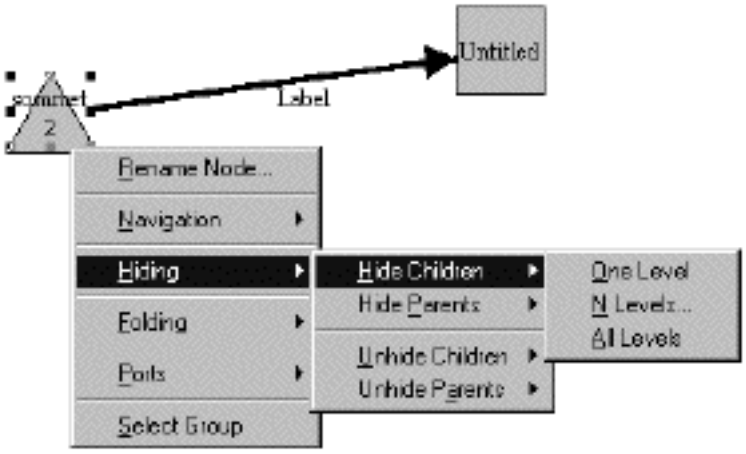


Figure 2.18 : Le menu associé à un sommet est obtenu par simple clic du bouton droit de la souris.

Toutes les manipulations sur le graphe s'effectuent directement grâce à la souris et avec le bouton gauche. A proximité d'un objet et en appuyant sur le bouton droit de la souris, un dialogue apparaît, permettant de modifier l'apparence de l'objet ou d'effectuer certaines commandes. Plusieurs types de commandes sont alors disponibles. Si une opération est interdite, alors elle apparaît en grisée. Un inconvénient existe. La commande faisant réapparaître les objets cachés dessine de nouveau le graphe au milieu de la fenêtre en changeant tous les déplacements d'objets réalisés préalablement.

Les arêtes multiples existent entre les sommets étiquetés "sommet" et "vertex" dans la *figure 2.17*. Les arêtes sont toutes étiquetées avec le nom "Label". Les arêtes ne peuvent pas être manipulées parce qu'elles sont toutes dessinées au même endroit. Pour déplacer les arêtes, il faut dessiner le graphe suivant certaines configurations, en utilisant par l'exemple les algorithmes "Incremental Layout" ou "Hierarchical Layout", afin que des points de contrôle associés aux arêtes deviennent visibles. Alors seulement, les arêtes peuvent être déplacées.

Le mode d'interaction est modal. L'utilisateur choisit le mode construction. Il désigne alors le type d'objets à construire, un sommet ou une arête. Il opte pour le mode déplacement ; les objets peuvent être déplacés.

Graph Layout Toolkit a été développé par la société commerciale américaine Tomsawyer. Il est disponible sur les compatibles PC.

Groups & Graphs

Groups & Graphs [Kocay 90] possède une boîte d'outils pour construire et manipuler les graphes. Les outils sont représentés par les icônes à gauche de la fenêtre. Il affiche également diverses informations sur le graphe tel que le nombre de sommets et d'arêtes (au dessus des outils), et le nombre d'éléments de la sélection courante (en dessous). *Groups & Graphs* propose une approche des graphes par la théorie des groupes, associée à des aspects de symétries dans le dessin des graphes.

Une bibliothèque de graphes est à la disposition des utilisateurs pour la création de graphes. Elle se trouve dans le menu "File".

Visualisation : Seule la taille des sommets varie de 4 à 32 pixels. Les autres attributs classiques des sommets ou des arêtes (la couleur, la forme et la taille pour l'arête) n'existent pas. L'étiquette du sommet se résume à son numéro dans l'ordre de création du graphe, et elle est située à l'intérieur du cercle représentant le sommet.

On peut rechercher un sommet particulier grâce à son indice par l'intermédiaire d'une boîte de dialogue. L'étiquette des sommets peut être rendue invisible. Enfin, un système d'échelle existe pour agrandir ou réduire la taille du dessin d'un sous-graphe induit.

Fonctionnalité : Les opérations classiques sur les graphes sont présentes : la subdivision d'arête ou la fusion d'un ensemble de sommets (dans les menus "Edit", "Drawing" et "Subgraph"). *Groups & Graphs* calcule des invariants classiques (dans les menus "Drawing", "Graph" et "Subgraph"), vérifie des propriétés des graphes, comme l'hamiltonicité ou la planarité (dans les menus "Drawing", "Graph" et "Planar"). Il propose de représenter un graphe planaire par un dessin sans croisement d'arêtes. La face externe du graphe est un polygone régulier. La représentation du graphe dual au graphe planaire est possible.

Une étude particulière a été faite sur la théorie des graphes et celle des groupes (le menu "Group"). Des procédures de calculs sont proposées sur les automorphismes de groupes. Des vérifications sur des propriétés liées à la théorie des groupes (les sous-groupes commutatifs, les stables, les orbites, les générateurs du groupe, les éléments

des groupes de permutation) sont également à la disposition de l'utilisateur. Plusieurs transformations sont présentes : les isomorphismes de graphes, les cycles hamiltoniens, le graphe des arêtes, le graphe de voisinages. Il existe un générateur de graphes, dont les paramètres sont le nombre de sommets et le degré du groupe associé au graphe demandé. Une bibliothèque de graphes et de groupes est disponible.

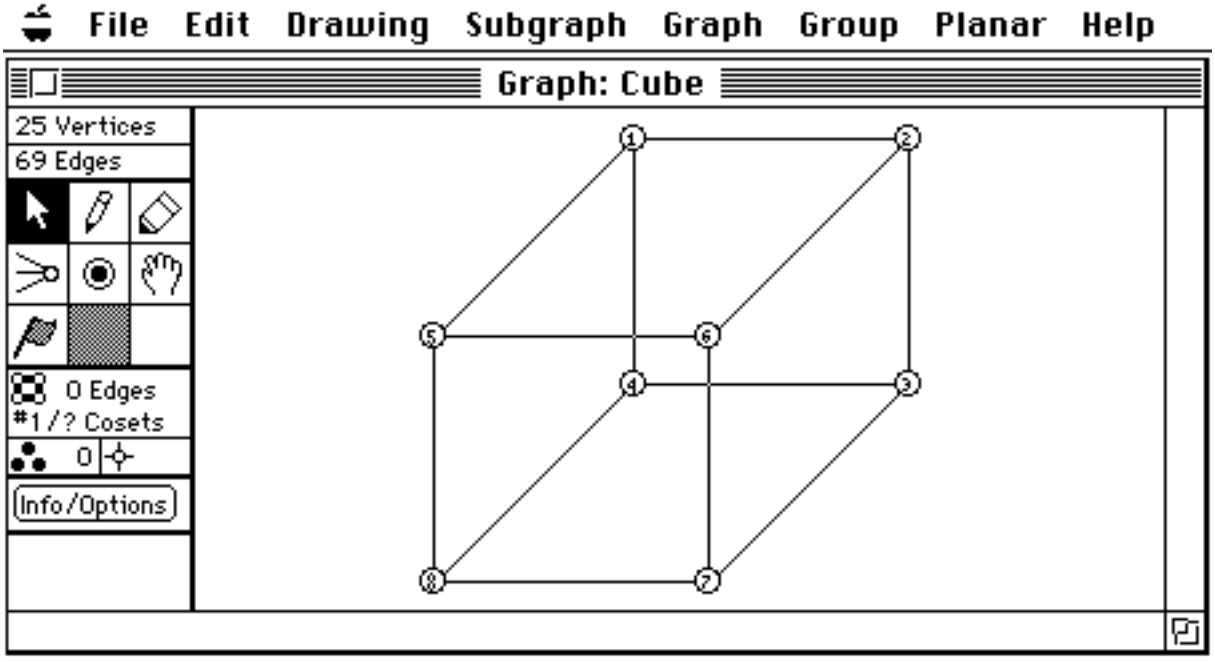


Figure 2.19 : Le graphe "Cube" dans l'environnement de *Groups&Graphs*.

Le service "couper-copier-coller" est présent, mais il est restreint parce que la copie d'un graphe est automatiquement visualisée au milieu de la fenêtre sans intervention de l'utilisateur. La commande "dupliquer" effectue une copie du sous-graphe sélectionné, et elle la dessine légèrement décalé par rapport à la sélection. Le service "Annuler/Répéter" n'existe pas.

Groups & Graphs sauvegarde le graphe dans un fichier texte avec une sortie Postscript, permettant des échanges avec d'autres programmes.

Mode d'interaction : Les objets manipulables sont limités uniquement aux sommets. La notion de graphe, comme un objet en lui-même, existe pour les commandes de copier-couper-coller et pour un déplacement global du graphe. *Groups & Graphs* autorise des manipulations sur la sélection de sommets (les menus "Edit", "Drawing" et "Subgraph").

Dans *Groups&Graphs*, l'utilisateur choisit pour le curseur des icônes associées à des états. Les outils de la visualisation sont variés. La "flèche" sélectionne les sommets et elle les déplace. Le "stylo" ajoute des arêtes en cliquant sur un sommet et en relâchant le bouton sur un autre sommet ; il permet de les supprimer. Le "stylo large" ajoute et

détruit des arêtes dans le sous-graphe. Le "cercle en gras" permet de créer des sommets par un simple clic de la souris et en pressant la touche "Shift" de détruire le sommet. Le "spoutnik" crée un sommet et le relie à la sélection courante de sommets. La main autorise le déplacement du graphe dans sa totalité. Le "drapeau" sélectionne et "illumine" une face d'un graphe planaire.

Il est disponible sur les Macintosh. Grâce aux simulateurs d'environnements Macintosh, *Group & Graphs* est disponible aussi sur les stations de travail et sur les compatibles PC.

Link

Link [Berry & al. 96] est un environnement informatique dans lequel des objets du domaine des mathématiques discrètes, représentant des problèmes réels, peuvent être manipulés et visualisés. Les objets créés sont des graphes et des hypergraphes et les graphes peuvent être orientés, avoir des arêtes multiples et posséder des attributs graphiques. *Link* a été conçu pour être un environnement éducatif utilisé dans l'enseignement des mathématiques discrètes[14] , un outil d'aide dans la recherche, et un outil de prototypage dans l'industrie.
Link est un environnement informatique sur la combinatoire et la théorie des graphes.

Visualisation : Dans *Link*, les sommets sont représentés par un cercle dont la taille et la couleur sont en option. Les arêtes possèdent trois types de représentation : un segment, un segment brisé ou une courbe de Bézier quadratique. L'option par défaut est le segment. L'étiquetage des sommets et des arêtes est automatique (par ordre chronologique de création des objets) et est attachée à son objet. Les étiquettes ne sont que des nombres. Ils commencent par le chiffre 1 pour les sommets et la valeur e0 pour les arêtes.

Des algorithmes de dessin de graphes sont proposés. Ils autorisent alors la modification de la représentation de l'arête. Les deux options pour la représentation des arêtes sont le segment brisé et une courbe de Bézier quadratique. Des outils de visualisation sont présents. Par exemple, *Link* montre ou cache les étiquettes des sommets et des arêtes. Il propose aussi la duplication de sous-graphes.

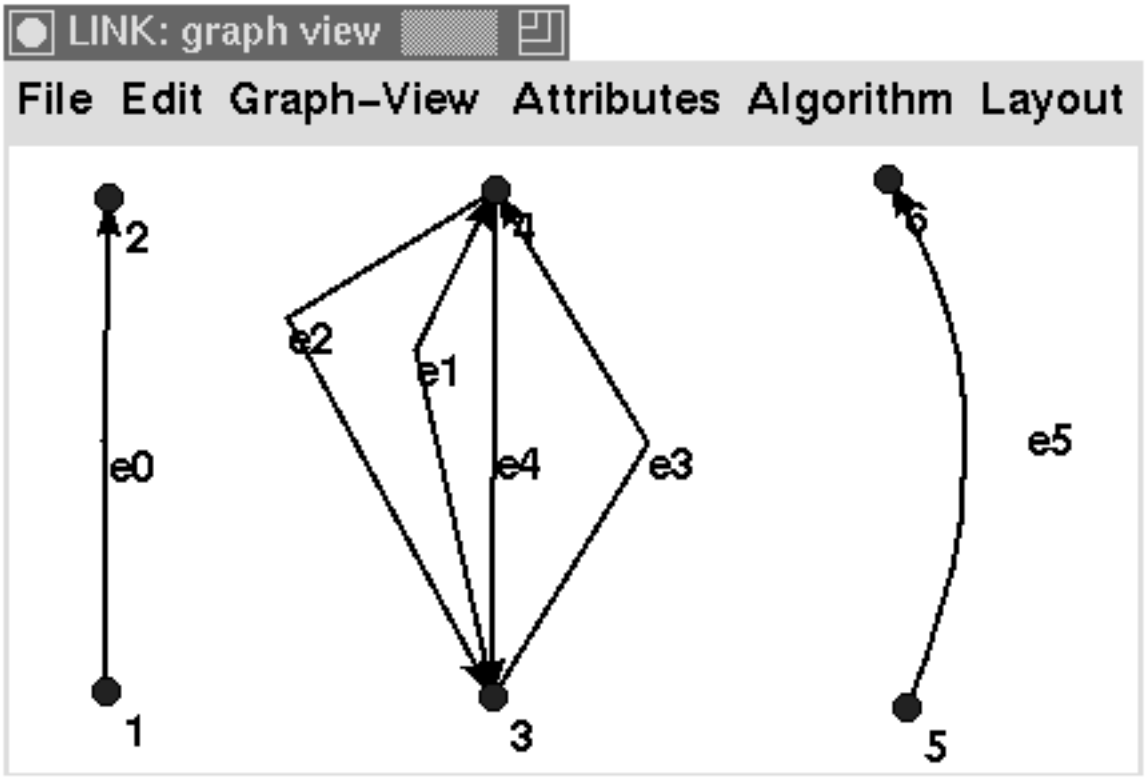
Fonctionnalité : Les opérations dans *Link* sont actuellement peu nombreuses. Ils se composent du complémentaire, du transposé, du produit, de la somme, de la vérification de l'isomorphisme entre deux graphes. Un générateur de graphes est disponible pour

[14] L'enseignement des mathématiques discrètes comme la combinatoire ou la théorie des graphes, commence dès le collège aux Etats-Unis.

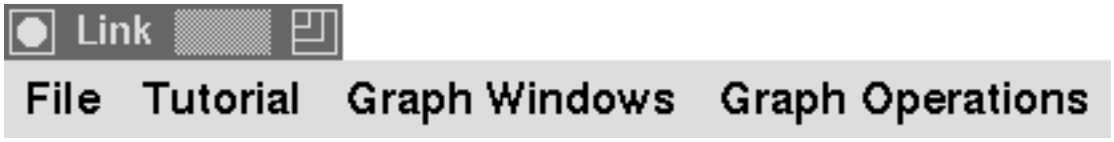
créer des cycles, des graphes complets, des grilles, des graphes aléatoires ou des tournois dans le cas orienté.

Des algorithmes de parcours de graphes (DFS, BFS et parcours topologiques) ont été programmés. L'unique programme d'optimisation présenté est un algorithme de recouvrement minimal d'un arbre.

La commande "dupliquer" effectue une copie du graphe.Une nouvelle fenêtre est créée dans laquelle la copie du graphe est dessinée. Les services "couper-copier-coller" et "Annuler/Répéter" n'existent pas.



(a) Différentes représentations des arêtes dans *Link*.

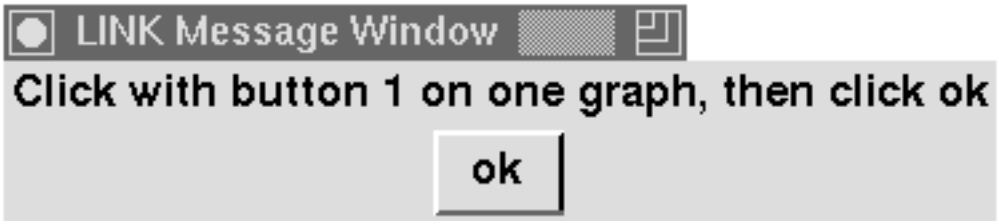


(b) : La fenêtre des commandes dans *LINK*.

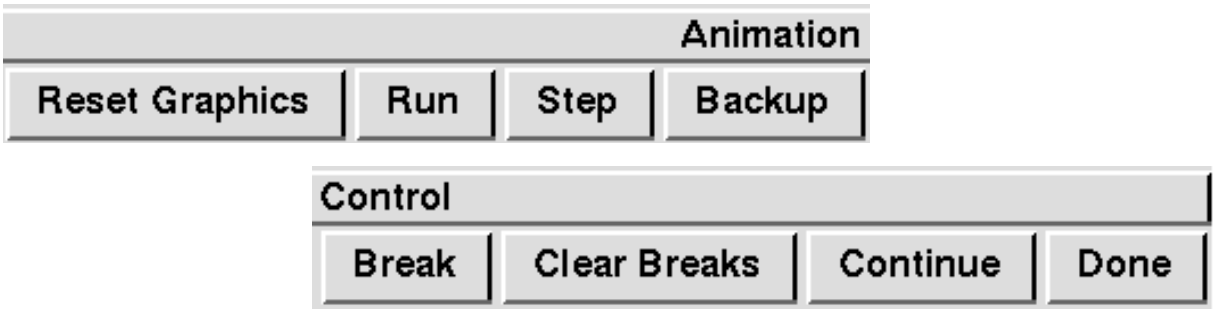
Figure 2.20 : L'environnement informatique *LINK*.

Il existe deux formats différents pour sauvegarder les données d'un graphe dans un fichier. Le format texte conserve uniquement la structure du graphe (la liste des sommets et la liste des arêtes), tandis que le format DIMAC ajoute en plus les caractéristiques graphiques et textuelles du graphe.

Mode d'interaction : Dans l'environnement *Link*, les objets manipulables sont le sommet et l'arête. La notion de sous-graphe permet uniquement de modifier l'apparence du sous-graphe, mais pas son déplacement. Pendant la manipulation d'un sommet, son précédent emplacement reste visible, mais les arêtes incidentes au sommet restent immobiles. Elles seront dessinées juste après la fin de déplacement du sommet. Le déplacement de l'arête s'effectue par l'intermédiaire de son étiquette, qui est associée au point de contrôle de l'arête.



(a) : Message affiché après toute commande demandée.



(b) Dialogue de contrôle pour l'animation sur un graphe.

Figure 2.21 : Des commandes dans *Link*.

Dans cet environnement informatique, le mode opératoire est le mode "verbe/nom" pour les opérations sur le graphe et le mode "nom/verbe" pour modifier l'apparence du graphe. Avant chaque opération, le message (a) de la *figure 2.21* apparaît ; et le résultat de l'opération est visualisé dans une nouvelle fenêtre. Ceci entraîne une certaine contrainte, puisqu'il faut constamment revenir dans la fenêtre pour cliquer sur le bouton "OK", exactement comme nous l'avons dit dans le chapitre 1. Les fenêtres sont indépendantes les unes par rapport aux autres.

Le dialogue de contrôle d'animation (voir la *figure 2.21* (*b*)) s'affiche uniquement pour quelques algorithmes spécialement conçus, comme les parcours sur les graphes.

LINK est développé au centre de recherche DIMACS ou "Discrete Mathematics and Theorical Computer Science", à l'université de Rutgers (New Jersey, Etats-Unis). Il est disponible uniquement sur les stations de travail.

Metanet

Metanet [Gomez & Goursat 90] est un environnement informatique sur les mathématiques discrètes. Il utilise le système *Scilab* [Delebecque & al. 96], qui est une bibliothèque de procédures sur les graphes et les réseaux. Deux fenêtres sont utilisées en permanence : une fenêtre *texte* pour les calculs sur les graphes, et une fenêtre *graphique* pour visualiser les graphes ou les réseaux.

Les modifications de graphes sont possibles à partir des deux fenêtres. *Metanet* est plus qu'un visualiseur de graphes, puisque la structure du graphe y est modifiable.

Visualisation : Les graphes sont orientés ou non, simples ou multiples. Ils peuvent aussi avoir des boucles. Seule la couleur est modifiable. Les étiquettes sont alphanumériques pour permettre des calculs de réseaux, comme par exemple le calcul de flots.

Fonctionnalité : L'utilisateur peut créer une animation en montrant successivement le sommet ou l'arête par une mise en évidence. Ceci est possible par l'intermédiaire d'un langage de commande adapté.

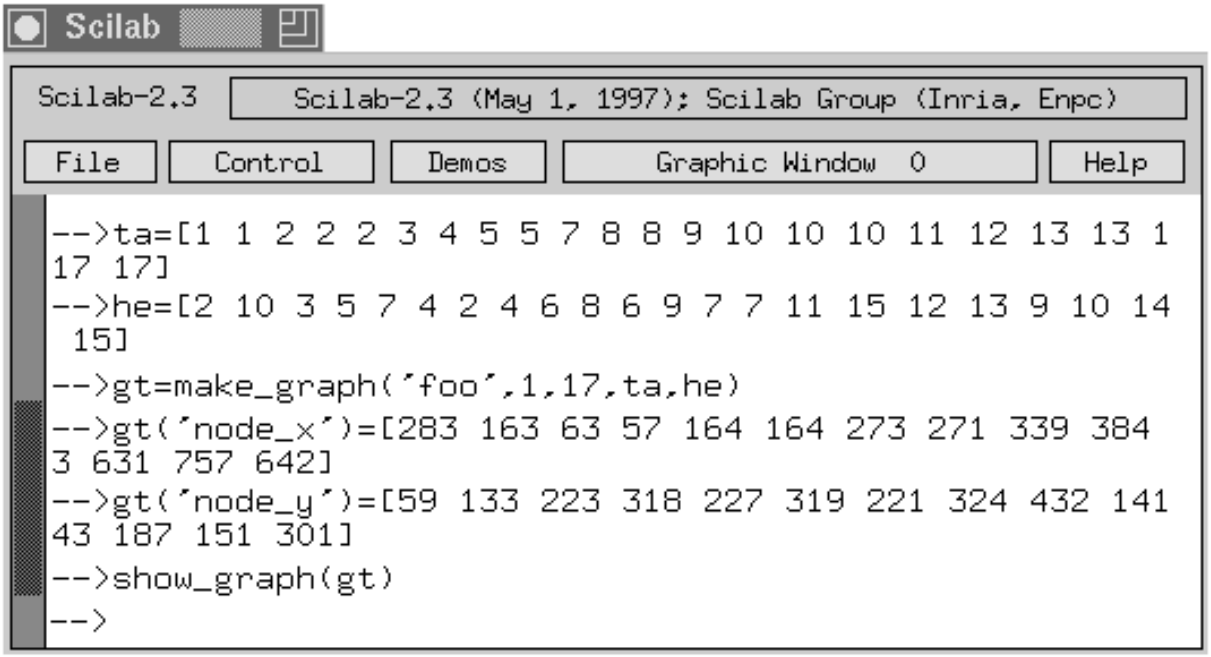


Figure 2.22 : La fenêtre textuelle de *Scilab*. Les diverses opérations pour construire un graphe s'effectuent par des mots d'un langage défini dans *Scilab*.

Le calcul des propriétés et les transformations classiques sont standard. Il existe peu de modifications graphiques directement accessibles dans *Metanet* ; ces transformations sont faites à partir de *Scilab*. *Scilab* possède des heuristiques de résolution de problèmes

difficiles : la triangulation de n points dans le plan, la résolution en 0-1 du problème du sac à dos multiple, le problème du voyageur de commerce, le problème d'affectation quadratique. Ces fonctions ne sont pas calculées directement sur le graphe ou le réseau.

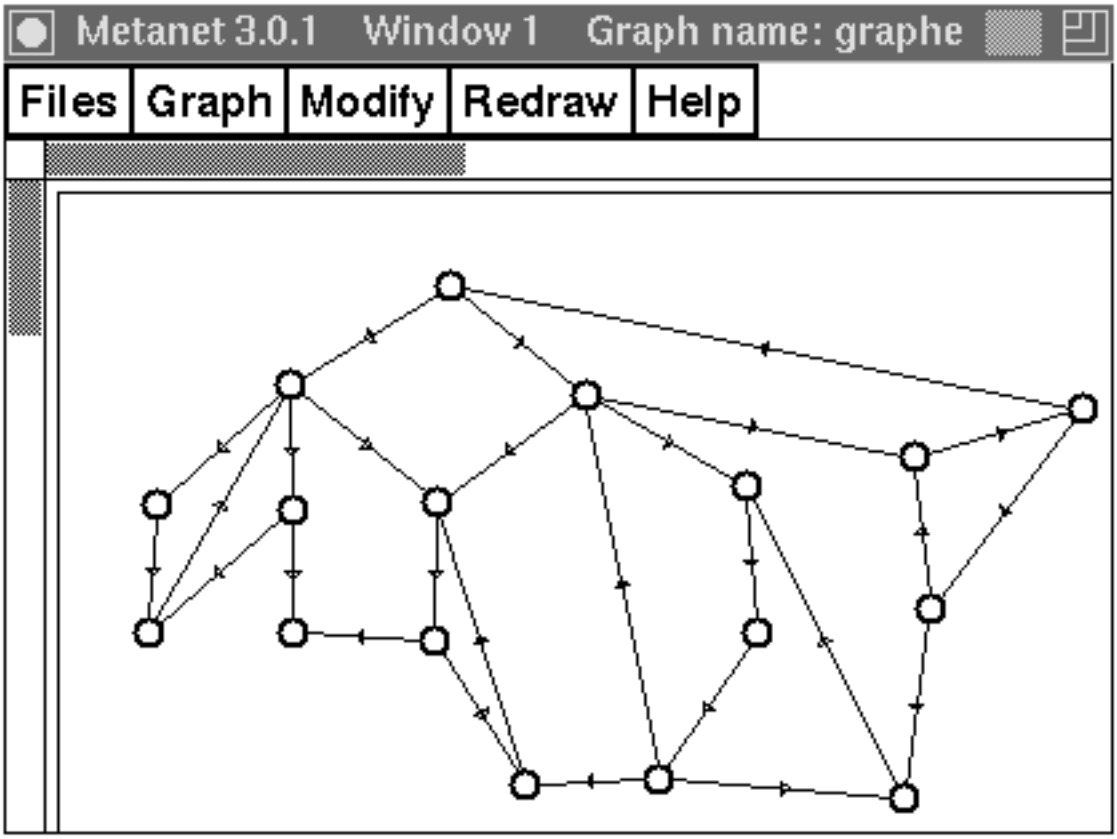


Figure 2.23 : La fenêtre graphique de *Metanet*. Le graphe représenté est le graphe défini dans la fenêtre de *Scilab*.

Mode d'interaction : Un point important est à signaler. Il n'existe aucun lien entre la représentation d'un graphe dans une fenêtre *Metanet* et les graphes utilisés dans *Scilab*. Donc, les graphes peuvent être créés ou modifiés dans une fenêtre de Metanet. Ils sont sauvés dans un fichier de type graphe et ce fichier est de nouveau ouvert dans *Scilab*. Inversement, quand un graphe est modifié dans *Scilab*, il peut être de nouveau visualisé dans une fenêtre *Metanet* en utilisant les commandes correspondantes.

Les deux types de manipulation sont disponibles par l'intermédiaire d'un langage de commandes via la fenêtre textuelle de *Scilab* ou par les menus déroulants et une manipulation des objets dans la fenêtre graphique de *Metanet.* De plus, il faut choisir dans les menus déroulants de la fenêtre graphique un mode d'interaction : la création ou le déplacement des objets. Ces derniers sont uniquement les sommets. La création des arêtes multiples s'effectue en équilibrant le nombre de liens entre deux sommets à partir de l'arête centrale (voir la *figure 2.24*).

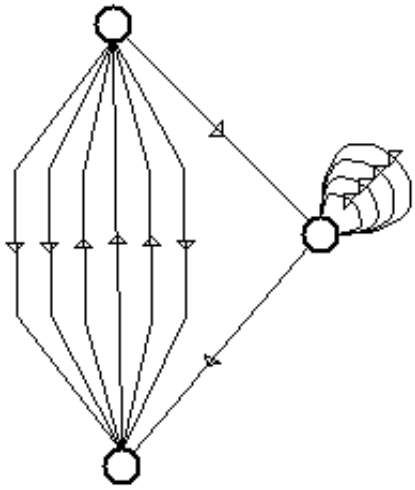


Figure 2.24 : Un graphe multiple dans *Metanet*.

La philosophie est que les calculs sont seulement faits dans *Scilab* et la fenêtre *Metanet* est seulement utilisée pour dessiner, créer et modifier les graphes. Donc, on utilise la boîte d'outils de *Scilab* indépendamment de la fenêtre *Metanet*.

Scilab et *Metanet* sont développés à l'INRIA-Rocquencourt (Institut Nationale de Recherche en Informatique et Automatique), avec comme responsable C. Gomez et M. Goursat pour *Metanet*. et J-P. Chancelier et C. Gomez pour *Scilab*. Il est écrit en Fortran et en C. *Scilab* est disponible sur toutes les machines, et *Metanet* uniquement sur les stations de travail.

3.4 D'autres types d'environnements

Nous étudions maintenant deux autres environnements informatiques très spécifiques. Le premier environnement est un système permettant de construire des éditeurs de graphes : *EGG II* d'Oliver Liechti [Lietchi 95]. Le second environnement est un environnement fonctionnant sur le réseau internet, *Graph Drawing Server* de Roberto Tamassia [Bridgemen & al. 96].

EGG II

Le but du projet *EGG II* [Liechti 95] est de spécifier et d'implémenter un outil, et de développer un éditeur interactif de graphes. La solution adoptée consiste en une programmation de la structure dans un langage orienté objet. A cette structure est attaché le comportement générique d'un éditeur de graphes.

Deux types d’utilisateurs peuvent être différenciés dans *EGG II* : le concepteur qui conçoit l’éditeur de graphes, et l'utilisateur qui les manipule pour construire des modèles de graphes. *EGG II* et un squelette générique d'éditeur de graphes. Il se décompose en deux parties : le "Designer Component" et le "Graph Editor Component".

Le concepteur utilise le "Designer Component". Il spécifie le type de sommets et d'arêtes, avec leurs attributs. Il ne peut pas définir des contraintes structurelles. Ces spécifications sont ensuite utilisées pour générer automatiquement un code de programmation. Le concepteur peut modifier ce code, avant de le lier avec le "Graph Editor Framework", pour construire le "Graph Editor Component".

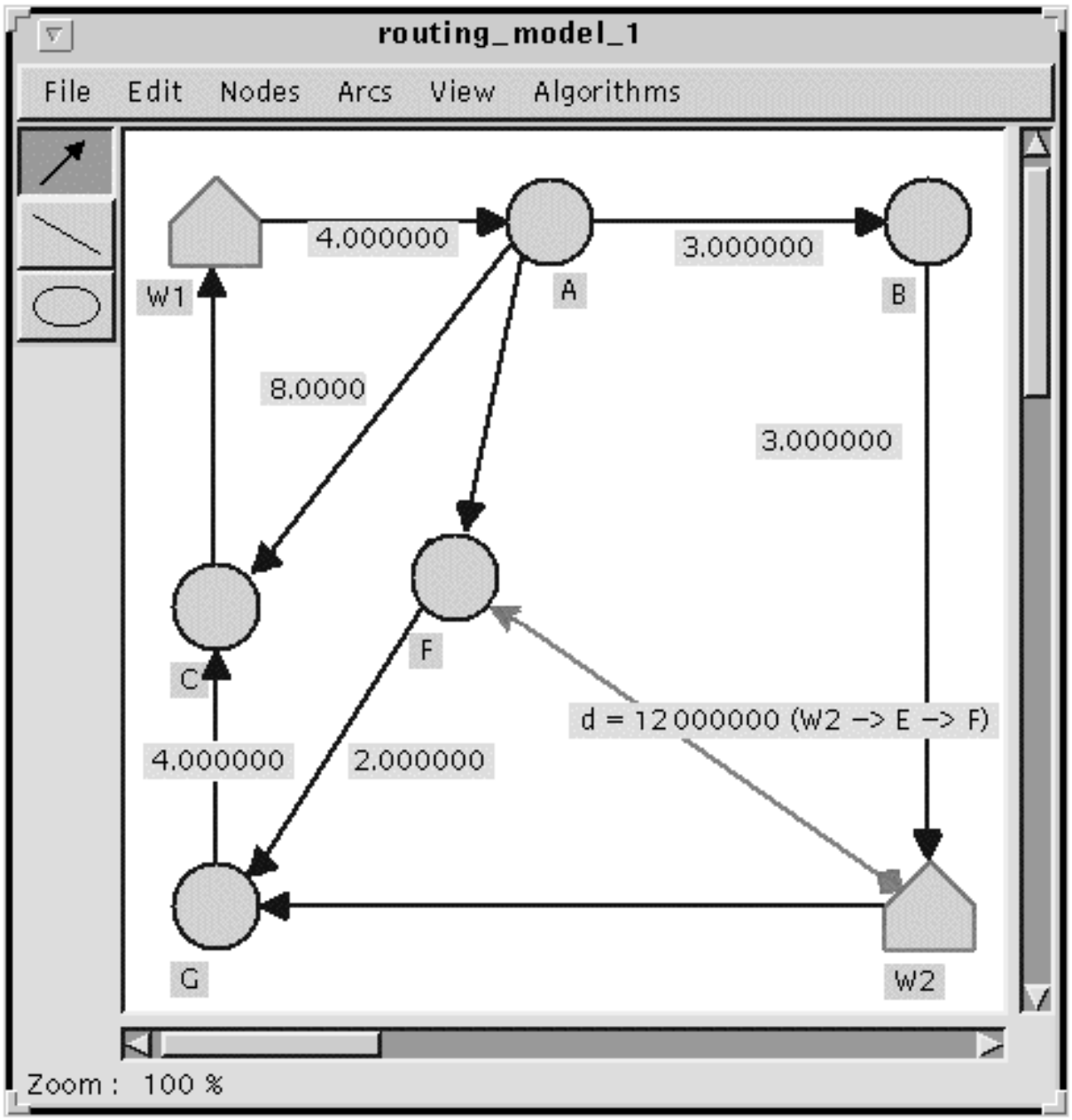


Figure 2.25 : Un exemple d'un éditeur simple construit avec *EGG II*.

L'utilisateur privilégié utilise le "Graph Editor Component". Avec cette application, il crée une instance de modèles de classes qui est spécifiée et liée au système. Les fonctionnalités sont standards et disponibles, avec des attributs classiques pour les sommets (la forme, la position topologique), la couleur du bord et de l'intérieur de l'objet, le facteur d'échelle horizontale et

verticale, l'angle de rotation) et pour les arêtes (la forme, le type de flèche en tête ou en queue, et la couleur des flèches au bord et à l'intérieur.), avec des étiquetages associés aux objets.

Un zoom, une grille magnétique, la rotation et le miroir sont les outils graphiques proposés. La sélection courante d'objets est l'ensemble des sommets et des arêtes sélectionnés. Les fonctionnalités associées à cette sélection sont l'ajout d'un ou de plusieurs objets. Un presse-papiers de type graphe a été créé pour permettre le service du copier-couper-coller.

EGG II est un logiciel développé à l'institut informatique de l'université de Fribourg (Suisse) avec comme responsable Olivier Liechti. Il est disponible sur des stations de travail.

Graph Drawing Server

Le système *Graph Drawing Server* [Bridgemen & al. 96] est un environnement informatique utilisant les capacités du réseau internet[15]. Il permet ainsi à tout utilisateur, ayant un accès au réseau, de tester des algorithmes de dessin de graphes.

Un critère supplémentaire de comparaison très important est la rapidité d'exécution des commandes. Or, *Graph Drawing Server* est tributaire des réseaux de communication, du nombre d'utilisateurs connectés au réseau internet, du serveur hébergeant le site de l'application, etc., ce qui ralentit son temps de réponse à toute manipulation. En effet, il est très agaçant pour un utilisateur d'attendre un long moment après avoir pressé une touche ou effectuer une sélection à la souris.

Visualisation : Les graphes sont de toute nature : des graphes orientés ou non, simples ou multiples et avec des boucles. Dans la représentation des graphes, les arêtes et les arcs sont mélangés. Les arêtes multiples et les boucles sont représentées par un segment brisé ou par une courbe définie par le segment brisé. Pour symboliser l'orientation, une flèche se trouve à l'extrémité terminale de l'arête. Les sommets admettent plusieurs formes possibles, même des figures comme le *"smile"*. La couleur et l'étiquetage sont aussi proposés.

La création des sommets s'effectue par un simple clic sur l'écran. Pour les arêtes, il faut commencer à partir d'un sommet par un clic et laisser le bouton de la souris enfoncé jusqu'à sortir de la représentation du sommet. Ensuite tant qu'un clic ne s'effectue pas sur un sommet, un nouveau coude est ajouté à la représentation de l'arête (un segment brisé ou une courbe en fonction de ses points de contrôle). Si les deux sommets sont identiques, alors une boucle a été créée. L'utilisateur écrit une étiquette qu'il place ensuite sur l'écran indépendamment des objets.

---

[15] L'adresse internet de ce serveur est : http://loki.cs.brow.edu:8081/graphserver/home.html, sinon accessible directement à partir de l'adresse de l'université de Brown, Providence, Etats-Unis.

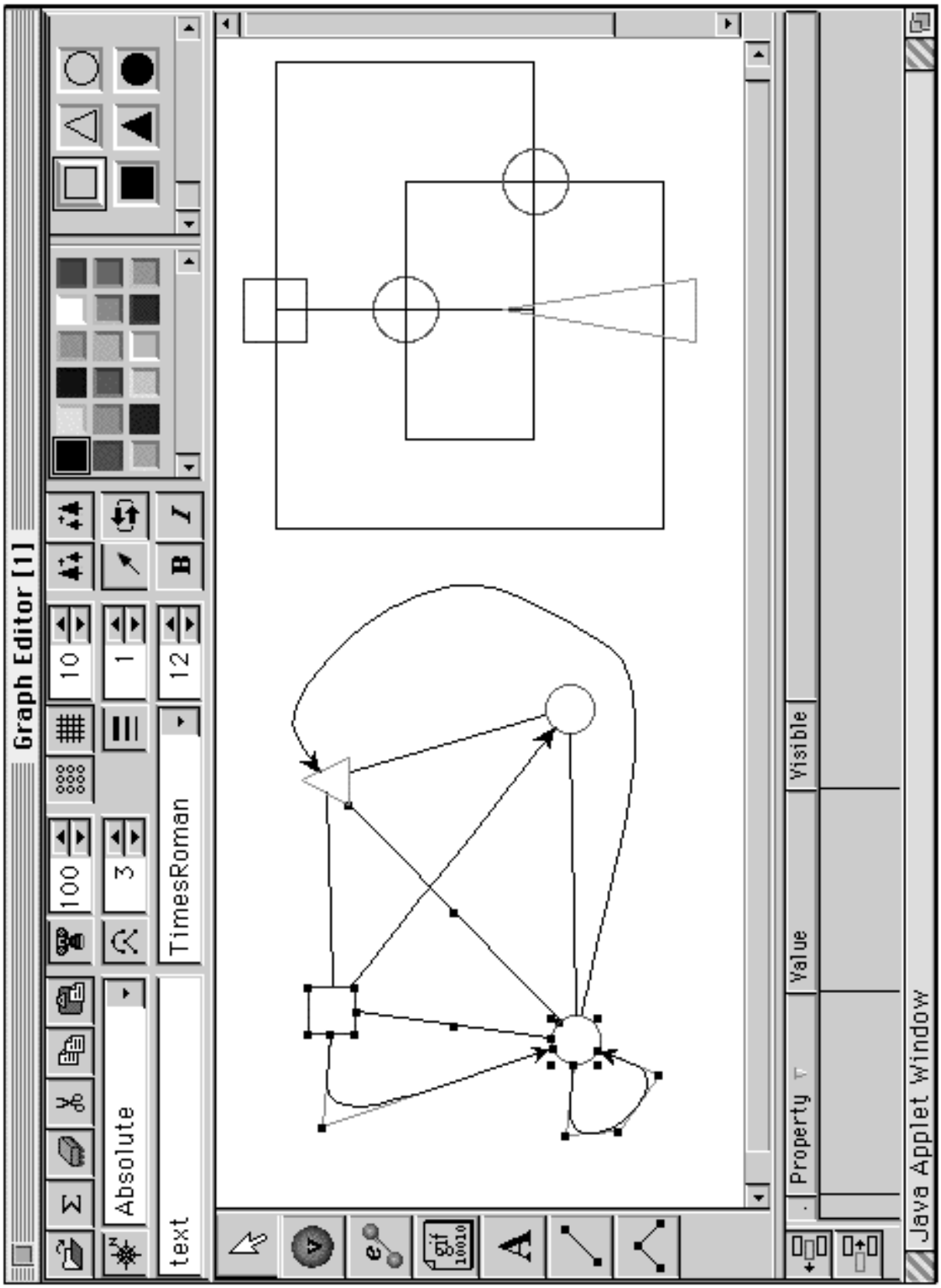


Figure 2.26 : *Graph Drawing Server* contient deux représentants différents du même graphe multiple. A droite, le graphe a été dessiné à la main, et à gauche par un algorithme de dessin automatique.

Fonctionnalité : Les principales fonctionnalités sont constituées de différents algorithmes de dessins de graphes. Les algorithmes sont, par exemple, l'algorithme de Giotto, le dessin orthogonal de type "upward" ou "non-upward", la planarisation d'un graphe ou l'algorithme de Sugiyama, dont la *figure 2.26* nous montre le résultat final (partie droite de la figure).

Pour accéder à ces algorithmes, il suffit de presser le bouton contenant comme icône la lettre grec Σ. En fait, cet environnement se veut être un système où les utilisateurs pourront tester et, peut être valider leurs propres algorithmes.

Plusieurs formats de graphes et une grande variété de familles de graphes (graphe dense ou clairsemé avec peu ou beaucoup de sommets) sont disponibles pour expérimenter les algorithmes existants.

Le service "couper-copier-coller" est présent, mais il est restreint parce que la copie ne s'effectue que pour le sous-graphe induit par la sélection des sommets, sans tenir compte des arêtes sélectionnées. Avec la commande coller, il est ensuite automatiquement visualisée dans la partie en haut à gauche de la fenêtre sans intervention de l'utilisateur. Le service "Annuler/Répéter" n'existe pas.

Mode d'interaction : Les objets manipulables sont évidemment les sommets, une sélection de sommets, l'arête par l'entremise de ses points de contrôle et les étiquettes. La représentation des arêtes est définie par des points de contrôle. La représentation est alors un segment ou une courbe définie par ces points, comme nous le montre la *figure 2.26*. La seule modification de représentation est le passage de l'une à l'autre.

Le mode d'interaction est de type modal. L'utilisateur choisit le mode construction. Il désigne alors le type d'objets à construire : un sommet, une arête ou une étiquette. Il opte pour le mode déplacement ; les objets peuvent être déplacés.

Graph Drawing Server est développé à l'université de Brown (Providence, Etats-Unis) avec le langage Java. Précisons que l'un des commanditaires est la société Tomsaywer, éditant *Graph Layout Toolkit*.

Conclusion

Nous résumons les possibilités des environnements informatiques étudiés dans le *paragraphe 3* pour la manipulation des objets. Pour simplifier l'écriture, nous utilisons les notations suivantes :

S : sommet

A : arête ou arc

F : flèche associée à un arc pour un graphe orienté

L : label ou étiquette. L_S : étiquette d'un sommet, L_A : étiquette d'une arête

E_S : ensemble de sommets, E_{SC} : sélection courante de sommets

E_A : ensemble d'arêtes, E_{AC} : sélection courante d'arêtes

P_t : point de contrôle pour une représentation de l'arête

Les visualiseurs de graphes

Pour les différents types de graphes disponibles, les notations sont :

G : graphe

M : multiple, et S : simple

O : orienté, et NO : non orienté

B : boucle

	Nature des graphes	**La manipulation des objets**	
		Sélectionner	Déplacer
da Vinci	M, B, O, NO, E	E_S si E_A non vide alors A, sinon une partie de A	S A par P_t s'il existe
Swan	NO	A S	/
VCG tool	M, O, E	A S	/

Tableau 2.1 : Tableau récapulatif suivant les critères d'analyse des visualiseurs de graphes.

Pour ces environnements informatiques, des algorithmes automatiques de dessin sont utilisés pour représenter les graphes. Des outils particuliers sont proposés pour visualiser les graphes : "une vue de poisson" pour *VCG tool*, des fenêtres multiples représentant le même graphe pour *da Vinci*, ou afficher dans une seule fenêtre deux aspects différents de la même structure pour *Swan*.

Peu d'objets peuvent être sélectionnés et déplacés : un sommet ou un ensemble de sommets et une arête. Les objets "flèche sur un arc" et "étiquette d'un sommet ou d'une arête" ne sont pas considérés. Les interactions avec les objets sont limitées par les appels à des menus, et par quelques actions directement effectuées par la souris.

Les éditeurs de graphes

Nous avons ajouté deux colonnes supplémentaires. Dans la première colonne, nous représentons les créations des objets dans la manipulation des objets, et dans la seconde des

services existants comme "couper-copier-coller", ou la possibilité d'animation sur le graphe. La gestion des ambiguïtés, pendant une sélection d'objets, est signalée lorsqu'elle existe.

Les notations ont été complétées :

CCC : le service de couper-copier-coller, il est "auto" (automatique) ou "manuel", et avoir des "options" .

GA : une gestion des ambiguïtés lors d'une sélection d'objets.

Annuler : la gestion du service "Annuler/Répéter".

Anim : une animation possible sur le graphe.

Fl : flèche d'un arc

T : un texte peut être affiché.

	Nature des graphes	**La manipulation des objets**			**Services**
		Créer	Sélectionner	Déplacer	
Cabri-graphes	M O NO B L	S A : segment brisé ou courbe	S A E_S E_A L_S L_A GA	E_S A Fl L_S L_A	CCC manuel et option Annuler
GraphEd	M O NO B L	S A : segment brisé	S A E_S E_A GA	E_S A	CCC auto
Graph Layout toolkit	M O NO B L	S A : segment simple L_S L_A et T	E_S E_A L_S L_A	E_S A L_S L_A	CCC manuel
Group & Graphs	S	S A : segment	E_S E_A	S G	CCC auto
Link	M O NO	S A : segment	E_S E_A	S A	L auto Anim
Metanet	M O B	S A : segment	E_S A	E_S	Anim
Graph Drawing Server	M O NO B	S A : segment brisé ou courbe T	E_S E_A T	E_S T	CCC auto

Tableau 2.2 : Tableau récapulatif suivant les critères d'analyse des éditeurs de graphes.

Remarques

1) La représentation des arêtes est modifiée par l'intermédiaire de leurs points de contrôle associés à la représentation. Ces points sont visibles après une sélection de l'arête.

2) Les environnements informatiques utilisent la manipulation directe ou les menus déroulants pour modifier les graphes. *Metanet* est le seul système à pouvoir employer un langage de commande pour effectuer aussi des transformations. *Metanet* utilise *Scilab* comme intermédiaire.

Les graphes édités sont des graphes multiples, orientés et ils acceptent les boucles. La création des sommets s'effectue par un simple clic. Plusieurs options ont été prises pour la création des arêtes. Dans la première option prise par *Graph Layout Toolkit, Group & Graphs* et *Link*, seul un segment est dessiné pour représenter l'arête. Dans la seconde option choisie par *GraphEd*, l'arête est représentée par une succession de segments en ajoutant des points de contrôle supplémentaires. La représentation de l'arête est appelée un segment brisé. Dans la dernière option adoptée par *Graph Drawing Server*, un lissage au niveau des coudes du segment brisé est optionnel. S'il est nul, alors l'arête est représentée est un segment brisé, sinon par une courbe. *Cabri-graphes* est le seul environnement à proposer trois représentations possibles pour l'arête (un segment, un segment brisé ou une courbe) et créer les extrémités de l'arête si elles n'existent pas.

Les environnements informatiques imposent un étiquetage automatique sur les objets comme *Link*, et parfois uniquement sur les sommets comme *Group & Graphs*. *Graph Layout Toolkit* intègre la création des étiquettes pour les sommets et les arêtes.

Les systèmes permettent de sélectionner les objets créés manuellement, un ensemble de sommets, un ensemble d'arêtes, ou une étiquette, mais aucun ne considère la flèche d'un arc comme un objet graphique. *Cabri-graphes* et *GraphEd* offrent une gestion des ambiguïtés lors de la sélection d'un objet. Dans *GraphEd*, cette gestion ne propose pas l'ensemble des objets correspondant à cette ambiguïté, mais elle parcourt cet ensemble demandant l'avis de l'utilisateur pour chaque élément. Dans *Cabri-graphes*, la gestion des ambiguïtés est sélective.

Les objets sélectionnés peuvent être déplacés généralement. Pour *Group & Graph* et *Link*, cette action s'applique à un sommet et non un ensemble de sommets. La représentation des arêtes est déformée par l'intermédiaire des points de contrôle, après la sélection d'une arête. La flèche représentant l'orientation d'un arc est fixée à l'extrémité terminale de l'arc sans possibilité d'être bougée. L'environnement *Graph Layout Toolkit* propose le déplacement des étiquettes (pour les autres systèmes, l'étiquette est fixe). L'étiquette suit le déplacement de l'objet associé, mais elle-même n'est pas restreinte dans son déplacement par rapport à l'objet associé.

Pour terminer, remarquons que des services classiques des interfaces sont faiblement présents. *Cabri-graphes* est le seul à proposer le service "Annuler/Répéter". L'unique service

offert est le "couper-copier-coller", collant systématiquement le graphe au dessus du graphe visualisé. *Cabri-graphes* dispose d'option pour ce service.

Nous nous proposons de concevoir et de construire un nouvel environnement informatique sur les graphes, poursuivant le travail effectué pour un éditeur de graphes simples.

Chapitre 3

Modélisation d'une interface sur la théorie des graphes et description de *Cabri-graphes*

Dans ce chapitre, nous décrivons les modifications apportées à l'environnement informatique *Cabri-graphes*. Elles sont fondées sur les réflexions théoriques exposées au chapitre 1, en particulier au mode d'interaction, et dans le domaine des éditeurs de graphes décrits au chapitre 2. Il s'agit donc d'une description du cahier des charges ayant régi la construction et l'évolution d'une interface de manipulation directe de graphes

Pour cette présentation, nous avons choisi d'exposer dans un premier paragraphe les choix relatifs au niveau du domaine. Dans le second paragraphe, nous décrivons le niveau conceptuel en commençant par les relations attachées à un graphe. Avec les attributs informatiques, nous formalisons ces relations et nous sélectionnons des structures de données abstraites possibles pour la représentation interne des objets du domaine. Nous affinons les structures de données avec leurs représentations par les attributs graphiques. Finalement, nous proposons des opérations sur les objets. Au troisième paragraphe, nous nous intéressons au niveau relationnel aux attributs de dépendance, et en particulier aux attributs de représentation (la création, la sélection et le déplacement des objets du domaine). Nous fournissons un aperçu de l'interface sur les graphes, et nous finissons par les structures de données abstraites adaptées aux graphes. Dans le quatrième et dernier paragraphe, nous décrivons le niveau informatique avec la programmation de ces structures de données, les choix dans les attributs visuels et la description de notre environnement informatique *Cabri-graphes*.

1. Le niveau du domaine

Nous présentons le domaine que nous modélisons en apportant quelques précisions sur la notion d'arêtes multiples, et nous définissons les objectifs de l'éditeur de graphes avec un cahier des charges sur les multiples facettes de l'environnement informatique à construire.

1.1 Définition

Nous rappelons la définition d'un graphe[1]. Un graphe $G = (V, E)$ est un couple constitué par un ensemble $V = \{ v_1, v_2, \ldots, v_n ; n \geq 1 \}$ et par une famille $E = \{ e_1, e_2, \ldots, e_m ; m \geq 0 \}$ d'éléments du produit cartésien $V \times V = \{ (u, v) / u \in V, v \in V \}$. Un élément (u, v) de $V \times V$ peut apparaître plusieurs fois dans la famille E.

Les élements de V sont appelés les sommets de G, et le nombre de sommets de G est appelé *l'ordre* de G.

Les éléments de E sont appelés les arcs de G. Un arc de la forme (u, u) est appelée une *boucle*. Si plusieurs éléments de la forme (u, v) de $V \times V$ apparaissent dans E, alors G est appelé graphe *multiple*. Pour un arc $e = (u, v)$, l'élément u est son extrémité initiale, et v est son extrémité terminal.

Si aucun ordre n'est précisé sur les extrémités des arcs, alors les élements de E sont appelées des arêtes et le graphe G est dit non orienté, sinon le graphe G est dit orienté.

Un graphe est dit fini lorsque l'ensemble V et la famille E sont de cardinalité finis. Dans la suite de notre étude, nous nous intéressons exclusivement à des graphes finis.

Par définition, un graphe simple ne comporte aucune arête multiple, ni aucune boucle. Pour des raisons pratiques et techniques dans les algorithmes de parcours de graphes et dans la visualisation des graphes, une distinction supplémentaire est aussi nécessaire entre les arêtes multiples et les boucles.

---

[1] Pour les notions utilisées dans ce chapitre, nous renvoyons le lecteur au chapitre 2 et au chapitre 4 pour les défintions complémentaires.

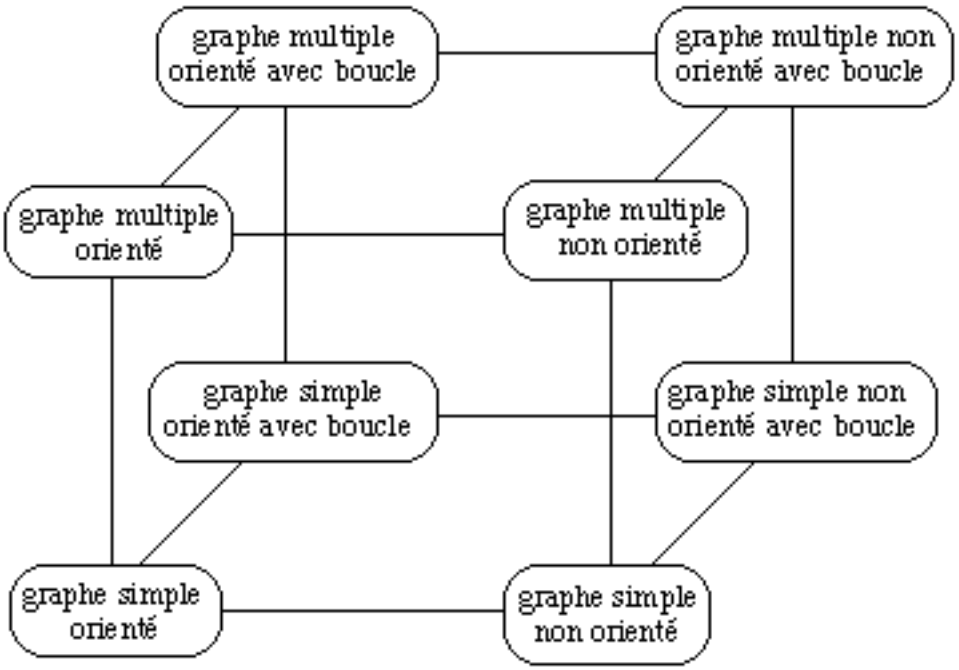


Figure 3.1 : Les huit types de graphes possibles.

Il existe ainsi huit types possibles de graphes représentés dans la figure 3.1 par le graphe suivant : chaque sommet du graphe représente un type de graphe particulier, et une arête est incidente à deux sommets s'ils possèdent une seule propriété en contradiction.

1.2 Les objectifs

Nous définissons un cahier des charges pour un éditeur de graphes. Nous construisons une interface de manipulation directe sur les graphes, pour être un outil utilisable par les chercheurs et les étudiants en théorie des graphes. La visualisation des graphes est aussi importante que leur manipulation dans notre environnement informatique.

Le domaine modélisé est une partie du domaine de la théorie des graphes. Il concerne plus précisément les graphes finis multiples ou simples, orientés ou non, avec ou sans boucle. Ces graphes sont représentés dans le plan et ils acceptent un étiquetage des sommets et des arêtes.

La création des graphes est interactive et ils sont manipulés comme dans un environnement papier-crayon, mais avec toutes les facilités et la puissance que des ordinateurs apportent. Les graphes sont dessinés aussi bien manuellement qu'automatiquement, en mettant en évidence des propriétés géométriques ou structurelles. Un intérêt principal réside dans les comportements interactifs de l'environnement, lors de la création ou la manipulation des objets, grâce à la manipulation directe de tous les objets : les objets intrinsèques au domaine, les objets graphiques, et les objets de l'interface.

Une boîte d'outils est associée à notre environnement. Ils effectuent différents calculs tels que l'évaluation d'invariants sur les graphes. Il propose des opérations sur les graphes et des transformations classiques. Les transformations graphiques portent sur la représentation des graphes, comme la rotation ou la symétrie par rapport aux axes du plan. Les transformations sur les structures des graphes sont par exemple le changement de nature des graphes, ou les produits de graphes.

2. Le niveau conceptuel

Choix des objets du domaine

Les objets de base du domaine sont naturellement les sommets, les arêtes, et les graphes. En partant de la définition du graphe, l'ajout de l'objet ensemble paraît indispensable. Dans ce paragraphe, nous déterminons le mode de représentation interne des connaissances ou des objets du domaine (les attributs informatiques) et leurs représentations externes (les attributs graphiques).

Dans un premier temps, nous nous attardons sur les relations associées à un graphe, ce qui nous indiquera un type de représentation possible. Nous décrivons les structures de données sur les graphes. L'aspect graphique associé aux contraintes de la manipulation directe affinera notre étude sur les structures de données abstraites à utiliser.

2.1 Relations attachées à un graphe

Les relations associées à un graphe reflètent deux aspects des problèmes étudiés sur les graphes : une approche globale et une approche locale.

2.1.1 L'approche globale

Nous considérons deux ensembles : l'ensemble des sommets et l'ensemble des arêtes. Un sommet $v \in V$ et une arête $e \in E$ sont dits *incidents* si v est une extrémité de e. Deux sommets sont dits *adjacents* lorsqu'ils sont incidents à une même arête. On définit ainsi une relation interne à V, appelée *relation d'adjacence*. De même, deux arêtes sont dites *incidentes* lorsqu'elles ont une extrémité commune (un sommet). On définit ainsi une relation interne à E, appelée *relation d'incidence*.

2.1.2 L'approche locale

De manière analogue, nous pouvons considérer l'ensemble des arêtes incidentes à chaque sommet. Ainsi, nous pouvons conserver une trace de toutes les arêtes du graphe en conservant pour tous les sommets *u* du graphe l'ensemble des arêtes incidentes à *u*., ou, de façon équivalente si *G* est simple, l'ensemble des sommets adjacents. Nous obtenons une autre définition pour le graphe.

Un graphe $G = (U, \Gamma)$ est constitué d'un ensemble *U* appelé l'ensemble des sommets de *G* et pour tout *u* de *U* un sous-ensemble Γ_u de *U*, appelé l'ensemble des sommets adjacents à *u*. Cet ensemble de sommets est aussi appelé ensemble des voisins de *u* ou voisinage de *u*. Il est également noté $N(u)$ (*N* pour "Neighbourhood" ou voisinage en anglais). Dans le cas de boucles au niveau du sommet *u*, nous avons que $u \notin \Gamma_u$. Nous définissons la notion de *voisinage étendu* en ajoutant le sommet à l'ensemble de ses voisins.

Le voisinage d'un sous-ensemble *S* de *V* est noté $N(S)$, et il correspond à l'ensemble des sommets de *V-S* voisins d'au moins un sommet dans *S*.

2.2 Attribut informatique

L'introduction de ces relations d'incidence, d'adjacence et de voisinage suggère un mode de représentation des graphes par une matrice correspondante à l'approche globale (cette approche est la plus souvent adoptée [Xuong 92] et [Gondran & Minoux 85]) et par des ensembles pour l'approche locale (cette approche est moins utilisée [Kruse & al 89]). Dans les exemples cités, le mode de représentation dépend de la modélisation du problème étudié.

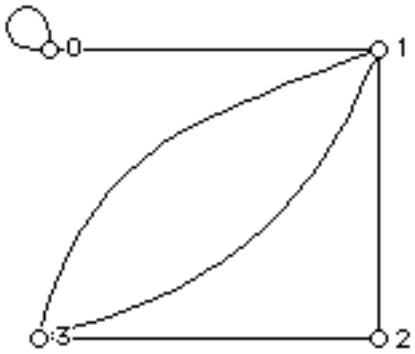
2.2.1 Style de représentation

Matrice d'adjacence ou matrice sommet-sommet

A tout graphe $G = (V, E)$ avec *n* sommets, on associe une matrice carrée de taille $n \times n$, notée $\mathrm{Adj}(G) = [a_{ij}]$ dont les lignes et les colonnes sont indexées par les éléments de *V*. Elle est définie suivant la nature du graphe :

. pour un graphe *G* non orienté, a_{ij} est le nombre d'arêtes ayant v_i et v_j comme extrémités,

. pour un graphe *G* orienté, a_{ij} est le nombre d'arcs d'extrémité initiale v_i et d'extrémité terminale v_j .



	0	1	2	3
0	1	1	0	0
1	1	0	1	2
2	0	1	0	1
3	0	2	1	0

Figure 3.2 : Graphe non orienté G et sa matrice d'adjacnece Adj(G).

Si *G* est non orienté, alors Adj(G) est une matrice symétrique à éléments entiers, sinon Adj(G) est une matrice antisymétrique pour *G* orienté. Si *G* est sans boucle alors la diagonale de la matrice n'est formée que de 0. Si *G* est simple non orienté, alors les éléments a_{ij} de la matrice Adj(G) appartiennent à l'ensemble {0,1}, sinon à l'ensemble {-1,0,1} pour G orienté.

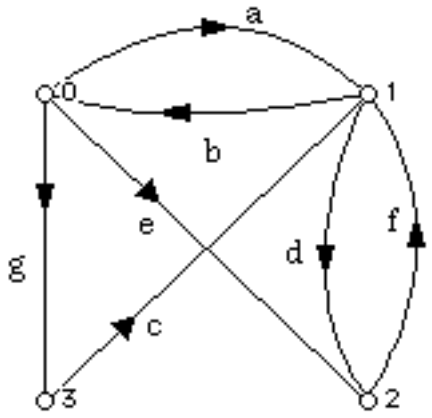
Matrice d'incidence ou matrice sommet-arête

A tout graphe $H = (V, E)$ avec n sommets et m arêtes, on associe une matrice de taille $n \times m$, notée Inc(H) = $[i_{jk}]$ dont les lignes sont indexées par les éléments de V et les colonnes par ceux de E. La matrice Inc(H) est définie suivant la nature du graphe :

.pour un graphe H non orienté, i_{jk} est le nombre de fois où le sommet v_j est incident à l'arête e_k

. pour un graphe H orienté, nous avons que :

$$i_{jk} = \begin{cases} -1 & \text{si } v_j \text{ est l'extrémité terminale de l'arc } e_k \\ +1 & \text{si } v_j \text{ est l'extrémité initiale de l'arc } e_k \\ 0 & \text{si } e_k \text{ est une boucle ou si le sommet } v_j \text{ et l'arc } e_k \text{ ne sont pas incidents} \end{cases}$$



	a	b	c	d	e	f	g
0	+1	-1	0	0	+1	0	+1
1	-1	+1	-1	+1	0	-1	0
2	0	0	0	-1	-1	+1	0
3	0	0	+1	0	0	0	-1

Figure 3.3 : Un graphe orienté H et sa matrice d'incicence $Inc(H)$ associée.

Pour un graphe *H* non orienté, le terme e_{jk} de la matrice Inc(H) appartient à l'ensemble {0,1,2} et la somme des éléments d'une colonne est égale à 2. Pour un graphe *H'* orienté, le terme e_{jk} de la matrice Inc(H') appartient à l'ensemble {-1,0,1} et la somme des éléments d'une colonne est égale à 0.

Voisinage

Dans un graphe $G = (V, E)$, à chaque sommet v est associé l'ensemble de ses sommets adjacents. En reprenant le graphe *G* de la figures 3.2, nous obtenons les tableaux de la figure 3.4.

0
1
2
3

0	1	
0	2	3_2
1	3	
1_2	2	

(a) Représentation sans répétition

0
1
2
3

0	1		
0	2	3	3
1	3		
1	1	2	

(b) Représentation avec répétition.

Figure 3.4 : Deux types de tableaux pour représenter le voisinage des sommets du graphe. Le nombre "2" en indice dans le tableau (a) signifie que le nombre d'arêtes entre le sommet et son sommet-voisin est 2

Pour un graphe simple, le premier tableau n'est qu'une forme condensée de la matrice d'adjacence. Par contre le deuxième tableau expose le problème de la multiplicité des arêtes. Effectivement, la question est de savoir d'une part si la redondance d'informations est nécessaire en ajoutant plusieurs fois un même élément dans le voisinage d'un sommet (tableau b). D'autre part, une seule représentation d'un sommet voisin (tableau a) est suffisante en ajoutant pour les éléments redondants des informations supplémentaires au niveau du sommet considéré, comme l'ensemble des arêtes multiples incidentes à ces deux sommets.

Nous commençons à distinguer deux styles de représentation possibles : par les matrices d'adjacence ou d'incidence et par des ensembles simples ou multiples. La multiplicité des arêtes et l'orientation possible du graphe sont deux situations à examiner plus précisément.

2.2.2 Multiplicité des arêtes

Opération de base

Il existe plusieurs opérations élémentaires sur les graphes, mais les principales sont liées à des contraintes sur le graphe lui-même. Il existe trois contraintes d'ordre structurel suivant la nature des graphes (multiple ou simple, orienté ou non, avec ou sans boucle), et une contrainte d'ordre temporel. En effet, le nombre d'arêtes et de sommets évolue dans le temps en fonction des

opérations de bases sur le graphe avec l'insertion et la suppression d'un élément. Une autre opération élémentaire sur les graphes est la recherche d'un élément dans un ensemble.

Les graphes sont définis en terme d'ensembles et les opérations de bases sont également des opérations sur les ensembles. Il paraît naturel de choisir le type ensemble comme structure de données. Le premier est l'ensemble des sommets et le second des arêtes comme paires de sommets.

Dans une application de type classique, c'est-à-dire sans la visualisation du graphe, la multiplicité des arêtes est similaire à l'adjonction d'un nouvel élément dans un ensemble, soit un nouveau sommet dans le voisinage d'un sommet, soit une nouvelle arête dans l'ensemble des arêtes. Par contre, pour les structures de données, il faut distinguer "les ensembles" n'admettant pas de redondance, avec "les ensembles" qui autorisent la multiplicité de leurs éléments, c'est-à-dire étudier le type de donnée *ensemble* et le type *famille*.

Les types ensemble et famille

La différence fondamentale entre une famille et un ensemble est la possibilité pour une famille d'avoir plusieurs fois le même élément, tandis que l'ensemble ne possède ses éléments qu'en un seul exemplaire. Ceci entraîne des tests supplémentaires pour les méthodes associées, et les conséquences suivantes :

- ajouter à un ensemble un élément déjà présent est inopérant,
- retirer à une famille un élément n'enlève qu'une occurrence de ce dernier et pas nécessairement toutes ses occurrences.

Nous devons étudier trois types de données : un ensemble, une famille et une famille avec un "classement" sur les éléments, c'est-à-dire que les occurrences d'un même élément sont regroupées ensemble. Nous comparons ces trois types de données par rapport à des opérations simples qui sont aussi appliquées à un graphe : l'insertion, la recherche d'un élément et la suppression.

Considérons un ensemble ou une famille de cardinalité n. Dans une famille, l'insertion est rapide. Cette opération est immédiate et sa complexité en temps s'écrit en O(1) au lieu de n, le temps de parcours complet de l'ensemble ou de la famille avec tri. La recherche d'un élément demande dans tous les cas le parcours du type de données. La suppression d'un élément requiert une recherche complète sur la famille afin de supprimer toutes les occurrences de l'élément, soit un temps en n . La suppression de la première occurrence dans une famille équivaut à la suppression de cet élément dans un ensemble ou dans une famille avec tri, cette opération demande un temps de n dans le pire des cas et en $n/2$ en moyenne.

Pour les recherches et les suppressions, des familles de plus grande taille sont traitées que dans le cas des ensembles ou des familles avec tri. La différence sur la longueur dépend de la fréquence d'insertion d'un élément déjà présent dans l'ensemble.

	Opérations		
Type d'ensemble	insertion	suppression	recherche
simple	$n/2 \rightarrow n$	$n/2 \rightarrow n$	$n/2 \rightarrow n$
famille	0	n	$n/2 \rightarrow n$
multiple et trié	$n/2 \rightarrow n$	$n/2 \rightarrow n$	$n/2 \rightarrow n$

Tableau 3.1 : Comparaisons entre trois types d'ensembles.

Ce tableau permet de comparer les types ensemble, famille et famille avec tri en fonctions des opérations de base. Décider d'une méthode dépend de l'environnement en construction. Si les insertions prédominent alors il faut autoriser les répétitions ; par contre si des répétitions peuvent se produire et si les recherches et les suppressions prévalent alors on a intérêt à vérifier la présence de l'élément avant de l'insérer.

Multiplicité des arêtes

La structure de données ensemble est intéressante et importante dans l'étude des graphes. Plusieurs catégories d'ensembles sont nécessaires. Les familles au niveau du voisinage conservent la multiplicité des liens entre deux sommets, et des ensembles sont des ensembles d'objets standards.

La dissociation entre les deux types ensemble et famille est très contraignante, d'autant plus que, dans les manipulations des graphes, le passage du type ensemble au type famille (et inversement) peut être assez fréquent. Par exemple, un graphe simple ou multiple est créé. L'ajout ou la suppression d'une arête déjà existante obligent éventuellement la création d'un nouveau type de graphe correspondant à la nouvelle situation présente.

Le graphe est composé d'un ensemble pour les sommets et d'une famille pour les arêtes. La nature de la relation d'adjacence définit le type du graphe (simple ou multiple), mais cette relation est attachée à chaque sommet. Or cet ensemble de sommets-voisins est de nature différente suivant le genre de graphe choisi : un ensemble pour un graphe simple et une famille pour un graphe multiple. Séparer les types ensemble et famille entraîne la création d'un type particulier de sommet pour chaque type de graphe.

Par conséquent, même sans choisir une représentation particulière, la distinction entre le graphe simple et le graphe multiple n'est pas facile à effectuer.

2.2.3 Orientation des arêtes

Orientation par défaut

L'orientation établit un ordre sur le couple des extrémités d'une arête. Par défaut, et lors d'une construction graphique, une arête est créée en précisant l'ordre de ses deux extrémités. Un choix naturel s'impose pour désigner l'extrémité initiale et terminale, l'ordre de rencontre ou de construction de ces dernières : la première pour l'extrémité initiale, la seconde pour l'extrémité terminale.

Par contre, cette distinction ou cette orientation modifie au niveau du sommet la structure de données et les informations associées. Le voisinage d'un sommet regroupe plusieurs types d'information (des informations structurelles, des informations graphiques, des informations sur l'étiquetage, etc.). Comme le voisinage d'un sommet est l'ensemble des sommets adjacents, sans autre précision sur la nature de l'adjacence. Ainsi une distinction doit apparaître entre les prédécesseurs et les successeurs d'un sommet pour les graphes orientés.

Caractéristiques du voisinage

Le problème de l'information "successeur" et "prédécesseur" est important. Les sommets successeurs et prédécesseurs peuvent être conservés dans deux ensembles séparés, ou dans un même ensemble en ajoutant une variable supplémentaire à l'élément afin de connaître la nature de l'adjacence. Une autre solution est de préserver uniquement les sommets successeurs ou prédécesseurs à chaque sommet du graphe.

Il existe donc quatre choix possibles d'ensembles pour conserver l'orientation au niveau des sommets. Le premier choix est l'ensemble E_1 des successeurs d'un sommet, et le second choix est l'ensemble E_2 des prédécesseurs. Le troisième type est l'ensemble E_3 composé des successeurs et des prédécesseurs. Dans le dernier cas, chaque élément de l'ensemble E_4 est constitué de l'ensemble de ses successeurs et de l'ensemble de ses prédécesseurs.

Nous étudions ces quatre types d'ensembles par rapport à des opérations élémentaires qui sont aussi appliquées à un graphe simple : l'insertion, la suppression et la recherche d'un élément.

Opération et voisinage

Avec des ensembles simples de cardinalité n, la recherche d'un élément nécessite le parcours de cet ensemble. La complexité en temps est alors en $n/2$ en moyenne et en n dans le pire des cas. Dans les graphes simples orientés, on a souvent besoin de trouver tous les successeurs ou tous les prédécesseurs d'un sommet donné. Avec un ensemble de type E_1 au niveau des sommets pour la sauvegarde de l'orientation, il faut parcourir le voisinage de tous les sommets du graphe pour trouver tous les successeurs de chaque sommet. La complexité en temps est alors $n^2/2$ en

moyenne à n^2 dans le pire des cas. L'opération est immédiate avec les ensembles de type E_2 et E_4, puisque l'ensemble recherché est déjà présent au niveau du sommet. Avec un ensemble de type E_3, un parcours complet de l'ensemble des voisins est aussi nécessaire. La complexité en temps est alors de *n* en moyenne à *2n* dans le pire des cas.

	Types d'ensemble			
Opération	$\mathbf{E}_1$	$\mathbf{E}_2$	$\mathbf{E}_3$	$\mathbf{E}_4$
chercher un arc	$n/2 \rightarrow n$	$n/2 \rightarrow n$	$n \rightarrow 2n$	$n/2 \rightarrow n$
Trouver les successeurs	$n^2/2 \rightarrow n^2$	0	$n \rightarrow 2n$	0
Trouver les prédécesseurs	0	$n^2/2 \rightarrow n^2$	$n \rightarrow 2n$	0

Tableau 3.2 : Comparaison des opérations suivant la nature des ensembles conservant l'orientation au niveau de chaque sommet dans un graphe.

Le tableau 3.2 résume le temps nécessaire pour effectuer les opérations élémentaires sur les graphes. La solution adoptée est de conserver au niveau du sommet deux ensembles de sommets-voisins : les successeurs et les prédécesseurs.

A cette difficulté de l'orientation, s'ajoute aussi pour nous la complexité précédente des ensembles multiples. La complication est donc rencontrée avec les graphes multiples orientés. Nous avons pour l'instant considéré uniquement les questions de temps. La conservation des informations admet aussi une complexité en place mémoire. Il faut alors trouver un compromis entre les deux situations.

A ce problème de distinction entre le type de voisinage, un autre problème aussi important s'impose dans le passage entre un graphe orienté et non orienté, avec la préservation de la multiplicité des arêtes.

Pour conclure ce paragraphe, nous avons adopté les ensembles multiples et triés. Ils nécessitent un même temps de calcul pour toutes les opérations. Pour l'orientation, nous conservons l'ensemble de successeurs et l'ensemble des prédécesseurs pour chaque sommet du graphe. Les notions de multiplicité et d'orientation deviennent des propriétés du graphe, afin de faciliter l'exécution des algorithmes par une simple vérification globale au niveau du graphe, et non une vérification locale en parcourant l'ensemble des sommets et des arêtes pour obtenir l'information.

2.3 Attribut graphique

Les structures de données précédentes ne s'occupent que de la structure interne des graphes. La visualisation du graphe nous oblige à poursuivre et à affiner cette étude de structures de données aux attributs graphiques.

Aux caractéristiques associées à la théorie des graphes (le voisinage, le degré d'un sommet, l'étiquetage, etc.), des informations purement graphiques (la forme, la taille, la couleur, l'algorithme de dessin, ...) sont associées aux sommets et aux arêtes.

2.3.1 Différence entre sommet et voisin

Un sommet-voisin est un élément du voisinage d'un sommet dans le graphe. Afin d'éviter des confusions, une distinction entre sommet et sommet-voisin s'impose pour distinguer ces deux notions.

Un sommet-voisin est aussi un élément de V, mais il est de nature totalement différente. Il exprime une relation, un lien entre deux sommets du graphe. Ce n'est pas un sommet intrinsèque au graphe, mais une instance du sommet qu'il représente.

sommet	**sommet-voisin**
. voisinage	. lien avec le sommet représenté
. étiquette	. lien avec l'arête associée
. valuation	
. dessin	

Tableau 3.3 : Distinction entre sommet et sommet-voisin.

Cette différence se retrouve donc dans les structures de données.

2.3.2 Différence entre voisin et arête

Dans des cas simples, la distinction entre le sommet-voisin et l'arête n'a pas d'utilité d'être : une programmation sans visualisation du graphe, un graphe sans étiquetage des arêtes, une représentation d'un graphe simple par exemple.

La multiplicité des arêtes entre deux sommets n'implique pas la redondance de sommet-voisin dans le voisinage, mais bien de multiples liens entre un sommet et ses sommets-voisins. Si le sommet-voisin représente un élément du voisinage, il ne représente absolument pas l'arête. De plus, il est souhaitable de distinguer les arêtes entre elles. En effet, elles possèdent leurs propres propriétés graphiques (la forme, la taille, la couleur, l'étiquette, l'algorithme de dessin, etc.) et les arêtes sont des objets manipulables (déplacement des arêtes, modification de la représentation de l'arête, déplacement de la flèche associée à un arc, etc.).

Il ne faut pas attribuer les caractéristiques de l'arête au sommet-voisin.

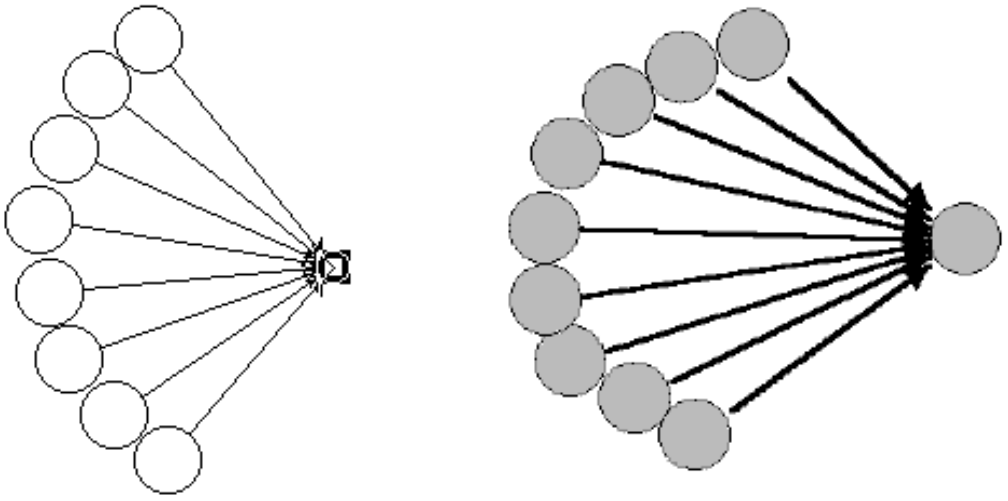
sommet-voisin		**arête**
. ensembles des successeurs et des prédécesseurs -->	. lien avec le sommet représenté . multiplicité . ensemble des arêtes associées	. sommet initial . sommet terminal . orientation . étiquette . graphisme

Tableau 3.4 : Distinction entre voisin et arête.

Afin de différencier les sommets des arêtes, nous proposons une double structure : un ensemble de sommets et un ensemble d'arêtes offrant un traitement direct sur les arêtes. Nous distinguons et nous conservons les relations dans la structure de données sur le graphe. La relation d'incidence se situe au niveau des arêtes et la relation d'adjacence au niveau du voisinage associé à chaque sommet.

2.3.3 Flèche

Pour les graphes orientés, et de façon standard, les éditeurs de graphes proposent de représenter l'orientation par une flèche à l'extrémité terminale de l'arête. Ainsi, il arrive fréquemment de ne plus distinguer des sommets, parce qu'un grand nombre d'arcs est incident à un même sommet, et que leur nombre efface alors une partie du graphe (voir la figure 3.5).



(a) dans *GraphEd* (b) dans *Graph Layout Toolkit*

Figure 3.5 : Visibilité et graphe orienté.

Il nous est donc apparu nécessaire sinon indispensable de pouvoir faire glisser cette représentation de l'orientation, c'est-à-dire la flèche, le long de la représentation de l'arête.

Evidemment, lors de la création d'un arc, une position arbitraire doit être choisie pour la représentation initiale de la flèche sur l'arc.

Très peu d'éditeurs offrent un service sur l'arête, et également sur l'orientation. Les commandes sont relatives aux modifications des attributs graphiques (la forme de la flèche, sa taille, sa couleur qui est intrinsèquement associée à celle de l'arête) et aux possibilités d'offrir plusieurs types d'étiquetage (une chaîne alphanumérique, un entier ou un réel) et plusieurs étiquettes.

2.3.4 Etiquetage

L'étiquetage d'un graphe reflète une importance primordiale, puisqu'elle est fortement associée au terme d'isomorphisme dans la théorie des graphes. Par définition, deux graphes simples $G = (V, E)$ et $G' = (V', E')$ sont isomorphes si et seulement si il existe une bijection f de V dans V' telle que :

$$\forall u \in V, \forall v \in V \; (e = (u, v) \in E \Leftrightarrow e' = (f(u), f(v)) \in E')$$

Par extension, le concept de graphe pourra désigner, soit l'objet mathématique défini par ses ensembles de sommets et d'arêtes, soit la classe d'isomorphisme d'un tel objet. Dans le premier cas, nous parlerons de graphes étiquetés, la différenciation des sommets se faisant par leur étiquette, dans le second de graphes non étiquetés.

L'étiquetage est souvent numérique et chronologique, et une étiquette est associée à un sommet ou à une arête. Le premier sommet créé possède le numéro 0. Sauf cas particulier, un sommet nouvellement créé sera numéroté par le plus petit entier non encore utilisé. Pour les arêtes, un étiquetage chronologique est aussi effectué, mais il n'est pas visualisé sauf si l'utilisateur choisit cette option.

Les étiquettes sont arbitraires, sans aucune restriction ni dans la longueur ni dans le contenu et de plusieurs natures. Elles sont numériques comme le résultat d'un calcul, Elles peuvent être alors utilisées dans des calculs d'invariants ou dans des opérations. Elles sont aussi alphanumériques pour différencier les objets associés entre eux.

Chaque étiquette possède une fonte, un style, et une taille (des données qu'il est possible de modifier). De même, l'étiquette possède une position relative par rapport au sommet ou à l'arête et elle admet un certain degré de liberté dans son déplacement.

2.4 Premières opérations sur les objets

Nous proposons dans ce paragraphe quelques opérations de bases sur les objets déjà rencontrés. Les opérations sont d'ordre structurel et graphique.

2.4.1 Les sommets

Il s'agit naturellement de la classe d'opérations la plus commune à tous les éditeurs de graphes. Les opérations suivantes sont disponibles :

- créer un sommet,
- détruire un sommet,
- relier deux sommets,
- déplacer un sommet,
- modifier les attributs d'un sommet (la représentation, l'étiquette, ...),
- ouvrir/fermer un sommet (cette opération édite l'ensemble des informations concernant un sommet ; elle n'est pas encore opérationnelle).

L'opération "détruire un sommet" ne supprime pas uniquement le sommet, mais également toutes les arêtes qui lui sont incidentes.

2.4.2 Les arêtes/arcs

Peu d'éditeurs de graphes considèrent réellement les arêtes comme des objets, surtout pour leur manipulation. Il s'agit d'une classe d'opérations aussi importante que celle sur les sommets. Les opérations suivantes sont disponibles :

- créer une arête ex nihilo, i.e. en créant en même temps les deux extrémités,
- créer une arête incidente à un sommet existant, l'autre extrémité étant créée en même temps que l'arête,
- créer une arête incidente à deux sommets existants. Cette opération n'est pas sémantiquement différente de "relier deux sommets", mais elle se distingue par son interface,
- détruire une arête,
- subdiviser une arête,
- inverser l'orientation d'un arc,
- déplacer une arête,
- modifier les attributs d'une arête (la représentation, l'étiquette,...),
- ouvrir/fermer une arête (cette opération édite l'ensemble des informations sur une arête ; elle n'est pas encore opérationnelle).

Les opérations "inverser l'orientation d'un arc", "subdiviser une arête" et "détruire une arête" doivent s'appliquer uniquement à l'arête sélectionnée et non, dans le cas de multiplicité, à l'ensemble des arêtes entre les deux sommets extrémités de l'arête choisie.

2.4.3 La flèche

Les opérations suivantes sont disponibles :

- créer une flèche,
- détruire une flèche,
- déplacer une flèche le long de l'arête,
- modifier les attributs de la flèche (la forme, la taille, …),
- ouvrir/fermer une flèche (cette opération édite l'ensemble des informations sur une flèche).

Les opérations "créer une flèche " et "détruire une flèche " sont associées aux opérations de création et de suppression de l'arête associée. L'opération "cacher la flèche" n'existe pas ; elle est remplacée par la modification du graphe orienté en un graphe non orienté. Nous n'acceptons pas actuellement des graphes mixtes.

2.4.4 L'étiquette

Les opérations suivantes disponibles sont classiques pour tout environnement informatique sur des graphes étiquetés :

- créer une étiquette,
- détruire une étiquette,
- cacher une étiquette,
- déplacer une étiquette autour de l'objet associé,
- modifier les attributs d'une étiquette (police, taille, style, couleur),
- ouvrir/fermer une étiquette (cette opération édite l'ensemble des informations sur une étiquette).

L'opération "cacher une étiquette" est plutôt remplacée par les opérations "cacher les étiquettes des arêtes" et "cacher les étiquettes des sommets" afin d'obtenir une meilleure visibilité pour les graphes. Les graphes sont automatiquement étiquetés par des entiers naturels. Cet entier est le numéro de création dans l'ordre croissant des objets. Par défaut, l'étiquette est visible pour un sommet ; par contre elle est invisible pour les arêtes.

2.4.5 Le graphe

Les opérations associées à un graphe sont classiques et naturelles :

- créer un graphe,
- déplacer le graphe,
- détruire le graphe,

- ajouter un sommet,
- ajouter une arête,
- compléter le graphe,
- complémenter le graphe,
- supprimer des sommets du graphe,
- supprimer des arêtes du graphe,
- changement de nature du graphe

simple ↔ multiple
non orienté ↔ orienté
sans boucle ↔ avec boucle

Les opérations "ajouter un sommet" et "ajouter une arête" sont similaires à "créer un sommet" et "créer une arête".

Remarque
Même si l'ajout supplémentaire d'objets graphiques, tels que la flèche pour l'orientation ou l'étiquette pour les objets, complique encore un peu plus l'interface pour le concepteur, cet ajout augmente aussi la liberté de manipulation pour l'utilisateur. Nous remarquons déjà dans les opérations de base sur les éléments intrinsèques des graphes, qu'un certain nombre d'opérations ne s'effectuent pas sur un seul élément, mais sur un ensemble d'éléments. Ces relations entre éléments et ensembles sont définies dans le prochain paragraphe.

3. Le niveau relationnel

Choix des métaphores
Il est important de bien choisir les métaphores adaptées à la théorie des graphes et à l'utilisation finale du logiciel en cours de développement. Elles contribuent fortement à la modélisation d'un environnement de manipulation graphique d'objets abstraits [Laborde 85]. Les objets manipulés par un éditeur de graphes sont de trois catégories. Les objets dits intrinsèques sont relatifs au domaine de la théorie des graphes comme les sommets, les arêtes, ou les graphes. Les objets graphiques sont des instances des objets intrinsèques, auxquels il faut ajouter en supplément d'autres objets comme l'orientation ou la flèche d'une arête, ou les étiquettes. Et les objets de l'interface (les menus, les fenêtres, les ascenseurs, ...) offrent un environnement pour éditer les graphes.

L'identification des métaphores ne repose pas exclusivement sur les objets du domaine sur lesquels les actions directes de l'utilisateur vont être réalisées, mais elle s'appuie aussi sur les représentations des objets. De plus, le travail sur un graphe s'accompagne souvent de la notion

de sélection courante de sommets (un sous-graphe induit) et de sélection courante d'arêtes (un sous-graphe partiel). Ces deux objets particuliers, le sous-graphe et le sous-graphe partiel, sont aussi des graphes par définition.

3.1 Attribut de dépendance

Les attributs de dépendance sont associés aux relations de construction d'un objet de la figure avec les objets déjà existants. Dans la théorie des graphes, l'arête incidente à ses extrémités (les sommets) est le seul élément correspondant à ces contraintes. L'étiquette d'un sommet ou d'une arête et la flèche sur une arête sont des éléments graphiques fortement attachés aux objets associés. Le déplacement de l'objet entraîne le déplacement de ses éléments graphiques . Nous représentons le graphe dans le plan. Aucune des contraintes géométriques et graphiques de l'espace n'est présente, comme le plan de projection ou la gestion des objets cachés par exemple.

Aux objets intrinsèques déjà traités, il faut ajouter deux autres éléments importants : un sous-ensemble de sommets, appelé la sélection courante de sommets, et un sous-ensemble d'arêtes, appelé la sélection courante d'arêtes. Ces deux sous-ensembles permettent l'utilisation de la notion de sous-graphe induit et de sous-graphe partiel.

3.1.1 Ensemble de sommets

Les opérations de base sur la sélection courante de sommets ou le sous-graphe induit par cette sélection sont :

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

Comme pour les opérations des autres objets intrinsèques, ces opérations sont d'ordre graphique, sauf "vider la sélection". Pour des raisons pratiques, nous n'avons pas mis les opérations internes d'un ensemble. Ils sont donnés dans le dernier paragraphe.

3.1.2 Ensemble d'arêtes

Les opérations de base sur la sélection d'arêtes ou le sous-graphe partiel induit par cette sélection sont similaires aux fonctionnalités associées au sous-graphe :

- initialiser la sélection à une arête,
- initialiser la sélection à un ensemble d'arêtes inscrit dans un rectangle,
- différence symétrique entre la sélection et un singleton,
- différence symétrique entre la sélection et un ensemble d'arêtes inscrit dans un rectangle,
- complémenter la sélection par rapport à l'ensemble des arêtes,
- sélectionner toutes les arêtes,
- vider la sélection,
- détruire la sélection d'arêtes,
- subdiviser les arêtes de la sélection,
- inverser l'orientation des arcs de la sélection,
- expliciter le sous-graphe partiel d'une sélection d'arêtes

Nous considérons les mêmes remarques que pour un sous-graphe. Aucune distinction n'est effectuée entre un ensemble d'arêtes ou un ensemble d'arcs pour la plupart des fonctionnalités.

3.2 Attribut de représentation

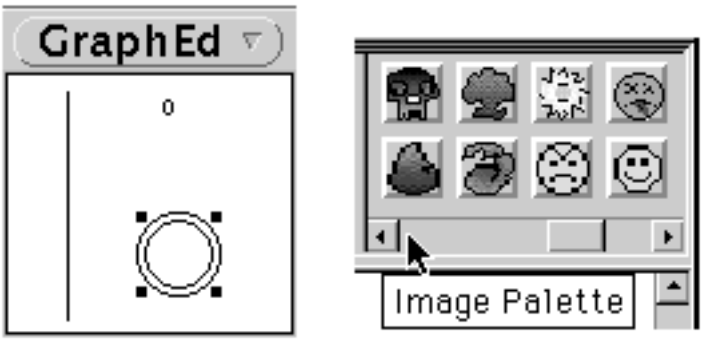
Nous analysons les différentes représentations des objets, en particulier l'interface entre le sommet et l'arête. Nous proposons plusieurs types de manipulation des objets. Les créations des objets, et plus spécialement de l'étiquette associée aux sommets, sont examinées. Des méthodes de mises en évidence d'une sélection d'objets sont présentées. Pour terminer, nous décrivons diverses méthodes pour déplacer les objets du domaine (une sélection de sommets ou un sous-graphe induit, la représentation d'une arête, la flèche représentant l'orientation d'un arc et l'étiquette associée aux objets).

3.2.1 Représentation des objets

Nous définissons les diverses représentations des objets dans un graphe, en prenant des exemples des environnements informatiques du chapitre précédent. Nous examinons le problème de l'interface entre le sommet et l'arête.

Sommet

La représentation du sommet est généralement une forme géométrique relativement simple (un carré ou un cercle) mais sans excès, contrairement à *Graph Drawing Server* en proposant des icônes fantaisistes (voir la figure 3.6 (b)). Pour admettre des icônes particulières, *da Vinci* offre un service particulier : permettre à l'utilisateur d'afficher une image de son choix à l'intérieur d'une forme géométrique de base.



(a) *GraphEd.* (b) *Graph Drawing Server.*

Figure 3.6 : Différentes représentations des sommet.

Une taille minimale de l'objet géométrique est nécessaire et doit être suffisamment grande, par exemple de 5 pixels, afin que le sommet représenté soit effectivement visible en tant qu'objet, et permettre une différence entre l'état "visible" ou "invisible" pour le sommet. Dans *GraphEd*, la taille minimale n'existe pas, un sommet peut même ne pas avoir de forme pour le représenter. Avec une faible taille, la représentation du sommet semble disparaître, et il est difficilement accessible pour une sélection simple et directe à la souris : l'exemple du trait vertical ou du "0" dans la figure 3.6 (a). Par contre, l'objet géométrique possède une largeur et une longueur propre, ce qui permet une grande variété de représentation possible. L'ajout de la couleur favorise aussi la différenciation entre les sommets.

Arête

Une arête est souvent représentée par un segment, parce qu'un minimum d'information est requis pour le dessin. Plusieurs représentations sont indispensables pour les diverses représentations des graphes. Les segments brisés interviennent pour les représentations orthogonales ou hiérarchiques, et les courbes sont disponibles pour éviter cette cassure des segments brisés. Des exemples de représentation des arêtes apparaissent dans les illustrations des environnements informatiques du chapitre précédent. Pour la représentation des graphes multiples, et donc des arêtes multiples, la représentation par segment brisé ou par courbe est indispensable.

Afin d'associer d'autres informations sur la représentation de la forme, différentes formes de pointillés sont utilisées, en plus de la couleur et de sa taille.

Flèche

La représentation d'une flèche indiquant l'orientation dépend de l'inclinaison des bords et du rattachement de son bord à l'arête (voir la figure 3.7). Seul *GraphEd* propose le paramètre de l'inclinaison.

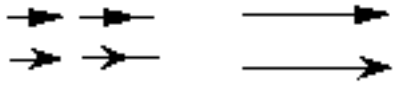


Figure 3.7 : Différentes formes pour la flèche.

La taille de la flèche est en fonction de la taille de l'arête et elle évolue entre deux valeurs à déterminer (un minimum et un maximum), afin d'illustrer l'orientation de l'arc dans un graphe. Cette information pertinente ne doit pas pour autant cacher par sa taille les autres éléments du graphe.

Interface sommet/arête

La taille de l'arête et du sommet devrait être en corrélation afin d'éviter de proposer une figure en forme de meccano entre les objets. La taille de l'arête ne doit pas être plus grande que celle du sommet, sinon il est difficile de représenter le sommet afin qu'il reste visible. Choisir une couleur différente pour les arêtes et les sommets (voir la figure 3.8) entraîne des contraintes dans la liberté des couleurs associées aux objets.

Figure 3.8 : Relation entre les tailles des arêtes et des sommets.

Notre domaine d'étude est la théorie des graphes, par conséquent les sommets sont aussi importants que les arêtes pour la visualisation des graphes. Une solution est d'imposer la taille du sommet comme une contrainte : sa valeur devient un maximum pour la taille des arêtes. Une limite plus faible est aussi possible, par exemple, la taille maximale d'une arête est au plus la moitié de la taille des sommets incidents. Ceci favorise la lisibilité du graphe.

Une autre solution consiste à enfermer la représentation du sommet dans une boîte et à dessiner l'arête à partir du bord de cette boîte. Cette solution a été adoptée par *Graph Layout Toolkit* (figure 3.9) ; chaque sommet est dessiné à l'intérieur d'un rectangle invisible, afin de gérer l'interface entre les sommets et les arêtes. *GraphEd* (voir la figure 3.10) propose même plusieurs types d'interface entre le sommet et l'arête.

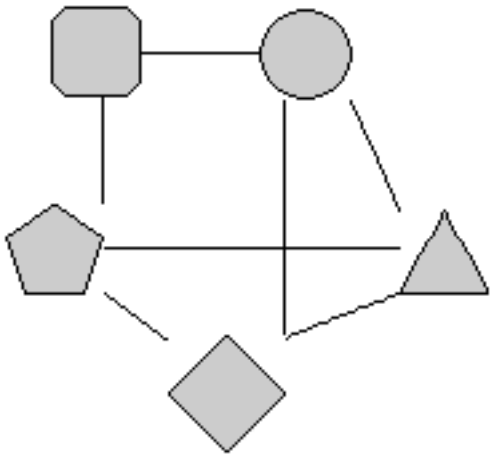


Figure 3.9 : Interface sommet/arête dans *Graph Layout Toolkit.*

Le choix de la boîte apporte aussi un début de solution aux problèmes des arcs dans les graphes orientés. En effet, si l'objet "flèche" ne peut être déplacé, il convient d'offrir une possibilité de mettre en évidence un sommet ayant beaucoup d'arcs incidents. Un autre atout offert par ce type de représentation est la facilité de dessiner des représentations des sommets et d'utiliser des algorithmes de dessin automatique.

L'inconvénient majeur est d'obtenir un dessin déséquilibré entre les sommets, et de ne plus rendre immédiatement apparentes les arêtes incidentes à un sommet.

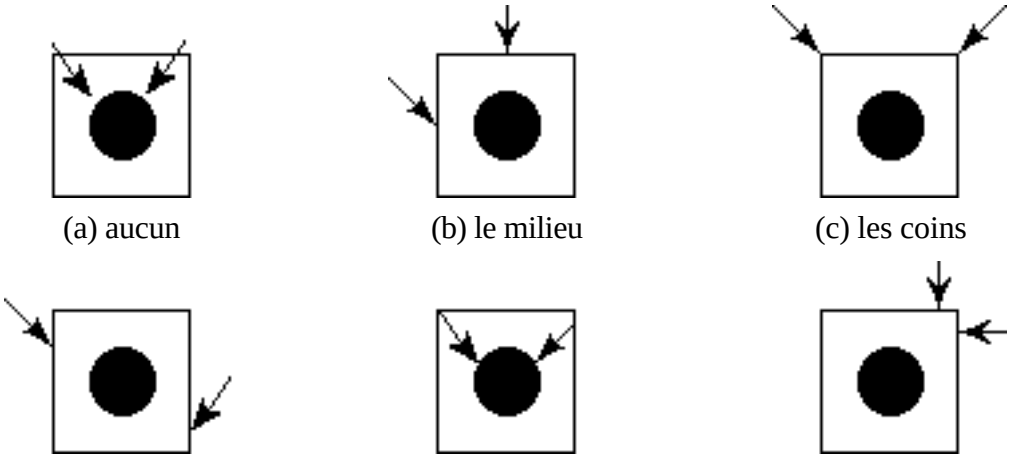


Figure 3.10 : Les six possibilités pour l'interface entre les sommets et les arêtes dans *GraphEd.*

Les différentes interfaces entre les sommets et les arêtes trouvent leurs utilités dans les algorithmes de dessin automatique et dans des représentations particulières des graphes.

Etiquette

Les étiquettes sont créées sans aucune restriction ni dans leur longueur ni dans leur contenu. Elles possèdent une fonte, un style et une taille arbitraire. Pour la plupart des environnements,

un étiquetage automatique, des sommets ou des arêtes, est proposé, généralement par des entiers et par ordre chronologique de création.

Plusieurs algorithmes sur les graphes nécessitent plus d'une étiquette sur les arêtes. Les calculs de flots ou la résolution d'un diagramme de Pert en sont des exemples. *Metanet* associe à chaque arête six étiquettes par des nombres, ce qui est suffisant pour les algorithmes existant, en plus d'une étiquette alphanumérique standard.

3.2.2 Manipulation des objets

Dans ce paragraphe, nous présentons la manipulation des objets par manipulation directe, pour les actions suivantes : la création, la sélection, le déplacement.

La création

Sommet

La création d'un sommet est simple et il s'effectue par un clic de la souris.

Aucune création de sommet n'est permise sur l'espace réservé par un sommet déjà existant ; de même aucune représentation de sommets n'en chevauche une autre. Dans le cas contraire, une gestion des ambiguïtés est nécessaire lors d'une sélection d'objets.

Arête

Cabri-graphes est le seul environnement informatique testé qui offre la possibilité de créer une arête ex nihilo. L'arête et ses deux extrémités sont créées en même temps s'ils n'existent pas, ou la construction d'une arête incidente à un sommet existant entraîne la création de l'autre extrémité inexistante en même temps que l'arête. La seule alternative proposée par les autres éditeurs de graphes est la création d'une arête incidente à deux sommets déjà existants.

Le premier clic de la souris désigne ou crée l'extrémité initiale de l'arête. Lors du déplacement de la souris, une nouvelle arête apparaît et se dessine en suivant le déplacement du curseur. L'extrémité du curseur est considérée comme l'extrémité terminale de l'arête tant qu'un deuxième sommet n'est pas choisi ou créé. Il faut éviter de désigner uniquement les sommets et dessiner ensuite l'arête, comme dans *Metanet,* sauf peut être dans le cas des arêtes réflexives. Le dessin de l'arête doit s'effectuer immédiatement après le choix du premier sommet et lors du mouvement de la souris.

Ce procédé construit le segment comme représentation de l'arête (*Graph Layout Toolkit* ou *Link),* alors que les arêtes ont la possibilité d'être modifiées en un segment brisé ou une courbe. Pour obtenir un segment brisé ou une courbe, des actions supplémentaires sont nécessaires. Les options possibles sont limitées. L'utilisateur appuie sur une touche modificatrice ou sur un bouton de la souris, exemple de *GraphEd* ou de *Graph Layout Server,* lors du tracé de l'arête, pour construire et dessiner un point de contrôle à l'emplacement du curseur.

Flèche

L'emplacement par défaut de la flèche, lors de la création d'un arc, se situe à l'extrémité terminale de l'arc (le choix adopté par la majorité des environnements informatiques), ou au milieu de la représentation de l'arête, le milieu du segment ou de la courbe (exemple de *Metanet),* ou encore sur un point de contrôle pour un segment brisé ou une courbe.

Pour pallier le problème de mobilité de la flèche, d'autres options sont proposées à l'utilisateur. La figure 3.10 illustre différentes approches pour bien "séparer" les flèches. Ces six possibilités écartent provisoirement l'inconvénient de beaucoup d'arcs incidents à un même sommet, comme le montrait la figure 3.5 (a).

Etiquette

Les étiquettes sont placées automatiquement, et elles peuvent être déplacées. Pour la plupart des environnements, un étiquetage automatique des sommets et des arêtes est proposé. Il est composé généralement par des entiers, et il progresse par ordre chronologique de création des objets.

Par défaut, il existe neuf possibilités initiales pour placer une étiquette à l'objet associé (voir la figure 3.11). Pour un sommet, *GraphEd* propose cinq choix : les quatre coins d'un carré et l'intérieur. Le problème de la position "intérieur du sommet" est pour la modification de l'étiquette. Sans une gestion des ambiguïtés de la sélection, elle doit être interdite.

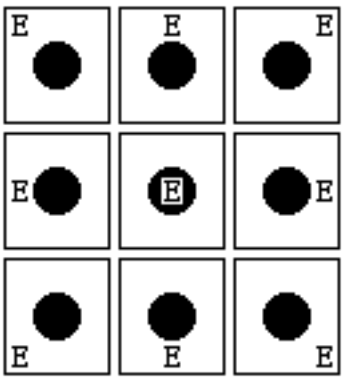


Figure 3.11 : Neuf choix initiaux pour placer une étiquette.

L'option "à l'intérieur" pour la représentation de l'étiquette du sommet ne doit pas être restrictive dans le sens où la taille de la représentation du sommet ne doit pas dépendre de l'étiquette associée, comme dans *da Vinci* (voir figure 2.9).

La figure 3.12 illustre l'étiquetage des sommets dans *Graph Layout Toolkit.* L'étiquette des sommets est placée à l'intérieur de l'objet représentant le sommet, mais sans contrainte pour la taille du sommet. La convention adoptée est d'afficher le texte de l'étiquette en fonction de la taille du sommet. Nous avons aussi une perte d'information sur l'étiquette si la taille du sommet est trop petite.

Figure 3.12 : Perte d'informations pour le texte de l'étiquette.

L'étiquetage des sommets à l'intérieur de l'objet entraîne une perte d'information sur l'étiquette ou une importance exagérée du sommet, même si ce dernier est pertinent. Or l'aspect visuel est très important, puisque l'œil est attiré par ce qui sort de l'ordinaire. Une perte de visibilité est aussi toujours possible pour l'ensemble du graphe : une mise en évidence d'un élément non pertinent par sa taille ou des difficultés à sélectionner entre sommet et étiquette.

Pour une arête, l'emplacement de l'étiquette est le milieu de la représentation de l'arête, pour un segment ou une courbe, exemple de *Graph Layout Toolkit*, ou à proximité d'un point de contrôle pour un segment brisé, exemple de *Link* (voir la figure 2.18 (a)).

La sélection

Les objets peuvent être sélectionnés grâce aux menus, par l'intermédiaire d'un rectangle de sélection, par un simple clic à proximité des objets, par vérification d'une propriété, etc. Le choix est très large.

Un problème de la sélection est la mise en évidence des objets sélectionnés. Plusieurs alternatives sont possibles, et elles dépendent du type d'objets sélectionnés. L'apparence des sommets sélectionnés apparaît de différentes façons. Le sommet est colorié en noir (exemple dans *Metanet)* ou il est dessiné avec la couleur associée mais en plus foncée (exemple dans *Link).* Une inversion du noir et blanc est utilisée avec des moniteurs noirs et blancs (ou monochrome), ou par un effet de surbrillance avec les couleurs. L'encadrement du sommet est noirci (exemple dans *GraphEd,* voir la figure 3.13), ou une ombre est ajoutée à la représentation du sommet (voir la figure 2.9 de *da Vinci).*

Le déplacement

Dans une interface de manipulation directe, les déplacements des objets du domaine sont continus, et immédiatement visibles à l'écran.

Sommet et ensemble de sommet

Un ensemble de sommets sélectionnés correspond au concept mathématique du sous-graphe induit. Déplacer une sélection de sommets équivaut donc à déplacer le sous-graphe induit. Ainsi, les arêtes incidentes sont transformées et les arêtes du sous-graphe sont translatées en conséquence lorsque le sous-graphe est déplacé.

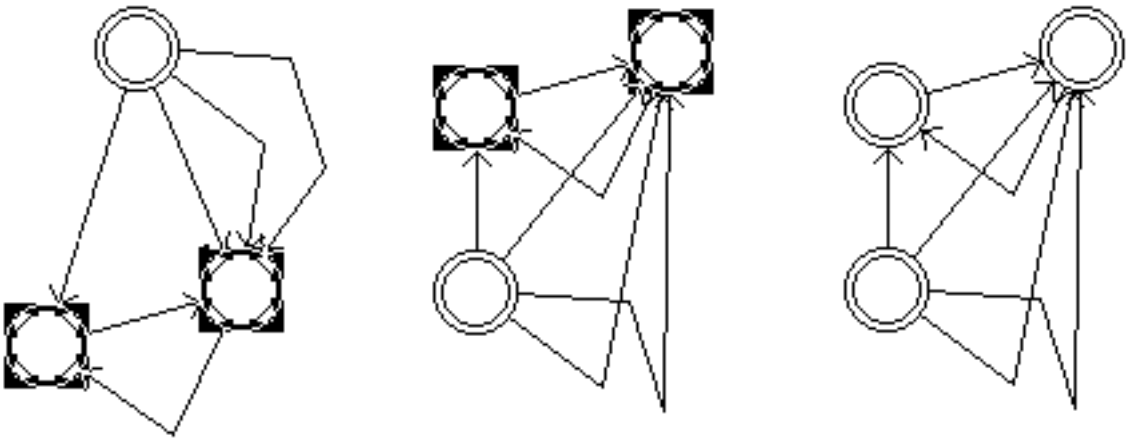


Figure 3.13 : Déplacement d'une sélection de sommets, exemple dans *GraphEd*

Arête

Il existe trois types de déplacement ou de manipulation d'une arête : par l'intermédiaire d'un sommet incident à l'arête (paragraphe précédent), par un point de contrôle de la représentation de l'arête (pour un segment brisé ou pour une courbe), et par un point quelconque de l'arête.

• *Déplacer un point de contrôle*

Après une sélection de l'arête, les points de contrôle de sa représentation apparaissent. Déplacer un des points de contrôle modifie la forme de l'arête. Un exemple est donné par la figure 2.19 sur l'environnement informatique *GraphEd.*

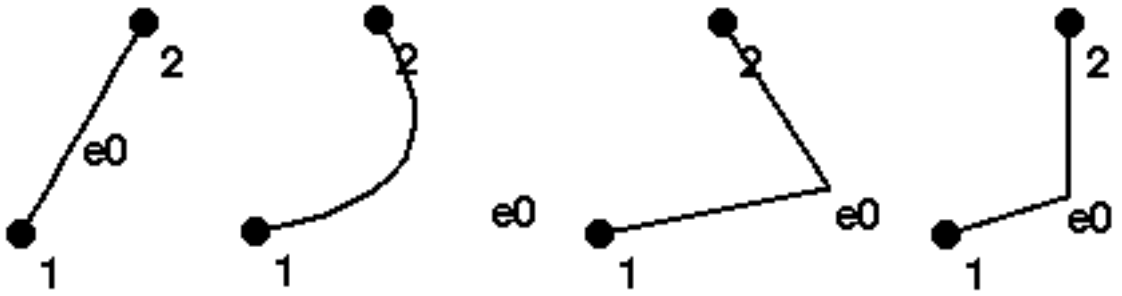


Figure 3.14 : Déplacement d'un point de contrôle d'une arête, exemple dans *Link*.

Un choix à éviter est d'utiliser l'étiquette comme moyen pour déformer l'arête, comme le cas de *Link.* Les arêtes courbes sont des courbes de Bézier quadratiques. Pour modifier leur représentation, l'étiquette (e0 dans notre exemple de la figure 3.14) est associée au point de contrôle de la courbe. Le déplacer est synonyme de déformation de l'arête. De plus, l'exemple de la figure 3.14, montre quelques inconvénients dont l'éloignement de l'étiquette par rapport à l'arête avec un risque plus grand de ne plus associer une arête et son étiquette.

• *Déplacer un point quelconque de l'arête*

Pour un segment brisé, il s'agit d'ajouter un point supplémentaire particulier dans l'ensemble des points de contrôle. *GraphEd* permet uniquement de déplacer les points de contrôle, même si le segment est brisé. Deux nouveaux points apparaissent alors (voir la figure 2.14 dans le chapitre 2).

Pour une courbe, il s'agit soit d'un déplacement standard, soit d'ajouter aussi un point de contrôle et agrandir ainsi le nombre de points définissant la courbe. Nous avons donc deux solutions : déformer la courbe existante ou la modifier en ajoutant un point.

Le choix de la représentation de l'arête et la manière de la déformer doivent avoir un minimum d'interférence entre elles. Il faut que le passage entre les différents types de représentation soit le plus simple possible, par exemple à l'aide d'un menu pour revenir à un segment en partant d'un segment brisé ou d'une courbe.

flèche

Déplacer la flèche sur une arête équivaut à déplacer un point sur l'arête. Une méthode classique consiste à décomposer l'arête en une succession de segments. Ensuite, la position de la souris est projetée orthogonalement sur les segments construits précédemment. On choisit la distance la plus courte entre l'emplacement de la souris et de son projeté. La flèche est ainsi dessinée sur sa nouvelle position. La flèche suit alors le mouvement de la souris.

Cette flèche se déplace le long de la droite ou de la courbe. Sa position, définie par un réel t compris entre 0 et 1, est un élément graphique de l'arête.

Etiquette

L'étiquette est associée aux objets par un effet de magnétisme, interdisant à cette dernière de se trouver trop éloignée de son centre d'attraction, et de refléter par conséquent une information fausse puisqu'elle est associée à un autre objet. Il faut remarquer que l'utilisation de cet aimant ne diminue que l'étendue du problème sans le résoudre pour des objets relativement proches entre eux. Le déplacement de l'étiquette n'est pas stoppé par le bord d'une figure géométrique entourant l'objet, mais il glisse le long de ce bord en suivant le déplacement du curseur. L'étiquette est affichée au point de projection orthogonale du curseur sur le bord de la figure.

Deux options supplémentaires sont à prendre en compte. La première option est de permettre à l'utilisateur de modifier cette attraction. La seconde option est d'autoriser sous des conditions particulières (initialisées dans un fichier préférence, en utilisant une touche modificatrice, ou par l'intermédiaire d'un article dans un menu) l'éloignement de l'étiquette en dehors de la zone définie pour les éléments.

• *pour un sommet*

L'étiquette associée à un sommet ne peut se mouvoir qu'à l'intérieur d'un disque de centre le sommet (en fait, le centre de la représentation du sommet) et de rayon un entier défini au préalable.

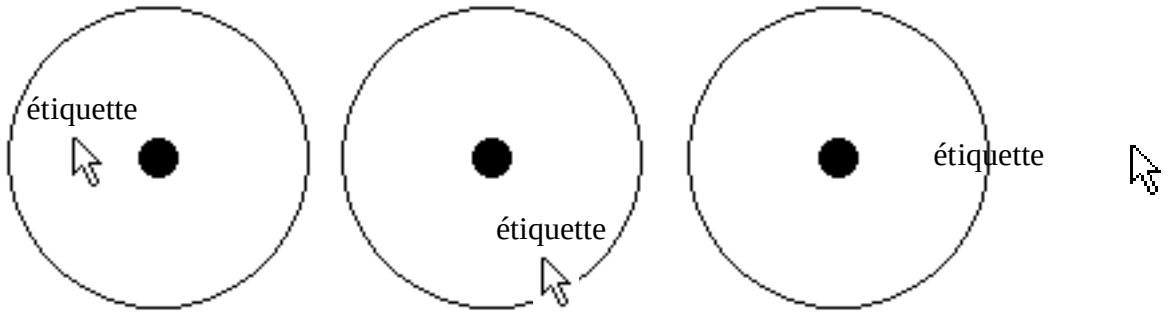


Figure 3.15 : Déplacement de l'étiquette pour un sommet.

• *pour une arête*

L'étiquette d'une arête représentée par un segment ne peut se mouvoir qu'à l'intérieur d'un cylindre. Son axe est le segment brisé représentant l'arête et son rayon est un entier défini au préalable. La courbe est décomposée en plusieurs segments. La méthode utilisée est similaire au déplacement d'une flèche sur l'arête. Contrairement à la flèche, l'étiquette est déplacée à l'intérieur d'une bande. La distance maximale entre la courbe et l'étiquette est déterminée un entier défini auparavant.

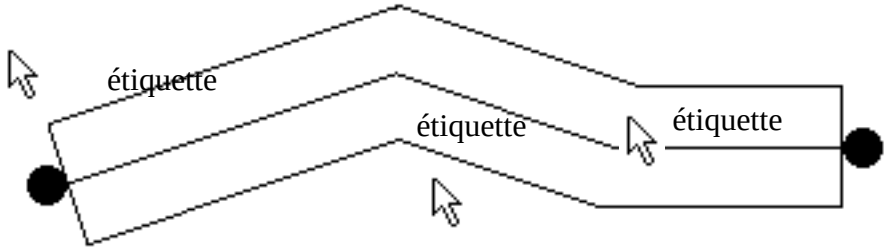


Figure 3.16 : Déplacement de l'étiquette associée à une arête à l'intétrieur d'une bande entourant le segment brisé.

Un autre problème intervient lors du déplacement de l'étiquette. Le texte de l'étiquette doit être contenu strictement à l'intérieur de la bande de déplacements, ou chevaucher légèrement le bord. Pour une gestion plus pratique, il faut définir quelle partie du texte, le début, la fin ou le milieu du texte (comme dans les figures 3.15 et 3.16), doit servir de limite.

Un choix doit être imposé dans un premier temps afin de favoriser la libre exploration de l'environnement informatique. Ensuite, et pour des utilisateurs confirmés, d'autres solutions peuvent être proposées grâce à un fichier préférence ou en dévoilant dans une l'aide les autres options possibles.

3.2.3 Aspects de l'interface

Différents espaces de travail

Nous esquissons dans ce paragraphe la forme générale de l'interface. Dans la fenêtre principale, nous avons décidé, évidemment, d'avoir un espace réservé au dessin. Un autre espace est utile pour communiquer facilement avec l'utilisateur. Un modèle de communication existe déjà pour les environnements de développements sur les compatibles PC (le texte en bas de la fenêtre, voir la figure 2.15 de *Graph Layout Toolkit)*. Il doit être entièrement construit sur les Macintosh et sur les stations de travail (voir la figure 2.12 sur *VCG tool)* , et il est complètement géré par le concepteur de l'environnement.

Il nous est apparu nécessaire d'afficher un message rappelant constamment le type de graphe en construction dans la fenêtre principale. Il est aussi indispensable que l'utilisateur puisse facilement modifier la structure interne du graphe, c'est-à-dire passer le plus simplement possible d'un graphe multiple à un graphe simple, d'un graphe orienté à un graphe non orienté, par exemple. Cette commande devrait être associée à chaque fenêtre et non globalement.

La figure 3.17 présente un modèle d'interface. Un cadre est réservé au dessin des graphes avec un message indiquant la nature du graphe. Des fenêtres optionnelles sont proposées. Des fenêtres, comme une barre d'outils symbolisée par des icônes, offrent des facilités à l'utilisateur (les barres d'outils dans l'éditeur de texte *Word6*).

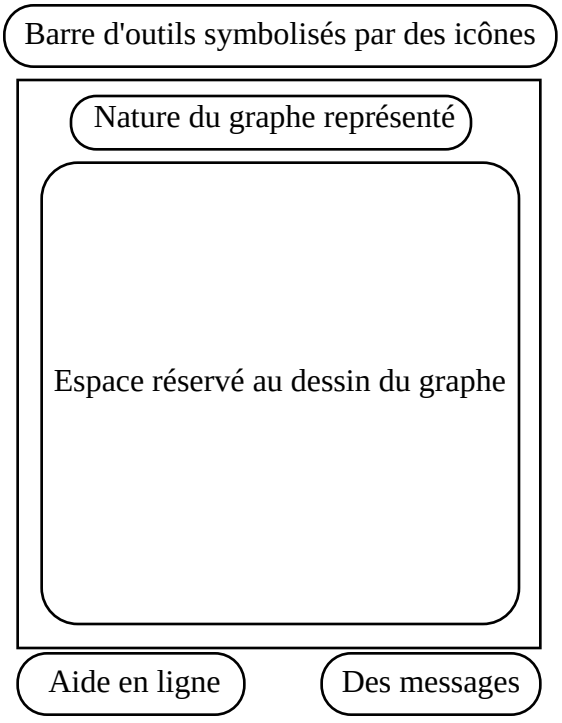


Figure 3.17 : Un modèle d'interface sur les graphes.

Il faut différencier la fenêtre de l'aide en ligne avec celle des messages utilisés par le système pour l'utilisateur. L'aide en ligne, comme son nom l'indique, renseigne de manière générale l'utilisateur sur les commandes, tandis que la boîte des messages est réservée à des retours d'information pour avertir les utilisateurs des risques d'erreur éventuelle, pour confirmer l'action de l'utilisateur, pour proposer des options sur l'opération en cours, etc.

Les retours d'information

Les retours d'information possibles, et actuellement disponibles, sont limités en nombre. Les deux domaines exploités sont les formes du curseur et les messages retournés à l'utilisateur. Les autres modes de sortie, le système audio par exemple, ne sont pas encore complètement intégrés aux systèmes.

Les différentes formes de curseur

L'icône représentant le curseur dépend à la fois du choix de l'opération à exécuter, de la position du curseur par rapport aux objets intrinsèques de la fenêtre active, de l'état des sélections courantes, de l'état des touches modificatrices ou du choix du bouton de la souris (dans le cas où la souris possède plus d'un bouton).

Prenons l'exemple de l'environnement de *Graph Layout Toolkit*. Dans l'espace réservé au dessin, nous avons deux icônes pour représenter les outils de construction :

 : pour la création d'un sommet

 : pour la création d'une arête

Il faut évidemment ne pas choisir la même icône pour séparer efficacement les différentes opérations. La flèche blanche ci-dessous est utilisée pour sélectionner les sommets et les arêtes du graphe, mais aussi pour choisir un outil dans la barre d'outils.

 : pour sélectionner un sommet ou une arête

Ces icônes renseignent déjà l'utilisateur. Un choix envisageable est une forme d'icône par famille d'action, avec peut être une légère nuance dans cette forme pour les actions au sein d'un même groupe. Le risque de la nuance dans une famille d'icône est sans doute une confusion chez l'utilisateur.

Les messages donnés à l'utilisateur

Ces messages sont simples, classiques, et de plusieurs natures :

- donner une explication d'une commande non valide ou non exécutée,
- confirmer une commande entraînant de graves conséquences pour la suite,

- demander des précisions sur la sélection en cours,
- utiliser l'aide en ligne pour apprendre les commandes de l'environnement.

D'autres sortes de messages sont encore possibles. Ils sont proposés par des boîtes de dialogue modales, par exemple la demande d'un fichier pour ouvrir ou sauvegarder un fichier de l'application, ou non modales pour les changements de formes des objets. Ces zones de dialogues peuvent aussi avoir un cadre réservé dans la fenêtre active (voir la figure 3.17), afin d'afficher pour cette fenêtre et pour le graphe représenté une information plus précise.

3.3 Les structures de données abstraites

La plupart des livres, traitant des structures de données abstraites ([Aho & al. 74] et [Froidevaux & al. 93]), séparent toujours le type abstrait graphe orienté du type abstrait graphe non orienté. Remarquons aussi l'absence de structures de données abstraites pour les graphes multiples. Contrairement à ces auteurs, nous nous proposons de concevoir une même et unique structure de données abstraites *graphe* afin de permettre le passage rapide et simple d'un type particulier de graphe à un autre, en conservant la même structure évitant ainsi des erreurs.

De plus, dans notre étude, le fait de représenter les graphes et de les manipuler directement, nous oblige à avoir aussi des structures de données abstraites indépendantes pour tous les objets du domaine (le sommet, l'arête ou l'arc, le sommet-voisin, la flèche représentant l'orientation et l'étiquetage des sommets et des arêtes).

Les avantages de l'approche des structures de données abstraites sont multiples. La conception est plus simple sans prendre en considération des détails de programmation. Les structures de données sont construites de manière définitive indépendamment de la représentation concrète choisie ultérieurement pour les types de structures abstraites.

3.3.1 Ensemble

Dans la notion d'ensemble d'éléments, la propriété importante est la présence ou l'absence d'un élément dans un ensemble particulier. Beaucoup d'algorithmes sur les graphes ne considèrent que cet aspect.

La définition et les propriétés des ensembles sont considérées connues. On veut pouvoir tester l'appartenance d'un élément à un ensemble, ajouter ou supprimer un élément, tester si cet ensemble est vide, etc. Cependant, dans certains cas, on peut être amené à considérer des répétitions d'éléments, comme dans les graphes multiples où plusieurs arêtes peuvent avoir comme extrémités les mêmes sommets.

Type : Ensemble
Utilise : Elément, Booléen, Entier
Opérations :

nouveau : → Ensemble
est_vide : Ensemble → Booléen
multiple : Ensemble → Booléen
ajouter : Ensemble × Elément → Ensemble
supprimer : Ensemble × Elément → Ensemble
appartenir : Ensemble × Elément → Booléen
cardinalité : Ensemble → Entier
occurrence_élément : Ensemble × Elément → Entier
ensemble_ occurrences : Ensemble × Elément → Ensemble

Plusieurs choix sont possibles pour le comportement des opérations "*ajouter*", dans le cas d'un élément déjà présent, et "*supprimer"* dans le cas d'un élément absent. On peut émettre un message d'erreur signifiant que les opérations "*ajouter* " et "*supprimer* " sont des opérations partielles, ou on peut considérer que ces opérations sont sans effet, ou on peut considérer les deux.

Nous avons décidé de ne construire qu'un seul type d'ensemble pouvant être multiple, mais les occurrences multiples d'un même élément sont regroupées côte à côte. Dans la définition des opérations, un test préalable est obligatoire afin de déterminer le type d'ensemble pour les opérations "*ajouter* " et "*supprimer"* un élément.

Ensemble de sommets ou sous-graphe

Les opérations sur une sélection de sommets sont :

Type : Ensemble de sommets
Utilise : Ensemble d'arêtes, Ensemble de sommets, Graphe
Opérations :

voisinage : Ensemble de sommets → Ensemble de sommets
successeurs : Ensemble de sommets → Ensemble de sommets
prédécesseurs : Ensemble de sommets → Ensemble de sommets
arêtes_incidentes : Ensemble de sommets → Ensemble d'arêtes
arêtes_entrantes : Ensemble de sommets → Ensemble d'arêtes
arêtes_sortantes : Ensemble de sommets → Ensemble d'arêtes
arêtes_internes : Ensemble de sommets → Ensemble d'arêtes
arêtes_entres_som : Ensemble de sommets × Ensemble de sommets → Ensemble d'arêtes
transformer_som_graphe : Ensemble de sommets → Graphe

Les opérations sur un ensemble de sommets sont similaires à celles sur un sommet par la recherche du voisinage de l'ensemble, de ses successeurs ou de ses prédécesseurs.

Ensemble d'arêtes ou sous-graphe partiel

Les opérations sur une sélection d'arêtes sont :

Type : Ensemble d'arêtes
Utilise : Ensemble d'arêtes, Ensemble de sommets, Graphe
Opérations :

sommets_composés	: Ensemble d'arêtes → Ensemble de sommets
arêtes_réflexives	: Ensemble d'arêtes → Ensemble d'arêtes
supprimer_multiples	: Ensemble d'arêtes → Ensemble d'arêtes
supprimer_boucles	: Ensemble d'arêtes → Ensemble d'arêtes
transformer_art_graphe	: Ensemble d'arêtes → Graphe

Les opérations sur un ensemble d'arêtes sont aussi importantes. L'ensemble des arêtes est le résultat d'opérations et de recherche sur les ensembles de sommets.

La spécification des ensembles peut être enrichie par des opérations d'union, d'intersection, de concaténation, et encore d'autres.

Pour distinguer par la suite les sommets, les sommets-voisins ou les arêtes d'un graphe, on les nomme de manière interne, soit par des chaînes de caractères, soit par des numéros. C'est cette deuxième convention qui est suivie dans la spécification ci-dessous. Chaque opération de ces objets est pour un graphe particulier qu'il faudrait préciser (_ × Graphe). Afin d'alléger la rédaction, il a été omis.

3.3.2 Sommet

Le sommet est un élément important de notre étude. Sa structure de données se décompose en trois parties. La première partie est interne, et elle représente ses relations avec les autres éléments du graphe.La seconde partie est graphique pour la représentation du sommet à l'écran. La dernière partie concerne l'étiquetage.

Type : Sommet
Utilise : Entier, Ensemble de voisins, Ensemble d'arêtes, Etiquette, Graphisme
Opérations :

nouveau	: → Sommet
récupérer	: Entier → Sommet

numéro : Sommet → Entier
degré : Sommet → Entier
nombre-de-liens : Sommet × Sommet → Entier
successeur : Sommet → Ensemble des voisins successeurs
prédécesseur : Sommet → Ensemble des voisins prédécesseurs
boucle : Sommet → Ensemble d'arêtes
étiquetage : Sommet → Etiquette
représentation : Sommet → graphisme

Dans le cas d'un graphe non orienté, l'ensemble des prédécesseurs sera un ensemble vide. L'ensemble des arêtes réflexives est conservé au niveau du sommet et il est considéré comme un ensemble particulier évitant des tests complémentaires pour les algorithmes classiques sur les graphes comme les parcours de graphes, par exemple.

3.3.3 Sommet-voisin

Le sommet-voisin, comme nous l'avons indiqué précédemment, est un élément qui permet de faire le lien entre l'ensemble des sommets et l'ensemble des arêtes. Il conserve l'information de multiplicité.

Type : Voisin
Utilise : Booléen, Ensemble d'arête, Sommet
Opérations :

nouveau : → Voisin
multiple : Voisin → Booléen
arêtes-multiples : Voisin → Ensemble d'arêtes

sommet-associé : Voisin → Sommet
arêtes-associées : Voisin → Ensemble d'arêtes

ou

sommet-associé : Ensemble × Elément → Sommet
arêtes-associées : Ensemble × Elément → ensembles d'arêtes

Le sommet-voisin permet de récupérer toutes les arêtes associées dans le cas de multiplicité. Nous avons indiqué uniquement les opérations différentes de celles associées au type ensemble. L'ensemble des voisins d'un sommet est un ensemble particulier de sommets.

3.3.4 Arête

L'arête est notre deuxième élément important dans notre étude. Sa structure de données se décompose, elle aussi, en trois parties et pour les mêmes raisons. Dans la partie graphique,

nous sauvegardons les informations relatives aux *flèches* pour l'orientation et aux représentations de l'arête sur l'écran (ses extrémités, et s'il est nécessaire, les points de contrôle d'un segment brisé ou d'une courbe).

Type : Arête
Utilise : Booléen, Entier, Etiquette, Graphisme, Sommet
Opérations :

nouveau	: → Arête
récupérer	: Entier → Arête
numéro	: Arête → Entier
sommet-initial	: Arête → Sommet
sommet-terminal	: Arête → Sommet
boucle	: Arête → Booléen
orienté	: Arête → Booléen
étiquetage	: Arête → Etiquette
représentation	: Arête → graphisme

Pour les utilisations des graphes sans visualisation, cette structure de données peut paraître superflue, ce qui est inexact qu'en partie. En plus de conserver l'information relative à sa représentation, l'arête conserve toutes les étiquettes qui lui sont associées. Les étiquettes sont composées d'une chaîne alphanumérique ou d'un nombre servant à des calculs sur les graphes.

3.3.5 Graphe

Le type "*graphe*" est notre principale structure. Un graphe orienté est un ensemble de sommets et un ensemble d'arêtes ou d'arcs (des paires ordonnées de sommets). Il y a des similarités entre ce type abstrait et celui pour les ensembles. Il y a cependant des différences. Elles sont dues essentiellement au maintien constant de la cohérence entre les deux ensembles. Ajouter ou retirer un sommet peut affecter l'ensemble des arêtes. Ajouter ou retirer une arête peut affecter l'ensemble des sommets, mais concerne principalement le voisinage des sommets qui sont les extrémités de l'arête.

Type : Graphe
Utilise : Arête, Booléen, Ensemble de sommets, Ensemble d'arêtes, Sommet
Opérations :

nouveau	: → Graphe
graphe_vide	: Graphe → Booléen
graphe-orienté	: Graphe → Booléen
graphe-multiple	: Graphe → Booléen

graphe-avec boucle : Graphe → Booléen

ajouter-un-sommet_à : Sommet × Graphe → Graphe
ajouter-une-arête_à : Arête × Graphe → Graphe
est-un-sommet-de : Sommet × Graphe → Booléen
est-une-arête-de : Arête × Graphe → Booléen
supprimer-un-sommet_à : Sommet × Graphe → Graphe
supprimer-une-arête_à : Arête × Graphe → Graphe

sommets-associés : Graphe → Ensemble de sommets
arêtes-associées : Graphe → Ensemble d'arêtes

orienté-à-non-orienté : Graphe → Graphe
non-orienté-à-orienté : Graphe → Graphe
simple-à-multiple : Graphe → Graphe
multiple-à-simple : Graphe → Graphe
avec-vers-sans-boucle : Graphe → Graphe
sans-vers-avec-boucle : Graphe → Graphe

Des conventions se sont avérées nécessaires pour les opérations d'*ajout* et de *suppression*. La première est plutôt un test qui consiste à connaître le type de graphe manipulé :

1) Quand un sommet est ajouté, il est isolé ; il n'y a aucune arête incidente.

2) Quand une arête (ou un arc) est ajoutée, le premier sommet est considéré comme l'extrémité initiale de l'arête et le second sommet comme l'extrémité terminale ; l'arête est orientée de fait, lors de sa création, par l'ordre implicite des sommets rencontrés.

3) Quand une arête est ajoutée, si les sommets incidents à cette arête n'appartiennent pas au graphe, alors ils sont ajoutés.

4) Par contre, quand une arête est supprimée, les sommets incidents ne sont pas supprimés,

5) mais quand un sommet est supprimé, toutes les arêtes incidentes sont supprimées.

En partant de ces opérations de base, on peut définir d'autres opérations qui sont souvent utilisées dans les algorithmes opérant sur les graphes. Nous avons réalisé une structure unique de donnée abstraite "*graphe* " ayant comme informations des variables booléennes permettant de connaître si le graphe possède des arêtes multiples, s'il est orienté ou s'il admet des boucles.

En conclusion, les graphes sont représentés par un ensemble de sommets et une famille d'arêtes. Pour une plus grande souplesse, le type abstrait *ensemble* proposé gère aussi bien les ensembles que les familles. Il existe trois ensembles de sommets-voisins au niveau du sommet. Le premier est l'ensemble des successeurs et le second l'ensemble des prédécesseurs. Un

troisième ensemble est présent pour les boucles. Pour un graphe non orienté, seul l'ensemble des successeurs est utilisé dans les algorithmes. Le sommet-voisin sauvegarde l'ensemble des arêtes ayant comme extrémités le sommet et son voisin. Le graphe converse aussi les propriétés de multiplicité, d'orientation et la possibilité de posséder des arêtes réflexives. Des tests supplémentaires sont donc nécessaires pour tout algorithme employant les structures de données du graphe.

Les options adoptées dans la manipulation des objets du graphe (la création, la sélection et le déplacement) et les différents aspects de l'interface de notre environnement informatique, *Cabri-graphes*, sont décrits dans le prochain paragraphe.

4. Le niveau informatique

Choix de la programmation

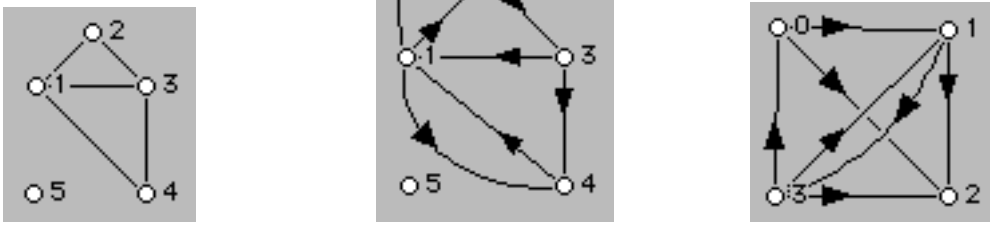
Nous avons choisi le langage C pour implémenter notre éditeur de graphes, pour des raisons de portabilité vers d'autres types de machines et avec d'autres systèmes d'exploitation. Nous décrivons l'aspect interne et externe de notre environnement.

L'aspect interne détaille la programmation des structures de données abstraites déterminées précédemment. Nous analysons les différentes représentations des graphes en machine. L'aspect externe désigne tous les attributs relatifs à notre interface de manipulation directe sur les graphes.

4.1 Attribut d'implémentation

On convient que les sommets d'un graphe G d'ordre n sont numérotés par les entiers à partir du chiffre 1 jusqu'à n. Les représentations en machine les plus courantes de G dépendent de l'énumération de la liste des extrémités des arêtes, de celle des voisins d'un sommet et de la programmation de ces listes.

On distingue deux types de représentations. La première représentation est tabulaire avec une allocation mémoire séquentielle. Les données sont stockées dans des cellules contiguës dans la mémoire, afin de permettre un accès direct par un adressage calculé. La seconde représentation s'effectue par des listes chaînées permettant une gestion dynamique des informations auxquelles l'accès est séquentiel.



(a) Un graphe simple G (b) Un graphe orienté H (c) Un graphe orienté H'

Figure 3.17 : Différents types de graphes

Nous illustrons les différentes techniques de représentation machine par l'intermédiaire des exemples de graphes de la figure 3.17.

4.1.1 Représentation tabulaire

Dans les quatre méthodes proposées, nous utilisons des matrices carrées de taille $t \times t$ ou des tableaux de taille t, t étant un entier défini au début du programme. La taille du graphe est donc limitée par la valeur de t.. Pour un graphe $G = (V, E)$ quelconque, nous avons que $|V| = n \leq t$ et que $|E| = m \leq t$. Nous donnons les places mémoires réellement prises dans chacune des méthodes.

La première méthode consiste à représenter G par sa matrice d'adjacence. Elle demande n^2 cases mémoires, et au maximum t^2 places. Cette méthode est insuffisante pour des graphes multiples. La matrice d'incidence est alors nécessaire. Cette matrice est de taille $n \times m$.

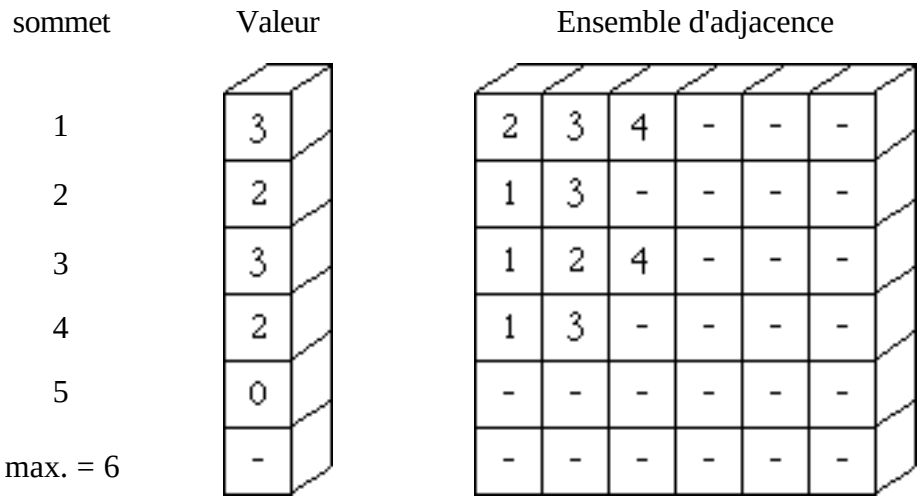


Figure 3.18 : Graphe sous forme de matrice.

La deuxième méthode consiste à remplir un tableau et une matrice carrée. Le tableau de valeur contient le nombre de sommets adjacents à un sommet donné, et la ligne correspondante

de la matrice contient l'ensemble de ses sommets-voisins. Considérons le graphe *G* de la figure 3.17(a). *G* possède cinq sommets et cinq arêtes.

Cette méthode demande n cases mémoires pour les sommets et m^2 pour les arêtes. Au maximum, on a $t + t^2$ cases mémoires disponibles, même si le nombre de cases mémoires utilisées est souvent inférieur au nombre total de places. Comme la méthode précédente, elle ne permet pas de gérer des graphes multiples.

La troisième méthode énumère l'ensemble des sommets et l'ensemble des arêtes en utilisant deux tableaux afin d'énumérer les extrémités des arêtes. Ces tableaux admettent une longueur *t* fixée aussi au début du programme. En prenant pour exemple le graphe $H = (V, E)$ de la figure 3.17 (b), nous avons que :
Le tableau S[1.. n] est pour les sommets :

S	1	2	3	4	5

Pour les arêtes nous avons les tableaux I[1 .. m] et T[1 .. m] suivant :

I[i] est le sommet initiale de l'arête a_i

T[i] est le sommet terminal de l'arête a_i

I	1	1	2	2	3	3	4
T	2	4	1	3	1	4	1

Cette représentation nécessite *n* cases mémoire pour les sommets et 2*m* cases mémoire pour les arêtes, soit au maximum 3*t* places en mémoire.

La quatrième méthode établit la table d'adjacence indiquant les successeurs ou les voisins d'un sommet. On représente l'ensemble des arcs en énumérant, pour chaque sommet *i*, l'ensemble des arêtes ayant *i* comme extrémité. Ces listes sont rangées les uns après les autres, dans l'ordre naturel des sommets qui leur sont associés, dans un tableau S de longueur *m*. Pour repérer le premier élément de chaque liste, un tableau *P* de longueur *n* est utilisé. P[i] pointe sur le premier élément de la i-ème liste. L'absence de successeur ou de voisin pour un sommet est codée par un 0 dans le tableau P.

	1	2	3	4	5
P	1	3	5	7	0

	1	2	3	4	5	6	7
S	2	4	1	3	1	4	1

Dans l'exemple du graphe *H* de la figure 3.17 (b), la table des successeurs du sommet 2 commence par l'indice 3 du tableau *S* et elle se termine avec l'indice du premier successeur du

prochain sommet dans *P*, c'est-à-dire à l'indice 7. La complexité en espace est de *n* pour les sommets et de *m* pour les arêtes, avec un maximum de 2*t* places mémoires.

4.1.2 Représentation par listes chaînées

Les représentations tabulaires ont l'avantage de la simplicité, et elles sont mises en œuvre rapidement et facilement. Par contre, elles exigent de fixer la taille maximale des graphes. Cette rigidité de l'allocation séquentielle est mal adaptée pour la manipulation des graphes, dont les opérations fréquentes sont l'insertion et la suppression d'éléments. De plus, les graphes représentés peuvent être multiples, et le nombre maximum d'arêtes entre deux sommets ne doit pas être imposé par le concepteur de l'environnement informatique.

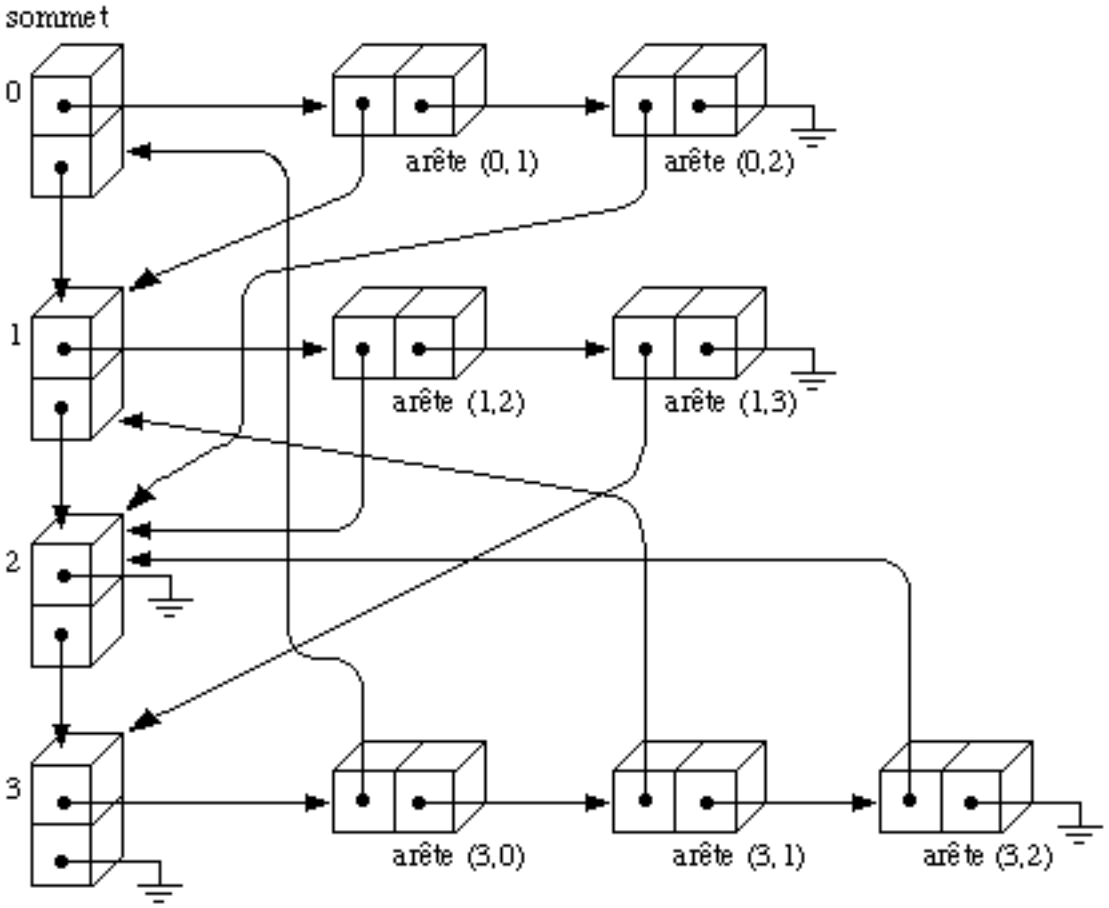


Figure 3.19 : Une implémentation par des listes chaînées simples du graphe *H'* de la figure 3.17 (c).

Pour pallier à ce principal inconvénient, on a recours à la représentation chaînée des listes où les pointeurs permettent de maintenir l'ordre des listes. Il existe plusieurs types de listes chaînées : les listes chaînées simples, les listes chaînées circulaires, les listes doublement chaînées et les listes doublement chaînées circulaires.

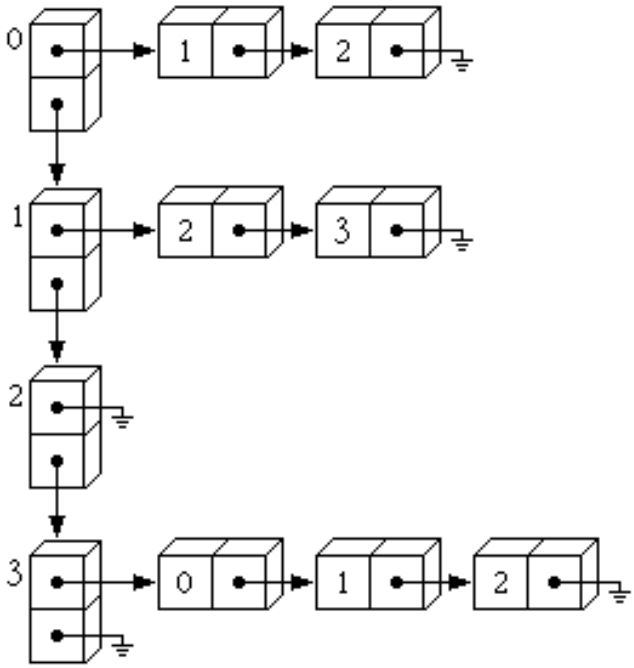


Figure 3.20 : Une autre implémentation par les listes chaînées du graphe H'.

La représentation par des listes chaînées simples nécessite *n+m* cases mémoires. Les deux exemples de représentation dans les figures 3.19 et 3.20. montrent que la place mémoire demandée pour la sauvegarde de la structure du graphe évolue constamment en fonction des ajouts et des suppressions des sommets et des arêtes.

4.1.3 Association du tableau et de la liste chaînée

Opérations sur les graphes

On peut distinguer des différences entre les deux représentations en machine si l'on considère des opérations simples sur les graphes. Une opération de base consiste à déterminer s'il existe une arête entre deux sommets. Avec une matrice d'adjacence, implémentée par des tableaux, la rechercher nécessite une complexité en temps de O(1).

Avec des listes d'adjacence, un temps de O(1) est nécessaire pour trouver l'en-tête de la liste d'adjacence d'un des sommets. Il faut alors parcourir cette liste en totalité dans le cas où l'autre extrémité de l'arête n'est pas présente. Le temps de parcours est de *n* dans le pire des cas, et de *n/2* en moyenne, si le sommet est présent dans la liste. Si le graphe est multiple et s'il possède *m* arêtes et *n* sommets, le temps de parcours est alors de *m/n* en moyenne pour une recherche. Plus un graphe est dense plus la matrice d'adjacence est préférable aux listes d'adjacence pour la recherche d'une arête.

Pour le cas orienté on a souvent besoin de trouver tous les successeurs d'un sommet donné. Avec une liste d'adjacence on accède à l'en-tête du sommet et on parcourt la liste en moyenne *m/n* fois pour trouver tous les successeurs. Avec une matrice d'adjacence, le temps de

parcours est alors de *n* quelque soit *m*, puisque seule la ligne de la matrice associée au sommet correspondant est examinée. Donc pour les graphes clairsemés, les listes d'adjacence sont bien plus rapides que les matrices d'adjacence, lorsque l'on doit examiner tous les successeurs d'un sommet donné.

Cependant, supposons que l'on cherche tous les prédécesseurs d'un sommet *v* donné. Avec une matrice d'adjacence, on peut examiner la colonne v ; cette recherche prend un temps de O(1). La représentation par des listes d'adjacence ne facilite pas cette recherche des prédécesseurs. On doit examiner la liste d'adjacence pour tout sommet *u*, pour savoir si elle contient *v*. Ainsi on peut examiner toutes les cellules de toutes les listes d'adjacence . Le temps de recherche est en *m*. Les matrices d'adjacence sont donc préférables pour les graphes denses.

Opération	Graphe dense	Graphe clairsemé
chercher un arc	matrice	matrice ou liste
Trouver les successeurs	matrice ou liste	matrice
Trouver les prédécesseurs	matrice	matrice ou liste

Tableau 3.5 : Comparaison entre un graphe dense et un graphe clairsemé en fonction du type de représentation (matrice d'adjacence ou liste d'adjacence) pour l'orientation.

La matrice d'adjacence ne considère ni la multiplicité des arêtes ni les arêtes réflexives. La matrice d'incidence est alors nécessaire pour conserver les informations associées. Un avantage des matrices par rapport aux listes chaînées est un accès direct à l'information demandée.

Combinaison du tableau et de la liste chaînée

La combinaison des structures de tableaux de longueur *k* et de listes chaînées réserve, en cas de besoin, non pas un seul élément mais un tableau de *k* éléments comme le suggère la figure 3.21.

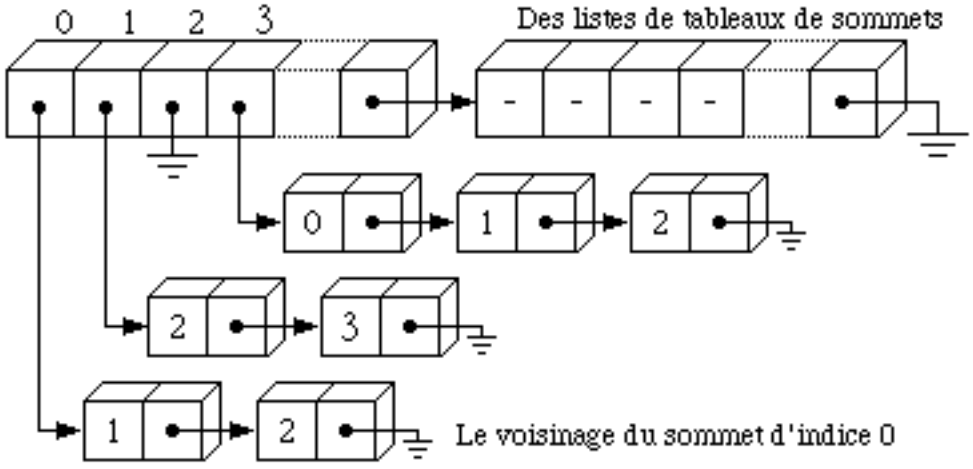


Figure 3.21 : La structure d'un graphe utilisant la combinaison des tableaux et des listes chaînées.

Les éléments des ensembles sont indexés par un entier i. Le procédé de combinaison autorise un accès rapide aux éléments. Soit deux entiers p et r tels que $i = p \times k + r$. La recherche de l'élément i s'effectue regardant l'élément d'indexé par r dans le p-ème tableau.

La combinaison des différents types de structures de données (les tableaux ou les matrices avec les listes) associe alors la rapidité dans la recherche d'information avec la réservation mémoire dynamique.

Vecteur caractéristique

Dans la plupart des algorithmes de parcours d'un graphe comme l'exploration en profondeur ou en largeur, l'important est de savoir si deux sommets sont adjacents. Par contre, dans d'autres algorithmes, il faut connaître explicitement le voisinage des sommets. Nous construisons alors pour chaque ensemble un outil appelé le vecteur caractéristique.

Le vecteur caractéristique d'un ensemble est un tableau indexé par les éléments de cet ensemble. Les valeurs appropriées du tableau sont vraies ou fausses, ce qui permet un codage avec les valeurs 0 et 1. Le vecteur est associé à chaque sommet du graphe, et comme il est représenté par un tableau, le nombre de sommets est donc fini mais très grand. Ce vecteur représente la matrice d'adjacence.

Le vecteur caractéristique est associé à tous les types d'ensemble (simple ou multiple) et pour les objets (les ensembles de sommets, les ensembles de sommets-voisins, les ensembles d'arêtes). L'utilisation du vecteur caractéristique facilite les parcours dans les graphes. Les structures de listes chaînées définies précédemment sont présentes afin d'obtenir l'ensemble du voisinage et de ses propriétés.

4.1.4 Structures de données

La structure de données de *Cabri-graphes* est présentée dans la figure 3.22.

Nous donnons un aperçu de la structure de données. Pour les ensembles de successeurs et de prédécesseurs, seul le premier ensemble a été représenté afin de ne pas alourdir la figure (le second ayant une représentation similaire). De même, nous n'avons pas détaillé les informations relatives à l'étiquetage et les éléments graphiques permettant de dessiner un sommet ou une arête.

Il faut bien faire la différence entre un algorithme sur le papier et la programmation de l'algorithme. La complexité, en mémoire ou en temps d'un algorithme, a toutes les chances de s'accroître si la structure de données est mal adaptée au problème posé.

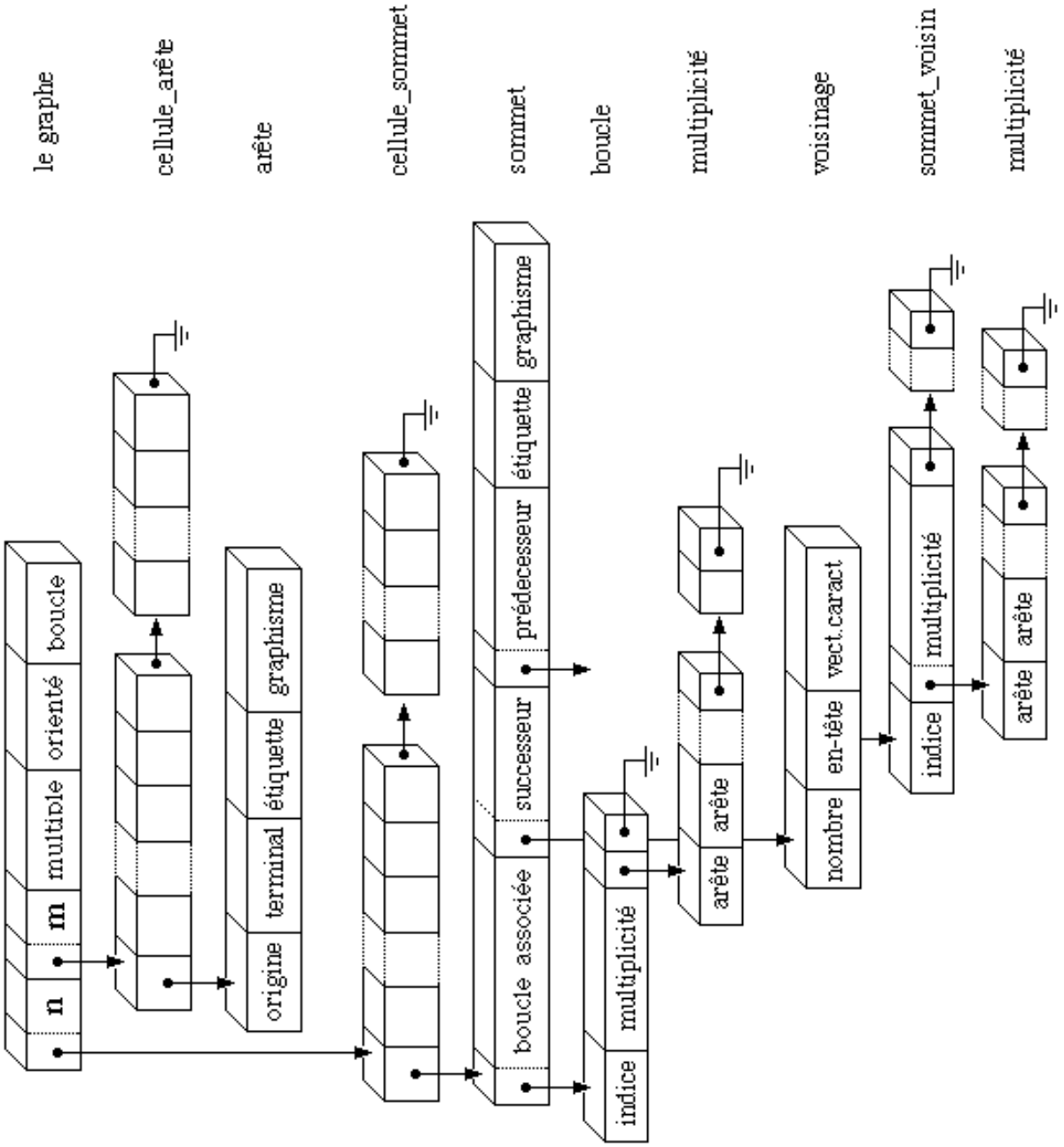


Figure 3.22 : Structure de données de *Cabri-graphes*.

Simplification dans les opérations

Nous proposons de simplifier quelques opérations de bases sur les objets déjà mentionnés.

Opérations sur les sommets

Les opérations "détruire un sommet" et "fusionner deux sommets" n'existent pas directement dans *Cabri-graphes.* Elles sont remplacées respectivement par des opérations plus globales : "détruire le sous-graphe" et "fusionner le sous-graphe".

Opérations sur les arêtes/arcs

Les opérations "inverser l'orientation d'un arc", "subdiviser une arête" et "détruire une arête" n'existent pas directement dans *Cabri-graphes.* Elles sont remplacées respectivement par "inverser l'orientation d'un sous-graphe partiel", "subdiviser le sous-graphe partiel" et "détruire les arêtes du sous-graphe partiel".

Remarque

Dans notre environnement *Cabri-graphes,* nous avons aussi offert à l'utilisateur la possibilité de manipuler plusieurs ensembles de sommets ou plusieurs ensembles d'arêtes. Cette capacité complique légèrement la gestion des ensembles pour toute suppression d'un élément vis-à-vis des autres ensembles, mais elle apporte une souplesse dans le maniement des ensembles d'objets sélectionnés et elle fournit une plus grande liberté d'utilisation. Une étude rigoureuse est indispensable pour une gestion propre.

4.2 Attribut visuel

Après l'aspect théorique, l'expérimentation nous semble absolument nécessaire pour permettre de définir correctement l'interface, pour l'affiner et continuer son développement et son évolution.

Cabri-graphes a été programmé sur Macintosh. Sauf dans certains cas particuliers, les solutions proposées sont communes aux environnements informatiques modernes. Dans le dernier paragraphe, nous décrivons de manière presque exhaustive *Cabri-graphes*.

Quelques aspects des interfaces

Les interfaces graphiques utilisateur dispose comme outil de désignation une souris munie d'un bouton ou de plusieurs[2] et qu'il possède un clavier comportant des touches modificatrices. Ces touches de clavier sont au nombre de quatre. Ils ne créent pas d'événements, mais leur état, enfoncé ou relâché, peut être testé à tout moment. *Cabri-graphes* n'utilise que les touches modificatrices présentes sur tous les claviers récents : la touche *commande* représentée par le symbole ⌘, la touche *option* représentée par le symbole ⌥, la touche *contrôle* représentée par les lettres ***ctrl*** et la touche *majuscule* représentée par le symbole ⇧.

Les environnements informatiques, respectant les conventions standards sur les interfaces, utilisent tous des menus déroulants. Afin d'améliorer l'ergonomie de ces environnements, il est

[2] La souris du Macintosh ne possède qu'un seul bouton. Sur un compatible PC, la différenciation entre les deux boutons de la souris conduit à d'autres choix dans la gestion des boutons et des touches modificatrices; un

possible d'appeler une commande présente dans un menu par un raccourci-clavier en appuyant conjointement sur la touche commande et une touche caractère. Le caractère à utiliser est précisé dans le menu correspondant.

4.2.1 Les aspects de l'interface

Avant de présenter une analyse par cas de l'interface de manipulation directe sur les graphes, nous exposons l'aspect général de l'interface et nous examinons les outils associés (voir les figures 3.23, 3.24 et 3.25).

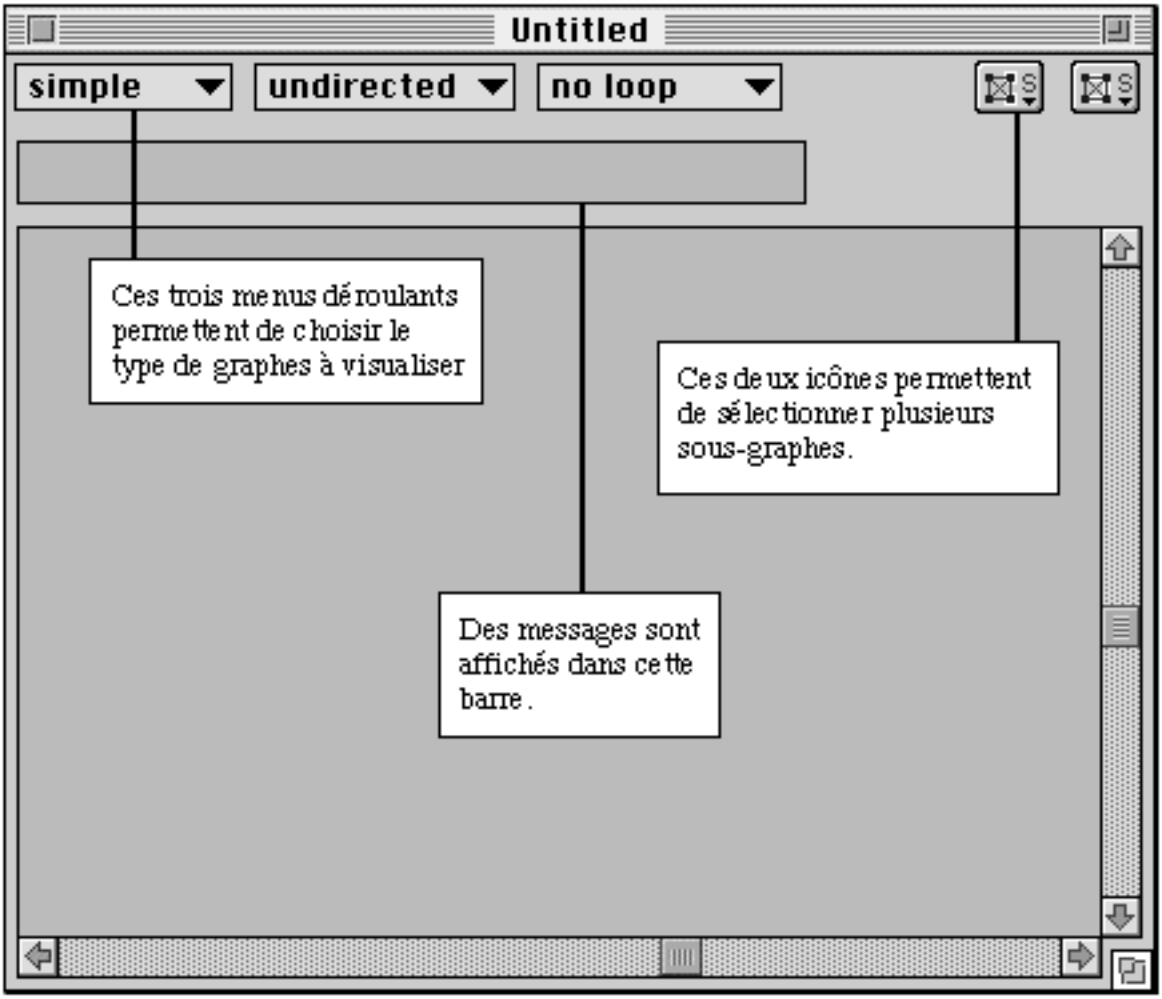


Figure 3.23 : L'interface de Cabri-graphes au démarrage : le graphe est simple, non orienté et sans boucle.

L'utilisateur modifie facilement la structure interne du graphe, c'est-à-dire qu'il passe le plus simplement d'un graphe multiple à un graphe simple, d'un graphe orienté à un graphe non

---

exemple est donné par *Graph Layout Toolkit* . De même, sur une station de travail, la souris avec trois boutons autorise une gestion différente, exemple de *GraphEd*.

orienté, par exemple. Cette commande devrait être associée à chaque fenêtre, et non globalement comme un article d'un menu. Un message rappelant constamment le type de graphe en construction est donc aussi présenté (voir les figures 3.23 et 3.24).

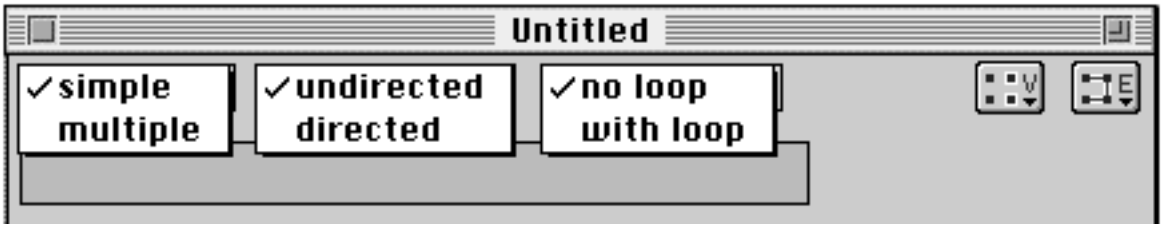


Figure 3.24 : Les menus déroulants permettent de changer la nature structurelle du graphe : simple ↔ multiple, orienté ↔ non orienté, avec boucle ↔ sans boucle.

L'interface de gestion de plusieurs ensembles de sommets et d'arêtes s'effectue par l'intermédiaire d'un bouton associé à un menu déroulant.

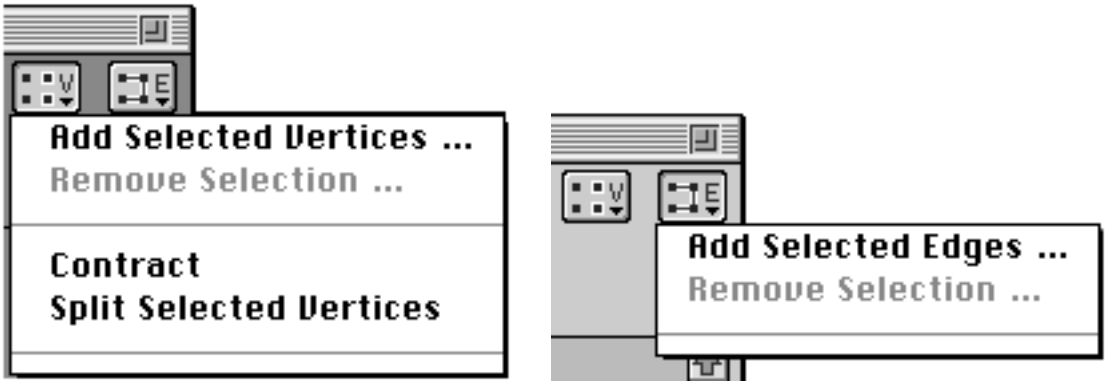


Figure 3.25 : Les boutons-menus d'ajout et de suppression d'ensembles de sommets et d'arêtes.

Multiplicité des interfaces

Les opérations dites secondaires sont toutes appelées par l'intermédiaire des menus déroulants. Certaines nécessitent l'utilisation de boîtes de dialogue intermédiaires (pour plus de détails, le lecteur intéressé se rapportera au rapport technique [Carbonneaux & al. 96] ou aux manuels utilisateurs [Bordier90], [Carbonneaux 97]).

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

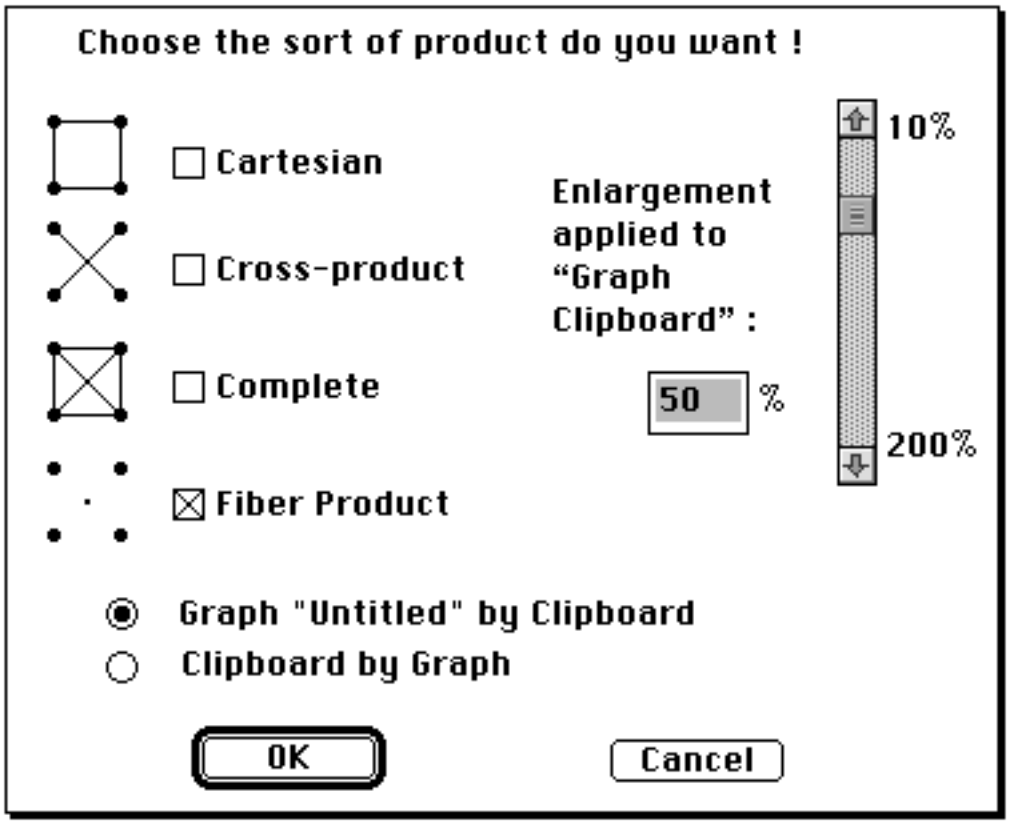


Figure 3.26 : La fenêtre de dialogue de la commande "Product by Clipboard".

Il est possible d'utiliser directement le produit cartésien par K_2 en appelant la fonction "Product by K_2" du menu "Tranform" ou de copier K_2 dans le presse-graphe[3] et d'appeler la fonction "Product by Clipboard" du même menu "Transform" (voir la figure 3.26). Dans le premier cas, la suite d'opérations s'apparente à celle du collage du graphe du presse-graphe (cet outil est décrit plus loin). Dans le second cas, une fenêtre de dialogue apparaîtra, permettant de spécifier qu'il s'agit d'un produit cartésien (et non du produit croisé, du produit complet ou du produit fibré) et de paramétrer en partie la représentation du graphe produit obtenu.

Représentation des objets

Sommet

La représentation du sommet est une forme géométrique parmi le carré, le cercle et le triangle. La taille du sommet varie entre 3 et 19 pixels et sa couleur est à choisir parmi les huit couleurs

[3] Le presse-graphes correspond au presse-papiers (Clipboard). Le presse-papiers est une mémoire tampon qui sert de zone de stockage du dernier élément coupé ou copié. Les éléments stockés dans le presse-papiers peuvent être collés dans des documents. Dans *Cabri-graphs*, nous avons étendu cette commande aux graphes afin de conserver le dernier graphe coupé ou copié. Evidement, le dernier élément est soit un graphe, soit une chaîne de caractère.

fondamentales (le blanc, le noir, le vert, le rouge, le bleu, le cyan, le magenta, le jaune), avec initialement le blanc.

Arête

La représentation de l'arête est un segment entre les deux sommets incidents, un segment brisé ou une courbe de Bézier cubique. La taille de l'arête est un chiffre impair compris entre 1 et 15 pixels. L'interface entre le sommet et l'arête est un modèle standard : l'arête est dessinée à partir des points de l'écran représentant ses extrémités. La couleur de l'arête est à choisir parmi les huit couleurs fondamentales, et elle est par défaut le noir.

Flèche

La flèche représentant l'orientation d'un arc est uniforme. La taille de l'arête associée détermine la taille de la flèche. Elle est dessinée par défaut à une distance d'un tiers pour un segment et à l'emplacement du premier point de contrôle d'une courbe. Elle peut être glissée le long de la représentation de l'arête.

Etiquette

Cabri-graphes autorise des étiquettes arbitraires pour les sommets et les arêtes sans aucune restriction ni dans la longueur ni dans le contenu de l'étiquette. Cette dernière est placée automatiquement à gauche du sommet et au milieu de la représentation d'une arête. Une étiquette possède de manière arbitraire une fonte, un style et une taille.

Un éditeur de sommets

La commande "Edit Vertices ..." est plus qu'une simple édition de sommets. En effet, activer cette commande fait apparaître une boîte de dialogue (voir la figure 3.27). Les modifications standard possibles sont sur le sommet (la forme, la taille, la couleur), sur l'étiquette alphanumérique et ses attributs (la police, le style, le format et la taille). La fenêtre est centrée sur le sommet édité et une vue partielle du graphe est alors disponible.

Les modifications effectuées sur les sommets sont aussi regroupées dans un éditeur de sommets (en rénovation). Un éditeur d'arêtes est en cours de construction.

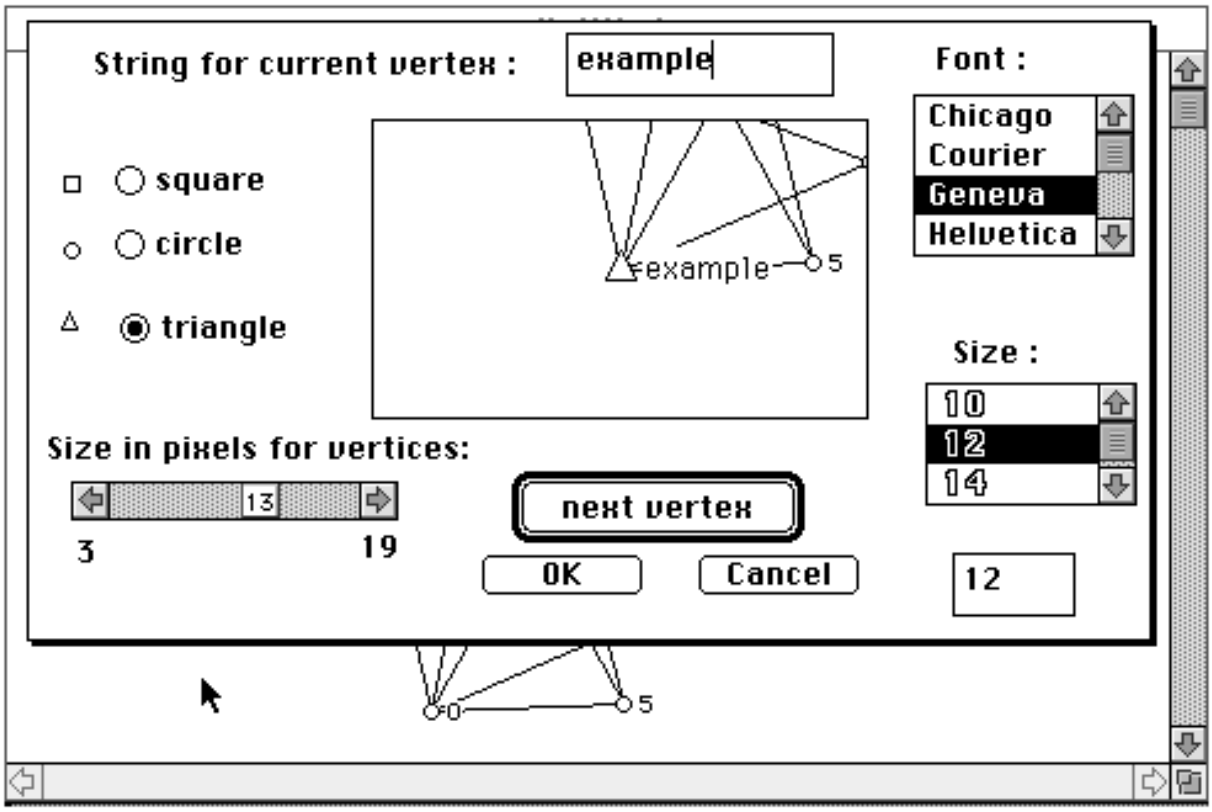


Figure 3.27 : L'éditeur des sommets dans *Cabri-graphes*.

4.2.2 Les retours d'information

Les retours d'information disponibles sont limités. Les quatre domaines exploités sont les formes du curseur, les classiques boîtes de dialogue, des messages retournés à l'utilisateur par le cadre réservé, et les problèmes liés à la sélection.

Les différentes formes du curseur

Le choix de l'opération à exécuter dépend à la fois de la position du curseur par rapport aux objets intrinsèques de la fenêtre active, et en particulier dans le cadre de la figure, de l'état des sélections, et de l'état des touches modificatrices.

Afin de signaler à l'utilisateur, au moins partiellement, la prochaine opération exécutée ou en cours d'exécution, *Cabri-graphes* modifie l'icône représentant le curseur :

Dans l'espace réservé au dessin

. des outils de construction :

+ : création d'un sommet,

: création d'une arête,

: insertion d'un caractère dans une étiquette.

. des outils de déplacement :

: déplacement de la lucarne,

: déplacement d'un sommet ou d'une étiquette,

: déplacement d'un point sur la représentation d'une arête, ou d'un point de contrôle d'une arête,

: sélection d'un sommet ou d'une arête.

. des outils divers :

et : main "tournante" pendant une rotation d'un ensemble de sommet

: s'il existe une ambiguïté dans la sélection des arêtes,

: pour définir un rectangle lors du service coller.

Dans l'espace de la fenêtre active, autre que le cadre réservé au dessin, nous avons :

: pour déplacer les ascenceurs,

: pour choisir un article d'un menu,

: pour appuyer sur un bouton.

Lorsque le curseur a la forme d'une montre , cela signifie que l'utilisateur doit patienter.

Les messages donnés à l'utilisateur

Dans la figure 3.24, un cadre est présent pour diffuser des informations à l'utilisateur. Ces informations sont de plusieurs natures :

(a) Un message simple dans le cadre réservé de la fenêtre active, donnant une information à l'utilisateur, comme par exemple l'explication d'une commande non valide ou non exécutée.

(b) Un message avec une demande de réponse. Il comprend les fameux boutons étiquetés "Cancel" et "Ok", afin de confirmer une commande entraînant de graves conséquences dans la structure ou le dessin.

(c) Un message permettant l'arrêt de l'action en cours, avec un bouton contenant le message étiqueté "Stop".

Dans le cas de changement de structure du graphe, modification profonde du graphe, un bouton ayant le message "Confirm" apparaît pour confirmer ce choix. Dans la majorité des cas, le message est "Cancel" pour annuler l'action en cours.

4.2.3 Divers services

Les entrées-sorties

Le fait de pouvoir sauvegarder des graphes sur un support externe représente un avantage non négligeable par rapport à l'archivage des feuilles de brouillon. Il serait fort intéressant de pouvoir bénéficier d'un stockage *intelligent* sous la forme d'une base de données. Cela nécessite à notre avis un travail de réflexion important si l'on veut arriver à un résultat satisfaisant. En particulier, il se pose le problème de l'isomorphisme, puisqu'il n'existe pas à l'heure actuelle d'algorithme polynomial permettant de tester si deux graphes sont isomorphes.

Le stockage sous forme de fichier texte nous paraît tout à fait indispensable pour les raisons exposées plus haut. Il faut, d'autre part, que l'automate de lecture d'un tel fichier laisse le maximum de liberté à un utilisateur pour décrire son graphe. L'automate actuel est présenté dans l'*annexe 1*.

Annuler/Répéter

Dans *Cabri-graphes*, nous avons opté pour un service simple de "Annuler/Répéter", c'est-à-dire que répéter une deuxième fois la commande "Annuler" correspond à annuler le premier "Annuler". La commande "Répéter" est en étude.

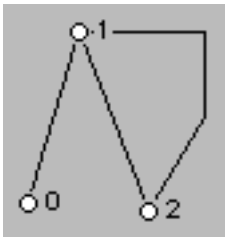
Nous avons aussi décidé d'afficher un message aussi significatif que possible lors de l'annulation de la dernière commande, afin de permettre une meilleure utilisation de ce service. Nous donnons quelques exemples.

Copier-couper-coller

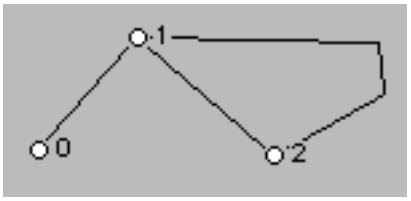
La commande appelée par l'article "Paste" du menu "Edit" permet de coller le graphe du presse-graphe dans la fenêtre active. Il est nécessaire de spécifier un rectangle de destination pour le graphe à coller. Un message apparaît dans la partie réservée au dialogue de la fenêtre pour signaler cette attente avec à proximité un bouton permettant l'annulation de l'opération. De plus, le curseur est représenté par deux petites flèches orthogonales :  .

Lorsque l'utilisateur appuie sur le bouton de la souris, la position du curseur détermine un premier point *p* du rectangle de destination. Le curseur prend alors la forme de deux petites flèches orthogonales. Lors du relâchement du bouton, la position de la souris détermine un deuxième point *p'*. Soit R(p,p') le plus petit rectangle contenant les points *p* et *p'*. Tant que le bouton de la souris reste enfoncé, un rectangle en surbrillance apparaît. Il correspond au rectangle obtenu avec comme second point, la position courante du curseur. Si le rectangle de destination est jugé trop petit, la procédure de spécification du rectangle est de nouveau initialisée, et un message demandant un rectangle plus grand est affiché dans le rectangle réservé

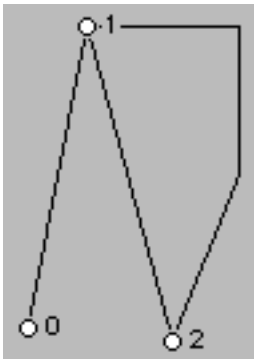
au dialogue (voir la *figure 1.4* (*b*) du chapitre 1). Evidemment cette opération peut à tout moment être arrêtée.



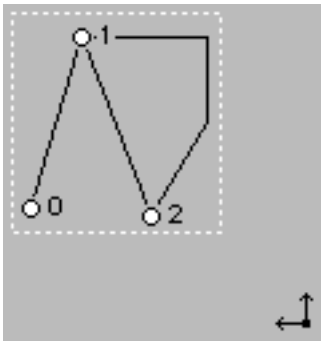
(a) Le graphe initial G.



(b) Aucune touche n'est enfoncée .



(c) La touche *majuscule* est enfoncée.



(d) La touche*e commande* est enfoncée.

Figure 3.28 : Un graphe G et diverses copies du graphe.

Si aucune touche modificatrice n'est enfoncée lors du relâchement du bouton de la souris, alors R(p,p') est le rectangle de destination. Si la touche *majuscule* est enfoncée, alors le graphe collé subit uniquement une déformation par similitude (le rapport hauteur/largeur reste constant). Le rectangle de destination est alors le plus petit rectangle respectant cette contrainte en contenant le rectangle R(p,p'). Si la touche *commande* est enfoncée, alors le graphe collé ne subit aucune déformation. Le rectangle de destination est le rectangle défini par le point *p*, et un autre point de la diagonale passant par *p* étant le plus proche possible de *p'* (voir les représentations des graphes copiés dans la figure 3.28).

Presse-graphes

Cabri-graphes dispose d'un presse-papiers permettant ainsi le transfert du dessin de graphe dans les différents environnements graphiques ou les éditeurs de texte (voir la figure 3.29).

Ce presse-graphes ne sert pas uniquement pour le service couper-copier-coller de *Cabri-graphes*, mais aussi dans des opérations sur les graphes, comme les produits de graphes ou le plongement de graphes. En effet, il est très difficile de sélectionner deux parties d'un même ensemble, et l'utilisation du presse-graphes facilite l'interface des opérations pour l'utilisateur.

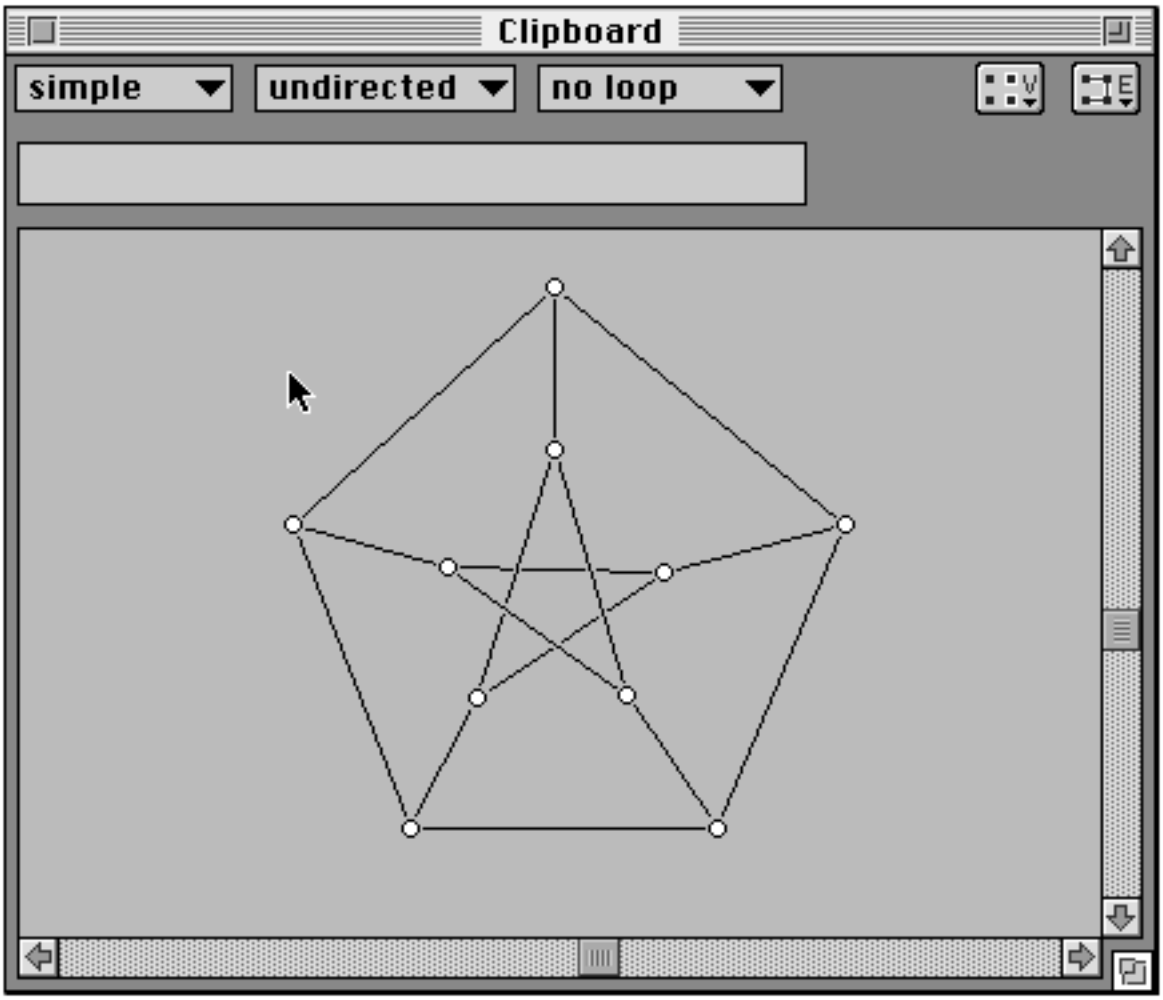


Figure 3.29 : Le presse-graphes ou "clipboard" de *Cabri-graphes*.

4.3 Manipuler les graphes

La manipulation des graphes s'effectue le plus simplement et naturellement possible. Elle est interactive en utilisant les pointeurs de désignations disponibles (en l'occurrence la souris) pour les commandes de création, de sélection et de déplacement de tous les objets du domaine.

4.3.1 Dessiner un graphe

Il existe plusieurs façons de dessiner des arêtes multiples dans un graphe. Nous avons introduit dans *Cabri-graphes* la possibilité de dessiner des arêtes par un segment brisé ou par une courbe de Bézier cubique. Les différentes formes de la courbe de Bézier sont la boucle, le lacet et l'arc de courbe, plus des formes dégénérées (pour plus de détails, voir l'*annexe 2*).

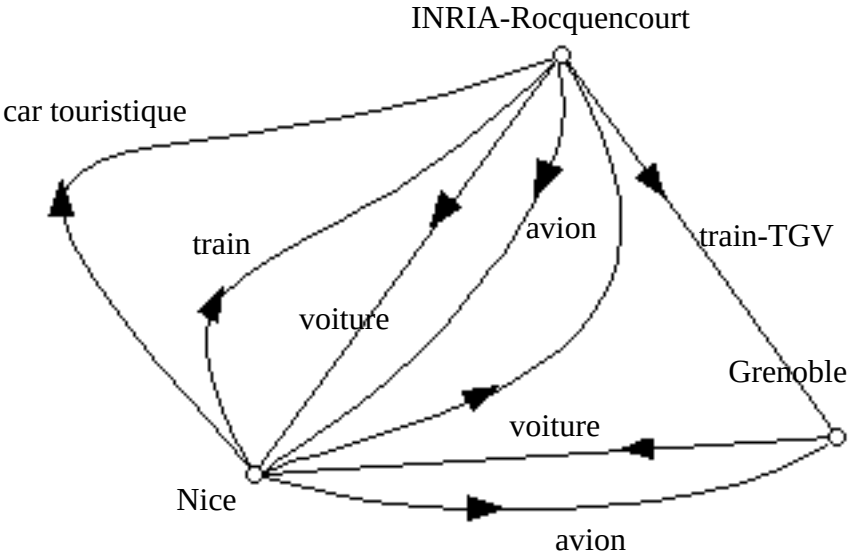


Figure 3.30 : Un graphe multiple, orienté et étiqueté.

L'arête admet trois états possibles : le segment, le segment brisé et la courbe (voir la figure 3.30). Ces trois représentations graphiques de l'arête ne s'opposent pas puisqu'il est possible de passer d'un état à l'autre. Une courbe de Bézier cubique est définie par ses quatre points de contrôle externes. Dans *Cabri-graphes*, l'utilisateur détermine interactivement quatre points appartenant à la courbe. Ces points sont appelés des points de contrôle interne de la courbe (pour plus de détails sur la détermination d'une telle courbe de Bézier, voir l'*annexe 2*).

L'utilisateur possède deux possibilités pour créer les objets du graphe : manuellement ou automatiquement. Nous étudions dans ce paragraphe seulement la manière manuelle. Pour le dessin automatique de graphes, le lecteur intéressé se reportera aux manuels d'utilisation ([Bordier 90] pour la version précédente de *Cabri-graphes* et [Carbonneaux & al. 97] pour la version présente). La création manuelle d'un graphe se traduit par celle des sommets et des arêtes. Leur construction s'effectue par manipulation directe.

La création des sommets

Le curseur, sous la forme d'une croix, doit se trouver éloigné de tout autre sommet et les touches modificatrices sont toutes relevées. De plus, aucun sommet ne doit être sélectionné. Dans ce cas, si l'on appuie sur le bouton de la souris dans l'espace réservé au dessin, un cercle représentant le nouveau sommet apparaît ainsi qu'une mire après un certain temps (voir figure 3.31).

L'autre cas est pendant une création d'une arête, si au moins une de ses deux extrémités n'existe pas.

(a) Forme du curseur pour créer un sommet. (b) Représentation après un clic .

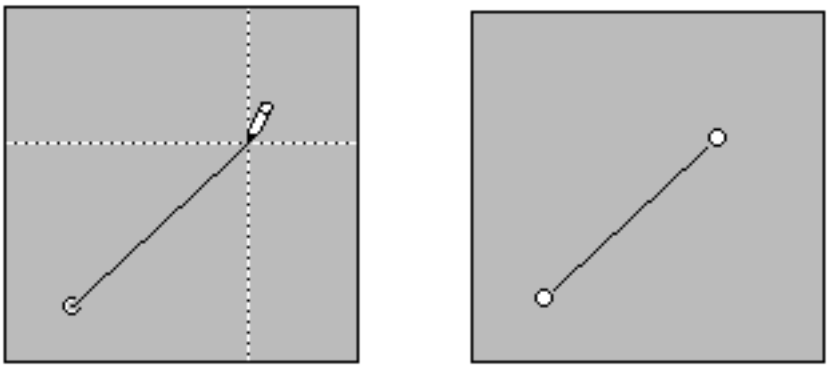
Figure 3.31 Création d'un sommet dans *Cabri-graphes*.

La création des arêtes

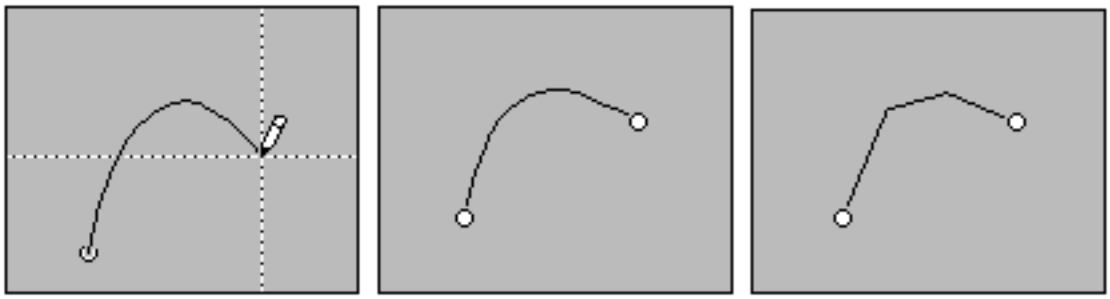
Si la touche *option* est enfoncée, alors *Cabri-graphes* considère que l'on cherche à créer une arête et le curseur se présente alors sous la forme d'un crayon.

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

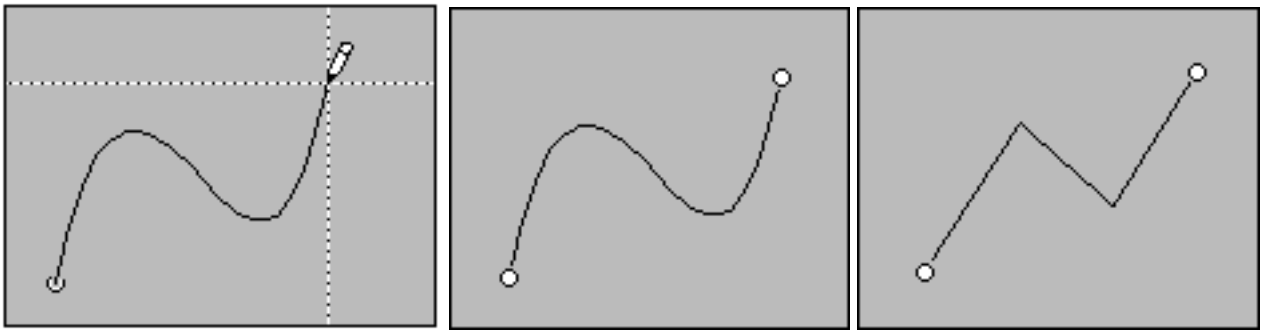
Lors du déplacement de la souris, le bouton restant appuyé, la nouvelle arête apparaît et sa deuxième extrémité suit le déplacement du curseur, sauf dans le cas où celui-ci se rapproche d'un sommet existant. La deuxième extrémité de l'arête est alors *attirée* par ce sommet. Cela signifie que, si l'utilisateur relâche la souris dans cette situation, alors ce sommet déjà existant formera la deuxième extrémité de la nouvelle arête, sinon un nouveau sommet sera créé. Plusieurs cas peuvent alors intervenir pendant le déplacement du curseur sous forme de stylo (voir la figure 3.32 de la prochaine page)



(a) Si le bouton de la souris est relâché proche d'un sommet, alors l'arête joint ce sommet sinon elle joint le nouveau sommet créé.



(b) En déplaçant le curseur, si une touche du clavier est enfoncée alors un point interne de la courbe est dessiné. Cette arête est une courbe de Bézier ou un segment brisé, suivant l'option choisie.



(c) Si, une autre touche du clavier est encore enfoncée, alors le second point interne est créé. Une courbe de Bézier cubique ou un segment brisé est alors construite. A la fin le bouton de la souris doit être relâché.

Figure 3.32 : Construire une nouvelle arête.

Si l'arête dessinée est incompatible avec les propriétés du graphe (par exemple vouloir créer une arête multiple dans un graphe simple), son dessin est alors supprimé.

La création de la flèche

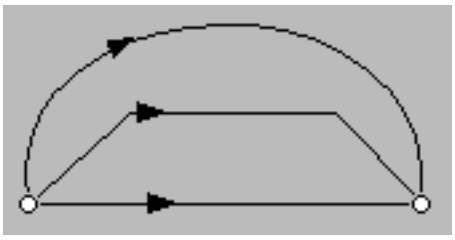


Figure 3.33 : La position des arcs créés par défaut.

Lorsqu'un arc (une flèche sur une arête) est créé, sa position initiale est à 1/3 de la distance entre les deux sommets extrémités pour une arête représentée par un segment ou par un segment brisé et au point-fixe au temps t = 1/3 pour une arête représentée par une courbe de Bézier.

La création des étiquettes

Dans *Cabri-graphes* , il existe un étiquetage numérique et un étiquetage alphanumérique. Ces deux étiquetages ne peuvent être visualisés en même temps. Par contre, l'étiquetage numérique est conservé lorsque l'on passe en mode alphanumérique et inversement (pour l'étiquetage numérique résultat d'une commande). Enfin, il est possible de visualiser le graphe sans aucun étiquetage.

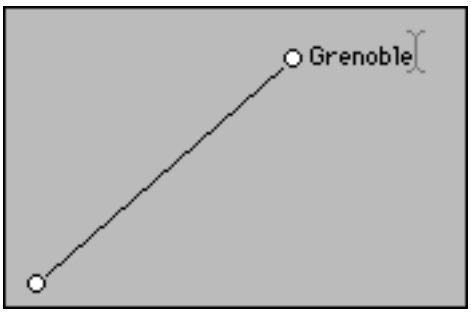


Figure 3.34 : Etiquetage des sommets. Par defaut, l'étiquette est placé à gauche et au milieu de la représentation du sommets.

Par défaut, l'étiquetage est numérique et chronologique. Le premier sommet créé possède le numéro 0. Sauf cas particulier, un sommet nouvellement créé sera numéroté par le plus petit entier non encore utilisé. Il existe donc des trous dans la numérotation si l'on supprime des sommets au cours de la création du graphe.

Pour les arêtes, un étiquetage chronologique est aussi effectué, mais il n'est pas visualisé sauf si l'utilisateur choisit cette option.

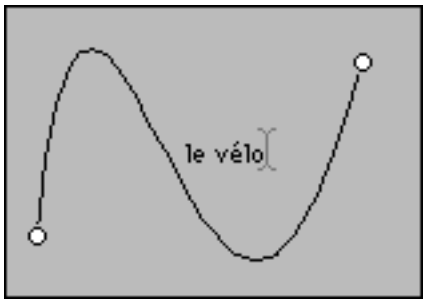


Figure 3.35 : Etiquetage des arêtes. Par défaut l'étiquette est placé au "milieu" de la représentation de l'arête ou pour un temps t = 1/2.

Les étiquettes numériques sur les arêtes servent principalement aux calculs, par exemple les calculs de flots, et les calculs sur tous les réseaux de transport où les étiquettes numériques sont utiles. L'étiquetage alphanumérique est aussi disponible.

4.3.2 Sélectionner des objets

Pour la théorie des graphes, il existe que peu d'objets, nous avons donc adoptée une attitude particulière pour la sélection, en comparaison à des environnements informatiques dans d'autres domaines. Par exemple, pouvoir différencier les sommets et les arêtes avec une sélection par rectangle grâce à une touche modificatrice du clavier.

Gestion des ambiguïtés

Dans l'environnement de *Cabri-graphes*, un ordre de préférence sur les objets a été adopté, pour une gestion simple des ambiguïtés. Cet ordre est le suivant : le sommet, l'arête, la flèche, et l'étiquette.

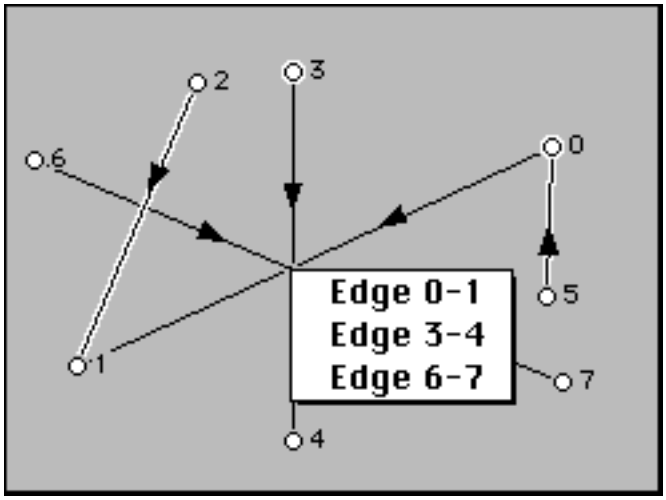


Figure 3.36 : Un graphe orienté dans l'environnement de *Cabri-graphes*. Les deux arêtes et le sommet d'indice 4 sont en surbrillance, parce qu'ils sont sélectionnés.

Par contre, lorsque des objets de même nature sont sélectionnés, un choix est laissé à l'utilisateur. Dans la figure 3.36, le menu déroulant affiche à l'intersection des segments trois choix possibles d'arcs afin que l'utilisateur désigne explicitement l'arête sur laquelle il voudra effectuer une opération.

Désignation d'un ou plusieurs sommets

Pour plusieurs commandes, il est nécessaire de désigner un ou plusieurs sommets. Un exemple est donné par la sélection des sommets *s* et *s'* de l'intervalle I(s,s'). Cette procédure est appelée par l'article "Interval" du menu "Select".

Après l'appel de cette fonction, un message apparaît, demandant la désignation d'un premier sommet *s*, alors que le curseur prend la forme d'une flèche. Le message est de plus accompagné d'un bouton permettant l'annulation de l'opération. Lorsque l'on déplace le curseur et que celui-ci s'approche d'un sommet, ce sommet est entouré par un petit cercle. Le sommet *s* sera le sommet entouré au moment où l'utilisateur appuiera, puis relâchera le bouton de la souris. Si aucun sommet n'est entouré, le clic n'est pas pris en compte. Après la désignation du sommet *s*, un deuxième dialogue apparaît en demandant de sélectionne un deuxième sommet *s'*. De son côté, le sommet *s* est mis en évidence par un effet de surbrillance ou en étant entouré par un petit cercle gras. La désignation du sommet *s'* est identique à celle de *s*.

Si la sélection courante *S* de sommets n'est pas vide au moment de l'appel de la fonction, alors seuls les sommets sélectionnés doivent être pris en compte. La procédure est alors appliquée au sous-graphe induit pas l'ensemble *S*. Les sommets de la sélection sont entourés d'un cercle gras au moment du passage du curseur à leur proximité.

4.3.3 Déplacer les objets

Comme nous l'avons souvent répété, tous les objets de *Cabri-graphes* sont manipulables et donc le sommet, l'arête, l'orientation, l'étiquette et un sous-graphe peuvent être déplacés.

Les sommets

Un simple clic près d'un sommet le fait apparaître en noir et sélectionne ce sommet. Un ensemble de sommets peut être sélectionné par les menus ou par l'intermédiaire d'un rectangle.

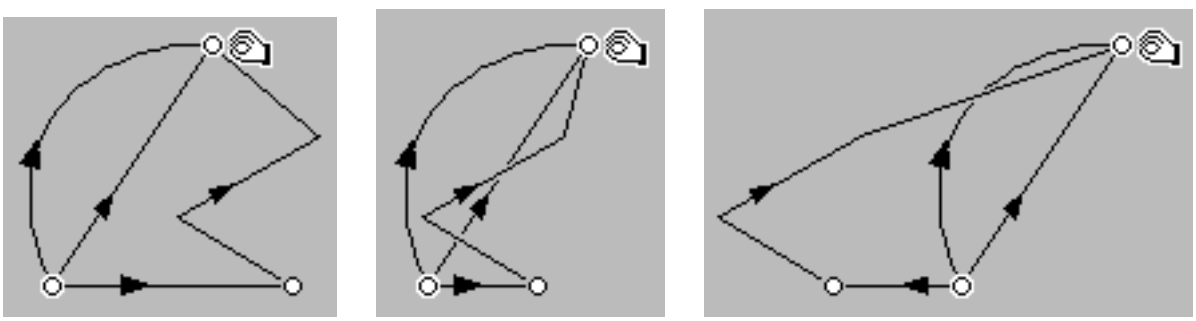


Figure 3.37 : Le déplacement d'une sélection de sommets.

Un ensemble de sommets sélectionnés correspond au concept mathématique de sous-graphe induit (voir la figure 3.37). L'apparence des sommets sélectionnés est en noir pour la version en noir et blanc, et en surbrillance dans la version couleur. Ceci s'avère très utile pour restreindre une action donnée à une partie du graphe.

Lorsqu'un sous-graphe (ou une sélection de sommets) est déplacé, les arêtes incidentes sont modifiées et les arêtes du sous-graphe sont translatées en conséquence.

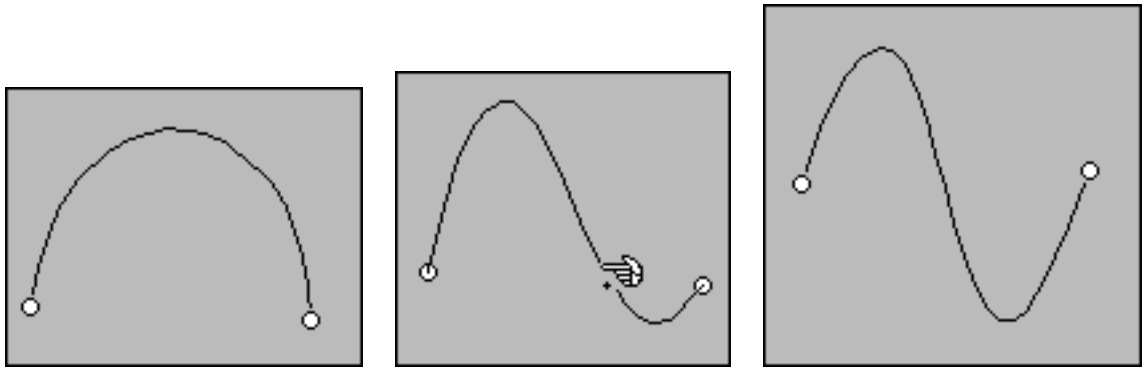
Les arêtes

Il existe trois manières de manipuler une arête. La première est par l'intermédiaire d'un sommet incident à l'arête. La seconde façon s'effectue directement par un point interne de l'arête courbe. La troisième manière utilise un point quelconque appartenant à la courbe.

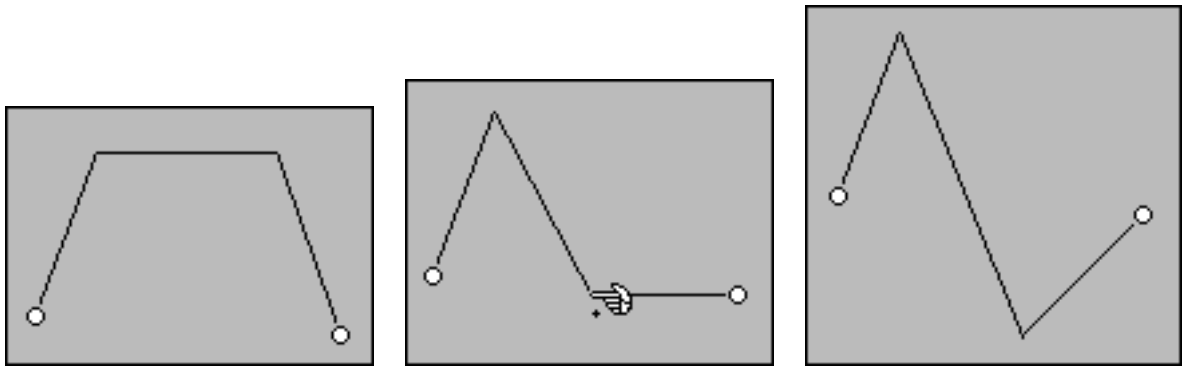
Déplacer une extrémité d'une arête

Cette commande est équivalente au déplacement d'un sommet du graphe.
Toutes les arêtes incidentes au même sommet sont déformées lors du déplacement du sommet.

Déplacer un point de contrôle interne d'une arête



(a) La représentation de l'arête est une courbe de Bézier cubique.



(a) La représentation de l'arête est un segment brisé.

Figure 3.38 : Transformation d'une arête.

En appuyant sur la touche de contrôle, le curseur se transforme en crayon s'il est proche d'une arête. Un des points internes de la courbe est alors prêt à être déplacé. Son déplacement modifie la courbure de l'arête (Pour plus de détails, se reporter à l'*annexe 2*). Dans la figure 3.38, nous avons ajouté "à la main" les points de contrôle internes de la courbe et en pointillé son ancienne position.

Déplacer un point quelconque d'une arête représentée par une courbe

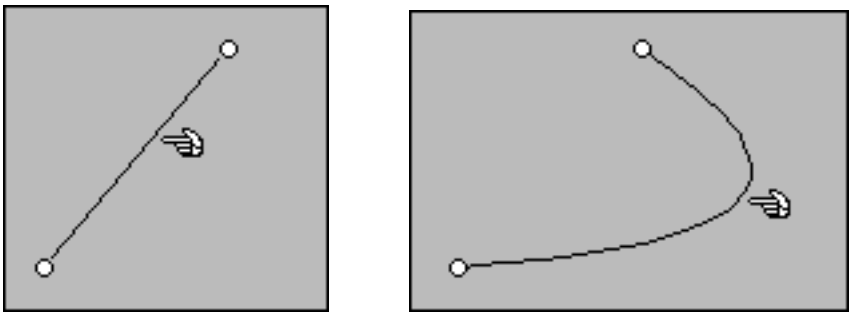


Figure 3.39 : Déformation d'une arête par "translation" d'un de ses points.

En déplaçant la souris au voisinage de l'arête, un point quelconque de l'arête est désigné (le point au milieu de l'arête dans la figure 3.39). La forme de la courbe de Bézier cubique s'adapte pour suivre le déplacement de ce point.

Le procédé est une version simplifiée pour notre cas de la méthode de Fowler et Bartels [Fowler & Bartels 93]. Une courbe de Bézier cubique est définie par :

$Q(t) = \sum_{i=0}^{3} P_i B_i(t)$, où P_i sont les points de contrôle, $B_i(t)$ les fonctions de base et *t* un réel de l'interval [0,1] (pour plus de détails se reporter à l'*annexe 2*).

Pour tout *t* de [0,1], correspond un unique point de la courbe, et réciproquement. Alors, au moment du clic sur le bouton de la souris pour déplacer la courbe, un réel t_0 est fixé et par conséquent un point $Q_0(t_0)$ de la courbe. Un déplacement ΔQ_0 de ce point Q_0 implique la résolution du système suivant, afin de dessiner de nouveau la courbe :

$\Delta Q(t_0) = \sum_{i=0}^{3} \Delta P_i B_i(t_0)$, or P_0 et P_3 sont des points de contrôle fixes de la courbe

donc $\Delta P_0 = \Delta P_3 = 0$

et il faut résoudre le système : $\Delta Q_0() = \Delta P_1.B_1(t_0) + \Delta P_2.B_2(t_0)$

La solution de cette équation différentielle est appelée *la solution de longueur minimum*. Les nouveaux points de contrôle ΔP_1 et ΔP_2 de la courbe de Bézier sont donnés par :

$$\Delta P_1(t_0) = \frac{B_1(t_0)}{B_1^2(t_0) + B_2^2(t_0)} \Delta Q(t_0) \text{ et } \Delta P_2(t_0) = \frac{B_2(t_0)}{B_1^2(t_0) + B_2^2(t_0)} \Delta Q(t_0)$$

Les points de contrôle internes de la courbe doivent être de nouveau calculés.

Déplacer un point quelconque d'une arête représentée par un segment brisé

Nous avons deux cas possibles.

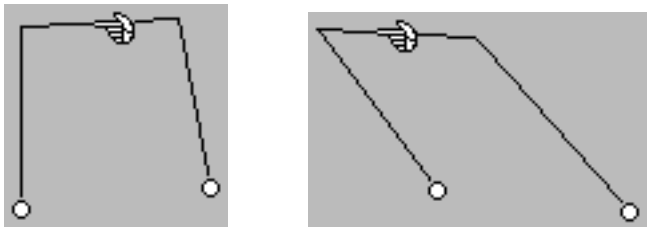


Figure 3.40 : Déformation d'une arête par "translation" d'un de ses points.

Dans le premier cas, on déplace un point d'un segment "non adjacent" à une extrémité de l'arête associée (voir la figure 3.40). Ce segment est translaté en fonction de la position du curseur. Les autres segments attachés à ce segment sont ainsi agrandis ou diminués.

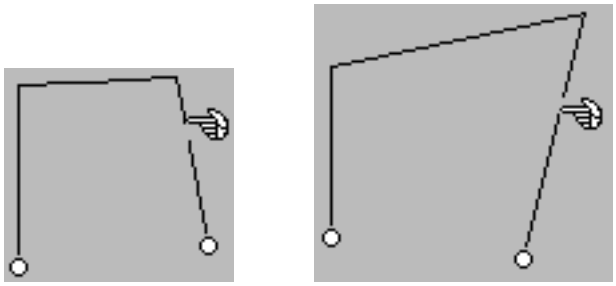


Figure 3.41 : Déformation d'une arête par "translation" d'un de ses points.

Dans le second cas, on déplace un point d'un segment "adjacent" à un sommet. Lors du déplacement du curseur, on conserve la proportionnalité de distance entre la position du curseur et les extrémités du segment. L'autre segment, attaché par une extrémité, est donc aussi modifié. Par contre, les autres segments, dans notre exemple de la figure 3.41, un seul autre segment, ne sont en aucun cas transformés.

Orientation

Cette flèche se déplace le long du segment ou de la courbe. La position de la flèche est déterminée par la projection orthogonale de la positon du curseur, représenté par une main, sur la représentation de l'arête . Sa position, le long de l'arête, est définie par un réel *t* compris entre 0 et 1, et est un élément graphique de l'arête (voir la figure 3.42).

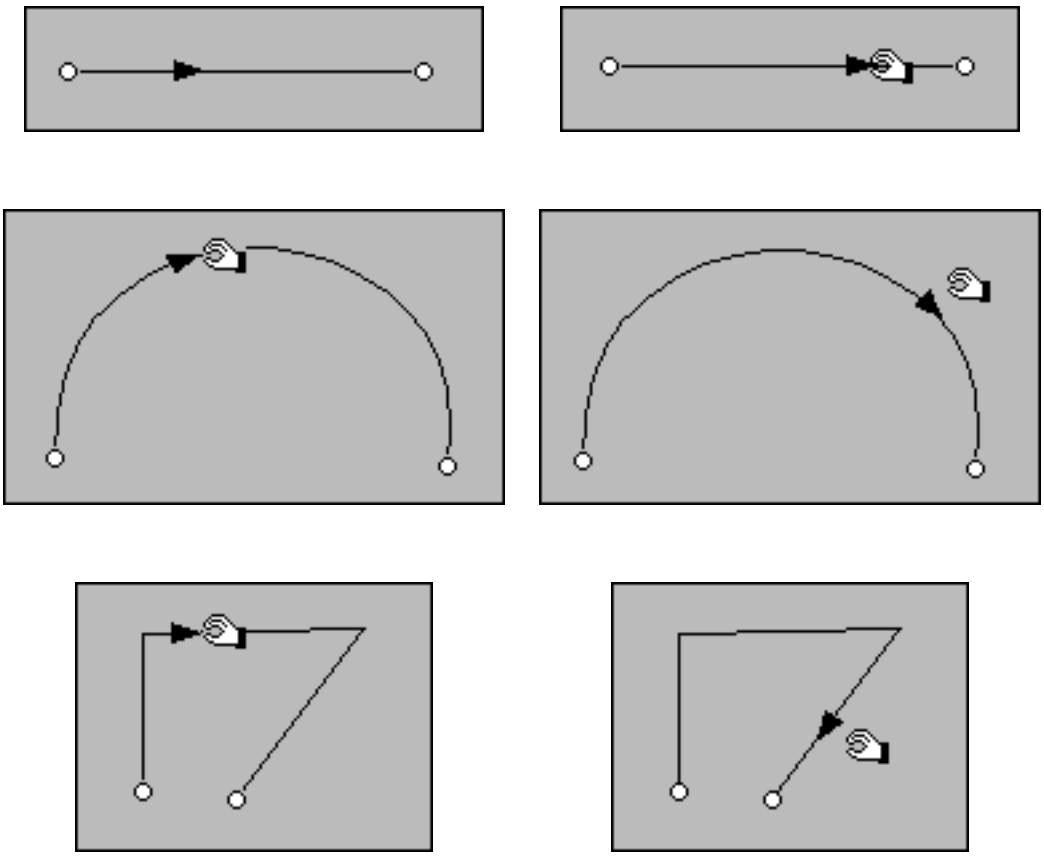


Figure 3.42 : Déplacement de la flèche.

Etiquetage

L'étiquette est associée aux objets par un effet de magnétisme. Cet effet interdit à l'étiquette de se trouver trop éloignée de son centre d'attraction, et par conséquent de refléter une information fausse en étant associée à un autre objet. Il faut remarquer que cet aimant ne diminue que l'étendue du problème sans le résoudre pour des objets relativement proches entre eux.

L'étiquette d'un sommet ou d'une arête est une chaîne alphanumérique. Elle est créée initialement à droite de l'objet associé et elle est magnétisée avec ce dernier.

En appuyant sur une touche modificatrice, l'étiquette échappe à l'attraction de l'aimant. Sa position est libre sur l'écran. Pour les autres futurs déplacements, il faut enfoncer une autre touche modificatrice afin que l'étiquette soit de nouveau sujet à l'action de l'aimant. Ces actions sont en cours de développements.

pour un sommet

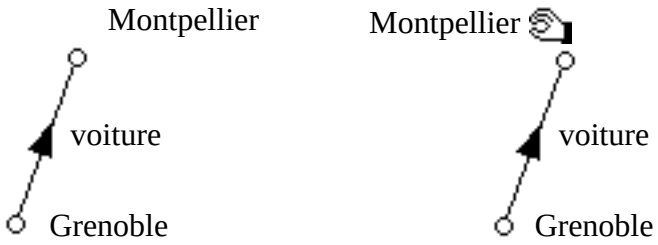
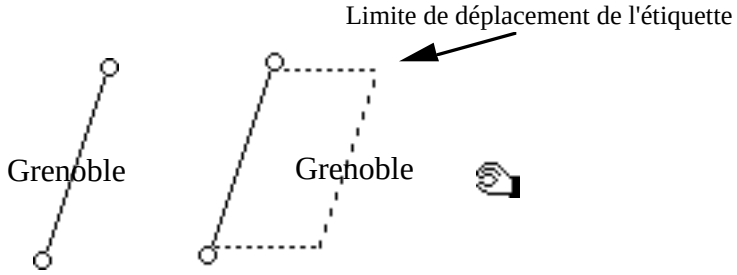


Figure 3.43 : Le dplacement de l'étiquette associée à un sommet.

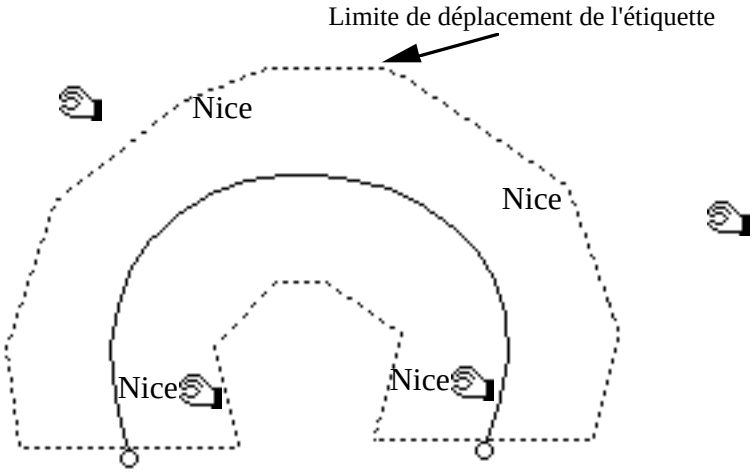
L'étiquette associée à un sommet ne peut se mouvoir qu'à l'intérieur d'un disque de centre le sommet et de rayon un entier défini par défaut. Ce paramètre se trouve dans un fichier préférence et sa valeur est modifiable.

pour une arête

Dans notre environnement informatique, le mouvement de l'étiquette est géré indépendamment de la représentation de l'arête. La forme de l'arête est décomposée en une succession de segments. Pour chaque segment, est défini un cylindre ayant ce segment pour axe et comme rayon un entier déterminé auparavant.



(a) l'arête est un segment.



(b) l'arête est une courbe de Bézier cubique.

Figure 3.44 : Le déplacement de l'étiquette associée à une arête.

L'étiquette d'un segment se déplace à l'intérieur du cylindre associé. Elle est dessinée au point déterminé par la souris. Si le curseur de la souris est en dehors du cylindre, alors la position du curseur est projetée orthogonalement sur le cylindre et l'étiquette est dessinée au bord du cylindre.

Pour un segment brisé ou une courbe, le déplacement de l'étiquette est similaire. La position du curseur se trouve à l'intérieur de la bande, alors l'étiquette est dessinée à partir de ce point, sinon elle est représentée au bord de la bande.

Une bande est définie par l'ensemble des cylindres et elle est associée à l'arête. la position du curseur est projetée orthogonalement sur l'ensemble des segments. Si la distance entre le point projeté et l'emplacement de la souris est inférieure au rayon des cylindres, alors l'étiquette est dessinée à la position du curseur. Sinon on considère la plus courte distance entre la position du curseur et les différents points de projections (un point pour chaque segment) en conservant la position du bord de la bande comme nouvel emplacement de l'étiquette.

4.3.4 La manipulation directe

Dans les prochains tableaux, nous présentons une analyse par cas des manipulations directes sur les objets de notre environnement. Cette analyse ne se veut pas exhaustive, mais seulement représentative du plus grand nombre de cas possible.

Cinq paramètres décident de l'opération à effectuer :

1) L'action effectuée sur la souris, avec l'enfoncement du bouton et le déplacement de la souris. Cette action est décrite dans la colonne **Action sur la souris**.
2) La position du curseur par rapport aux objets intrinsèques de la fenêtre active (le curseur est toujours dans cette fenêtre). Cette position est décrite dans la colonne **Objet**.
3) L'état de la sélection courante de sommets. Cet état est décrit dans la colonne **S**.
4) L'état de la sélection courante d'arêtes. Cet état est décrit par la colonne **A**.
5) L'état des touches modificatrices. Cet état est décrit dans la colonne **Touches**.

La colonne **Curseur** donne les différentes formes du curseur pendant la durée de l'opération. Les effets de l'opération exécutée sont décrits dans la colonne **Effet**.

Exemple

Action sur la souris = Enfoncer et relâcher au point p

Objet = aucun

S = Ø

A = non précisée

Touches = aucune

Curseur = +

Effet = Créer un sommet au point p

Interprétation

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

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

Enfin, afin de ne pas surcharger les tableaux, déjà fort longs, nous nous sommes contentés de référencer les actions possibles sur un graphe valué (étiquetage numérique) et d'ajouter en fin de tableau les principales actions spécifiques à l'étiquetage alphanumérique.

4.2.3.1 Création des objets de bases

Action sur la souris	**Objets**	**V**	**E**	**Touches**	**Curseur**	**Effet**
Enfoncer et relâcher au point *p*	Aucun	$\varnothing$	$\varnothing$	Aucune	+	Crée un sommet au point *p*
Enfoncer et relâcher au point *p*	Aucun	$\neq\varnothing$		Ctrl		Crée un sommet au point *p* ayant V pour voisinage
Enfoncer au point *p*, déplacer et relâcher au point *p′*	Aucun	$\varnothing$		Ctrl		Créer une arête d'extrémités en *p* et *p′*. S'il n'existait pas de sommets en *p* ou *p′*, alors ces sommets sont crées.
Enfoncer au point *p*, et déplacer	Aucun	$\varnothing$		Ctrl		Crée une arête d'extrémités en *p* et au point du curseur. S'il n'existait pas de sommets en *p* , alors ce sommet est créé.
Déplacer .	Aucun			Une touche quelconque		Dessine l'arête comme une courbe de Bézier quadratique, de points de contrôle *p*, *q* et au point du curseur.
Déplacer .	Aucun			Une touche quelconque		Dessine l'arête comme une courbe de Bézier cubique, de points de contrôle *p*, *q*, *r* et au point du curseur.
Relâcher au point *p′*	Aucun					Dessine l'arête comme une courbe de Bézier cubique, de points de contrôle *p*, *q*, *r* et *p′*. L'arête est créée si elle est conform à la nature du graphe.

4.2.3.2 Gestion des ensembles d'objets

Action sur la souris	Objets	V	E	Touches	Curseur	Effet
Enfoncer et relâcher au point *p*	Aucun	$\neq \varnothing$		Aucune	+	$V \leftarrow \varnothing$
Enfoncer et relâcher au point *p*	Aucun	$\varnothing$	$\neq \varnothing$	Aucune	+	$E \leftarrow \varnothing$
Enfoncer et relâcher au point *p*	Sommet *v*	$s \notin V$		Aucune		$V \leftarrow \{s\}$
Enfoncer et relâcher au point *p*	Sommet *v*					$V \leftarrow V \Delta \{s\}$
Enfoncer et relâcher au point *p*	Arête *e*	$e \notin E$		Aucune		$E \leftarrow \{e\}$
Enfoncer et relâcher au point *p*	Arête *e*					$E \leftarrow E \Delta \{e\}$

4.2.3.3 Sélection par rectangle

Action sur la souris	**Objets**	**V**	**E**	**Touches**	**Curseur**	**Effet**
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Aucun			Aucune	+	V ← Rect{p,p'} E ← Rect{p,p'}
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Aucun			⇧	↱	V ← V Δ Rect{p,p'} E ← E Δ Rect{p,p'}
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Aucun			⌥	↱	V ← Rect{p,p'}
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Aucun			⇧ ⌥	↱	V ← V Δ Rect{p,p'}
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Aucun			Ctrl ⌥	↱	E ← Rect{p,p'}
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Aucun			⇧ Ctrl ⌥	↱	E ← E Δ Rect{p,p'}

4.2.3.4 Déplacement des objets

Action sur la souris	Objets	V	E	Touches	Curseur	Effet
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Sommet v	$v \notin V$		Aucune	, puis	$V \leftarrow \emptyset$ Déplace le sommet *v* de *p* en *p'*
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Sommet v	$v \in V$		Aucune		Effectue sur les éléments de V une translation de vecteur $\overrightarrow{pp'}$
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Arête *e*			Ctrl		Déplace un point p quelconque de l'arête e au point p', La forme de l'arête est modifiée.
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Arête *e*					Déplace au point *p'* le point de contrôle de la courbe, représentant l'arête *e*, qui le plus près du point *p* , La forme de l'arête est modifiée.
Enfoncer au point *p*, déplacer et relâcher au point *p'*	La flèche *f* sur l'arête *e*					Déplace la flèche le long de la représentation de l'arête.

Action sur la souris	**Objets**	**V**	**E**	**Touches**	**Curseur**	**Effet**
Enfoncer au point *p*, déplacer et relâcher au point *p′*	Sommet *v* Sommet s'	$s \notin V$		Aucune		$V \leftarrow \emptyset$ Fusionne *s* et *s′*. La fusion est étiqueté comme *s*
Enfoncer au point *p*, déplacer et relâcher au point *p′*	Sommet *v* Sommet s'	$s \notin V$				$V \leftarrow \emptyset$ Fusionne *s* et *s′*. La fusion est étiqueté comme *s′*
Enfoncer au point *p*, déplacer et relâcher au point *p′*	Sommet *v* Sommet s'			Pas		Fusionne *s* et *s′*. La fusion est étiqueté comme *s*
Enfoncer au point *p*, déplacer et relâcher au point *p′*	Sommet *v* Sommet s'					Fusionne *s* et *s′*. La fusion est étiqueté comme *s′*
Enfoncer au point *p*, déplacer et relâcher au point *p′*	Ancun sommet					Effectue une translation de vecteur $\overrightarrow{pp'}$ de la feuille

4.2.3.6 Les commandes relatives à l'étiquette

Action sur la souris	**Objets**	**V**	**E**	**Touches**	**Curseur**	**Effet**
Enfoncer et relâcher au point *p*	Aucune étiquette					Rend active l'étiquette associée à l'objet proche du point p.
Enfoncer et relâcher au point *p*	Etiquette					Insère le curseur dans le texte de l'étiquette.
Enfoncer au point *p*, déplacer et relâcher au point *p'*	Etiquette					Déplace l'étiquette autour de l'objet associé.

Conclusion

Nous avons construit un environnement informatique de manipulation directe sur les graphes. Il est complété par des bibliothèques de procédures. Elles ont été conçues initialement pour des graphes simples non orientés. Ces bibliothèques ont déjà été modifiées pour répondre aux normes des nouvelles structures de données abstraites.

En effet, il existe une unique structure pour le type graphe dans laquelle les natures possibles des graphes sont devenues des propriétés (simple ou multiple, orienté ou non, avec ou sans boucle). Cette méthode a simplifié les problèmes de visualisation et augmenté la complexité des algorithmes, comme les parcours de graphes, par des ajouts de tests afin de connaître la nature exacte des graphes et, par conséquence, d'améliorer au mieux les algorithmes.

La nouvelle interface offre une grande souplesse dans la manipulation des graphes, et par conséquent une plus grande liberté d'action pour l'utilisateur.

Chapitre 4

Problèmes de graphes

Dans ce chapitre, nous allons aborder quelques problèmes relatifs à la théorie des graphes. Après un rappel de quelques définitions, en particulier de la notion d'homomorphisme, nous étudions le produit fibré de graphes et la contraction d'un ensemble de sommets dans un graphe.

Hell [Hell 72] utilise les concepts classiques de la théorie (algébrique) des catégories dans la catégorie des graphes. Il vérifie que le produit fibré de graphes existe [Hell 79]. Khelladi [Khelladi 92] suggère que ce produit de graphes, peu étudié, peut être une nouvelle approche, à la conjecture d'Hedetniemi [Hedetniemi 66], sur le nombre chromatique du produit cartésien (ou croisé) de graphes. Une première approche de cette question est parue dans [Semri 93] avec l'étude des propriétés du produit fibré sur des cycles impairs. Nous établissons quelques propriétés du produit fibré et nous détaillons le lien existant entre le produit fibré et le produit croisé. Le produit fibré a été programmé dans notre environnement informatique.

La contraction est un outil facilitant la visualisation des graphes. La contraction d'un ensemble de sommet dans un graphe est une opération qui consiste à identifier les sommets en conservant l'ancienne structure du graphe. Ainsi, la décontraction qui est l'opération inverse dessine de nouveau cet ensemble de sommets et les anciens liens existants. L'introduction des opérations contraction et décontraction de sommets est une utilisation de "haut niveau" de l'opérateur "Construire/Annuler".

1. Définitions complémentaires

Dans ce paragraphe, nous donnons des définitions complémentaires qui ont été traitées par un environnement informatique sur les graphes ou qui sont utilisées dans ce chapitre. Nous avons de nouveau utilisé [Harary 69], [Berge 73] et [Xuong 92].

1.1 Graphes particuliers

1.1.1 Chaînes et cycles, chemins et circuits

Une *chaîne* (resp. *chemin*) C de longueur k dans un graphe $G = (V, E)$ non orienté (resp. orienté) est une séquence alternée de sommets et d'arêtes $C = v_0\ e_1\ v_1\ e_2 \ldots\ v_k$ telle que pour tout i, $1 \leq i \leq k$, les extrémités de e_i sont v_i et v_{i+1} , c'est-à-dire que $e_i = (v_i, v_{i+1}) \in E$.

Si tous les termes e_i sont distincts, alors la chaîne (resp. chemin) C est *simple*. Si tous les termes v_i sont distincts, alors la chaîne (resp. chemin) C est *élémentaire*.

Un *cycle* (resp. *circuit*) est une chaîne (resp. chemin) dont les extrémités sont confondues. Un *cycle* (resp. *circuit*)*élémentaire* est un cycle (resp.circuit) dont tous les sommets, à l'exception des extrémités, sont distincts. En particulier, la longueur d'un cycle élémentaire est égale au nombre de sommets.

Graphe eulérien

Soit $G = (V, E)$ un graphe simple tel que $|E| = m$, s et t deux sommet différents de G. Une chaîne (resp. chemin) de s à t est dite *eulérienne* si elle est simple et de longueur m ; elle passe une fois et une seule par chaque arête du graphe.

Un cycle (resp. circuit) de longueur n est *un cycle* (resp. *circuit*)*eulérien.* Un graphe est dit *eulérien* s'il possède un cycle (resp. circuit) eulérien.

Graphe hamiltonien

Soient $G = (V, E)$ un graphe simple non orienté tel que $|V| = n$, s et t deux sommet différents de G. Une chaîne de s à t est dite *hamiltonienne* si elle est élémentaire et de longueur $n-1$; elle passe une fois et une seule par chaque sommet du graphe.

Un cycle élémentaire de longueur n est appelé un *cycle hamiltonien*. Un graphe est dit *hamiltonien* s'il possède un cycle hamiltonien.

1.1.2 Connexité

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

Un *ensemble d'articulation* est un ensemble de sommets dont la suppression disconnecte le graphe et qui est minimal pour cette propriété. Un point d'articulation est un sommet v tel que {v} est un ensemble d'articulation. Soit k un entier naturel, G est dit *k-connexe* si tous ses

ensembles d'articulation sont de cardinal supérieur ou égal à k. La *connectivité* d'un graphe est le minimum des cardinaux de ses ensembles d'articulation.

1.1.3 Planarité

Il existe une classe de graphes vérifiant la propriété suivante : ils peuvent être dessinés sur un plan de manière que les sommets soient des points du plan, les arêtes des courbes simples, et deux arêtes quelconques n'ont pas d'intersections communes autres que les sommets qu'elles peuvent avoir en commun. Ces graphes sont appelés des *graphes planaires.*

1.2 Homomorphisme de graphes

Après un bref rappel théorique sur la catégorie des graphes non orientés, nous donnons différentes définitions sur l'homomorphisme et ses propriétés.

1.2.1 Définitions de base

Définition 1

Soient $G = (V, E)$ et $G' = (V', E')$ des graphes non orientés, un *homomorphisme f* de G dans G' est une application de V dans V' telle que des sommets u et v adjacents dans G ont pour images deux sommets $f(u)$ et $f(v)$ adjacents dans G'.

Remarque

En particulier, s'il existe une boucle au sommet u de G, alors il existe une boucle au sommet $f(u)$ de G'.

Un exemple important d'homomorphisme est la donnée d'une *coloration* des sommets d'un graphe G avec n couleurs. D'une manière plus précise, soit n un entier, un graphe $G = (V, E)$ est dit *n-colorable* si on peut associer à chaque sommet de G une couleur prise dans $\{1, \ldots, n\}$ de sorte que deux sommets adjacents aient une couleur différente. Une telle coloration peut être interprétée comme un homomorphisme c de G dans le graphe complet K_n construit sur les sommets $\{1, \ldots, n\}$, et réciproquement, un homomorphisme $c : G \rightarrow K_n$ définit une n-coloration de G.

La notion d'homomorphisme de graphes permet de définir la catégorie des graphes notée *Graphe* comme étant la catégorie dont les objets sont des graphes $G = (V, E)$ non orienté sans boucle et sans arêtes multiples et dont les morphismes de G dans G' sont les homomorphismes de G dans G'. Le lecteur pourra trouver une introduction à la théorie de la catégorie des graphes dans [Hell 79].

1.2.2 Autre formulation

En vue d'étendre la notion d'homomorphisme au graphe multiple, nous donnons une définition légèrement différente à la notion d'homomorphisme de graphes. Cette définition particulière servira pour caractériser le produit fibré étudié au prochain paragraphe et l'opération de contraction de graphes définie dans le dernier paragraphe.

Définition 2

Soient $X = \{x_1, x_2, \ldots, x_n\}$ et $Y = \{y_1, y_2, \ldots, y_m\}$ deux ensembles. A toute application f de X dans Y est associée une matrice $n \times m$, notée $M(f)$ dont l'élément $m_{i,j}$ est égal à 1 si $f(x_i) = y_j$ et 0 sinon. Cette matrice $M(f)$ est une matrice 0-1 ayant exactement un seul 1 par ligne. Cette représentation matricielle des applications est utilisée par la suite pour caractériser les matrices représentatives d'homomorphisme de graphes.

Définition 3

Soient $G = (V, E)$ et $G' = (V', E')$ des graphes non orientés, possédant éventuellement des boucles et des arêtes multiples. Un homomorphisme h de G dans G' est la donnée d'un couple d'applications (f,g) où f est une application de l'ensemble V dans l'ensemble V' et g une application de l'ensemble E dans l'ensemble E' telles que quelque soit l'arête e d'extrémités u et v alors l'arête $g(e)$ a pour extrémités $f(u)$ et $f(v)$:

$$e = (u, v) \Leftrightarrow g(e) = (f(u), f(v)).$$

Remarques

1) Soient G et G' sont des graphes simples et f une application de V dans V' telle que pour toute arête $e = (u,v)$ de G, $(f(u),f(v))$ est une arête de G'. Si on définit l'application g de E dans E' par $g(e) = (f(u),f(v))$ pour $e = (u,v)$, alors $h = (f,g)$ est un homomorphisme de G dans G', au sens de la définition 3.

2) Soit $h = (f,g)$ un homomorphisme de G dans G'. De manière générale, f bijective n'implique pas nécessairement g bijective. Une contre-exemple est le cycle C_5 dans le graphe complet K_5.

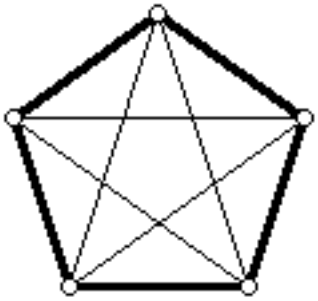


Figure 4.1 : Cycle C_5 (en gras) dans le graphe K_5.

3) Soit $h = (f,g)$ un homomorphisme de G dans G'. De manière générale, g bijective n'implique pas f bijective. Pour un graphe simple G non complet l'identification de deux sommets non adjacents fournit un contre-exemple, comme le montre la figure 4.2.

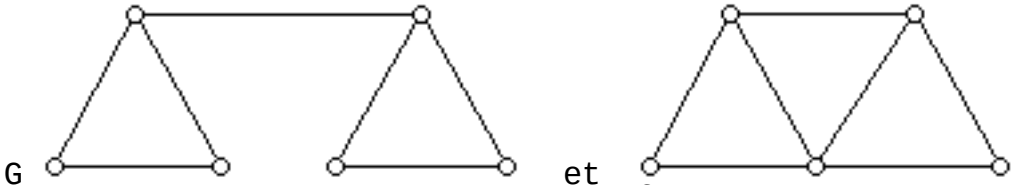


Figure 4.2 : Deux graphes G et G'.

4) Soit $h = (f,g)$ un homomorphisme de G dans G'. Si les applications f et g sont toute deux bijectives alors h est un isomorphisme dans la catégorie *Graphe*.

5) Les définitions 1 et 3 sont équivalentes lorsqu'on se limite aux graphes simples.

Théorème

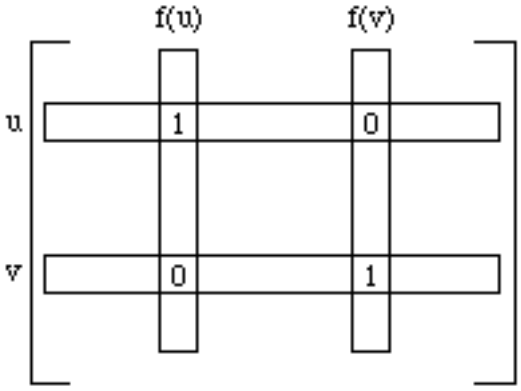
Soient G et G' deux graphes sans boucles, et deux applications f de V dans V' et g de E dans E'. L'application $h = (f,g)$ est un homomorphisme de graphes si et seulement si :

$$Inc(G').M(g)^t = M(f)^t.Inc(G)$$

où Inc(G) est la matrice d'incidence du graphe G.

Remarque

Pour tout homomorphisme $h = (f,g)$ de G dans G' la matrice $M(f)$ possède la propriété suivante: pour toute arête $e = (u,v)$ de G, on a :



Puisque $e = (u,v)$ est une arête (u et v sont adjacents) alors on a que $f(u) \neq f(v)$, ce qui est symbolisé par le dessin ci-dessus.

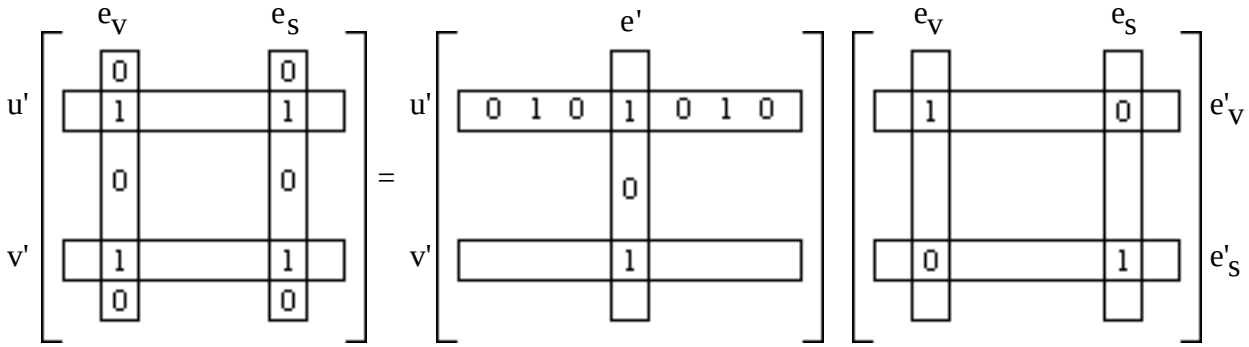
Démonstration

Soit h un homomorphisme de graphes entre $G = (V, E)$ et $G' = (V', E')$. Regardons plus précisément les produits matriciels de $\mathrm{Inc}(G').M_g^t$ et de $M_f^t.\mathrm{Inc}(G)$ en détails.

• Posons $A = \mathrm{Inc}(G').M(g)^t$, avec la matrice $A = (A_{u'e})$, $u' \in V'$ et $e \in E$

La matrice A est de taille $n' \times m$ et elle est définie par :

$$A_{u'e} = \begin{cases} 1, \text{ s'il existe e' = (u',.) dans E' telle que g(e) = e'} \\ O, \text{ sinon} \end{cases}$$



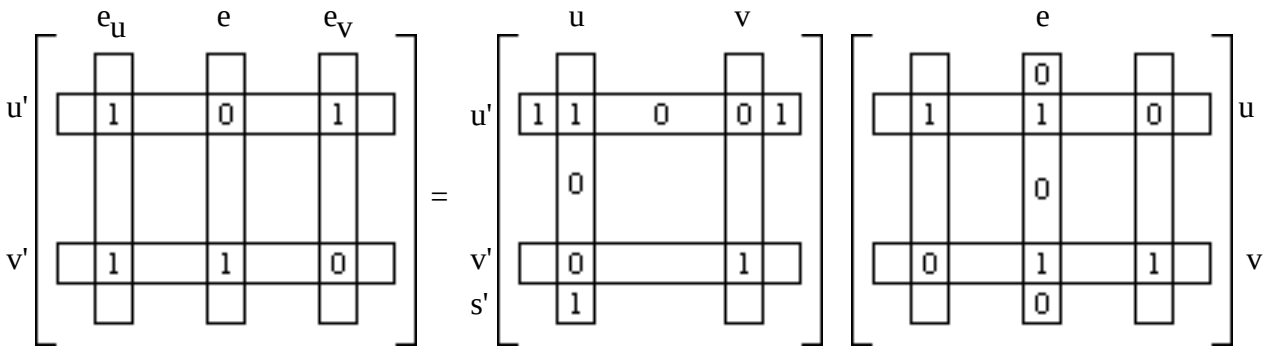
Pour le nouveau graphe associé à la matrice A, le sommet u' est incident à l'arête e s'il existe une arête $e'=(u',.)$ dans E' telle que $g(e)=e'$. Cette matrice ne possède que deux éléments égaux à 1 par colonnes.

• Posons $B = M(f)^t.\mathrm{Inc}(G)$, avec la matrice $B = (B_{u'e})$, $u' \in V'$ et $e \in E$

Les valeurs possibles pour $B_{u'e}$ sont 2, 1 ou 0. La valeur 2 est à exclure par définition d'un homomorphisme.

La matrice *B* est de taille *n'×m* et elle est définie par :

$$B_{u'e} = \begin{cases} 1, \text{ s'il existe u dans V telle que f(u) = u' et e = (u,.)} \\ O, \text{ sinon} \end{cases}$$



Pour le nouveau graphe associé à la matrice *B*, le sommet *u'* est incident à l'arête *e* s'il existe un sommet *u* dans V telle que *f(u)=u'* et *e=(u,.)* . Cette matrice ne possède que deux 1 par colonne.

Les matrices *A* et *B* sont identiques. Etant chacunes des matrices 0-1, de même dimension n'×m et ayant chacune deux 1 par colonne, il suffit de voir que les 1 se correspondent :

A(*u'*,*e*) = 1 si et seulement si il existe une arête *e'* = (*u'*,.) de *E'* telle que *g*(*e*) = *e'* ; or par définition de l'homomorphisme *h* = (*f*,*g*) on a alors les égalités suivantes :
(*u'*,.) = *e'* = *g*(*e*) = *g*((*u*,.)) = *f*(*u*)*f*(.) = (*u'*,*f*(.))
et donc *B*(*u'*,*e*) = 1.

Soit *h* = (*f*,*g*) une application entre deux graphes *G* et *G'* telle que :
$(D_{u'e}) = Inc(G').M(g)^t = M(f)^t.Inc(G) = (H_{u'e})$

Les matrices produits *A* et *B* sont des matrices *n'×m* en 0-1. Par le calcul précédent et la remarque (i), l'application *h* définit bien un homomorphisme.

On peut remarquer que la matrice produit représente la matrice d'incidence du sous-graphe partiel multiple (image de *G* dans *G'* par *h*).

1.3 Produit de graphes

Dans la catégorie des graphes, plusieurs opérations sur les objets que sont les graphes peuvent être définies. Nous présentons deux de ces opérations considérées comme classiques.

Définition

Soient $G = (V, E)$ et $G' = (V', E')$ deux graphes, le *produit croisé* de G et G' est le graphe noté $G \times G' = (V \times V', E_{cr})$ tel que :

$$((x,x'),(y,y')) \in E_{cr} \Leftrightarrow (x,y) \in E \text{ et } (x',y') \in E'$$

Le produit ainsi défini, possède les propriétés canoniques de produit cartésien dans la catégorie des graphes. C'est pourquoi Berge [Berge 83] utilise le terme de *produit cartésien*.

Le produit suivant est sans doute celui qui a été le plus étudié :

Définition

Soient $G = (V, E)$ et $G' = (V', E')$ deux graphes, le *produit carré* de G et G' est le graphe noté $G \square G' = (V \times V', E_{ca})$ tel que :

$$((x,x'),(y,y')) \in E_{ca} \Leftrightarrow \begin{cases} x = y \text{ et } (x',y') \in E' \\ \text{ou} \\ x' = y' \text{ et } (x,y) \in E \end{cases}$$

Berge utilise le terme de *somme cartésienne* pour ce produit, cependant dans la littérature on peut trouver le terme de *produit cartésien.* Afin d'éviter toute confusion, nous avons choisi les termes de produit carré et produit croisé.

2. Le produit fibré

L'évaluation du nombre chromatique du produit carré de deux graphes est connue (théorème de Vizing, voir [Berge 82]), alors que la même question, pour le produit croisé proposée par Hedetniemi [Hedetniemi 66], reste encore un problème ouvert.

Dans ce paragraphe, nous étudions un produit initialement introduit par Hell [Hell 79], le produit fibré [Khelladi & al. 96].

2.1. Définition

Etant donné un graphe H, Hell [Hell 79] a défini une nouvelle catégorie notée Gra_H où les objets sont les homomorphismes d'un graphe G dans H et les morphismes de Gra_H de l'objet $f_1 : G_1 \rightarrow H$ dans l'objet $f_2 : G_2 \rightarrow H$ sont les homomorphismes de graphes $g : G_1 \rightarrow G_2$ tels que $f_1 = g \circ f_2$ (voir la figure 4.3).

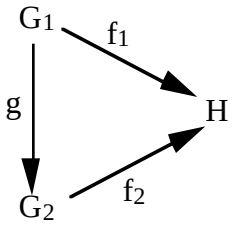


Figure 4.3 : Diagramme.

Une question sur les catégories de graphes est de trouver le produit cartésien correspondant. Dans le paragraphe sur les produits de graphes, on a vu que le produit croisé est le produit cartésien dans la catégorie des graphes. Hell [Hell 72] vérifie que le produit cartésien dans la catégorie Gra_H , nommé *produit fibré*, existe. Ce produit est donc un graphe et un morphisme de ce graphe dans le graphe *H*. Le graphe produit fibré est défini comme suit :

Définitions 4

Soient *H*, *G* et *G'* trois graphes, $h : G \to H$ et $h' : G' \to H$ deux homomorphismes. Le *produit fibré* de *G* et *G'* par rapport à *h* et *h'* est le graphe noté $G \times_H G'$ où l'ensemble des sommets est :

$$V(G \times_H G') = \{ (u,u') \in V(G) \times V(G') \mid h(u) = h'(u') \},$$

et deux sommets (*u*,*v*) et (*u'*, *v'*) sont adjacents si et seulement si *u* est adjacent à *v* dans *G* et *u'* est adjacent à *v'* dans *G'*.

Remarques

1) L'écriture $G \times_H G'$ sous-entend l'existence des homomorphismes *h* et *h'*. Sauf cas contraire, ces homomorphismes ne sont pas expliciter, afin de simplifier l'écriture. Le graphe $G \times_H G'$ est ainsi bien défini.

2) Par définition, le graphe $G \times_H G'$ est le sous-graphe de $G \times G'$ induit par $V(G \times_H G')$. Nous établirons plus précisément la relation entre le produit croisé et le produit fibré.

Comme le produit fibré est le produit cartésien dans la catégorie Gra_H il hérite des propriétés canoniques du produit cartésien. On rappelle quelques unes de ces propriétés :

Propriétés 2.1:

(i) Pour tous graphes *H*, G_1, G_2 et G_3 on a

$$(G_1 \times_H G_2) \times_H G_3 = G_1 \times_H (G_2 \times_H G_3) \qquad \text{(associativité)}$$

(ii) Si G'_i (i = 1, 2) est un sous-graphe (resp. graphe partiel) de G_i alors $G'_1 \times_H G'_2$ est un sous-graphe (resp. graphe partiel) de $G_1 \times_H G_2$.

(iii) Si $h : H \to H'$ est un homomorphisme alors tout graphe $G \times_H G'$ est un sous-graphe de $G \times_{H'} G$ défini par composition avec h des homomorphismes qui définissent $G \times_H G$.

(iv) Si H est un graphe d'ordre n, alors le produit fibré $G \times_H G'$ défini par les homomorphismes $f : G \to H$ et $f' : G' \to H$ est égal au produit fibré $G \times_{K_n} G'$ défini par les homomorphismes $f \circ I_H$ et $f' \circ I_H$, où I_H est l'injection canonique de H dans K_n.

Cette dernière propriété nous permet de supposer que la fibre H est une clique à |V(H)| sommets. De plus, un homomorphisme de G dans K_n est en fait une n-coloration de G. Donc par la suite, étant donné une n-coloration de G et une n-coloration de G' on étudiera le graphe $G \times_{K_n} G'$.

On donne une interprétation de la définition de $G \times_{K_n} G'$ en introduisant de nouvelles notations :

(1) Les sommets de $G \times_{K_n} G'$ sont partitionnés en n stables notés $N_i = S_i \times S'_i$ (i = 1, …, n) où S_i (resp. S'_i) est l'ensemble des sommets de G (resp. G') coloriés par la couleur i.

(2) L'ensemble des arêtes liant N_i à N_j dans $G \times_{K_n} G'$ est définie par : si $(u, v) \in E(G)$ avec $u \in S_i$, $v \in S_j$ et si $(u', v') \in E(G')$ avec $u' \in S'_i$, $v' \in S'_j$ alors $((u, u'), (v, v')) \in E(G \times_{K_n} G')$

(3) Si p est le nombre d'arêtes entre S_i et S_j dans G, q le nombre d'arêtes entre S'_i et S'_j dans G', alors le nombre d'arêtes entre N_i et N_j dans $G \times_{K_n} G'$ est $r = p.q$.

(4) Si G est un graphe k-chromatique ($k \leq n$) alors $G \times_{K_n} K_n$ est isomorphe à G quelle que soit la n-coloration de G choisie.

La figure 4.4 illustre les propriétés (1) et (2)

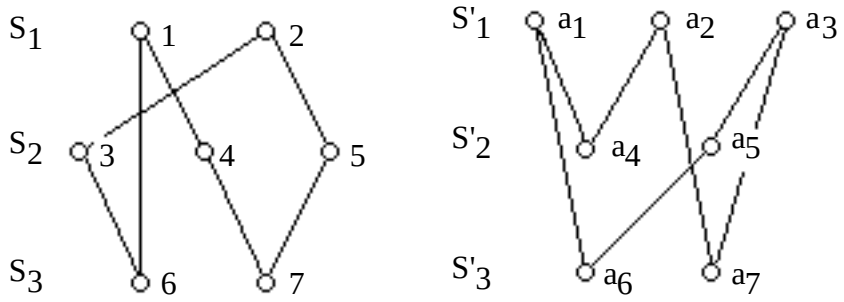


Figure 4.4 : Une 3-coloration de deux cycles impairs.

Alors, le graphe $G \times_{K_n} G'$ peut être obtenu par la partition de n stables $N_i = S_i \times S'_i$ (où le symbole $\times$ correspond au produit cartésien usuel entre deux ensembles) et les arêtes entre N_i et N_j de ce graphe sont définies par le produit croisé des paires S_i, S_j et S'_i, S'_j (voir la figure 4.5). Cela signifie que le sous-graphe de $G \times_{K_n} G'$ réduit par $N_i \cup N_j$ est le sous-graphe du produit croisé $G \times G'$ induit par $S_i \times S'_i \cup S_j \times S'_j$.

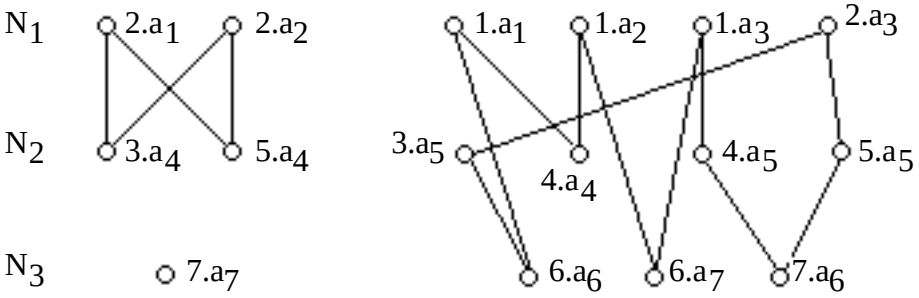


Figure 4.5 : Le produit croisé des cycles impairs de la figure 4.4.

Cet exemple montre que le produit fibré de deux graphes hamiltoniens n'est pas nécessairement hamiltonien. De cette nouvelle interprétation plusieurs propriétés apparaissent directement (par définition du produit croisé):

Propriété 2.2

Soient G et G' deux graphes n-colorables

(i) Si G^* est un sous-graphe partiel de G alors $G^* \times_{K_n} G'$ est un sous-graphe partiel de $G \times_{K_n} G'$.

(ii) Si G^* est un sous-graphe induit de G alors $G^* \times_{K_n} G'$ est un sous-graphe induit de $G \times_{K_n} G'$.

Propriété 2.3

Si p est le nombre d'arêtes entre S_i et S_j dans G, q le nombre d'arêtes entre S'_i et S'_j dans G', et n le nombre d'arêtes entre N_i et N_j dans le graphe produit $G \times_{K_n} G'$ alors $n = p.q$.

Propriété 2.4

Si G et G' sont deux graphes n-chromatiques alors $\chi(G \times_{K_n} G') \leq n$.

Propriété 2.5

Si G est un graphe k-chromatique ($k \leq n$) alors $G \times_{K_n} K_n$ est isomorphe à G.

2.2. Propriétés

Dans cette section, nous donnons quelques propriétés sur le produit fibré de graphes eulériens ou hamiltoniens.

2.2.1. Propriétés eulériennes

On dira qu'un graphe G est *eulérien* (non nécessairement connexe) si tous les sommets de G sont de degré pair.

Proposition 2.6

Soient G et G' deux graphes n-chromatiques et c (resp. c') une n-coloration de G (resp. G'). Si u (resp. u') est un sommet de G (resp. G') tel que $c(u) = c(u')$ et si r_i (resp. q_i) est le nombre de voisins de u (resp. u') dans S_i (resp. S'_i) alors le degré du sommet (u, u') dans $G \times_{K_n} G'$ est égal à $\sum_{i=1}^{n} r_i.q_i$.

Démonstration

La proposition 2.6 est une conséquence de la *propriété 2.3*, lorsqu'on restrient le compte des arêtes entre deux niveaux du sous-graphe induit par le sommet (u, u') et son voisinage dans $G \times_{K_n} G'$.

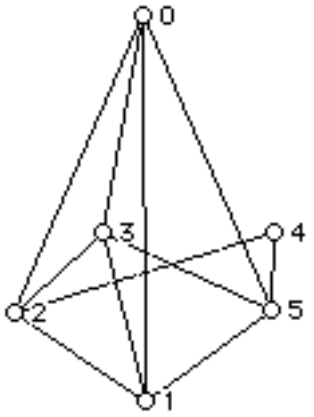
Corollaire 2.7

Soient G et G' deux graphes 3-chromatiques, si G et G' sont eulériens alors, pour toute 3-coloration de G et toute 3-coloration de G', le graphe $G \times_{K_3} G'$ est eulérien.

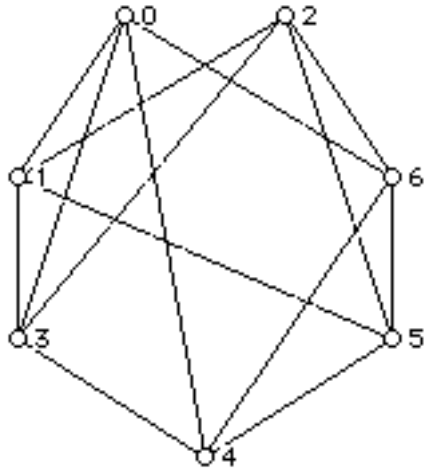
Démonstration

Soit (u, u') un sommet de $G \times_{K_3} G'$ Sans perdre de généralité on peut supposer que (u, u') est dans N_1 (c'est à dire $u \in S_1$ et $u' \in S'_1$). Par hypothèse, les graphes G et G' sont eulériens, donc, avec les mêmes notations que pour la *proposition 2.6*, r_2 et r_3 (resp. q_2 et q_3) ont même parité, car les graphes sont 3-coloriables. Ainis, $r_2.q_2$ et $r_3.q_3$ ont même parité et la *proposition 2.6* montre que le degré de (u, u') est $r_2.q_2 + r_3.q_3$ qui est un nombre pair.

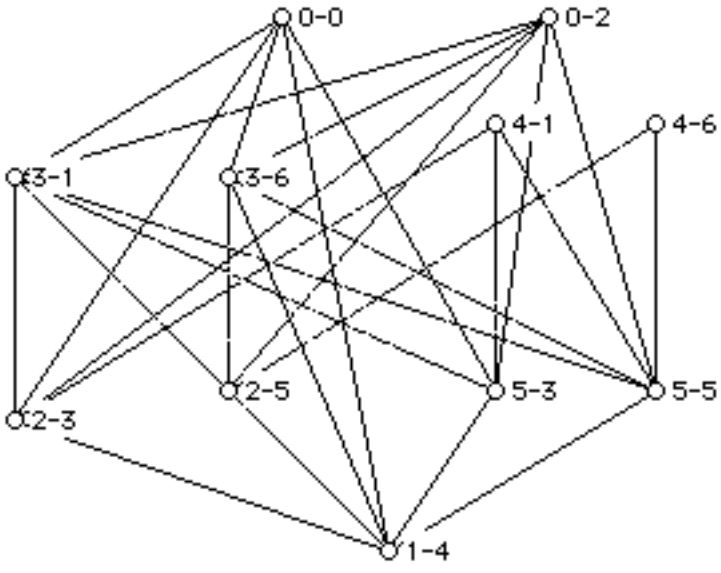
L'exemple de la figure 4.6 montre que le corollaire n'est valide que pour des graphes 3-chromatiques. En effet, pour deux graphes 4-chromatiques eulériens G et G', le produit fibré $G \times_{K_4} G'$ n'est pas nécessairement eulérien.



(a) G eulérien et 4-chromatique



(b) G' eulérien et 4-chromatique



(c) Le graphe $G \times_{K_4} G'$produit fibré n'est pas eulérien.

Figure 4.6 : Le produit fibré de graphe eulérien n'est pas nécessairement eulérien.

Pour les colorations décrites dans la figure 4.6 (une couleur par niveau), on peut vérifier que les sommets d'incide (0-0) ou (4-1) sont de degré respectif 5 et 3, donc $G \times_{K_4} G'$ n'est pas eulérien (même s'il est 4-chromatique).

2.2.2. Propriétés d'hamiltonicité

Le problème de déterminer si un graphe contient un cycle hamiltonien, est fondamental en théorie des graphes. Aubert et Schneider ([Aubert & Schneider 82]) ont étudié le produit carré de deux cycles hamiltoniens. Nous donnons un théorème analogue pour le produit fibré.

Théorème 2.8

Soit *C* un cycle et *G* un graphe dont l'ensemble des arêtes peut être décomposé en 2 cycles hamiltoniens. Alors $C \times_{K_2} G$ peut être décomposé en trois cycles hamiltoniens.

La décomposition du produit croisé en sous-graphes hamiltoniens a partiellement été étudiée par Kouïder et Kheddouci [Kouïder & Kheddouci 95]. Ils ont en particulier montré le théorème 2.9 :

Théorème 2.9

Les graphes suivants sont décomposables en cycles hamiltoniens :

- $K_{m,\,m} \times K_n$, pour tout entier $m \geq 3$ et tout entier n impair,
- $G \times C_n$, pour tout entier n impair et tout graphe *G* 3-régulier fortement hamiltonien[1].

Nous donnons un théorème analogue pour le produit fibré.

Théorème 2.10

Si *C* et *C'* sont deux cycles pairs alors, pour toute bicoloration de *C* et toute bicoloration de *C'*, le graphe $C \times_{K_2} C'$ est l'union arête-disjointe de deux cycles hamiltoniens.

Démonstration

D'abord, le produit fibré $C \times_{K_2} C'$ est connexe. Pour montrer la partition en deux cycles arête-disjoints, on considère la bicoloration des sommets de C (resp. C') suivante $S_1 = \{1, \ldots, k\}$, $S_2 = \{k+1, \ldots, 2k\}$ (resp. $S'_1 = \{a_1, \ldots, a_{k'}\}$, $S'_2 = \{a_{k'+1}, \ldots, a_{2k'}\}$), les arêtes de C (resp. C') étant de la forme $i(i + k)$, $1 \leq i \leq k$

[1] *G* est fortement hamiltonien si et seulement si par chaque arête de *G* il passe un cycle hamiltonien.

et i(i + k - 1), $2 \le i \le k$ et l'arête liant 1 à 2k (resp. $a_i a_{k'+i}$, $a_i a_{k'+i-1}$ et l'arête liant a_1 à $a_{2k'}$ dans C'). Le premier cycle hamiltonien est obtenu comme l'union des chaînes P_i liant les sommets (i,a_1) à $((i+k),a_{2k'})$ $(i = 1, \dots, k)$ décrites par les séquences de sommets du produit fibré :

$(i,a_1), ((i+k),a_{k'+1}), (i,a_2), ((i+k),a_{k'+2}), \dots, (i,a_{k'}), ((i+k),a_{2k'})$.

Les chaînes P_i et P_{i-1} $(i = 2, \dots, k)$ sont liées par les arêtes $((i+k-1), a_{2k'})(i,a_1)$ dans $C \times_{K_2} C'$ et P_1 et P_k par l'arête $(1,a_1)(2k,a_{2'})$.

Le second cycle hamiltonien est obtenu comme l'union des chaînes Q_i $(i = 2, \dots, k)$ liant les sommets (i,a_1) aux sommets $((i+k-1),a_{2k'})$ décrites par les séquences de sommets du produit fibré :

$(i,a_1), ((i+k-1),a_{k'+1}), (i,a_2), ((i+k-1),a_{k'+2}), \dots, (i,a_{k'}), ((i+k-1),a_{2k'})$, et la chaîne $Q_1 = (1,a_1), (2k,a_{k'+1}), (1,a_2), (2k,a_{k'+2}), \dots, (1,a_{k'}), (2k,a_{2k'})$.

Les chaînes Q_i et Q_{i-1} $(i = 1, \dots, k)$ sont liées par les arêtes $(i,a_1)(k+i,a_{2k'})$ dans $C \times_{K_2} C'$.

Par construction, ces deux cycles sont arête-disjoints. De plus, par la proposition 2.1, le graphe $C \times_{K_2} C'$ est 4-régulier, ce qui complète la preuve.

Corollaire 2.11

Si *G* et *G'* sont des graphes bipartis qui peuvent être partitionnés en *k* (resp. *k'*) cycles hamiltoniens alors $G \times_{K_2} G'$ peut être partitionné en *2kk'* cycles hamiltoniens.

Démonstration

On note $G = \bigcup_{i=1}^{k} C_i$ et $G' = \bigcup_{j=1}^{k'} C'_j$ où C_i et C'_j sont des cycles hamiltoniens. D'après le théorème 2.10 appliqué à chaque paire (C_i, C_j), on obtient une décomposition de $G \times_{K_2} G'$ en 2kk' cycles hamiltoniens, car $G \times_{K_2} G'$ est l'union des $C_i \times_{K_2} C'_j$ $(i = 1, \dots, k$ et $j = 1, \dots, k')$. De plus, G (resp. G') est 2k (resp. 2k') régulier et biparti, donc d'après la proposition 2.6, $G \times_{K_2} G'$ est 4kk' régulier.

Nous pouvons reformuler le corollaire 2.11 en remplaçant cycles hamiltoniens par 2-facteurs[2].

---

[2] Un 2-facteur du graphe *G* est un graphe partiel de *G* où les sommets sont tous de degré 2.

2.2.3. Lien entre produit fibré et produit croisé

On a remarqué au paragraphe 2.1 que le produit fibré de deux graphes est un sous-graphe du produit croisé. Dans cette section, nous détaillons ce lien existant entre produit fibré et produit croisé. On introduit au préalable une opération binaire :

Définitions

1) Un graphe G à n sommets est dit *étiqueté* si et seulement si G est muni d'une numérotation sans ambiguïté (deux sommets distincts ont une étiquette différente) des sommets par une liste de n symboles.

2) Soient G et G' deux graphes étiquetés, la *somme* de G et G' notée $G \oplus G'$ est le graphe obtenu après avoir pris une copie de G et une copie de G', avoir identifié les sommets ayant la même étiquette et supprimé la multiplicité des arêtes.

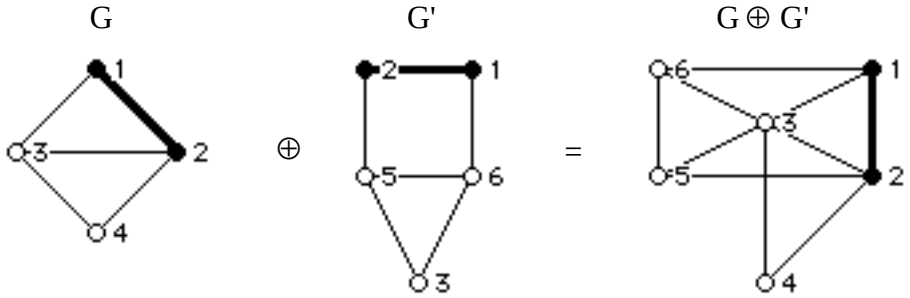


Figure 4.7 : Le graphe $G \oplus G'$.

Cette opération est une simple extension de la notion d'identification (contraction) de sommets au cas de graphes étiquetés. Le graphe $G \oplus G'$ admet un étiquetage "compatible" avec ceux de G et G' (voir exemple de la figure 4.7) induit par le fait que les graphes G et G' sont des graphes partiels de $G \oplus G'$.

Contrairement aux sections précédentes, nous avons besoin de préciser les homomorphismes mis en jeu dans le produit fibré, c'est pourquoi nous introduisons la notation supplémentaire suivante :

Etant donné une n-coloration c_1 de G_1, une n-coloration c_2 de G_2 et une permutation σ de $\{1, \ldots, n\}$, le produit fibré de G_1 par G_2 par rapport à c_1 et σ o c_2 est noté $G_1 \times_{K_n} G_2^{\sigma}$. On munit $G_1 \times_{K_n} G_2^{\sigma}$ de l'étiquetage induit par la définition du produit fibré (voir le paragraphe 2.4.4 *Mise en œuvre* de l'algorithme pour plus de détails).

Le résultat suivant signifie que le produit fibré est un produit croisé de stables des graphes considérés.

Théorème 2.12

Etant donné une *n*-coloration de G_1 et une *n*-coloration de G_2, si σ_n est l'ensemble des permutations sur $\{1, \ldots, n\}$ alors, pour tout étiquetage de G_1 et tout étiquetage de G_2, on a $G_1 \times G_2 = \bigoplus_{\sigma \in \sigma_n} (G_1 \times_{K_n} G_2^{\sigma})$.

Démonstration

On note pour $i = 1, 2$, V_i l'ensemble des sommets, n_i le nombre de sommets et $S_i^1, \ldots, S_i^n$ la n-coloration c_i de G_i. Sans perte de généralité on peut supposer que l'étiquetage de G_i est décrit par $V_i = \{1, \ldots, n_i\}$. Notons V l'ensemble des sommets et E l'ensemble des arêtes de $\bigoplus_{\sigma \in \sigma_n} (G_1 \times_{K_n} G_2^{\sigma}$.

Par définition du produit fibré et de l'étiquetage compatible de la somme, on a que $V \subset V_1 \times V_2$. Soit $(u,x) \in V_1 \times V_2$, il existe i $(1 \le i \le n)$ et j $(1 \le j \le n)$ tels que $u \in S_1^i$ et $x \in S_2^j$. On définit la permutation σ telle que $\sigma(i) = j$, $\sigma(j) = i$ et pour tout $k \ne i$ et $k \ne j$ on pose $\sigma(k) = k$ (σ est la transposition de i et j). On obtient alors $(u, x) \in V(G_1 \times_{K_n} G_2^{\sigma}) \subset V$. Donc $V = V(G_1 \times G_2)$.

Soit $((u, x), (v, y))$ une arête de $G_1 \times G_2$. On peut supposer que $u \in S_1^1$, $v \in S_1^2$, $x \in S_2^i$ et $y \in S_2^j$, par définition du produit croisé $i \ne j$. On définit la permutation σ telle que $\sigma(i) = 1$, $\sigma(j) = 2$, $\sigma(1) = i$, $\sigma(2) = j$ et pour tout $k \ne 1, 2, i,$ et j on pose $\sigma(k) = k$, comme $G_1 \times_{K_n} G_2^{\sigma})$ est un sous-graphe de $G_1 \times G_2$ on a $((u, x), (v, y)) \in E$. Donc $E \supset E(G_1 \times G_2)$.

Soit $e = ((u, x), (v, y))$ une arête de E. Par définition de la somme, il existe $\sigma \in \sigma_n$ tel que $e \in E(G_1 \times_{K_n} G_2^{\sigma})$, de plus $G_1 \times_{K_n} G_2^{\sigma}$ est un sous-graphe de $G_1 \times G_2$, donc $e \in E(G_1 \times G_2)$.

Lorsque n = 2, si on note $G_i = (V_i, E_i)$ avec $V_i = S_i^1 \cup S_i^2$ pour i = 1, 2 où S_i^k est le stable des sommets colorés k dans la bipartition de G, alors d'après le *théorème 2.12*, on a $G_1 \times G_2 = \bigcup_{\sigma \in \sigma_2} G_1 \times_{K_2} G_2^\sigma$ car $V(G_1 \times G_2)$ est l'union disjointe des $S_1^i \times S_2^j$ pour tout $1 \le i, j \le 2$. En d'autres termes, lorsque les G_i sont connexes alors les deux composantes connexes de $G_1 \times G_2$ sont complètement définies par les deux produits fibrés $G_1 \times_{K_2} G_2^\sigma$.

Au vu de cette remarque et du *corollaire 2.11*, on en déduit le corollaire immédiat suivant :

Corollaire 2.13
Si le graphe G_1 (resp. G_2) est biparti et décomposable en k (resp. k') 2-facteurs alors le graphe $G_1 \times G_2$ est décomposable en $4kk'$ 2-facteurs.

Le théorème 2.13 donne une nouvelle approche pour aborder le produit croisé. Il justifie que l'on s'intéresse au produit fibré. Et cette nouvelle perception du produit croisé va permettre de développer certains résultats sur le problème d'hamiltonicité.

2.3. Coloration du produit fibré

Une k-coloration des sommets d'un graphe G est une application $c : V \to \{1, \ldots, k\}$ telle que pour toute arête $e = (u, v)$ de G $c(u) \neq c(v)$. Le *nombre chromatique* $\chi(G)$ est le plus petit entier k tel que G admette une k-coloration. Pour la classe de graphes considérés, différents produits sont définis [Turzik 83] et pour chaque produit le problème du calcul du nombre chromatique en fonction des nombres chromatiques des graphes composants est posé. Pour certains produits dont le produit carré (appelé aussi somme cartésienne dans [Berge 73]) ce problème est résolu. Dans ce travail quelques uns de ces résultats sont rappelés.

Pour deux graphes G et G', on peut vérifier que $\chi(G \times G') \le \min(\chi(G), \chi(G'))$ en utilisant les projections canoniques $p : G \times G' \to G$ et $p' : G \times G' \to G'$. Hedetniemi [Hedetniemi 66] a conjecturé l'égalité. Cette conjecture peut être reformulée ainsi :

Conjecture d'Hedetniemi (1966)
Pour deux graphes n-chromatiques quelconques G et G' on a $\chi(G \times G') = n$.

Des résultats partiels ont été obtenus sur cette conjecture, elle a été montrée lorsque $n \le 4$ (voir [Burr & al. 76], [Duffud & al. 85] et [El-Zahar & Sauer 85]), mais elle reste ouverte pour

$n \geq 5$. Khelladi suggère d'étudier le produit fibré et propose dans [Khelladi 92] une conjecture plus forte que celle d'Hedetniemi.

Conjecture de Khelladi (1992)

Pour tous graphes n-chromatiques G et G' et toute n-coloration de G et n-coloration de G', le graphe $G \times_{K_n} G'$ est aussi n-chromatique.

On sait que : - $G \times_{K_n} G'$ est un sous-graphe de $G \times G'$ et

$$- \chi(G \times_{K_n} G') \leq \chi(G \times G') \leq \min(\chi(G), \chi(G')),$$

donc la conjecture de Khelladi implique celle de Hedetniemi. Mais on pourrait donner une conjecture plus faible : "Pour tous graphes n-chromatiques G et G', et il existe des n-colorations de G et de G' telles que le graphe $G \times_{K_n} G'$ est aussi n-chromatique". Cette conjecture implique aussi la conjecture d'Hedetniemi.

La difficulté de la conjecture de Khelladi est due au fait que l'on doit vérifier l'assertion pour toutes les n-colorations des graphes G et G'. Dans sa thèse[Semri 93], Semri expose quelques propriétés du produit fibré, dont le résultat suivant :

Proposition 3.1

Si G et G' sont deux graphes n-chromatiques avec $\omega(G) = n$ alors pour toutes les n-colorations de G et de G' on a $\chi(G \times_{K_n} G') = n$.

Démonstration

On sait déjà que $\chi(G \times_{K_n} G') \leq n$. Et d'autre part, si G_n est une n-clique de G alors $G_n \times_{K_n} G'$ est isomorphe à G', d'où le résultat.

Pour $n = 3$, on utilise le fait que tout graphe 3-chromatique contient un cycle impair afin de démontrer la conjecture de Khelladi.

Théorème 3.2

Si C et C' sont deux cycles impairs alors

(i) chaque sommet de $C \times_{K_3} C'$ est de degré 0, 2 ou 4,

(ii) $C \times_{K_3} C'$ est un graphe eulérien et il possède un nombre impair d'arêtes.

Démonstration

Soient S_i (resp. S'_i) pour i = 1, 2, 3 les trois stables de C (resp. C'). soit u un sommet de S_i et u' un sommet de S'_i, le degré de (u, u') dans $C \times_{K_3} C'$ est égal à :

- 0 si les deux voisins de u sont dans S_k et les deux voisins de u' sont dans S'_t avec $t \neq k$.
- 4 si les deux voisins de u sont dans S_k et les deux voisins de u' sont dans S'_k.
- 2 sinon.

Afin d'achever la preuve, nous avons besoin des lemmes suivants :

Lemme 3.3

Si $G = (V, E)$ est un graphe 3-arête colorable avec des sommets de degré soit un soit trois, alors pour toute 3-arête coloration de G et pour tout ensemble X de sommets de G de degré trois, si E_i est l'ensemble des arêtes colorées i entre X et V-X, alors |X| et $|E_i|$ ont la même parité.

Démonstration

L'ensemble V peut être partitionné en $S \cup W$ où S est l'ensemble des sommets de degré un et W l'ensemble des sommets de degré trois. Si |X| = 0 alors la propriété est valide. Maintenant supposons que |X| > 0. Soit v un sommet de X et considérons l'ensemble X' = X-{v}. La cardinalité de l'ensemble E'_i des arêtes colorées i liant X' à V-X' a la même parité que |X'| par récurrence sur |X'|. Soit $e = (v, x)$ l'unique arête colorée i adjacente à v. Si x appartient à X' alors e appartient à E'_i, donc $E_i = E'_i$-{e} ; sinon $E_i = E'_i \cup \{e\}$. Dans tous les cas $|E_i| = |E_i'| \pm 1$ a la même parité que |X| = |X'| + 1.

Lemme 3.4

Pour toute 3-coloration de sommets d'un cycle impair C en S_1, S_2, et S_3, et pour $i, j = 1, 2, 3$ ($i \neq j$), le nombre d'arêtes reliant S_i à S_j est impair.

Démonstration

On note les sommets de C par la suite $u_1, \ldots, u_{2r+1}$. On oriente C selon l'ordre induit par la numérotation modulo 2r + 2 de ses sommets. Si $c : C \rightarrow K_3$ est une 3-coloration de C, on peut obtenir une 3-arête-coloration de C en affectant la couleur c(u) à l'arc (u, v). On définit un nouveau graphe non orienté noté G en ajoutant à C pour tout i un nouveau sommet u'_i incident à u_i (i = 1, ... , 2r + 1). On étend la 3-coloration c à G en affectant la couleur non incidente à u_i à l'arête (u_i, u'_i) de telle sorte que le nombre d'arêtes colorées

1 dans G est égal au nombre d'arêtes reliant les sommets colorés 2 aux sommets colorés 3 dans la 3-coloration de C.

On applique maintenant le Lemme 3.3 à G avec X = V(C), et on obtient alors que le nombre d'arêtes colorées 1 de G et reliant X à V(G)\X (qui est $|E_1|$) est impair comme |X|. Le même argument appliqué aux couleurs 2 et 3 termine la preuve.

Pour terminer la démonstration du théorème 3.2, d'après le Lemme 3.4, le graphe $C \times_{K_3} C'$ a un nombre impair d'arêtes entre deux N_i. De plus ce graphe a trois stables N_i, il a donc un nombre total d'arêtes impair.

Corollaire 3.5

Si *G* et *G'* sont 3-chromatiques alors, pour toute 3-coloration de *G* et toute 3-coloration de *G'*, on a $\chi(G \times_{K_3} G') = 3$.

Démonstration

G (resp. G') est 3-chromatique donc il contient un cycle impair C (resp. C'). Si on munit C (resp. C') de la 3-coloration de G (resp. G') restreinte à C (resp. C') alors d'après le théorème 3.2, au moins une des composantes connexes de $C \times_{K_3} C'$ est un graphe eulérien impair donc contient un cycle impair.

On achève la preuve en remarquant que $C \times_{K_3} C'$ est un sous-graphe de $G \times_{K_3} G'$.

La preuve de la conjecture de Khelladi pour $n = 3$ utilise fortement le fait que l'on connaît bien les graphes minimaux 3-chromatiques (les cycles impairs). Ce type de preuve ne semble pas pouvoir s'étendre pour $n > 3$ car la description des graphes minimaux n-chromatiques n'est pas connue.

2.4. Algorithmes

Nous proposons deux méthodes de construction du graphe produit fibré. Dans la première méthode, les sommets sont d'abord construits et ensuite les arêtes du graphe produit fibré ; dans la seconde les sommets et les arêtes sont construits simultanément.

2.4.1 Principe

La construction du graphe produit fibré $G \times_{K_n} G'$ s'effectue en trois étapes :

1 - choix de la coloration des graphes *G* et *G'*

. au choix de l'utilisateur,

. par une heuristique,

. par un algorithme exact,

2 - vérification que les deux graphes G et G' admettent bien une bonne n-coloration,

3 - construction du graphe $G \times_{K_n} G'$ produit fibré de G et G'.

Les sommets des deux graphes G et G' sont décomposés en stable (un stable par couleur). Nous obtenons une structure par niveau comme le montrent les figures 4.4 et 4.5 du paragraphe 2.2.

Par définition du produit fibré par niveau, si G (resp. G') admet une partition de l'ensemble des sommets V de G (resp. V' de G') en stables S_i (resp. S'_i) alors le graphe $G \times_{K_n} G'$ est obtenu par la partition de n stables $N_i = S_i \times S'_i$ (où $\times$ est le produit cartésien usuel entre deux ensembles). Une indexation des sommets des graphes G et G' induit de façon naturelle une indexation pour les sommets de $G \times_{K_n} G'$. Si α est l'étiquette de u et β celle de r, alors pour (u, r) l'étiquette est (α, β). Les sommets sont étiquetés par des entiers. Par abus du langage et pour simplifier l'écriture, l'indice du sommet et l'étiquette du sommet sont confondus.

Construction séparée des sommets et des arêtes

La construction des arêtes du graphe produit fibré s'effectue après celle des sommets. Les sommets (v,s) de $G \times_{K_n} G'$ qui sont dans le même stable que le sommet (u,r) ne sont pas considérés. Seuls les sommets, appartenant à des stables d'indice plus petit que celui de (u,r), sont traités. La vérification de l'existence des arêtes $e = (u,v)$ de E et $e' = (r,s)$ de E' entraine la création d'une arête $((u,r),(v,s))$ pour le graphe $G \times_{K_n} G'$.

Construction combinée des sommets et des arêtes

Les sommets sont partitionnés en stables. Après la construction d'un nouveau sommet dans le graphe produit fibré $G \times_{K_n} G'$(il est noté G_f dans l'algotithme pour simplifier l'écriture), pour chaque sommet, nous nous intéressons uniquement aux sommets-voisins de couleurs plus petites. S'il existe des sommets déjà construits de couleur identique, alors nous construisons une nouvelle arête dans $G \times_{K_n} G'$.

2.4.2 Texte de l'algorithme

Pour la construction du graphe produit fibré, nous présentons deux algorithmes. Dans le premier algorithme, l'ensemble des sommets est d'abord construit et ensuite l'ensemble des arêtes. Dans le second algorithme, la construction des sommets et des arêtes s'effectue en même temps.

Nous présentons les algorithmes à l'aide d'un langage de description de programmation. A chaque procédure sont présentés une liste des données d'entrée et de sortie, et parfois des commentaires.

Début

Données

graphe $G = (V, E)$

$G' = (V', E')$

entier γ nombre chromatique de G

γ' nombre chromatique de G'

ensemble de sommets S_i /* les stables de G, $1 \leq i \leq \gamma$ */

S'_i /* les stables de G', $1 \leq i \leq \gamma'$ */

Variables

graphe $G_f = (V_f, E_f)$

entier : γ_f /* nombre maximal de couleurs utilisées pour G et G' */

ensemble de sommets N_i /* les stables de G_f, $1 \leq i \leq \gamma_f$ */

Initialisation

Initialisation

$V_f = Ø, E_f = Ø$

$\gamma_f = \max (\gamma, \gamma' \}$

Corps du programme

/* Construction séparée des sommets et des arêtes */

V_f = construire_sommet(G, G')

E_f = construire_arête(G, G')

ou

/* Construction combinée des sommets et des arêtes */

G_f = construire_graphe_produit(G, G')

Fin

Construction séparée des sommets et des arêtes

procédure *construire_sommet*
entrée
graphe : G, G'
sortie
ensemble de sommets : V
variables
entier : i
sommets : u, r
débutp
$V = \emptyset$
Pour i = 1 j**usqu'à** γ_f **de_pas** 1 **faire**
$N_i = \emptyset$
Pour_tout u $\in$ S_i **faire** /* S_i stable de G */
Pour_tout r $\in$ S'_i **faire** /* S'_i stable de G' */
/* création d'un sommet du stable N_i de $G \times_{K_n} G'$ */
$N_i = N_i \cup \{(u,r)\}$
fpour_tout
fpour_tout
$V = V \cup N_i$
fpour
finp

procédure *construire_arête*
entrée
graphe : G, G'
sortie
ensemble d'arêtes : A
variables
entier : i, j
sommets : $x_{(u, r)}$, $y_{(v,s)}$ /* les sommets x et y ont les indices (u,r) et (v,s) respectivement */

débutp
A = Ø
Pour i = 1 **jusqu'à** γ_f - 1 **de_pas** 1 **faire**
Pour_tout $x_{(u,r)} \in N_i$ **faire** /* N_i stable de G $\times_{K_n}$ G' */
Pour j = i+1 **jusqu'à** γ_p **de_pas** 1 **faire**
Pour_tout $y_{(v,s)} \in N_j$ **faire**
si (u,v) ∈ E et (r,s) ∈ E' **alors**
/* création d'une arête de G $\times_{K_n}$ G' */
$A = A \cup \{ (x_{(u,r)}, y_{(v,s)}) \}$
fsi
fpour_tout
fpour
fpour_tout
fpour
finp

Construction combinée des sommets et des arêtes

procédure *construire_graphe_produit*
entrée
graphe : G, G'
sortie
graphe : G_f
variables
entier : i
sommets : u, r, s, v, $x_{(u, r)}$, $y_{(r,s)}$
ensemble de sommets : N_G /* N_G(s) est le voisinage de *s* dans le graphe G */
débutp
$V_f = Ø$, $E_f = Ø$
N_1 = sommet_niveau(1, S_1, S'_1)
$V_f = V_f \cup N_1$
Pour i = 2 **jusqu'à** γ_p **de_pas** 1 **faire**
N_i = sommet_niveau(i, S_i, S'_i)

$V_f = V_f \cup N_i$

Pour_tout sommet $y_{(v,s)} \in N_i$ construit **faire**

Pour j = 1 **jusqu'à** i - 1 **de_pas** 1 **faire**

Pour_tout sommet $x_{(u,r)} \in N_j$ et **faire**

si $u \in N_G(v)$ et $r \in N_{G'}(s)$ **alors**

/* création de l'arête $(x_{(u,r)}, y_{(v,s)}) \in E(G \times G')$ */

$E_f = E_f \cup \{ (x_{(u,r)}, y_{(v,s)}) \}$

fsi

fpour_tout

fpour

fpour_tout

fpour

finp

procédure ***sommet_niveau***

entrée

ensemble de sommets : S_i, S'_i

entier : i

sortie

ensemble de sommets : N_i

variables

sommets : u, r

débutp

$N_i = \emptyset$

Pour_tout $u \in S_i$ **faire** /* S_i stable de G */

Pour_tout $r \in S'_i$ **faire** /* S'_i stable de G' */

/* création d'un sommet du stable N_i de $G \times_{K_n} G'$ */

$N_i = N_i \cup \{(u,r)\}$

fpour_tout

fpour_tout

finp

Cet algorithme est été mis en œuvre dans *Cabri-graphes.* Il est détaillé après le paragraphe sur la complexité.

2.4.3 Complexité

La complexité des opérations est calculée pour des graphes simples non orientés.

Soient $G = (V, E)$ et $G' = (V', E')$ deux graphes de nombre chromatique γ et γ' respectivement. Les deux graphes ont été décomposés en stable, notés N_i et N'_j avec $1 \leq i \leq \gamma$ et $1 \leq j \leq \gamma'$. nous notons par n (resp. n') la cardinalité de V (resp. V') et par m (resp. m') celle de E (resp. E').

Construction séparée des sommets et des arêtes

Dans la procédure *construire_sommet,* tous les sommets des graphes G et G' ne sont visités qu'une seuls fois (G et G' étant décomposés en au plus γ_f stables). Donc la complexité en temps est de $O(\gamma_f.n.n')$.

Dans la procédure *construire_arête,* quatre boucles sont imbriquées les unes dans les autres. La complexité en temps est donc de $O(\gamma_f^2\ n^2\ n'^2)$.

Procédure	Complexité en temps
construire_sommet	$O(\gamma_f.n.n')$
construire_arête	$O(\gamma_f^2\ n^2\ n'^2)$

Tableau 4.1 : La complexité des procédures.

En conclusion, la complexité en temps est s'écrit $O(\gamma_f^2\ n^2\ n'^2)$.

Construction combinée des sommets et des arêtes

La procédure *construire_graphe_produit* combine les deux procédures précedentes *construire_sommet* et *construire_arête,* sans modifier la complexité en temps. Ainsi, La complexité en temps de l'algorithme est aussi de $O(\gamma_f^2\ n^2\ n'^2)$.

2.4.4. Mise en œuvre

L'implémentation du produit fibré a été effectuée dans le cadre du projet *Cabri-graphes* [Carbonneaux 96]. Dans cet environnement informatique, il existe déjà un algorithme de calcul exact du nombre chromatique [Dvorak 95], ainsi qu'une heuristique de coloration. Nous avons décidé de les utiliser. La coloration manuelle ou automatique se traduit dans la fenêtre de

visualisation par un étiquetage des sommets avec des entiers ou une vraie couleur sur les sommet (si γ_f est inférieur ou égale à huit).

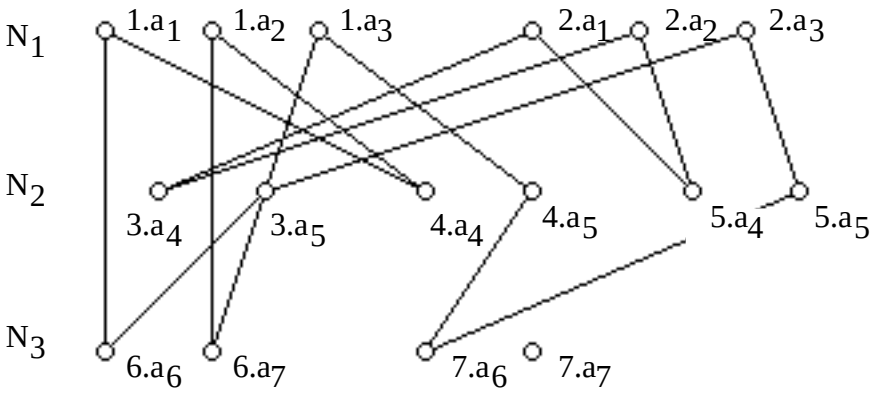


Figure 4.8 : Un bon ordre sur les sommets du produit fibré dans le graphe de la figure 4.4. La numérotation des sommets du graphe produit est le produit par ordre croissant des numérotations.

Les sommets des graphes G et G' sont étiquetés. Le choix d'un bon étiquetage des sommets du graphe produit, à partir de la structure par niveau, permet de mieux visualiser l'algorithme, c'est-à-dire un bon ordre des sommets par niveaux. Dans l'exemple donné par la figure 4.5, les sommets de $G \times G'$ sont bien étiquetés mais mal ordonnés. Il faut placer les sommets par niveaux comme le montre la figure 4.8 (en partant de la configuration initiale de la figure 4.4).

Le bon ordre des sommets de G $\times_{K_n}$ G' est induit naturellement par celui de G et G'. Ainsi, si N_i (resp. N'_i) désigne le nombre de sommets du niveau S_i (resp. S'_i) alors les sommets dans N_i peuvent se retrouver facilement et rapidement par les relations suivantes:

Notation

. pour chaque niveau de S_i les sommets sont numérotés de 1 à t

. pour chaque niveau de S'_i les sommets sont numérotés de 1 à t'

. Les sommets de N_i sont indexés par (x, y) où $1 \le x \le t$ et $1 \le y \le t'$, et par la proposition 2 le nombre total de sommets de N_i est $t \times t'$

L'ensemble des sommets étiquetés $(z,.)$ commence à partir du sommet d'indice $n_i \times (z-1) + 1$ dans N_i et se terminent au sommet d'indice $n_i \times (z)$. De même, l'ensemble des sommets d'étiquette $(.,y)$ est dispersé. Ces sommets ont comme indices dans N_i les valeurs (z,y), $(z + n_i, y)$, ... , $(z + t \times n_i, y)$. La figure 4.6 illustre ce procédé de numérotation.

Pour l'algorithme, ces ordres et étiquetages sont associés pour chaque sommet. Le niveau N_1 est construit. Pour tous les sommets (v,s) de $V(G \times_{K_n} G')$ créés au niveau N_i, la construction des arêtes utilisent uniquement les sommets (u,r) du niveau précédent N_{i-1}. Grâce à l'indexation sur les sommets, le parcours des sommets dans les niveaux précédents est accéléré, et la création des arêtes $((u,r),(v,s))$ de $E(G \times_{K_n} G')$ est rapide et immédiat.

Soient deux graphes $G = (V, E)$ et $G' = (V', E')$, une fonction c (resp. c') de coloration sur les sommets de G (resp. G'). Les graphes G et G' ont alors γ et γ' pour nombres chromatiques et les sommets de G et G' sont décomposés en des ensembles de stables (ou ensembles de sommets indépendants) S_i, $1 \leq i \leq \gamma$, et S'_j, $1 \leq j \leq \gamma'$.

Début

Données

graphe G = (V, E)
G' = (V', E')

entier γ nombre chromatique de G

γ' nombre chromatique de G'

ensemble de sommets S_i /* les stables de G, $1 \leq i \leq \gamma$ */

ensemble de sommets S'_j /* les stables de G', $1 \leq j \leq \gamma'$ */

Variables

graphe $G_f = (V_f, E_f)$

sommet : u, u', v, v', w, w', s_q

entier : γ_f /* nombre maximal de couleurs utilisées pour G et G' */

i, j, k, q

tableau d'entiers : P_{S_i} et $P'_{S'_j}$ /* $P_{S_i}[w]$ est la positon du sommet w dans S_i */

P /* P[k] nombre de sommets construits dans G_f */
/* jusqu'à la couleur k */

ensemble de sommets N_i /* les stables de G_f, $1 \leq i \leq \gamma_p$ */

Initialisation

/* classer les voisinages de tous les sommets de G et G' */

classer_voisinage(G)

classer_voisinage(G')

/* construire les tableaux de couleurs pour G et G' */

construire_tableau_couleur(G)

construire_tableau_couleur(G')

i = j = k = 1

$V_f = \emptyset$, $E_f = \emptyset$

$\gamma_f = \max (\gamma, \gamma')$

Corps du programme

/ Construction du graphe produit, les sommets et les arêtes ensemble */*

tant_que ($k < \gamma_f$) **faire**

/ construction du sommet indexé (v,v') au niveau k */*

$v = S_k(i)$ /* le i-ème élément du niveau S_k du graphe G */

$v' = S'_k(j)$ /* le j-ème élément du niveau S'_k du graphe G' */

$N_k = N_k \cup \{(v,v')\}$ /* un nouveau sommet de G_f est créé */

/ Construction du voisinage du sommet indexé (v,v'), dans G_f */*

Pour_tout $w \in N_G(v)$ **faire**

/ Construction du voisinage du sommet indexé (v,v'), dans G_f */*

Pour_tout $w \in N_G(v)$ **faire**

si $c(w) < c(v)$ **alors**

Pour_tout $w' \in N_{G'}(v')$ **faire**

/ recherche du premier w' tel que c(w') = c(w) */*

si $c(w') = c(w)$ **alors**

/ calcul de l'indice de position , en fonction de la remarque sur le "bon ordre" des sommets */*

$q = P_{S_i}[w'] + | S'_{c(w')} | \times P'_{S'_j}[w]$

/ indice du sommet-voisin dans le graphe produit : nombre de sommet construit jusqu'à la couleur c(w) plus la position du sommet-voisin */*

$s_q = P[\ c(w)\ ] + q$

/ ajout de la nouvelle arête */*

$E_f = E_f \cup \{\ (\ (v,v'),\ s_q\ )\ \}$

fsi

fpout_tout

fsi

fpour_tout

j = j + 1

Condition pour changer l'indexation des sommets

si $j > |S'_k|$ **alors**

i = i + 1

j = 1

fin_si

si $i > |S_k|$ **alors**

i = 1

k = k + 1

$V_f = V_f \cup N_k$

fin_si

fin_tant_que

Fin

procédure ***classer_coisinage()***

entrée : *G* un graphe.

sortie : le voisinage de chaque sommet a été modifié.

Cette procédure classe le voisinage de chaque sommet *u* suivant le bon ordre défini précédemment. Nous avons donc que :

- pour tout sommet $v \in N_G(u)$ on a $c(u) > c(v)$
- les sommets voisins de *v* sont classés par ordre croissant de leur coloration respective.

En imposant ce classement pour les sommets des graphes *G* et *G'*, nous avons alors que toutes les arêtes ne sont traitées qu'une seule fois. La complexité en temps de ce classement est au mieux en *n.logn* et au pire en $O(n^2)$.

procédure ***construire_tableau_couleur()***
entrée : G un graphe.
sortie : les variables globales P_S sont construites et initialisées.

Cette procédure construit les tableaux de position P_{S_i}, pour $1 \leq i \leq \gamma(G)$, utilisés pendant la construction des sommets et des arêtes de G_f.

Remarquons que si les ensembles de sommet S_i, pour $1 \leq i \leq \gamma(G)$, n'existent pas, alors ils sont construits dans cette procédure, après une vérification de la coloration du graphe.

La complexité en temps, pour la construction de tableau, est en $O(n^2)$.

Conclusion

Le théorème 2.12 donne une nouvelle approche pour aborder le produit croisé. Il montre un des intérêts du produit fibré. Cette nouvelle perception du produit croisé a permis de développer certains résultats sur le problème d'hamiltonicité [Gravier 96].

3. La contraction de graphes

Pendant les manipulations des graphes, il arrive souvent qu'une partie du graphe étudié soit bien connue et qu'on veuille l'isoler du reste du groupe. Il faudrait rassembler cet ensemble en un seul élément du graphe, tout en laissant la possibilité de le réintégrer. C'est l'idée du méta-graphe.

Un méta-graphe est un graphe où chaque sommet représente lui-même un graphe (un sommet unique étant un graphe avec un seul élément). Une opération intéressante est alors la possibilité de contracter, de décontracter ou d'éclater ensuite un ensemble de sommets au sein d'un même graphe. Le sommet n'est plus un objet uniquement intrinsèque du domaine, mais il devient lui même un objet abstrait dont la représentation à l'écran est identique à celle du sommet. Dans la suite de notre étude, nous nous intéresserons uniquement à des graphes finis et simples.

3.1 Utilisation

La contraction de graphes est souvent utilisée dans la structure théorique des graphes, et dans divers théorèmes comme principe de structure interdite. Dans la définition du graphe aux arêtes,

[Beineke 68] donne un théorème sur un graphe étant le graphe représentatif des arêtes d'un graphe simple en présentant neuf sous-graphes interdits.

3.1.1 Définitions

Une définition est donnée dans [Graham & al. 95]. Si S est un ensemble d'arêtes d'un graphe G, le graphe issu de G, par suppression de S et identification des extrémités des arêtes de S, est dit être obtenu par *contraction* de S, et il est noté par G/S. Si H est un sous-graphe de G, on note la contraction $G/E(H)$ simplement par G/H. Un *mineur* du graphe G est un graphe que l'on peut obtenir à partir de G par une séquence de suppression de sommets, d'arêtes et de contraction d'arêtes.

Les mineurs jouent un rôle très important dans la structure théorique des graphes. En particulier, d'après l'état de l'art effectué par [Roberton & Seymour 96] l'absence d'un graphe précis comme mineur a de profond effet sur les propriétés du graphe.

Nous avons adopté la définition classique :

Définition 5

Soient $G = (V, E)$ un graphe et S un ensemble de sommets de V. Le graphe *contracté* de G par rapport à S est le graphe noté $G/S = (V_S, E_S)$ où les sommets de l'ensemble S ont été remplacé dans G/S par un sommet s, c'est-à-dire :

(a) $V_S = (V\text{-}S) \cup \{s\}$

(b) E_S est définie par :

(i) si $u \in V\text{-}S$ et $v \in V\text{-}S$ tels que $e = (u, v) \in E(G)$, alors $e = (u, v) \in E_S$

(ii) si $u \in V\text{-}S$ et s'il existe $v \in S$ tel que $e = (u, v) \in E(G)$, alors on note encore $e = (u, s) \in E_S$

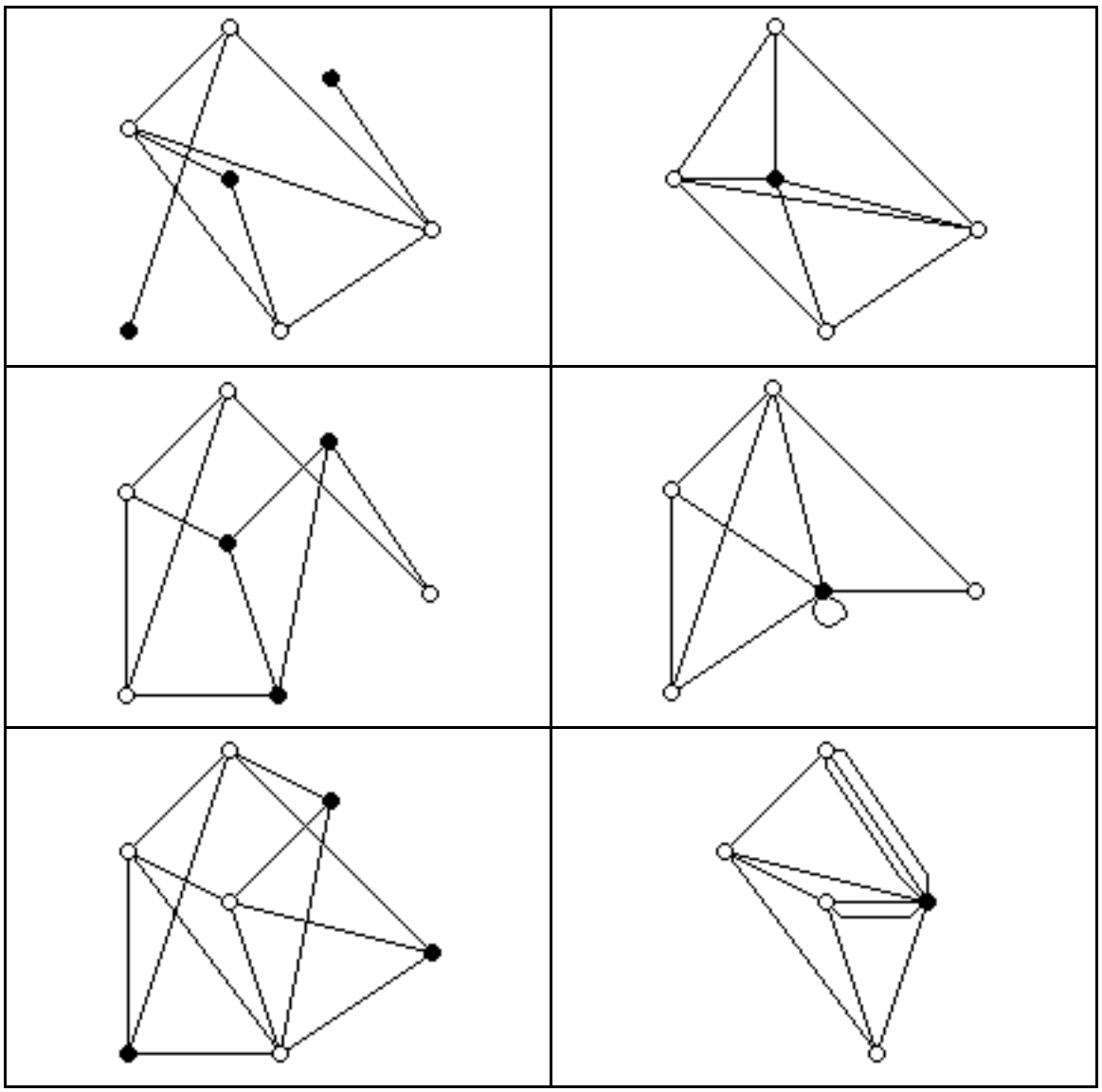
(iii) si $u \in S$ et $v \in S$ tels que $e = (u, v) \in E$, alors l'arête e est remplacé par la boucle $b_S = (s, s) \in E_S$

Remarques

1) Si l'ensemble S des sommets de G n'est pas un stable, alors G/S admet une boucle au sommet s (sommet contracté de S), et dans ce cas $E_S = (E - E(S)) \cup \{b_S\}$, sinon $E_S = E - E(S)$, avec E_S étant les arêtes du graphe induit par l'ensemble S de sommets..

2) Le graphe contracté G/S peut être un graphe multiple (des arêtes multiples et des boucles), même si le graphe G est simple. En particulier, le degré des sommets de V-S est le même dans G et dans G/S.

3) Si $|S| = 1$, alors $G/S = G$.



(a) L'ensemble S a contracté est en noir. (b) Le sommet contracté s est en noir.

Figure 4.9 : Exemples de graphes G et G/S par rapport à un ensemble S de sommets .

Les notions de contracté et de contraction sont utilisées dans plusieurs démonstrations et algorithmes, par exemple la recherche d'un couplage maximum [Gondran & Minoux 85].

Remarques

1) L'éclatement d'un ensemble de sommets est l'opération presque inverse de l'opération de fusion. L'éclatement consiste à remplacer le sommet considéré en plusieurs sommets, les liens entre ces nouveaux sommets étant les plus simples possibles (généralement le graphe complet) ou dépendants de critères choisis par l'utilisateur. Un exemple est d'éclater en deux les sommets d'un graphe orienté. Chaque sommet s est remplacé par deux sommets s_i et s_t. Le sommet s_i admet pour voisinage les sommets prédécesseurs de s, le sommet s_t les sommets successeurs de s. De plus, un arc $a = (s_i, s_t)$ est ajouté.

2) Une autre définition importante à connaître est la transformation appelée homéomorphisme [Reingold & al. 77]. Cette transformation est définie comme suit :

Définition

Deux graphes sont *homéomorphes* s'ils peuvent être obtenus à partir d'un même graphe par une séquence de subdivision des arêtes en chaînes.

Par exemple, deux cycles sont homéomorphes.

3.1.2 Propriété fondamentale

Soient $G = (V, E)$ un graphe et S un ensemble de sommets de V. L'opération de contraction du graphe G au graphe $G/S = (V_S , E_S)$, avec $V_S = (V\text{-}S) \cup \{s\}$ et E_S qui sont deux ensembles construits d'après la *définition 2*, peut être vu comme une paire d'applications $h = (f,g)$ où

$f : V \rightarrow V_S$ est donnée par :

. si $v \in V\text{-}S$ alors $f(v) = v$

. si $v \in S$ alors $f(v) = s$

$g : E \rightarrow E_S$ est définie comme suit :

. si $e \in E(S)$ alors $g(e) = b_S$

. si $e \notin E(S)$ ou $e \in E\text{-}E(S)$ alors $g(e) = e$.

Remarque

Dans le troisième cas pour l'application g, pour des graphes multiples, nous avons le résultat suivant : si $e = (u, v)$ est telle que $u \in S$ et $v \in S$ alors $g(e) = (s, s)$ est une boucle du graphe G/S.

Proposition

L'application de contraction $h = (f, g)$ définit un homomorphisme du graphe G dans le graphe G/S (au sens de la définition 3).

Démonstration

En effet, les applications f et g vérifient la définition de l'homomorphisme donnée précédemment.

Nous allons maintenant comparer les matrices d'adjacence et d'incidence des graphes G et G/S.

Les matrices d'adjacence

Pour la matrice d'adjacence M = Adj(G) ou la matrice sommet-sommet, après une renumérotation des lignes et des colonnes par les sommets libres de *1* à k et les sommets contractés de $k+1$ à n.

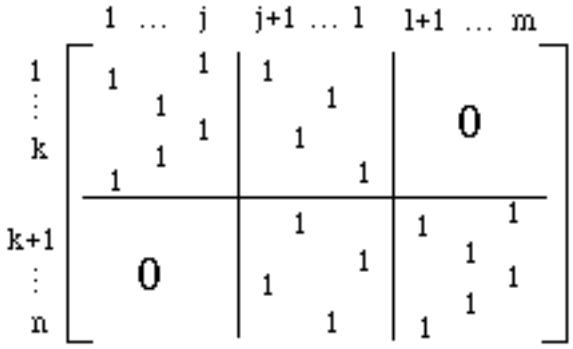
La matrice d'adjacence M' = Adj(G/S) est composée de plusieurs sous-matrices que nous détaillons :

1) pour $1 \leq i, j \leq k$, alors m'(i,j) = m(i,j)
2) pour tout i compris entre 1 et k
   m'(i,k+1) = 1 s'il existe j dans [k+1,…,n] tel que m(i,j) = 1
   = 0 sinon
3) pour tout j compris entre 1 et k
   m'(k+1,j) = 1 s'il existe i dans [k+1,…,n] tel que m(i,j) = 1
   = 0 sinon
4) m'(k+1,k+1) = 1 s'il existe i et j dans [k+1,…,n] tel que m(i,j) = 1
   = 0 sinon

Si les graphes G et G/S sont des graphes simples non orientés, les matrices M et M' sont des matrices en 0-1, symétriques avec des 0 sur la diagonale.

Les matrices d'incidence

Nous appliquons le même procédé pour la matrice d'incidence Inc(G), ou matrice sommet-arête. En effectuant la même renumérotation des lignes et des colonnes, nous obtenons la matrice d'incidence de G :



Chaque colonne comporte exactement deux 1. Nous avons donc classés les arêtes de telle sorte que les arêtes d'indice *i*, avec $1 \le i \le j$, sont inchangées par l'opération

1) pour $1 \le i \le j$, les arêtes e_i sont dans $E\text{-}E(S)$

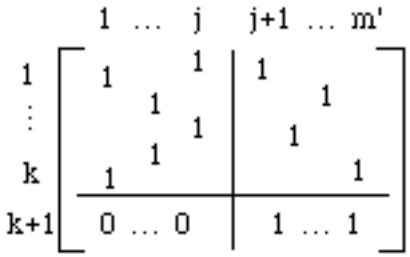
2) pour $j+1 \le i \le l$, l'arête $e_i = (u, v)$ avec $u \in V\text{-}S$ et $v \in S$

3) pour $l+1 \le i \le m$, les arêtes e_i sont dans $E(S)$

Dans le cas général, nous avons la matrice d'incidence Inc(G/S) suivante :

Les arêtes d'indice e_i, pour $l+1 \le i \le m$, sont remplacées par la fameuse boucle $b_S = e_{l+1}$ et $e_{l+1} = (u_{k+1}, u_{k+1})$ (le sommet *s* est ici noté u_{k+1}) et les arêtes multiples de G/S sont les arêtes e_i indexées par $j+1 \le i \le l$.

Dans le cas des graphes simples pour *G* et G/S, nous avons la matrice d'incidence Inc(G/S) suivante :



En reprenant la matrice précédente, la boucle b_S ou l'arête e_{l+1} est supprimée. Pour les arêtes e_i multiples de G/S indexées par $j+1 \le i \le l$, un seul représentant est conservé (nous avons donc que $j \le m' \le l$).

3.2 Conservation de propriétés du graphe

Dans ce paragraphe, nous étudions la conservation de quelques propriétés de graphes après cette opération de contraction de graphe.

3.2.1. Eulérien

Soient $G = (V, E)$ un graphe non orienté eulérien et S un ensemble de sommet de V. Nous donnons une condition nécessaire et suffisante pour que le graphe contracté G/S soit encore eulérien.

Lemme

Soient $G = (V, E)$ un graphe eulérien et S un ensemble de sommets de V. Le nombre d'arêtes entre G-S et S est pair.

Démonstration

Soient $G = (V, E)$ un graphe simple, non orienté, connexe, eulérien, S un ensemble de sommets de V et $\omega(S, G\text{-}S)$ l'ensemble des arêtes reliant S à G-S. L'idée de la démonstration est de compter le nombre d'arêtes entre S et G-S (voir la figure 4.10).

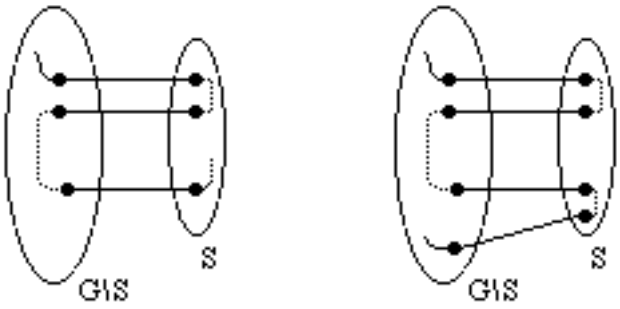


Figure 4.10 : Nombre d'arêts entre S et G-S : impair ou pair ?.

Faisons une récurrence sur $|V(G)\text{-}S| = r$.

. $r = 0$ alors $S = V(G)$, donc $\omega(S) = \emptyset$ et $|\omega(S)| = 0$ (paire).

. si $r \geq 1$, alors montrons que : $\forall u \in S$ $|\omega(S\text{-}u, G\text{-}S)|$ est pair.

. Soit $u \in S$, alors $\omega(S, G\text{-}S) = \omega(u, G\text{-}S) \cup \omega(S\text{-}\{u\}, G\text{-}S)$

. donc $|\omega(u, G\text{-}S)| + |\omega(S\text{-}\{u\}, G\text{-}S)| = 2k$, k un entier (par hypothèse de récurrence)

. $\omega(S\text{-}\{u\}, (G\text{-}S) \cup \{u\}) = \omega(S\text{-}\{u\}, G\text{-}S) \cup \omega(u, S\text{-}\{s\})$

. or $\omega(u, G-S) \cup \omega(u, S-\{s\}) = \omega(u)$

. donc $|\omega(u, G-S)| + |\omega(u, S-\{s\})| =$ degré(u) = 2k', *k'* entier (par définition *G* est eulérien)

. alors $|\omega(u, S-\{u\})| + |\omega(S-\{u\}, G-S)| = 2\ (k+k' - |\omega(u, G-S)|)$

. ainsi $|\omega(S-\{u\}, (G-S) \cup \{u\})| = |\omega(u, S-\{u\})| + |\omega(S-\{u\}, G-S)| = 2p$, *p* entier.

Ceci étant vrai pour tout sommet *u* de *S*, le lemme est bien démontré.

Proposition

Soient $G = (V, E)$ un graphe eulérien, *S* un ensemble de sommets de *V* ($S \neq V$). Alors *G*/*S* le graphe multiple obtenu par contraction de *S* est eulérien

Démonstration

Par définition de la contraction, pour tout sommet *u* de *V*-*S* dans *G* nous avons que deg(*u*, *G*) = deg(*u*, *G*/*S*) et ce degré est pair (*G* est eulérien). D'après le lemme précédent, le nombre d'arêtes entre *G*-*S* et *S* est pair, donc le degré du sommet *s*, le contracté de *S*, est pair dans *G*/*S*. Ainsi *G*/*S* est eulérien.

Proposition

Soient $G = (V, E)$ un graphe eulérien, *S* un ensemble de sommets de *V* et *G*/*S* le graphe simple obtenu par contraction de *S*. Alors les assertion suivantes sont équivalentes :

(i) le voisinage de *S* dans *G*, *N*(*S*), est de cardinalité pair et, pour tout sommet *u* de *N*(*S*), le nombre d'arêtes entre *u* et *S* est impair.

(ii) *G*/*S* est eulérien

Démonstration

Soient $G = (V, E)$ un graphe eulérien, *S* un ensemble de sommets de *V* et *G*/*S* le graphe contracté.

(i) $\Rightarrow$ (ii) : Pour tout sommet *u* de *N*(*S*), le nombre d'arêtes entre *u* et *S* est impair. Donc dans le graphe *G*/*S* et, par définition de la contraction, les sommets *u* de *N*(*s*) sont encore de degré pair. Comme le nombre de sommets dans *N*(*s*) est pair, nous avons aussi que le sommet *s* est de degré pair dans *G*/*S*. Donc *G*/*S* est bien un graphe eulérien.

(ii) $\Rightarrow$ (i) : *G*/*S* est eulérien alors tous les sommets *u* de V_S sont de degré pair, en particulier le sommet contracté *s*. Donc la cardinalité de *N*(*s*) dans *G*/*S* est pair. De plus, le voisinage de *s* dans *G*/*S* correspond au voisinage de *S* dans *G* eulérien.

Dans le graphe G/S, on a :

pour tout sommet u de N(s) : $\omega(u, G/S) = \omega(u, G/S - \{s\}) \cup \omega(u, s)$

alors $|\omega(u, G/S)| = |\omega(u, G/S - \{s\})| + |\omega(u, s)| = 2k$, k entier

et $|\omega(u, s)| = 1$, G/S étant simple

donc $|\omega(u, G/S - \{s\})| = 2p + 1$, p entier et $p < k$

Dans le graphe G, on a aussi :

pour tout sommet v de N(S) : $\omega(v, G) = \omega(v, G-S) \cup \omega(v, S)$

alors $|\omega(v, G)| = |\omega(v, G-S)| + |\omega(v, S)| = 2k'$, k' entier et $k' \geq k$

or $|\omega(v, G-S)| = |\omega(v, G_S - \{s\})| = 2p + 1$, pour v de $N(S)$ et v de $N(s)$ (sommet "inchangé" par la contraction)

donc $|\omega(v, S)| = 2(k'-p) - 1 > 0$

Ainsi, la cardinalité du voisinage de S dans G est pair et pour tout sommet u de $N(S)$ dans G, le nombre d'arêtes entre u et S est impair.

(a) Graphe G et S les sommets en noir. (b) G/S simple est non eulérien.

Figure 4.11 : La cardinalité de S est impair, le graphe G/S simple est non eulérien.

Remarque

La condition sur la cardinalité de $N(S)$ dans la proposition précédent est primordiale. La figure 4.11 nous fournit un exemple.

3.2.2. Hamiltonicité

Soient $G = (V, E)$ un graphe simple non orienté hamiltonien et S un ensemble de sommet de V. Nous donnons une condition nécessaire pour que le graphe contracté G/S soit hamiltonien.

Lemme

Soient $G = (V, E)$ un graphe hamiltonien et S un ensemble de sommets de V. S est un ensemble de sommets d'articulation alors G/S n'est pas hamiltonien.

Démonstration

Cela revient à dire que G-S doit rester connexe pour que G/S n'ai pas comme point d'articulation le sommet contracté s, sinon il n'existe pas de cycle hamiltonien et le graphe contracté est donc non hamiltonien.

3.2.3. Planarité

Dans un graphe planaire 2-connexe, une face est une région du plan limitée par des arêtes et dont l'ensemble constitue la frontière. D'une manière générale, cette frontière est constituée d'un cycle élémentaire.

Définition

Deux faces sont dites *adjacentes* lorsque leurs frontières ont au moins une arête commune, et pas seulement des sommets communs.

Définition

Deux faces sont dites *faiblement adjacentes* lorque leurs frontières ont un sommet en communs, mais aucune arête.

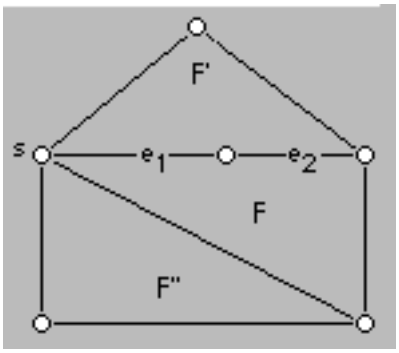


Figure 4.11 : Un graphe planaire.

Dans la figure 4.11, le graphe dessiné est planaire. Les faces F et F' sont adjacentes, puisqu'elles ont en commun les arêtes e_1 et e_2. Les faces F' et F'' sont faiblement adjacentes car le sommet s est le seul élément en commun.

Lemme

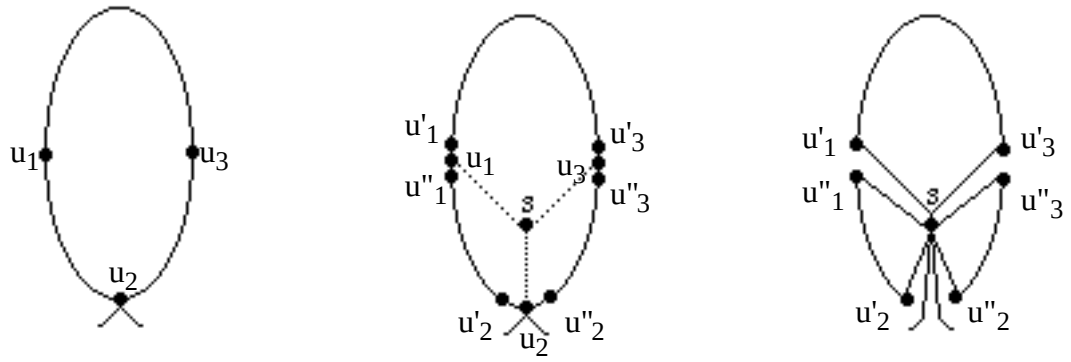
Soient $G = (V, E)$ un graphe planaire 2-connexe et F une face de G. Le graphe contracté G/F est planaire.

Démonstration

Comme les représentations des graphes planaires sont isomorphes, la face *F* du graphe *G* est choisie comme n'étant pas la face externe. La contraction de *F* en un sommet s_F consiste à supprimer une face de *G*, sans modifier la planarité du graphe.

Corollaire

Soient *G* = (*V*, *E*) un graphe planaire 2-connexe et *S* un ensemble de sommets de *V*. Si les éléments de S appartiennent tous à une même face alors *G*/*S* est planaire.



(a) Face d'un graphe avec les sommets *t*,*u*,*v* à contracter.

(b) Les trois sommets seront contractés en un sommet *s*.

(c) Les trois sommets ont été contractés en un sommet *s*.

Figure 4.12 : Les phases de contraction d'éléments d'une face.

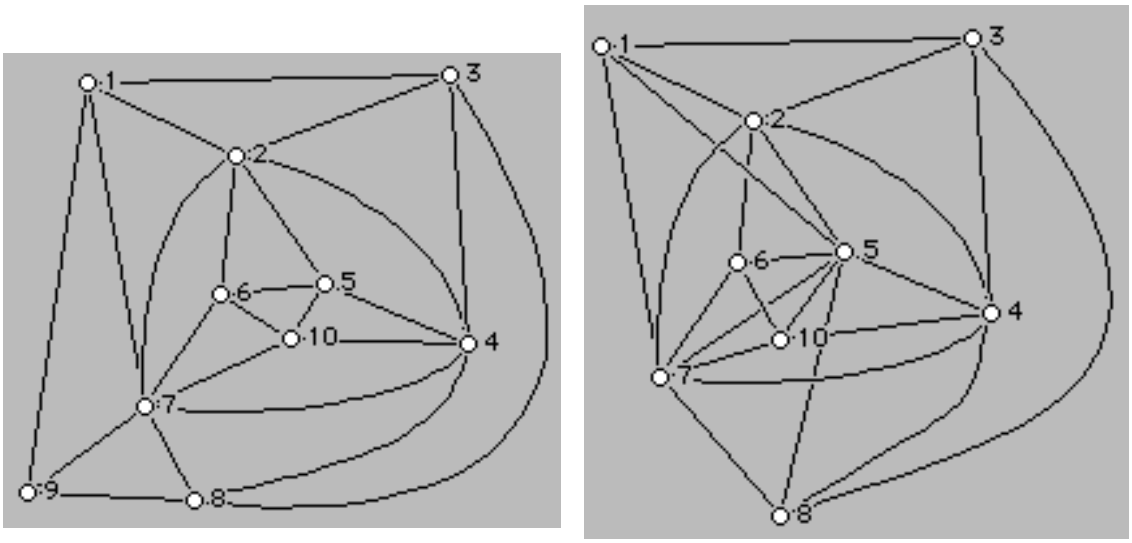
Démonstration

Soient des sommets u_1, u_2, u_3 sur une face *f* d'un graphe planaire. Notons par u_i' et u_i'' les sommets adjacents de u_i sur la face *F*. Nous avons alors deux cas à étudier.

Dans le premier cas, le sommet u_i est incident à uniquement deux faces adjacentes, comme les sommets *u* et *t* dans l'exemple de la figure 4.12. La second face *F'* est composé d'un cycle élémentaire définie par $x \ldots u_i'' \; u_i \; u_i' \ldots x$. Après la contraction de l'ensemble *S* en un sommet *s*, le cycle élémentaire s'écrit $x \ldots u_i'' \; s \; u_i' \ldots x$.

Dans le second cas, u_i est commun à des faces faiblement adjacentes ; c'est l'exemple du sommet *v* de la figure 4.12. Les faces sont décrites par des chaînes dont le sommet u_i est remplacé par le sommet *s*.

La face initiale *F* est découpée en plusieurs sous-faces définies à partir des sommets de la face*F* et des sommets de l'ensemble *S*.



(a) G graphe planaire, S = { 5, 9 }. (b) Graphe contracté G/S non planaire.

Figure 4.13 : Contraction de sommets n'appartenant pas à des faces adjacentes ou faiblement adjacentes. Le nouveau graphe est alors non planaire.

Remarques

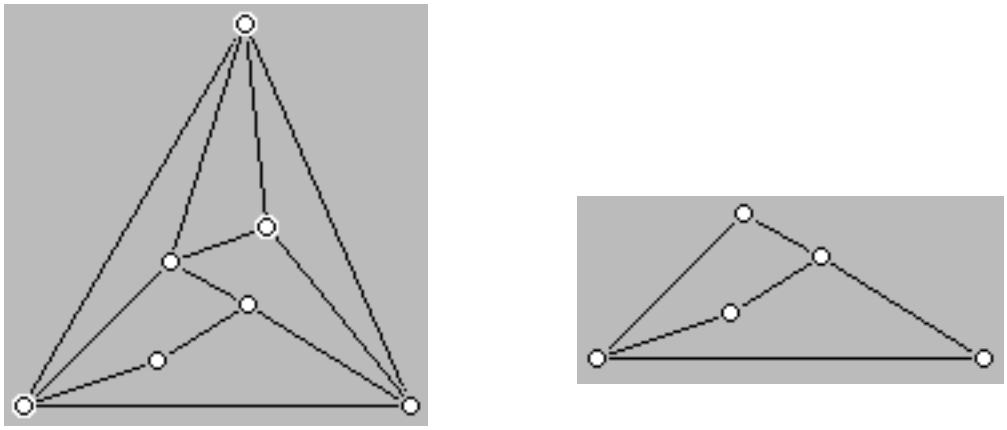
1) La contraction de ces sommets conserve la planarité et elle augmente le nombre de faces.

2) Soit G un graphe planaire. Si les sommets de S appartiennent à deux faces de G non adjacentes par une arête ou un sommet alors G/S est non planaire (exemple de la figure 4.13).

Lemme

Soient $G = (V, E)$ un graphe planaire 2-connexe et S un ensemble de sommets de V. Notons par $\{u_1, u_2, \ldots, u_p\}$ avec $p \leq |V|$ les éléments de S et par V_i le voisinage étendu de u_i, c'est-à-dire $V_i = N(u_i) \cup \{u_i\}$. Si les voisinages V_i sont tels que :

$$V_1 \subseteq V_2 \subseteq \ \ldots \ \subseteq V_p$$ alors G_S est planaire.



(a) Ensemble de sommets à contracter. (b) Graphe contracté.

Figure 4.14 : Graphe planaire dont les sommets en surbrillance sont contractés.

Démonstration

Considérons un graphe vérifiant les propriétés du lemme. Puisque le voisinage étendu de chaque sommet est inclus dans le suivant, seul l'ensemble V_p du dernier sommet u_p est utilisé lors de la contraction (voir figure 4.14).

Alors $N(S) \subset V_p$. Les arêtes entre les ensembles de sommets S et $N(V)$ ont toutes pour une des deux extrémités le sommet u_p. Pour la contraction de S, le sous-graphe induit par les sommets $\{u_1, u_2, \ldots, u_{p-1}\}$ est supprimé du graphe initial. Donc le graphe contracté G/S est encore planaire.

Remarque

Dans le cadre du lemme précédent, la contraction est équivalente, pour la représentation du graphe, à la suppression du sous-graphe induit par l'ensemble $\{u_1, u_2, \ldots, u_{p-1}\}$.

Définition

Soient $G = (V, E)$ un graphe planaire et s un sommet de V. On appelle ombrelle de s l'ensemble des faces associée à s dans une représentation planaire de G. Cet ensemble est noté Us.

Théorème

Soient $G = (V, E)$ un graphe planaire 3-connexe et s et r deux sommets de V. Si $S_u \cap S_v = \emptyset$ alors $G/\{s,r\}$ est non planaire.

Démonstration

Soit $G = (V, E)$ un graphe planaire 3-connexe et s et r deux sommets de V. Le plongement planaire de G est unique. G étant 3-connexe, les sommets s et r sont au moins de degré 3, et notons par s_i $i = 1, 2, 3$ et r_j $j = 1, 2, 3$ respectivement trois sommets-voisins de s et r, avec $s_i \neq r_j$ pout tout $i, j = 1, 2$ ou 3, puisque Us et Ur sont disjoints.

Par le théorème de Menger, comme G est un graphe 3-connexe, il existe trois chemins P_1, P_2 et P_3 arêtes-disjointes entre s et r. Sans perdre de généralité, et avec les notations précédentes, les chemins P_i admettent comme extrémités $s - s_i - \ldots - r_i - r$ avec $i = 1, 2,$ ou 3.

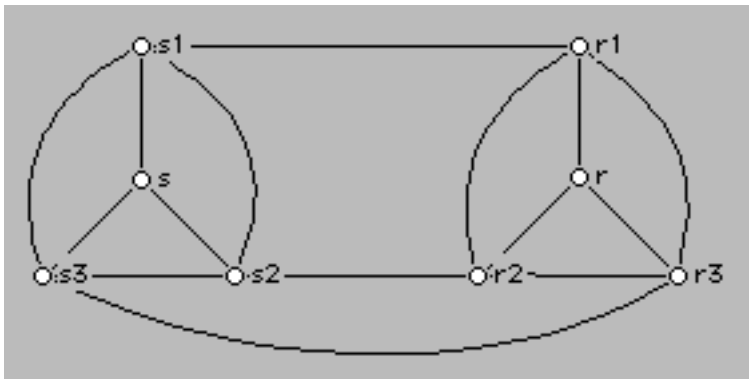


Figure 4.15 : Une configuration initiale de *G*, graphe planaire 3-connexe.

En contraction les sommets *s* et *r*, le graphe $G/\{s,r\}$ possède un sous-graphe homéomorphe à K_5, comme nous le montre la figure 4.16 :

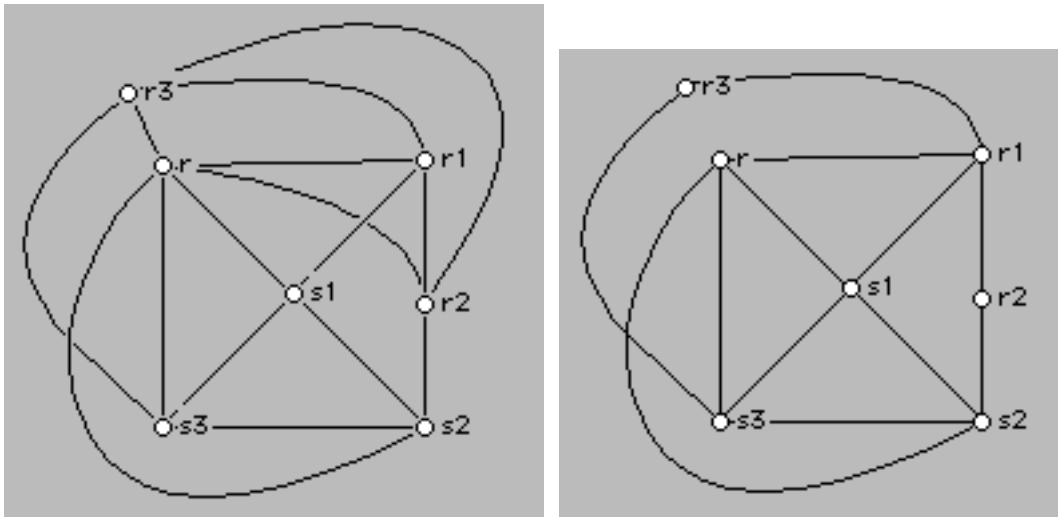


Figure 4.16 : Le graphe $G/\{s,r\}$ et un sous-graphe homéomorphe à K_5.

Ainsi la contraction de deux sommets, dont les ombrelles sont disjointes dans un graphes 3-connexe, implique que le graphe contracté n'est pas planaire.

Remarques

(i) Pour prolonger ce résultat au graphe 2-connexe, une étude plus précise de la structure du graphe est indispensable, afin d'obtenir un théorème similaire contenant des contraintes supplémentaires sur le graphes.

(ii) Il existe aussi d'autres configurations pour les sommets dont la contraction de l'ensemble entraîne que le graphe G/S ne soit plus planaire (voir figure 4.17).

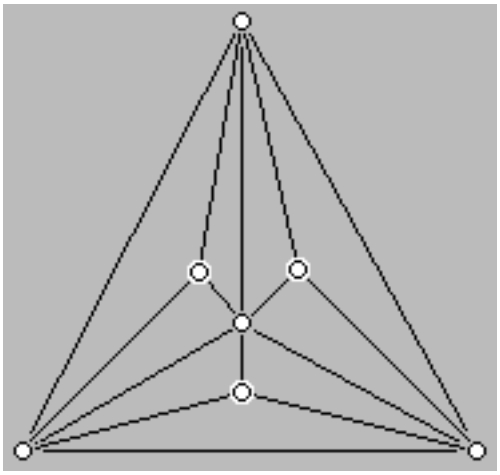


Figure 4.17 : La contraction de deux ou trois sommets sélectionnés (en surbrillance sur la figure) entraîne la création d'un graphe non planaire.

Nous allons maintenant établir un algorithme sur la contraction et sur la décontraction de sommets. Avant de programmer les algorithme, nous étudions plus précisèment ces opérations afin de connaître exactement les informations à conserver.

3.3 Algorithmes

Comme nous l'avons constaté au paragraphe précédent, l'opération *contraction* fusionne un ensemble de sommets S et elle crée un nouveau sommet s pour le remplacer. Les anciennes relations entre les sommets sont préservées entre le voisinage de S et s, afin de permettre de *décontracter* le graphe et de lui *redonner* sa précédente représentation. Pour permettre ce travail, il faut préserver pour chaque nouveau sommet deux ensembles : l'ensemble de sommets à contracter, c'est-à-dire le sous-graphe induit par cet ensemble, et le voisinage de cet ensemble par rapport graphe restant.

La contraction, sans la décontraction, n'est autre que l'opération d'identification ou de fusion d'un ensemble de sommets. Nous nous intéressons donc à une opération de type fusion, mais pouvant à un moment donné être réversible, c'est-à-dire supprimer le sommet contracté et rendre de nouveau visibles le sous-graphe induit et les arêtes entre le graphe et le sous-graphe. Dans ce cas, deux contraintes supplémentaires nous sont en plus imposées : visualiser en permanence un graphe contenant des sommets simples et des sommets contractés, permettre d'éclater les sommets contractés dans un ordre différent de leur contraction, et de rétablir les relations d'incidence sur les arêtes supprimées par la contraction (en particulier les arêtes multiples apparues lors de la contraction).

Dans la suite de notre étude, nous nous sommes uniquement intéressé au grahe G/S simple. Une attention toute particulière du cas général, G/S multiple est en cours d'étude.

3.3.1. Conserver les informations

Il existe deux types de contraction possibles : la contraction à un niveau ou la contraction sur plusieurs niveaux. Dans le premier cas, un sommet représentant déjà un ensemble contracté de sommets ne peut pas être de nouveau contracté. Dans le second cas, cette possibilité de contraction à répétition des ensembles contenant déjà des sommets contractés est admise.

Contraction à un niveau

La contraction d'un ensemble de sommets *S* en un seul sommet *s* exige de préserver au niveau du sommet *s* les informations suivantes : le sous-graphe G_S induit par *S* et les arêtes entre *Gs* et le graphe *G-S*.

Pour décontracter un sommet, il faut transformer le sommet *s* en un sous-graphe G_S, et avoir de nouveaux les arêtes entre G_S et les sommets du graphe restant. Plusieurs autres problèmes se posent alors :

- si plusieurs sommets sont contractés, alors il faut pouvoir les décontracter dans un autre sens que celui de la contraction ; ceci impose d'avoir une certaine indépendance entre les sommets contractés,

- dans ce cas, la préservation des arêtes s'avère plus délicate que prévue ; il faut associer un sommet contracté à chaque sommet pour préserver les liens d'adjacence entre les sommets lors de la décontraction de l'un deux. Donc une indépendance des sommets est nécessaire.

Remarques

1) Soit $G = (V, E)$ un graphe. Pour la contraction à un niveau, l'ensemble des sommets visibles à l'écran est au plus $|V|/2$, un sommet unique n'étant pas considéré comme un ensemble.

2) Avec les nouvelles informations associées à chaque sommet, il est inutile de conserver l'ensemble des arêtes. En effet, ces dernières le sont grâce aux données préservées au niveau du sommet et en fonction du graphe initial.

Contraction sur plusieurs niveaux

Notons S_i un sommet contracté dans un graphe *G*, et *S* un ensemble de sommets par rapport au graphe initial. Pour ce type de contraction sur plusieurs niveaux, des interactions sont possibles entre les différents ensembles S_i. Cette relation est un emboîtement possible des S_i, c'est-à-dire qu'un sommet contracté S_i pourra contenir d'autres sommets déjà contractés S_j et cet empilement est réalisable sur plusieurs niveaux.

Remarques

1) Pour un graphe $G = (V, E)$, il existe au maximum ($|V|$-1) emboitements successifs d'ensembles de sommets S_i, un sommet unique n'étant pas considéré comme un ensemble.

2) Soient un graphe G et un graphe G_S contracté sur plusieurs niveaux et deux sommets contractés u et v de G_S. Notons par S_u (resp. S_v) l'ensemble des sommets de base associés à u (resp. v) dans le graphe G. Il existe une relation entre S_u et S_v :

. les ensembles sont disjoints, $S_u \cap S_v = \emptyset$,

. ou, l'un est inclus dans l'autre, $S_u \subset S_v$ ou $S_v \subset S_u$

Il faut conserver des informations supplémentaires afin de pouvoir dessiner de nouveau le graphe correctement après avoir "décontracté" un sommet. Pour mettre en œuvre cette opération, les informations associées à un ensemble de sommets contractés se décomposent en deux parties : un état, visible ou invisible, et un ensemble de sommets. Cet ensemble est lui-même composé de sommets normaux (voir le niveau 1) et de sommets contractés.

Les informations associées au niveau des sommets doivent aussi contenir des informations relatives aux arêtes, et toujours en fonction du graphe initial. Si l'ordre de décontraction n'est pas l'ordre inverse de contraction, des nouvelles relations ou arêtes entre sommets seront nécessaires pour refléter la relation existant dans le graphe initial entre les différents ensembles.

3.3.2 Principe

L'algorithme s'applique uniquement à des graphes simples, éventuellement orientés. Nous définissons trois graphes. Le premier graphe est le graphe de base, noté $G_b = (V_b, E_b)$, correspondant au graphe initial avant les diverses opérations de contraction (à plusieurs niveaux) et de décontraction. Le second graphe est le graphe affiché à l'écran. Il est noté par $G_a = (V_a, E_a)$. Le troisième graphe est le graphe virtuel $G_v = (V_v, E_v)$ contenant toutes les données nécessaires aux opérations, en particuliers des sommets et des arêtes nécessaires aux décontractions n'étant ni des sommets de base (resp. des arêtes de base) ni des sommets affichés (resp. des arêtes affichées).

Lors de la contraction d'un ensemble S de sommets de G_a, un nouveau sommet s est créé pour les graphes G_a et G_v, et de nouvelles arêtes correspondantes aux relations entre S et G-S sont construites pour G_a et G_v .

Remarques

1) Les graphes traités sont simples, accessoirement orientés. Les trois graphes G_b, G_v et G_a admettent une relation au niveau de leur ensemble de sommets :

$V_b \subset V_v$ et $V_a \subset V_v$

Le graphe virtuel peut être considéré comme le plus important des trois graphes.

2) Le nombre d'emboitement maximum de contractions est $|V_b| - 1$. Alors, nous avons que :

$$|V_b| \leq |V_v| \leq 2.|V_b| - 1$$

et $1 \leq |V_a| \leq |V_b|$

3) De même pour les arêtes, nous avons :

$$|E_b| \leq |V_b|.(|V_b| - 1)/2$$

alors $|E_b| \leq |E_v| \leq |V_v|.(|V_v| - 1)/2 = 2.|V_b|.|V_b|$

et $|E_a| \leq |E_b|$

Ces inégalités sont vraies dans le cadre d'un graphe contracté simple et sans boucle.

principe de *contraction*

La contraction d'un ensemble S de sommets s'effectue en plusieurs étapes. Pendant la première étape, on récupère le voisinage N de S dans G_a et des ensembles d'arêtes. Ces ensembles sont l'ensemble des arêtes induites par S dans G_v, et l'ensemble des arêtes incidentes entre S et N dans G_v. Dans la seconde étape, le sommet contracté s et les arête associées sont construits. La dernière étape consiste à mettre à jour les différents graphes utilisés.

principe de *décontraction*

La décontraction d'un ensemble S de sommets se décompose en plusieurs étapes. Pendant la première étape, on récupère des ensembles d'objets. Les ensembles de sommets sont le voisinage N de S dans G_a, et l'ensemble W des sommets dont la contraction correspond à S. Les ensembles d'arêtes sont l'ensemble des arêtes induites par S dans G_v, et l'ensemble des arêtes incidentes entre les ensembles S et N dans G_v. Dans la seconde étape, les sommets de W sont contruits. Pour les arêtes incidentes aux sommets de W, et les arêtes incidentes entre W et N, une recherche est réalisée dans G_v , éventuellement en décontraction jusqu'au niveau le plus bas les éléments de W et de N afin de trouver éventuellement une arête. Dans le cas d'une découverte positive, une arête est alors construite. La dernière étape consiste à mettre à jour les différents graphes utilisés.

3.3.3 Texte de l'algorithme

Dans ce paragraphe, nous donnons les algorithmes sous la forme d'un langage de description de programmation. Pour chaque procédure, les données d'entrée et de sortie sont présentées, et parfois des commentaires.

Début

Données

graphe de base $G_b = (V_b, E_b)$

Variables

graphe virtuel $G_v = (V_v, E_v)$

graphe affiché $G_a = (V_a, E_a)$

ensemble de sommets S

Initialisation

$G_v \leftarrow G_b$

$G_a \leftarrow G_b$

Corps du programme

au_choix

contracter_sommets(S)

décontracter_sommets(S)

Fin

Contraction

Pour l'opération contraction , nous avons détaillé uniquement la procédure la plus importante. Les autres procédures sont seulement commentées.

procédure contracter_sommets

entrée : S un ensemble de sommets

variables

ensemble de sommets : N

ensemble d'arêtes : $A_{i,}$ A_e, A_n

sommet : s

débutp

N = voisinage(S, G_a)

A_i = arêtes_induites(S, G_v)

A_e = arêtes_entre_ensembles(S, N, G_v)

A_n = créer_arêtes_voisinage(s, A_e)

s = nouveau_sommet(S, A_n)

$V_v = V_v \cup \{s\}$

$E_v = E_v \cup A_n$

$V_a = V_a - S$, $V_a = V_a \cup \{s\}$

$E_a = E_a - A_i$, $E_a = E_a - A_e$, $E_a = E_a \cup A_n$

finp

Les procédures suivantes sont des procédures classiques. Elles sont uniquement commentées ; les variables d'entrée et de sortie sont explicitement décrites.

procédure *voisinage*
entrée : S ensemble de sommets
G graphe
sortie : N ensemble de sommets
Cette procédure retourne l'ensemble N de sommets du voisinage de l'ensemble S, c'est-à-dire les sommets à distance 1 de S dans le graphe G.

procédure *nouveau_sommet*
entrée : S ensemble de sommets
A_v ensemble d'arêtes
sortie : s sommet
Cette procédure crée pour les graphes G_v et G_a le sommet contracté s en lui associant le sous-graphe induit par S et les arêtes de A_v entre les ensembles N(S) et S.

procédure *arêtes_induites*
entrée : S ensemble de sommets
G graphe

sortie : A_i ensemble d'arêtes

Cette procédure récupère l'ensemble A_i des arêtes du graphe G induit par S.

procédure ***arêtes_entre_ensembles***

entrée : S, V ensembles de sommets

G graphe

sortie : A_e ensemble d'arêtes

Cette procédure récupère l'ensemble des arêtes entre les ensembles de sommets S et V dans le graphe G.

procédure ***créer_arêtes_voisinage***

entrée : s un sommets

A_e ensemble d'arêtes

sortie : A_n ensemble d'arêtes

Cette procédure crée l'ensemble A_v des arêtes incidentes au sommet s, en fonction de la nature du graphe initial..

Décontraction

Pour l'opération decontraction , nous avons détaillé les procédures importantes. Les autres procédures sont uniquement commentées. Ils existent des procédures utilisées et décrites dans le paragraphe précédent.

procédure décontracter_sommets(fixe S ensemble de sommets)

variables

ensemble de sommets : N, W

ensemble d'arêtes : A_i, A_e, A_w, A_n

débutp

W = sommets_sous_cache(S, G_v)

N = voisinage(S, G_a)

A_i = arêtes_induites(S, G_a)

A_e = arêtes_entre_ensembles(S, N, G_a)

A_w = construire_arêtes_internes(W)

A_n = construire_arêtes_voisinage(W, N)

$V_v = V_v - S$, $V_v = V_v \cup W$

$E_v = E_v - \{A_i \cup A_e\}$, $E_v = E_v \cup A_w \cup A_n$

$V_a = V_a - S$, $V_a = V_a \cup W$

$E_a = E_a - \{A_i \cup A_e\}$, $E_a = E_a \cup A_w \cup A_n$

finp

procédure construire_arêtes_internes

entrée

ensemble de sommets : W

sortie

ensemble d'arêtes : A

variables

sommets : u, w

ensemble d'arêtes : A_b

débutp

pour_tout w ∈ W **faire**

pour_tout u ∈ W **faire**

A = contruire_relation(w, u)

$A = A \cup A_b$

fpour

fpour

finp

procédure construire_arêtes_voisinage

entrée

ensemble de sommets : W, N

sortie

ensemble d'arêtes : A

variables

sommets : u, w

ensemble d'arêtes : A_b

débutp

```
début p
    pour_tout w ∈ W faire
        pour_tout u ∈ N faire
            A = contruire_relation( w, u )
            A = A ∪ A_b
        fpour
    fpour
fin p
```

```
procédure construire_relation
entrée
    sommets : w, u
sortie
    ensemble d'arêtes : A
variables
    ensemble de sommets : V_u, V_w
début p
    si w ∈ V_a alors
        A = récupérer_arêtes( w, u, G_v )
        si A = Ø alors
            V_w = sommets_de_base({w} )
            V_u = sommets_de_base( {u} )
            A = créer_arête_base( V_w, V_u )
        fsi
    fsi
fin p
```

```
procédure sommets_de_base
entrée
    ensemble de sommet : V
sortie
    ensemble de sommet : W
variables
```

```
variables
    sommet : v
    ensemble de sommets : R
débutp
    W = V
    pour_tout v ∈ V faire
        si v ∈ V_b alors
            W = W ∪ {v}
        sinon
            W = W - {v}
            R = sommet_sous-cache( {v} )
            W = W ∪ sommets_de_base( R )
        fsi
    fpour
finp
```

Les procédures suivantes sont des procédures de bases dont les variables d'entrée et de sortie sont décrites.

procédure ***sommets_sous_cache***
entrée : S ensemble de sommets
sortie : W ensemble de sommets

Pour chaque sommet s de l'ensemble S, cette procédure récupére dans le graphe virtuel G_v les sommets w associés à *s* par une contraction et elle les insère dans l'ensemble W.

procédure ***créer_arêtes_base***
entrée : U, W ensembles de sommets
sortie : A ensemble d'arêtes

S'il existe une arête entre les ensemble U et W dans G_b, alors une arête est créée entre les sommets *u* (décontracté en U) et *w* (décontracté en W) dans G_a et G_v.

Nous poursuivons notre étude de ces opérations par le calcul de leur complexité respective.

3.3.4 Complexité

Soit $G = (V, E)$ un graphe simple non orienté, nous notons par n la cardinalité de V et par m celle de E.

Contraction

Dans la procédure *Voisinage*, le voisinage de chaque sommet de l'ensemble S est parcouru. En moyenne, S contient $n/2$ sommets et chaque sommet possède $n/2$ sommets-voisins. Donc, la complexité en temps de cette procédure est en $O(n^2)$.

Pour les procédures *Arêtes_induites* et *Arêtes_entre_ensembles*, l'ensemble des arêtes entre deux sommets est parcouru pour récupérer les informations nécessaires. Ainsi la complexité en temps de ces procédures est aussi en $O(n^2)$.

Pour la création du sommet contracté s et de son voisinage, la complexité en temps s'écrit en O(n).

Procédure	Complexité en temps
voisinage	$O(n^2)$
arêtes_induites	$O(n^2)$
arêtes_entre_ensembles	$O(n^2)$
créer_arêtes_voisinage	O(n)

Tableau 4.3 : La complexité des procédures intervenant lors de la contraction d'un ensemble de sommets.

En conclusion, la complexité en temps de l'algorithme de *contraction* s'écrit en $O(n^2)$.

Décontraction

Pour cette opération, la recherche et la sauvegarde des informations sont prédominant.

Pour connaître le nombre de sommets dépendant d'un sommet s de S, il suffit de lire les informations associées. Sa complexité est de O(n). Comme précédement, la récupération des sommets du voisinage s'effectue en $O(n^2)$.

La récupération des arêtes induites par la décontraction de S et la recherche des arêtes entre les ensembles N(S) et W(S) demandent un temps de l'ordre de $O(n^2)$.

La complexité de la procédure récursive *sommets_de_base* se calcule par l'étude du graphe A construit par le procédé suivant. Soit r un sommet de V_v la racine de A. Si r est un élément de V_b alors A est réduit à la racine r, sinon un ensemble de sommets S lui est associé (un ensemble S_r a été contracté en un sommet r dans une contraction antérieure). Les éléments s de S_r

représentent des nouveaux sommets pour *A*, et une arête est créée entre les sommets *r* et *s*. Le processus se poursuit récursivement tant que les nouveaux sommets *s* de S_i au niveau *i* ne sont pas des éléments de V_b. Donc *A* est un graphe dont toutes les feuilles sont des éléments de V_b et les nœuds intermédiaires des éléments de V_v. Le graphe ainsi construit est un arbre, puisque les ensembles S_i et S_j sont soit disjoints, soit l'un est inclut dans l'autre (aucun cycle n'étant créé).

Le récupération des feuilles de l'arbre *A* consiste à le parcourir récursivement comme pour une exploration en profondeur. Or, la complexité en temps de cet algorithme d'exploration, pour un graphe $G = (V, E)$, s'écrit O(|V|+|E|).

En utilisant les remarques sur la cardinalité des ensembles V_v et E_v, nous avons que :
$|V_v| \leq 2.|V_b| - 1$ et $|E_v| \leq 2.|V_b|.|V_b|$ et $|V_b| = n$
donc la complexité en temps de la procédure *sommets_de_base* est en $O(n^2)$.

La procédure *construire_relation* applique deux fois consécutifs l'algorithme précedent. Donc sa complexité s'écrit aussi en $O(n^2)$.

Dans les procédures *construire_arêtes_internes* et *construire_arêtes_voisinage* la construction éventuelle des arêtes entre les éléments des ensembles se fait en parcourant pour chaque élément du premier ensemble, les éléments du second. Les deux ensembles contiennant au plus n éléments, la complexité en temps de ces deux procédures est alors de $O(n^4)$.

Procédure	**Complexité en temps**
sommets_sous_cache	O(n)
voisinage	$O(n^2)$
arêtes_induites	$O(n^2)$
arêtes_entre_sommets	$O(n^2)$
sommets_de_base	$O(n^2)$
construire_relation	$O(n^2)$
construire_arêtes_internes	$O(n^4)$
construire_arêtes_voisinage	$O(n^4)$

Tableau 4.4 : La complexité des procédures intervenant lors de la décontraction d'un ensemble de sommets.

En conclusion, la complexité en temps de l'algorithme de *décontraction* s'écrit en $O(n^4)$.

3.3.5 Mise en œuvre

Nous avons intégré ces deux opérations dans notre environnement informatique.

Les structures de données

Comme nous pouvons le constater, il faut sauvegarder beaucoup d'informations, pour chaque ensemble de sommets à contracter, et lors de la décontraction, il faut toujours faire référence au graphe initial indépendamment du graphe représenté.

Pour simplifier au maximum, nous avons associé de nouvelles informations au niveau du sommet et de l'arête dans notre structure de données précédente. Ces informations sont :

pour le sommet

. une variable d'état, visible ou invisible, correspondant au fait que le sommet est représentée ou non,

. un ensemble S_c de sommets regroupant pour ce sommet l'ensemble des sommets associés. Si cet ensemble S_c est vide alors cela signifie que le sommet ne représente pas un sommet contracté.

pour l'arête

. une variable d'état, visible ou invisible, correspondant au fait que l'arête est représentée ou non.

Cet ajout au niveau des structures de données permet de sauvegarder un minimum d'informations, pour une meilleure gestion globale. Ainsi, nous avons qu'une seule structure de graphe représentant les trois graphes, et ce graphe est le graphe virtuel G_v. Les deux autres graphes s'en déduisent facilement. Les sommets du graphe de base G_b sont les sommets de V_b dont l'ensemble S_c associé est vide. Le graphe G_b est donc le sous-graphe de G induit par l'ensemble des sommets précédent. Le graphe affiché G_a est le graphe dont la variable d'état visible des sommets et arêtes est vraie.

Un problème est qu'il faut constamment vérifier quel type de sommet est utilisé dans les algorithmes employés au niveau interne, pour le calcul des invariants par exemples.

Exemple

Pour l'opération de décontraction, il faut effectuer plusieurs étapes : vérifier que tous les sommets dans l'ensemble S sont des sommets contractés, sinon les enlever, récupérer pour chaque sommet contracté son ensemble de sommets associés et les dessiner de nouveau, en s'efforçant de garder la configuration initiale de l'ensemble S par rapport au sommet s.

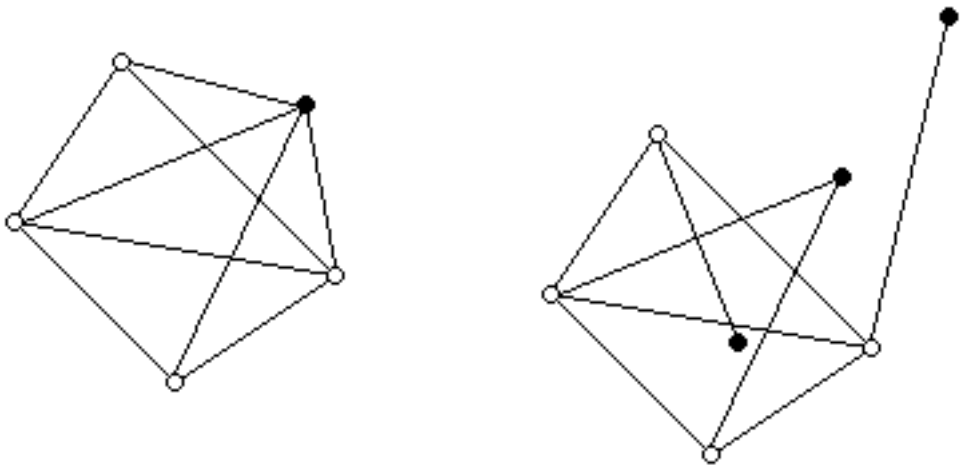


Figure 4.18 : Déplacement d'un ensemble contracté.

En reprenant l'exemple du graphe de la figure 4.9, le sommet contracté s (en noir) est déplacé, puis il est décontracté à son nouvel emplacement (représentation dans la figure 4.18). Le sommet contracté est le barycentre des points associés aux autres sommets. En considérant la position d'origine *p* du sommet s et son emplacement actuel *p'*, nous effectuons pour la décontraction une translation de vecteur $\overrightarrow{pp'}$. En cas d'ensemble de sommets à décontracter, un ajustement est nécessaire. L'éclatement du sommet s s'effectue dans un espace réduit fonction des autres sommets.

3.4. Différentes types de contraction

Dans le paragraphe précédent, nous avons étudié la conservation d'une propriété d'un graphe après la contraction d'un ensemble de sommets. Dans ce cette section, nous nous intéressons plus particulièrement à des contractions spécifiques qui s'avèrent être utilisées dans des algorithmes : contraction d'une composante fortement connexe dans un graphe orienté, sous-graphe dépendant d'un sommet dans un graphe orienté, etc.

3.4.1. Composante fortement connexe

Un graphe orienté $G = (V, E)$ est dit fortement connexe si, étant deux sommets quelconques *u* et *v* (dans cet ordre), il existe un chemin d'extrémité initiale *u* et d'extrémité terminal *v*.

Considérons la relation $u \mathrel{R} v$ définie par : il existe à la fois un chemin de *u* à *v* et un chemin de *v* à *u* ou $u = v$. Cette relation est une relation d'équivalence (réflexivité, symétrie, transitivité). Les classes d'équivalence induites sur *V* par cette relation forment une partition de *V* en $V_1, V_2, \ldots, V_q$.

q est appelé le nombre de connexité fort du graphe *G*. Les sous-graphes $G_1, G_2, \ldots, G_q$ de *G* engendrés par les sous-ensembles $V_1, V_2, \ldots, V_q$ sont fortement connexes et sont appelés

les composantes fortement connexes de *G*. Un graphe est fortement connexe si et seulement s'il n'a qu'une seule composante fortement connexe.

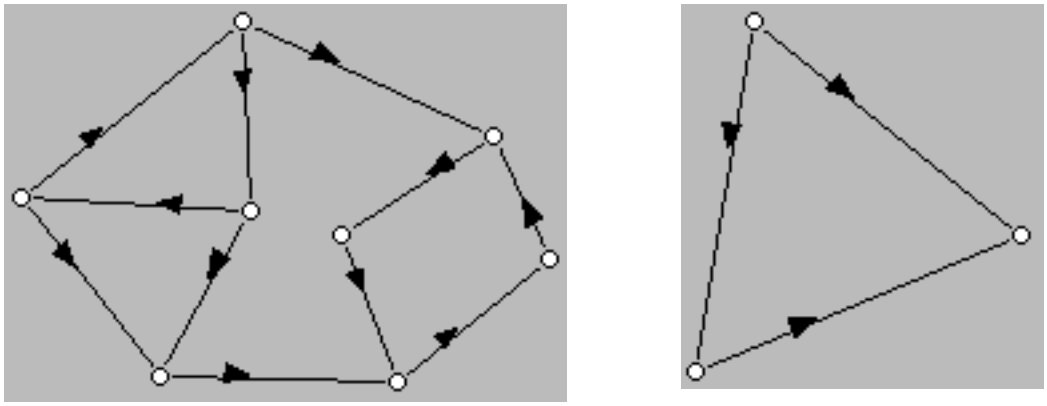


Figure 4.19 : Graphe orienté G et son graphe réduit Gr associé.

Si le graphe *G* n'est pas fortement connexe, alors on associe le graphe orienté simple $G_r = (V_r, E_r)$. Le graphe G_r appelé graphe réduit de *G*, il est défini par rapport à la relation d'équivalence. Le graphe réduit G_r n'a donc plus de circuit, comme le montre le graphe de la figure 4.19.

3.4.2. Ensemble indépendant

Dans un graphe *G*, certaines parties du graphe sont mieux connues et analysées que d'autres. On voudrait provisoirement "mettre de côté" cette partie du graphe afin de mieux analyser le reste du graphe. La contraction de cet ensemble est une des solutions possibles pour satisfaire cette contrainte.

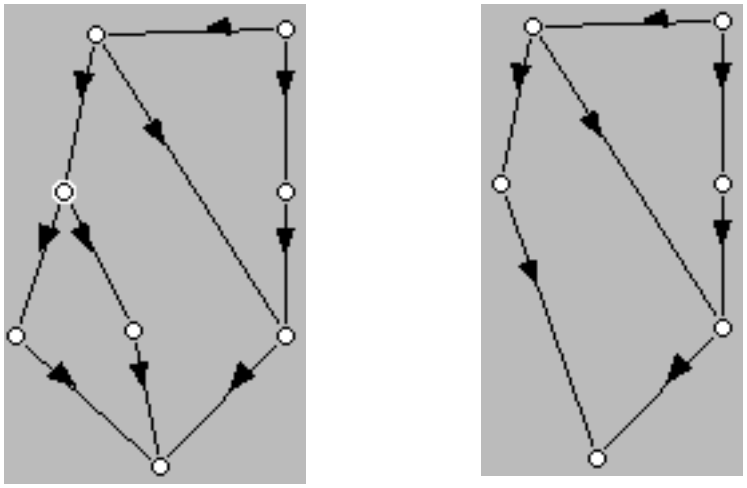


Figure 4.20 : Graphe orienté G et son graphe contracté à partir du sommet sélectionné (indiqué par une surbrillance).

Généralement, le sous-graphe à dissimuler par contraction est une partie indépendante d'un graphe restant. Dans l'exemple d'un graphe orienté, cet ensemble pourrait être la composante

fortement connexe dont il dépend, par exemple un sous-graphe ayant comme puits (ou racine dans le cas non orienté) un sommet sélectionné.

Cette fonctionnalité sera très utile pour notre collaboration avec le travail de Vanda Luengo et *Cabri-Euclide* [Luengo 97]. En effet, dans son environnement informatique, elle utilise *Cabri-graphes* pour représenter un graphe de preuve des relations mathématiques (les axiomes, les théorèmes, les hypothèses, etc.) que l'élève assemble suivant ses connaissances. Pour l'étude de certaines connaissances, la représentation du graphe de preuve est un graphe dense et nécessite justement la possibilité de "dissimuler" temporairement des morceaux de démonstrations afin d'examiner le reste du graphe (un exemple est donné par la figure 4.20).

Actuellement, la contraction de graphe est un outil employé par un seul autre environnement informatique sur les graphes, *da Vinci*. Les autres éditeurs ou les autres visualiseurs de graphes ne possèdent que l'identification d'un ensemble de sommets. La contraction dans *da Vinci* est particulière dans le sens où le sommet contracté est isolé des autres sommets, parce qu'aucune arête ne le relie aux autres sommets.

Conclusion

Dans ce chapitre, nous avons aussi abordé quelques problèmes relatifs aux graphes d'un point de vue théorique et aussi algorithmique. La définition d'homomorphisme proposé est bien plus restrictive que les autres. Nous obligeons une correspondance non pas uniquement avec les sommets, mais aussi avec les arêtes. Cette définition nous a permis d'identifier des opérateurs sur les graphes. L'introduction du produit fibré nous a amené à la conservation de propriétés pour le graphe produit, et en particulier sur le problème l'hamiltonicité.

Nous nous sommes aussi intéressés à des problèmes de visualisation des graphes. Notre démarche consiste à cacher les éléments les moins pertinents et donc mettre en évidence les autres. La démarche traditionnelle consiste à mettre en avant des parties du graphe par différentes méthodes : une vue multiple par plusieurs fenêtres ou une vue en œil de poisson. Ce nouvel outil comme la contraction de sommets, avec une sauvegarde suffisante d'informations pour décontracter ultérieurement les sommets après des manipulations graphiques, est un apport important dans le domaine de la visualisation de données abstraites.

La mise en œuvre des algorithmes est un domaine demandant beaucoup d'attention. Une étude détaillée est nécessaire afin de ne pas augmenter la complexité en temps des algorithmes et pouvoir les améliorer avec, par exemple, de nouvelles structures de données mieux adaptées aux problèmes.

Conclusion

La contribution de la thèse

Les interfaces de manipulation directe sont centrées sur l'utilisateur. Elles ont pour origine l'utilisation d'un mode d'interaction basé sur la métaphore du monde réel. Notre problématique est liée à la notion de manipulation directe et le type de métaphore sur lequel elle se fonde et s'appuie. Nous avons étudié les principes associés à la manipulation directe. Nous avons apporté des précisions sur ces principes, et nous en avons ajouté de nouveaux. En particulier, nous avons établi que le mode opératoire nom/verbe n'est pas une condition nécessaire associée à la manipulation directe. De plus, le mode opératoire utilisé dépend fortement du domaine modélisé et des objectifs de l'environnement informatique. Nous avons aussi remarqué qu'une difficulté majeure, du concepteur d'interface de manipulation directe, est l'identification des métaphores.

Il existe une multiplicité des environnements informatiques en mathématiques discrètes, et plus spécialement en théorie des graphes. Plusieurs contextes et critères ont été à l'origine de leur création, mais peu d'entre eux ont utilisé la manipulation directe comme mode d'interaction. Les environnements étudiés sur les graphes sont de plusieurs catégories, et nous nous sommes principalement attardés sur les éditeurs de graphes. Les éditeurs proposent tous la création manuelle des objets du domaine, mais avec plus ou moins de contraintes dans leur réalisation. Ils utilisent la manipulation directe ou un langage de commande comme mode d'interaction. *Link* est le seul environnement à considérer le mode opératoire verbe/nom, les autres utilisant le mode nom/verbe. La manipulation des objets est souvent restreinte, parce qu'elle est souvent difficile à mettre en œuvre et nécessite un investissement et une étude approfondie du problème. Nous avons observé différents choix pour résoudre les problèmes (le placement des étiquettes, des contraintes pour l'utilisateur dans la création et les manipulations possibles des arêtes, par exemple), mais les solutions proposées ne respectent pas toujours les principes de la manipulation directe.

Nous avons réalisé un environnement prenant en considération ces différentes contraintes liées à la manipulation directe. Nous avons apporté des solutions ou suggéré des voies de recherche. En particulier, nous avons utilisé une représentation des arêtes par une courbe de Bézier cubique avec des points de contrôle internes. Nous avons indiqué les difficultés d'associer une étiquette à un objet en proposant une solution résolvant partiellement le problème.

Nous avons conçu une unique structure pour le type "graphe" permettant de construire un large éventail de graphes : graphe simple ou multiple, orienté ou non, avec ou sans boucle. Cette méthode a permis de simplifier des problèmes de visualisation, mais elle a augmenté la complexité des algorithmes comme les parcours de graphes. Des tests supplémentaires ont été ajoutés pour connaître la nature exacte des graphes et permettre d'améliorer les algorithmes. La nouvelle interface offre une grande souplesse dans la manipulation des graphes, et par conséquent une plus grande liberté d'action pour l'utilisateur.

Nous nous sommes intéressés à deux applications sur les graphes : le produit fibré de graphe et la contraction de graphe (un outil de visualisation sur les graphes).

Notre travail du produit fibré a permis de vérifier pour un cas particulier de la conjecture d'Hédètniemi sur le nombre chromatique dans un produit de graphes. Dans nos deux applications, nous avons étudié des caractéristiques sur les graphes afin de conserver des propriétés simples sur le graphe résultant de l'opération.

Pour la visualisation des graphes, notre démarche a consisté à cacher les éléments les moins pertinents, contrairement à la démarche traditionnelle qui exhibe les parties pertinentes par différentes méthodes (une vue multiple avec plusieurs fenêtres ou une vue en œil de poisson par exemple). Notre outil, la contraction de sommets avec une sauvegarde suffisante d'informations pour décontracter les sommets après des manipulations graphiques, est un apport dans le domaine de la visualisation de données abstraites

Les perspectives de développement

Nos recherches actuelles s'orientent suivant plusieurs directions : apporter des solutions aux problèmes posés par les interfaces de manipulation directe, concevoir de nouveaux outils afin de visualiser de manières pertinentes et efficaces les graphes, éventuellement créer un environnement informatique de visualisation de graphes dans l'espace, et poursuivre l'étude du produit fibré, par exemple.

Interface de manipulation directe

Comme tout environnement informatique, *Cabri-graphes* continuera d'évoluer afin de répondre aux nouvelles exigences de l'utilisateur : améliorer l'interface, programmer de nouvelles fonctionnalités. Une attention toute particulière sera apportée pour entreprendre une communication avec d'autres environnements. Une étape a déjà été faite dans la collaboration avec *Cabri-Euclide* [Luengo 97]). De plus, il est intéressant de sauvegarder les informations sur les graphes pour communiquer avec d'autres éditeurs de graphes ou avec des logiciels de calcul formel.

Une possibilité est l'étude et la réalisation d'un environnement fonctionnant sur le réseau internet. L'environnement informatique devra tenir en compte des dernières technologies disponibles, et pouvant facilement intégrer les outils, par exemple les sorties sonores, ou les joysticks avec retour de force, dont certains sont déjà en vente dans des magasins grand public.

Des outils de visualisation

Des recherches dans plusieurs directions ont déjà apporté des solutions en proposant de nouveaux outils : des fenêtres multiples pour des vues multiples ou des effets optiques comme la vue en œil de poisson.

Deux axes principaux sont envisagés. Le premier axe est de poursuivre le travail engagé avec la contraction de graphes. Plusieurs problèmes restent ouverts, en particulier sur la conservation de propriété. Un objectif est de caractériser les ensembles de sommets dont la contraction, dans un graphe planaire, laisse le graphe planaire. Le second axe de recherche est une collaboration avec Safwan Qasem [Qasem 97] pour la réalisation d'un environnement informatique de visualisation de graphes dans l'espace.

Etudes théoriques et algorithmiques

La programmation des algorithmes nous a apparu comme une nécessité. Une partie de notre travail sur le produit fibré et sur la contraction de graphes (en particulier la conservation de propriétés pour le graphe contracté) a été possible grâce à l'implémentation des algorithmes proposés dans notre environnement.

La poursuite des recherches sur le produit fibré semble nécessiter une dimension informatique, par exemple la gestion des produits fibrés de graphes non triviaux, leurs visualisations et le calcul de leurs invariants. Un environnement informatique, comme *Cabri-graphes*, apporte de solides outils permettant de poursuivre ces études.

Bibliographie

[Aho & al. 74] Aho A.V., Hopcroft J.E. & Ullman J.D. (1974), Data Structures and Algorithms, Addison-Wesley.

[Akerman 95] Ackerman (1995), *Environnement Interactifs : Culture d'auteurs ou culture de zappeurs*, dans Guin, Nicaud et Py (Eds.) Environnements Interactifs d'Apprentissage avec Ordinateur, Actes des Quatrièmes journées EIAO de Cachan 95, Tome 2, Eyrolles, Paris, pp. 9-15.

[AOLpress 95] Logiciel édité chez America Online, Inc (1995-97).

[Apple 87] *Human Interface Guidelines : The Apple Desktop Interface*, Addison-Wesley Publishing Compagny, Paris (1987).

[Aubert & Schneider 82] Aubert J. & Schneider B. (1985), *Décomposition de la somme cartésienne d'un cycle et de l'union de deux cycles hamiltoniens en cycles hamiltoniens*, Discrete Math. n° 38, pp. 7-16.

[Baudon 90] Baudon O. (1990), *CABRI-Graphes, un CAhier de BRouillon Interactif pour la Théorie des Graphes*, Thèse de 3-ème cycle, Université J. Fourier, Grenoble I, France.

[Baulac 90] Baulac Y. (1990), *Un micro-monde de géométrie, Cabri-géomètre*, Thèse de 3-ème cycle, Université J. Fourier, Grenoble I, France.

[Beaudon-Lafon & al. 90] Beaudon-Lafon M., Berteaud Y. & Chatty S. (1990), *Créer des applications à manipulations directe avec Xtv*, IHM'90, Biarritz, France.

[Bellemain 92] Bellemain Franck (1992), *Conception, réalisation et expérimentation d'un logiciel d'aide à l'enseignement de la géométrie : Cabri-géomètre*, Thèse de 3-ème cycle, Université J. Fourier, Grenoble I, France.

[Benzaken 86] Benzaken C. (1986), *Un éditeur de graphe simple, d'aide à l'enseignement et à la recherche*, rapport technique n° 1, IMAG, Université J. Fourier, Grenoble I, mars 1986.

[Berge 73] Berge C. (1973), *Graphes et hypergraphes*, Dunod, Paris.

[Berge 83] Berge C. (1983), *Graphes*, Gauthiers-Villars, Paris.

[Bernat 94] Bernat Philippe (1994), *Conception et réalisation d'un environnement interactif d'aide à la résolution de problèmes*, Thèse de 3-ème cycle de l'université H. Poincarré, Nancy I.

[Bernat & al. 95] Bernat Philippe & Morinet-Lambert Josette (1995), *Spécificités et modélisation de l'interaction dans un EIAO*, Environnement Interactifs d'Apprentissage avec Ordinateur (tome 2), Eyrolles Paris, pp. 209-220.

[Berry & al. 96] Berry J., Dean N., Fasel P., Goldberg M., Johnson E., MacCuish J., shannon G. & Skiena S. (1996), *Link*, DIMACS Center, Rutgers University, Piscataway, NJ.

[Bobrow & al. 86] Bobrow D. G., Mittal S. & Stefik M.J. (1986), *Expert systems: Perils and promises*, Communication of the ACM, n° 29(9), pp. 880-894.

[Bordier 90] Bordier J. (1990), *Cabri Graph, Reference Manuel*, Laboratoire de Structures Discrètes et de Didactique, IMAG, Grenoble, France.

[Bordier & Laborde 91] Bordier J. & Laborde J-M. (1991), *An Interactive Tool for Graph Theory*, 7th Annual Apple EUC, Conference proceeding, Paris, France.

[Bouchard 97] Bouchard J. (1997), *Les graphes, guide d'apprentissage des mathématiques au secondaire*, Les éditions de la pomme, Québec, (Canada).

[Burdea & Coiffet, 93] Burdea Grigore & Coiffet Philippe (1993), *La Réalité Virtuelle*, Hermès, Paris, 1996.

[Bridgeman & al. 96] Bridgeman S., Garg A. & Tamassia R. (1996), *A Graph Drawing and Translation Service on the WWW*, Université de Brown (Providence, Rhode Island), rapport technique RI 02912-1910, Etats-Unis.

[Burr & al. 76] Burr S.A., Erdös P. & Lovàsz L. (1976), *On Graphs of Ramsey Type*, Ars Combinatoria 1, pp. 167-190.

[Buxton 93] Buxton Bill (1993), *HCI and the Inadequacies of Direct Manipulation System*, SIGCHI Bulletin, January 93, vol. 25, n° 1, pp. 21-22.

[Carbonneaux & al. 95] Carbonneaux Y., Madani M. & Laborde J-M. (1995), *Cabri-graph*: A *Tool for Research and Teaching, Graph Theory*, Graph Drawing, *in* Brandenburg F.J. (Ed.), Lecture Notes in Computer Science n° 1027, pp. 123-126.

[Carbonneaux & al. 96] Y. Carbonneaux, J.M. Laborde, R. M. Madani, *Cabri-graphes : un Outil pour la Recherche et l'Enseignement en Théorie des Graphe*s, rapport technique 961-I, octobre 1996, Grenoble.

[Carbonneaux 97] Carbonneaux Yves (1997), *Cabri 4.0 Manuel d'utilisateur*, Laboratoire Leibniz, Grenoble, France.

[Chatty 93] Chatty Stéphane (1993), *De la manipulation directe à l'animation : la dynamique de l'interface*, Centre Etude National Aérienne, PII, n° 93.830, juillet 1993.

[Coutaz 90] Coutaz Joëlle (1990), *Interfaces homme-ordinateur, Conception et réalisation*, Dunod informatique, Bordas, Paris 1990.

[Dao & al. 86] Dao M., Habib M., Richard J-P. et Tallot D. (1986) *"Cabri" an interactive system for graph manipulation, Graph Theoric Concepts in Computer Science*, Proceeding of the International Workshop WG'86, Bernried FRG, June 1986, Lecture Notes in Computer Sciences n° 286, pp. 58-67.

[Delebecque & al. 96] Delebecque F., Gomez C. & Goursat M. (1996) *High-level data structures in CACSD Example: Programming with graphs in Scilab*, IEEE International Symposium on CACSD, Dearborn (Michigan, USA), septembre 14-18 1996.

[Dix & al. 93] Dix Alan, Finlay Janet, Abowd Gregory & Beale Russell (1993) *Human-Computer Interaction*, Prentice Hall, London.

[Duffus & al. 85] Duffus D., Sands B. & Woodrow R.E. (1985), *On the chromatic number of the product of graphs*, J. of Graph Theory, Vol.9, pp. 487-495.

[Dvorak & al. 95] Dvorak T., Havel I., Hrebec Martin, Laborde J-M. & Liebl P. (1995), *On implementation of a fast algorithm for chromatic number*, rapport de recherche n° 951, université de Grenoble, France.

[Eades & al. 88] Eades P., Fogg I. & Kelly D. (1988), *SPREMB: A System for Developing Graph Algorithms*, rapport technique, université de Queensland, St Lucia, Queensland (Australie), septembre 88.

[Ehrig & al. 87] Ehrig H., Nagl M., Rosenfeld A. & Rozenberg G. (1987), *Graph-Grammars and Their application to Computer Science*, Editeur Rozenberg, Lecture Notes in Computer Science n°291.

[Engelbart & English 68] Engelbart D. C. & English W. K. (1968), *A Research Center for Augmenting Human Intellect*, réimprimé dans ACM SIGGRAPH Video Review, 1994, n° 106.

[English & al. 67] English W. K., Engelberg D. C. & Berman M. L. (1967), *Display Selection Techniques for Text Manipulation*, IEEE Transaction on Human Factors in Electronics, HFE-8(1).

[Fekete 96] Fekete Jean-Daniel (1996) *Les trois services du noyau sémantique indispensables à l'IHM*, actes des journées de travail d'IHM'96 à Grenoble.

[Fourneau 86] Fourneau J-M. (1986), *Unicorn, une maquette de Cabri pour les réseaux d'interconnexion*, rapport de recherche ISEM 051, université Paris-Sud.

[Fowler & Bartels 93] Fowler B. & Bartels R. (1993), *Constaint-Based Curve Manipulation*, IEEE computer Graphics & Applications, vol. 16, pp. 43-49.

[Froidevaux & al. 94] Froidevaux C., Gaudel M-C. & Soria M. (1994), *Types de données et algorithmes*, Ediscience international, Paris.

[Frölich 93] Frölich D.M. (1993), *The history and future of direct manipulation*, Behaviour & information technology, vol. 12, n° 6, pp. 315-329.

[Frölich & Werner 94] Fröhlich M. & Werner M. (1994) *Demonstration of the Interactive Graph Vizualisation System da Vinci*, in R. Tamassia & I.G. Tollis (Ed.), Lecture Notes in Computer Science n° 894, pp. 266-269.

[GAP 97] *The GAP Team, GAP -- Groups, Algorithms, and Programming, Version 4*. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, and School of Mathematical and Computational Sciences, University of St. Andrews, North Haugh, St. Andrews, Fife KY16 9SS, Scotland, 1997.

[Goldberg 88] Goldberg A. (1988), *A History of Personal Workstations*, Addision-Wesley Publishing Compagny, New-York, NY. 537.

[Gomez & Goursat 90] Gomez C. & Goursat M. (1990), *METANET: a system for network problems study*, Rapport technique n° 124, INRIA (France), novembre 1990.

[Gondran & Minoux 85] Gondran M. & Minoux M. (1985), *Graphes et algorithmes*, Eyrolles, Paris.

[Graham & al. 95] Graham R.L., Grötschel M. & Lovász L. (1995), *Handbook of Combinatoric*, The MIT Press, Elsevier, Lausanne.

[Graph Layout Toolkit] Logiciel édité par la société américaine : Tom Sawyer Software, 804 Hearst Avenue, Berkeley, CA 94710 USA.

[Gravier 96] Gravier S. (1996), *Coloration et Produits de Graphes*, Thèse de 3-ème cycle, Université J. Fourier, Grenoble I, France.

[Gravier 97] Gravier S. (1997), *Hamiltonicity of the cross-product of two Hamiltonian graphs*, Discrete Mathematics, vol. 170, pp. 253-257.

[Gritzman & al. 95] Gritzman M., Kluge A. & Lovett H. (1995), *Task Orientation in User Interface Design*, Human-Computer Interaction, Interact'95, *in* Nordby K., Helmersen P., Gilmore D. J. & Arnesen S. A. (Ed.), Chapman & Hall, Londres, pp. 97-102.

[Habib & Laborde 83] Habib M. & Laborde J-M (1983), *Document de constitution du groupe CABRI, PRC "Mathématiques et Informatique"*.

[Harary 69] Harary F. (1969), *Graph Theory*, Addison-Wesley, Reading, Massachusset.

[Hedetniemi 66] Hédètniemi S.T. (1966), *Homomorphisms of Graphs and Automata*, , rapport technique 03105-44. T, université de Michigan (, Etats-Unis).

[Hell 72] Hell P. (1972), Rétractions de graphes, thèse d'état, université de Montréal (Canada).

[Hell 79] Hell P.(1979), *An Introduction to Category of Graphs* N.Y. Ac. Sc., pp. 120-136.

[Himsolt 94] Himsolt M. (1994) *GraphEd: A Graphical Platform for the Implementation of Graph Algorithms*, Graph Drawing, *in* R. Tamassia & I.G. Tollis (Ed.), Lecture Notes in Computer Science n° 894, pp. 182-193.

[Himsolt 96] Himsolt M. (1996) *GraphLet: A Graphical Platform for the Implementation of Graph Algorithms*, Graph Drawing, *in* R. Tamassia & I.G. Tollis (Ed.), Lecture Notes in Computer Science n° 894, pp. 182-193.

[Hutchins & al. 85] Hutchins E. L., Holland J. D. & Norman D. A. (1985), *Direct manipulation interfaces*, Human Computer Interaction (1), pp. 311-338.

[IBM 88] Systems Application Architecture, Common User Access, Panel Design and User Interaction, IBM Corporation.

[Kay 69] Kay A. (1969), *The Reactive Engine*, thèse de l'université des sciences de l'Utah (Etats-Unis).

[Kay 77] Kay A. (1977), *Personal Dynamic Media*, IEEE Computer, n° 10(3), pp. 31-42.

[Khelladi 92] Khelladi A. (1992), *Sur une opération binaire de graphes*, communication à la 32ème Semaine de la Science, Damas (Syrie), nov. 1992.

[Khelladi & Semri 93] Khelladi A. & Semri A. (1993), *Sur quelques produits de graphes*, communication du 4ème colloque Maghrébin des méthodes numériques de l'ingénieur, Alger (Algérie), nov. 1993.

[Khelladi, Carbonneaux & Gravier 96] Khelladi A., Carbonneaux Y. & Gravier S. (1997), *Fiber Product* , publication de l'institut mathématiques d'USTHB, Alger (Algérie).

[Kheddouci & Kouïder 95] Kheddouci H. & Kouïder M. (1995), Produit de Kronecker de certains graphes par des cycles, 5-ième cooloque international de graphe et combinatoire, Marseille Luminy, (septembre 1995).

[Knuth 93] Knuth Donald E. (1993) *CWEB*, Addison-Wesley Publishing Company, New York, ACM Press.

[Knuth 94] Knuth Donald E. (1994) *The Stanford GraphBase, A Platform for Combinatorial Computing*, Addison-Wesley Publishing Company, New York, ACM Press.

[Kocay 90] Communication privée.

Pour plus d'informations sur le logiciel Group&Graph , contacter : Dr. Bill Kocay, Computer Sciences Department , University of Manitoba, Winnpeg, Manitoba, Canada, R3T 2N2 e-mail: bkocay@cm.umanitoba.ca

[Kruse, Leung & Tondo 89] Kruse Robert L., Leung Bruce P. & Tondo Clovis L. (1989), *Data Structures and Program Design in C*, Prentice-Hall International Editions, New-York.

[Laborde 87] Laborde Jean-Marie (1987), *"Cabri" a tool for research and teaching in graph theory*, colloque "La Combinatoire et l'informatique", Montréal (Québec, Canada).

[Laborde 89] Laborde Jean-Marie (1989), *Intelligent Microworlds and Learning Environments, in* Intelligent Learning Environments: The Case of Geometry, edited by J-M. Laborde, NATO Serie F: Computer and Systems Sciences, (1995) vol. 117, pp. 113-132.

[Laborde 95] Laborde Jean-Marie (1995), *Des connaissances abstraites aux réalités artificielles, le concept de micromonde Cabri*, Environnement Interactifs d'Apprentissage avec Ordinateur (tome 2), Eyrolles Paris, pp. 29-41.

[Liechti 95] Liechti Olivier(1995) *EGG, a Tool for Building Interactive Attributed Graph Editors*, thèse en informatique (août 1995), Institut d'informatique, de l'université de Fribourg, Suisse.

[Luengo 97] Luengo V. (1997), *Un micro-monde de preuve intégrant la réfutation : CABRI-EUCLIDE*, Actes des 5e journées EIAO de Cachan, Edition Hermès, Paris, pp.85-99.

[MacKay 90] MacKay Brendan D. (1990) *Nauty Users Guide (Version 1.5)*, Technical Report TR-CS-90-02, Computer Science Department, Australian National University (1990).

[Maple-V 91] Maple V, Language Reference Manuel (1991), Springer-Verlag, Paris.

[Myers 92] Myers Brad A. (1992), *Demonstrational Interfaces: A Step Beyond Direct Manipulation*, IEEE Computer, août 92, pp. 61-73.

[Myers 96] Myers Brad A. (1996), *A Brief History of Human Computer Interaction Technology*, rapport de recherche CMU-CS-96-163 et CMU-HCII-96-103 de l'université Carnegie Mellon, Pittsburgh (PA, Etats-Unis), décembre 1996.

[Nanard 90] Nanard Jocelyne (1990), *La manipulation directe en interface homme-machine*, thèse de doctorat d'état, Université des Sciences et Techniques du Languedoc, Montpellier II, 1990.

[NCTM 89] National Council of Teachers of Mathematics (1989), *Curriculum and Evaluation Standards for School Mathematics*, Reston, VA: NCTM.

[Newberry 93] Newberry Paulisch (1993) *Edge: The Design of an Extensible Graph Editor*, Lecture Notes of Computer Science, Springer-Verlag, n° 704.

[Norman & Draper 86] Norman D.A. & Draper S.W. (1986) *User Centered System Design*, Lawrence Erlbaum Associates, Publishers.

[Pisenski 95] Pisanski T. (1995), *VEGA Version 0.2: Quick Reference Manual and Vega Graph Gallery*, Department of Theorical Computer Science at Institute of Mathematics, Physics and Mechanics in Ljubljana, Slovenia, 1995.

[Reingold & al. 77] Reingold E.M., Nievergelt J. & Deo N. (1977), *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall.

[Roberton & Seymour 96] Roberton N. & Seymour P.D. (1996), *Graph Minors*, Journal of Combinatorial Theory, Series B, vol. 68, n°1 (septembre 96), pp. 112-148

[Sander 94] Sander George (1994) *Graph Layout Through the VCG Tool, in* Tamassia R., Tollis I.G. eds.: Graph Drawing, Proc. DIMACS Intern. Workshop GD'94, LNCS 894, pp. 194-205, Springer-Verlag.

[Sauer & Zhu 92] Sauer N.W., Zhu X. (1992), *An approach to Hedetniemi's conjecture*, J. of Graph Theory, Vol. 16 N°5, pp. 7-13.

[Scheifler & Gettys 86] Scheifler R. W. & Gettys J. (1986), *The X Window System*, ACM Transaction on Graphics, n° 5(2), pp. 79-109.

[Semri 93] Semri (1993), *Etude de produits de graphes*, thèse de magistère, USTHB, Institut des Mathématiques, Alger (Algérie).

[Shneiderman 82] Shneiderman Ben (1982), *The Future on Interactive System and the Emergence of Direct Manipulation*, Behavior and Information Technology (1), pp. 237-256.

[Shneiderman 83] Shneiderman Ben (1983), *Direct Manipulation: A Step Beyond Programming Languages*, IEEE Computer (16)8, pp. 57-69.

[Shneiderman 87] Shneiderman Ben (1987), *Designing the User Interface, Strategies for Effective Human-Computer Interaction*, Addison-Wesley Publishing Compagny, Paris.

[Skiena 95] Skiena S. (1990), *Implementing Discrete Mathematics: Combinatorics and Graph Theory in Mathematica*, Advanced Book Division, Addison-Wesley, Redwood City CA, June 1990.

[Smith 75] Smith D. C. (1975), *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*, 1977, Basel, Stuttgart: Birkhauser Verlag, thèse de l'université de Standford (, Etats-Unis), 1975.

[Smith & al. 82] Smith D. C., Irby C., Kimball R. & Verplank B. (1982), *Designing the Star User Interface*, BYTE, April 1982, pp. 242-282.

[Stallman 79] Stallman R. (1979), *Emacs: The Extensible, Customizable, Self-Documenting Display Editor*, MIT Artificial Intelligence Lab. Report, août 79.

[Sugiyama & al. 81] Sugiyama K., Tagawa S. & Toda M (1981), *Methods for visual understanding of hierarchical system structurs*, IEEE Trabsaction on Systems, Man and Cybernetics SMC-11, Feb. 1981 n°. 2, pp. 109-125.

[Sutherland 63] Sutherland Ivan (1963), *SketchPad: A Man-Machine Graphical Communication System*, in AFIPS Spring Joint Computer Conference, 1963, n° 23, pp. 329-346.

[Sutton & Sprague 78] Sutton J. & Sprague R. (1978), *A study of display generation and management in interactive business applications*, rapport technique n° RJ2392(31804), Laboratoire de recherche d'IBM San José (Etats-Unis), novembre 1978.

[Tallot 85] Tallot D. (1985), *Quelques propositions pour la mise en œuvre d'algorithmes combinatoires*, thèse de troisième cycle, université des sciences et techniques du Languedoc, Montpellier.

[Tankönyvkiadó 85] Magyar-francia tanitási nyeluü gimnázium, Exercices et problèmes de Mathématiques, Classes Ie - IVe, Edition Tankönyvkiadó, 1985.

[Tarjan 83] Tarjan R.E. (1983), *Data Structures and Network Algorithms*, Regional Conference Series in Applied Mathematic, SIAM.

[Teitelman 77] Teitelman W. (1977), *A Display Oriented Programmer's Assistant*, Intenational Journal of Man-Machine Studies, vol. 11, pp. 157-187 (1979), et aussi rapport technique de Xerox PARC, n° CSL-77-3, Palo Alto (Californie, Etats-Unis).

[Thimbleby 90] Thimbleby Harold (1990), *User Interface Design*, Frontier Series, Edition ACM Press, Amsterdam (Pays-Bas).

[This 97] This Hervé. (1997), *La téléolfaction*, Pour la science, n° 236, juin 1997, pp. 21.

[Turzik 83] Turzik D. (1983), *A Note on the Chromatic Number of Direct Product of Graphs*, Commentationes Mathematicae Universitatis Carolinae 24, 3.

[Williams 83] Williams G. (1983), *The Lisa Computer System*, Byte Magazine, n° 8(2), pp. 33-50.

[Williams 84] Williams G. (1984), *The Apple Macintosh Computer*, Byte, n° 9(2), pp. 30-54.

[Wolfberg 69] Wolfberg M. (1969), *An interactive graph theory system*, Ph. D. Thesis, université de Pennsylvania (Pennsylvanie, Etats Unis).

[Wolfram 91] Wolfram S. (1991), *Mathematica, A system for Doing Mathematics by Computer*, Addison-Wesley Publishing Compagny, 1991.

[Xuong 92] Xuong N.H. (1992), *Mathématiques Discrètes et Informatiques*, Masson, Paris.

[Yang & al. 96] Yang J., Shaffer C. A. & Heath L. S. (1996), *Swan: A Data Structure Visualization System*, Lecture Notes In Computer Science, n° 1027, pp. 520-523.

[Ziegler & al. 88] Ziegler J. E. & Fähnrich K.-P. (1988), *Direct Manipulation*, in Handbook of Human-Computer Intercaction, M. Helander (Eds.), Elsevier Science Publishers B. V., North-Holland, 1988, pp.123-133.

Annexe 1

Représentation d'un graphe dans un fichier texte

1. Description générale

CABRI Version 4.0, Nov 1997

graphe : le graphe

numéro_de_version : contenu dans l'en-tête, précise le numéro de la version du logiciel ayant créé le graphe. Il doit être compris entre 3 et 3.5 ou égale à 4.0 pour que *Cabri-graphes* accepte de lire le graphe.

no_labels : non_étiqueté, si non vide alors le graphe apparaîtra non étiqueté.
alpha_num : si non vide alors l'étiquetage du graphe sera alphanumérique sinon l'étiquetage est un entier.
without_undo : sans undo, si non vide alors la fonction Undo sera activée.

oriented : orienté, si non vide alors le graphe sera un graphe orienté.
multiple : multiple, si non vide alors le graphe sera un graphe multiple sinon simple.
loop : boucle, si non vide alors le graphe acceptera des boucles -> non
encore opérationnelle

grille : si non vide alors la maille de la grille doit être précisée sinon la grille sera inactive et de maille 20.
grid : maille_grille, maille de la grille; si elle est négative, la grille sera inactive
#vertices : sommet, nombre de sommets du graphe.

#edges : arête, nombre d'arêtes du graphe.
Ces valeurs permettent de réserver un morceau de la structure de données globale nécessaire pour le graphe (voir Vertices et Edges).

Vertices : ce mot consigne la suite du texte au sommet. Les sommets sont généralement "classés par ordre de création ; mais, pour fichier externe, il peut être utile d'avoir la liste des sommets non "classés".

sommet : un sommet, le nombre d'éléments de ce langage auxiliaire doit être égal à n.
degree : le degré du sommet pour un graphe non orienté avec occurrence des multiplicités, sinon le degré sortant d'un sommet dans le cas orienté.
un_voisin : un voisin du sommet pouvant apparaître autant de fois qu'il est adjacent au sommet ; pour chaque sommet, le nombre d'éléments de ce langage auxiliaire doit être égale au degré.
coord. : c_horizontale, c_vertical, les coordonnées horizontales et verticales du sommet. Elles ne peuvent être omises dans l'état actuel du logiciel, mais prévues comme optionnelles.
val. : valeur, valeur affectée au sommet en cas d'étiquetage numérique ; si elle est vide alors la valeur par défaut elle sera le numéro du sommet dans l'ordre de lecture.

shape : une_forme, la forme du dessin du sommet ; si aucune ou le nom "round" alors la forme est le cercle, si le nom est "squarre" alors la forme est un carré, et si le nom est "triangle" alors la forme est un triangle.
size : une_taille, la taille du dessin du sommet; par défaut elle sera égale à 7.
ident. : une_couleur, la couleur du sommet, par défaut le blanc.

color	couleur	numéro
white	blanc	0
red	rouge	1
green	vert	2
cyan	cyan	3
magenta	magenta	4
yellow	jaune	5
blue	bleu	6
black	noir	7

Cette table de correspondance entre les couleurs et le numéro interne n'est pas encore opérationnelle ; il faut utiliser les termes en anglais.

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

étiquetage : étiquette alphanumérique du sommet, si elle est vide alors cette étiquette est initialisée à une chaîne vide.

label : l'étiquette, la chaîne de caractères de l'étiquette est alphanumérique.

font : la police utilisée pour l'étiquette, un nom explicite ou un numéro seulement (voir la table suivante) :

police	numéro	police	numéro
chicago	0	San Fransisco	8
applFont	1	Toronto	9
New York	2	Cairo	10
Geneva	3	Los Angeles	11
Monaco	4	Times	20
Venice	5	Helvetica	21
London	6	Courrier	22
Athens	7	Symbol	23

Cette table de correspondance entre le nom des polices et le numéro interne. Il faut utiliser soit le nom de la police, soit le numéro.

size : une_taille_p, la taille de police utilisée.

face : un_style, le style utilisé pour l'étiquette, un nom explicite entre parenthèses ou un numéro seulement (voir table suivante) :

face	style	numéro
normal	normal	0
bold	gras	1
italic	italique	2
underline	souligné	3
outline	contour	4
shadow	ombragé	5
condense	condensé	6
extend	étendu	67

Cette table de correspondance entre le style de l'étiquette et le numéro interne. Il faut utiliser soit le nom en anglais, soit le numéro.

Edges : ce mot indique la fin des éléments relatifs aux sommets et commence les informations associées aux arêtes sortant de la norme, c'est-à-dire non un segment de droite (ou une courbe de Bézier cubique), de couleur différente du noir, de taille supérieure à 1 pixel ou pour un arc une flèche non positionnée à 1/3 de la longueur de la forme représentative de l'arête.

edge : une arête, mot clé désignant une nouvelle arête hors norme.
extrémité1 - extrémité2 : les extrémités définissant l'arête.

size : la taille du dessin de l'arête ; par défaut elle sera égale à 7. Pour des raisons pratiques cette valeur doit être impaire et comprise entre 1 et 15.
color : la couleur du sommet, par défaut le noir.
alpha : position_flèche, un entier compris entre 0 et 1000 ; la position de la flèche sur l'arête (en réalité cette position est comprise entre 0 et 1, mais pour des raisons pratiques j'ai préféré mettre un entier)

point_control : point_contrôle, mot clé désignant les deux points de contrôle non extremité d'une courbe de Bézier cubique.
c_horizontale, c_vertical : les coordonnées horizontales et verticales du deuxième point de contrôle
c_horizontale, c_vertical : les coordonnées horizontales et verticales du troisième point de contrôle

2. Exemples de fichier texte

Nous proposons quatre exemples de fichier texte regroupant les principaux choix possibles de l'utilisateur.

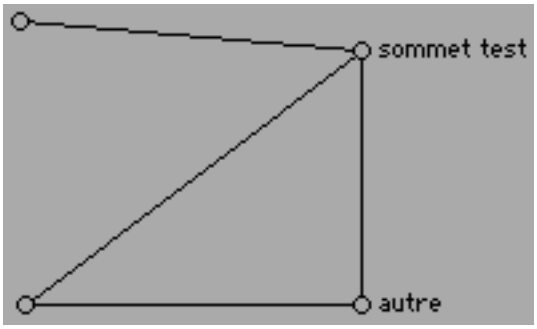
2.1 Graphe simple non orienté et étiqueté

CABRI Version 4.0, Nov 1997

alpha_num
grid : -20
#vertices: 4 #edges: 4

Vertices: 4
0 degree: 1 1
coord.: 112 78 val.: 0 shape: round size: 3 ident.: white offsets: 7 3
1 degree: 3 3 2 0
coord.: 240 89 val.: 1 shape: round size: 3 ident.: white offsets: 7 3
2 degree: 2 3 1
coord.: 240 184 val.: 2 shape: round size: 3 ident.: white offsets: 7 3
label: autre font: 1 (applFont) size: 9 face: 0 (normal)
3 degree: 2 1 2
coord.: 114 184 val.: 3 shape: round size: 3 ident.: white offsets: 7 3

Edges: 0



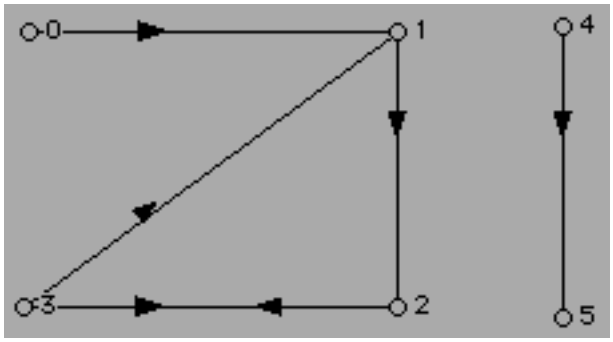
2.2 Graphe simple orienté

CABRI Version 4.0, Nov 1997

oriented
grid : -20
#vertices: 6 #edges: 6

Vertices: 6
0 degree: 1 1
coord.: 112 69 val.: 0 shape: round size: 3 ident.: white offsets: 7 3
1 degree: 1 2
coord.: 250 69 val.: 1 shape: round size: 3 ident.: white offsets: 7 3
2 degree: 1 3
coord.: 250 172 val.: 2 shape: round size: 3 ident.: white offsets: 7 3
3 degree: 2 2 1
coord.: 110 172 val.: 3 shape: round size: 3 ident.: white offsets: 7 3
4 degree: 1 5
coord.: 312 67 val.: 4 shape: round size: 3 ident.: white offsets: 7 3
5 degree: 0
coord.: 312 176 val.: 5 shape: round size: 3 ident.: white offsets: 7 3

Edges: 0



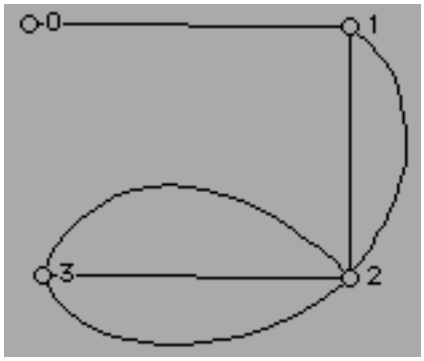
2.3 Graphe multiple et non orienté

CABRI Version 4.0, Nov 1997

multiple
grid : -20
#vertices: 4 #edges: 6

Vertices: 4
0 degree: 1 1
coord.: 147 88 val.: 0 shape: round size: 3 ident.: white offsets: 7 3
1 degree: 3 2 2 0
coord.: 267 89 val.: 1 shape: round size: 3 ident.: white offsets: 7 3
2 degree: 5 3 3 3 1 1
coord.: 267 183 val.: 2 shape: round size: 3 ident.: white offsets: 7 3
3 degree: 3 2 2 2
coord.: 152 182 val.: 3 shape: round size: 3 ident.: white offsets: 7 3

Edges: 3
edge 3 - 2 size: 1 color: black
point_control: 181 120 249 160
edge 3 - 2 size: 1 color: black
point_control: 164 213 224 221
edge 2 - 1 size: 1 color: black
point_control: 286 162 304 118



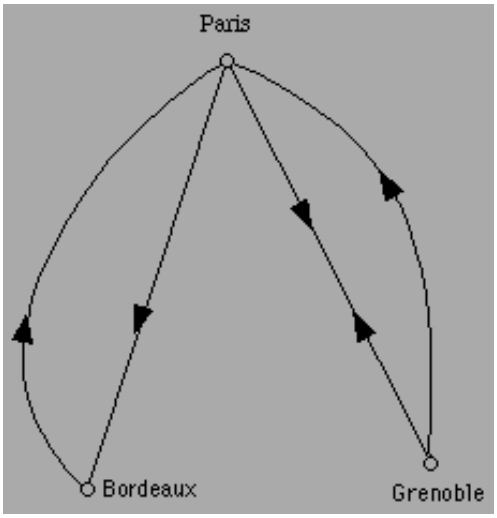
2.4 Graphe multiple orienté et étiqueté

CABRI Version 4.0, Nov 1997

alpha_num
oriented multiple
grid : -20
#vertices: 3 #edges: 5

Vertices: 3
0 degree: 1 1
coord.: 118 281 val.: 0 shape: round size: 3 ident.: white offsets: 7 4
label: Bordeaux font: 21 (Helvetica) size: 10 face: 0 (normal)
1 degree: 3 2 2 0
coord.: 183 81 val.: 1 shape: round size: 3 ident.: white offsets: -12 -13
label: Paris font: 20 (Times) size: 12 face: 0 (normal)
2 degree: 1 1
coord.: 278 269 val.: 2 shape: round size: 3 ident.: white offsets: -17 18
label: Grenoble font: 1 (applFont) size: 10 face: 0 (normal)

Edges: 5
edge 0 - 1 size: 1 color: black alpha: 333
point_control: 43 220 129 111
edge 1 - 0 size: 1 color: black alpha: 600
edge 1 - 2 size: 1 color: black alpha: 333
edge 1 - 2 size: 1 color: black alpha: 647
point_control: 277 212 294 125
edge 2 - 1 size: 1 color: black alpha: 333



Annexe 2

Caractérisation des courbes de Bézier basée sur les points fixes

La représentation des graphes ou "Graph Drawing" a passionné les chercheurs à travers le monde et donné lieu à une littérature très dense. Le but de notre étude est de décrire les différentes formes d'une courbe de Bézier dessinée interactivement en utilisant l'environnement informatique *Cabri-graphes*. Nous donnons une autre caractérisation géométrique des courbes de Bézier cubique en utilisant les propriétés des points de contrôle fixés sur la courbe.

La philosophie de *Cabri-graphes* est de créer interactivement des graphes, lesquels peuvent habituellement avoir des arêtes multiples. La représentation d'une arête courbe nécessite quatre points de contrôle, comme pour les courbes de Bézier cubiques. Deux points de contrôle sont les extrémités de la courbe et les deux autres points de contrôle sont des points externes à la courbe (ils n'appartiennent pas à la courbe), mais ils permettent de la définir. On remplace ces deux points de contrôle externes par deux points de contrôle "internes".

1. Définition des courbes de Bézier

La représentation générale d'un polynôme cubique est paramétrée par la fonction suivante : $Q(t) = (X(t), Y(t))$, où $X(t)$ et $Y(t)$ sont chacun des polynômes cubiques. La fonction Q admet comme fonctions dérivées $Q'(t) = (X'(t), Y'(t))$ et $Q''(t) = (X''(t), Y''(t))$ respectivement. Les courbes paramétrées sont définies pour les valeurs restreintes du paramètre. Ces valeurs sont prises généralement sur l'intervalle [0,1]. Deux représentations sont alors possibles.
La représentation de base pour la fonction Q(t) est appelée la représentation par puissance. Elle est définie par :

$$Q(t) = \sum_{i=0}^{3} \frac{1}{i!} \alpha_i t^i \qquad (1)$$

L'équation (1) est un cas spécial d'une forme plus générale. La courbe admet aussi une représentation par une combinaison de ses points de contrôle et des fonctions de base :

$$Q(t) = \sum_{i=0}^{3} P_i B_i(t) \quad (2)$$

où les P_i sont les points de contrôle et les $B_i(t)$ les fonctions de base.

Bien que les courbes planaires cubiques paramétrées soient très flexibles, il existe quelques contraintes dans la diversité de leur caractérisation. Pour nos travaux, nous avons utilisé les articles [3] et [4]. En particulier, les auteurs ont montré que la boucle, l'aiguille et le point d'inflexion sont mutuellement exclusifs. Ils ont aussi démontré que la présence d'une boucle, d'une aiguille ou d'un point d'inflexion peut être déterminée en examinant la fonction :

$$F(t) = \det(Q'(t), Q''(t)) = X'(t)Y''(t) - X''(t)Y'(t)$$

Cette fonction est proportionnelle au signe de la courbure de la courbe au point Q(t). Le zéro de F(t) indique la présence d'un point d'inflexion, puisque, pour un tel point, les vecteurs de la dérivée première et de la dérivée seconde sont linéairement dépendants. Cette dépendance montre que F(t) est une fonction cubique. De plus, en utilisant la représentation par les puissances pour la fonction Q(t), on montre que le terme cubique est nul. Ainsi, F(t) est une fonction quadratique et elle est de la forme :

$$F(t) = pt^2 - 2qt + 2r \quad (3)$$

avec $p = [a_2, a_3]$, $q = [a_3, a_1]$, $r = [a_1, a_2]$,

où [.,.] désigne le déterminant formé par les composants des deux vecteurs.

Les solutions pour les valeurs de t de l'équation F(t) = 0, indique aussi la présence d'un point d'inflexion. Le discriminant $G = q^2 - 2pr$ devient une valeur importante.

Wang, Su et Lu [5],[4] ont montré que la caractérisation de la courbe est entièrement déterminée à partir des valeurs *p*, *q* et *r* comme suit :

si p = 0, alors la courbe contient exactement un point d'inflexion associé à la valeur du paramètre t = r/q. Sinon, le point $Q(t_0)$ sur la courbe est

(1) une boucle si $G < 0$ et $t_0 = \frac{q+\varepsilon\sqrt{-3G}}{p}$ ($\varepsilon = \pm 1$)

(2) une aiguille si $G = 0$ et $t_0 = \frac{q}{p}$

(3) il existe deux points d'inflexion sur la courbe si et seulement si

$G > 0$ et $t_0 = \frac{q+\varepsilon\sqrt{G}}{p}$ ($\varepsilon = \pm 1$)

Su et Liu ont adapté le résultat ci-dessus à la représentation de la courbe de Bézier. Leur approche consiste à fixer six des huit degrés de liberté dans l'équation de la courbe (chaque point de contrôle possède une position dans le plan, et il engendre ainsi deux degrés de liberté). Les points des extrémités et les directions des vecteurs tangents, aux valeurs du temps t = 0 et t = 1, sont dessinés. Stone et DeRose [2] dessinent les trois premiers points de contrôle. Ils analysent les courbes cubiques planaires paramétrées lorsque le quatrième point de contrôle est déplacé dans le plan. La courbe est alors représentée sous la forme canonique, et la courbe est appelée la courbe canonique.

Dans les deux cas sont fixés le premier de ces points de contrôle ou le dernier et une direction du vecteur tangent à ces points (et même les deux autres points de contrôle). Notre étude repose sur le fait que nous fixons trois points de la courbe et non les points de contrôle excepté le premier de la courbe, qui est par définition un point de contrôle.

2. Courbes de Bézier et invariants

La description, pour générer le diagramme de caractérisation, procède d'abord par sélectionner une base et dans notre cas, des bases de Bézier. On choisit alors un système de coordonnées fixant les trois points de la courbe. Ces points sont le point P_0 (le premier point de la courbe), et les points R_1 et R_2 appartenant à la courbe. Il existe aussi deux réels t_1 et t_2, avec $t_1 \in]0,1[$ et $t_2 \in]0,1[$ associés aux points R_1 et R_2 respectivement, tels que :

$$Q(t_1) = R_1 = P_0B_0(t_1) + P_1B_1(t_1) + P_2B_2(t_1) + P_3B_3(t_1)$$
$$Q(t_2) = R_2 = P_0B_0(t_2) + P_1B_1(t_2) + P_2B_2(t_2) + P_3B_3(t_2) \qquad (4)$$

2.1 Courbe de Bézier cubique

La représentation d'une courbe de Bézier cubique planaire Q(t) est définie par quatre points de contrôle dans le plan, notés P_0, P_1, P_2, et P_3. D'après la définition de base (2) nous obtenons la relation :

$$Q(t) = \sum_{i=0}^{3} \binom{3}{i} t^i (1-t)^{3-i} P_i \qquad (5)$$

Les fonctions de base B_i sont les polynômes de Bernstein correspondant à :

$$B_0(t) = (1-t)^3 \qquad B_1(t) = 3t(1-t)^2$$
$$B_2(t) = 3t^2(1-t) \qquad B_3(t) = t^3$$

Ces courbes montrent l'influence de chaque point de contrôle dans la forme finale de la courbe. Pour une valeur particulière du paramètre *t* , on additionne la valeur obtenue correspondante aux quatre polynômes. On vérifie que P_0 est le point le plus influent pour la valeur de *t* égale à 0. Lorsque la valeur de*t* augmente, les polynômes B_0 et B_1 déterminent fortement la forme de la courbe de Bézier, et l'influence de B_2 et B_3 commencent. Les points de contrôles P_1 et P_2 produisent un effet maximal quand le paramètre t prend les valeurs 1/3 et 2/3 respectivement.

Pour ces valeurs $t_1 = 1/3$ et $t_2 = 2/3$, les polynômes B_i sont définis par :

$B_0(t_1) = B_3(t_2) = 8/27$ $B_1(t_1) = B_2(t_2) = 4/9$

$B_2(t_1) = B_1(t_2) = 2/9$ $B_3(t_1) = B_0(t_2) = 1/27$

Alors l'équation (4) est équivalente à :

$$R_1 = \frac{8}{27} P_0 + \frac{4}{9} P_1 + \frac{2}{9} P_2 + \frac{1}{27} P_3 \quad (6)$$

$$R_2 = \frac{1}{27} P_0 + \frac{2}{9} P_1 + \frac{4}{9} P_2 + \frac{8}{27} P_3 \quad (7)$$

Les points de contrôle P1 et P2 sont définis par :

$$P_1 = 3 R_1 - \frac{3}{2} R_2 - \frac{5}{6} P_0 + \frac{1}{3} P_3 \quad (8)$$

$$P_2 = -\frac{3}{2} R_1 + 3 R_2 + \frac{1}{3} P_0 - \frac{5}{6} P_3 \quad (9)$$

Et l'équation (5) peut s'écrire :

$$Q(t) = P_0 + (9 R_1 - 9/2 R_2 - 11/2 P_0 + P_3)t + (-45/2 R_1 + 18 R_2 + 9 P_0 - 9/2 P_3)t^2 + (27/2 R_1 - 27/2 R_2 - 9/2 P_0 + 9/2 P_3)t^3 \quad (10)$$

La courbe de Bézier cubique est entièrement définie avec seulement un paramètre *t* et les quatre points de la courbe.

2.2 Invariants

Définition

Un point-fixe est un point de la courbe de Bézier cubique dont la valeur du paramètre associé *t* a été fixée.

Alors nous donnons une autre définition des courbes de Bézier cubique. Elle est définie par quatre points R_0, R_1, R_2, et R_3 de la courbe associés respectivement aux valeurs suivantes de *t* : $t_0 = 0$, $t_1 = 1/3$, $t_2 = 2/3$, $t_3 = 1$.

Remarques

1) $R_0 = P_0$ et $R_3 = P_3$

2) Les valeurs de t_1 et t_2 peuvent être deux valeurs quelconques, mais les valeurs 1/3 et 2/3 sont les meilleures possibles pour les polynômes de Bernstein B_1 et B_2 (voir la section précédente).

3) L'équation (10) définit complètement la courbe de Bézier cubique.

Théorème

Il existe un point-fixe S_i pour chaque famille de droites P_iP_{i+1} avec i = 0, 1 ou 2, quand un point-fixe R_i se déplace.

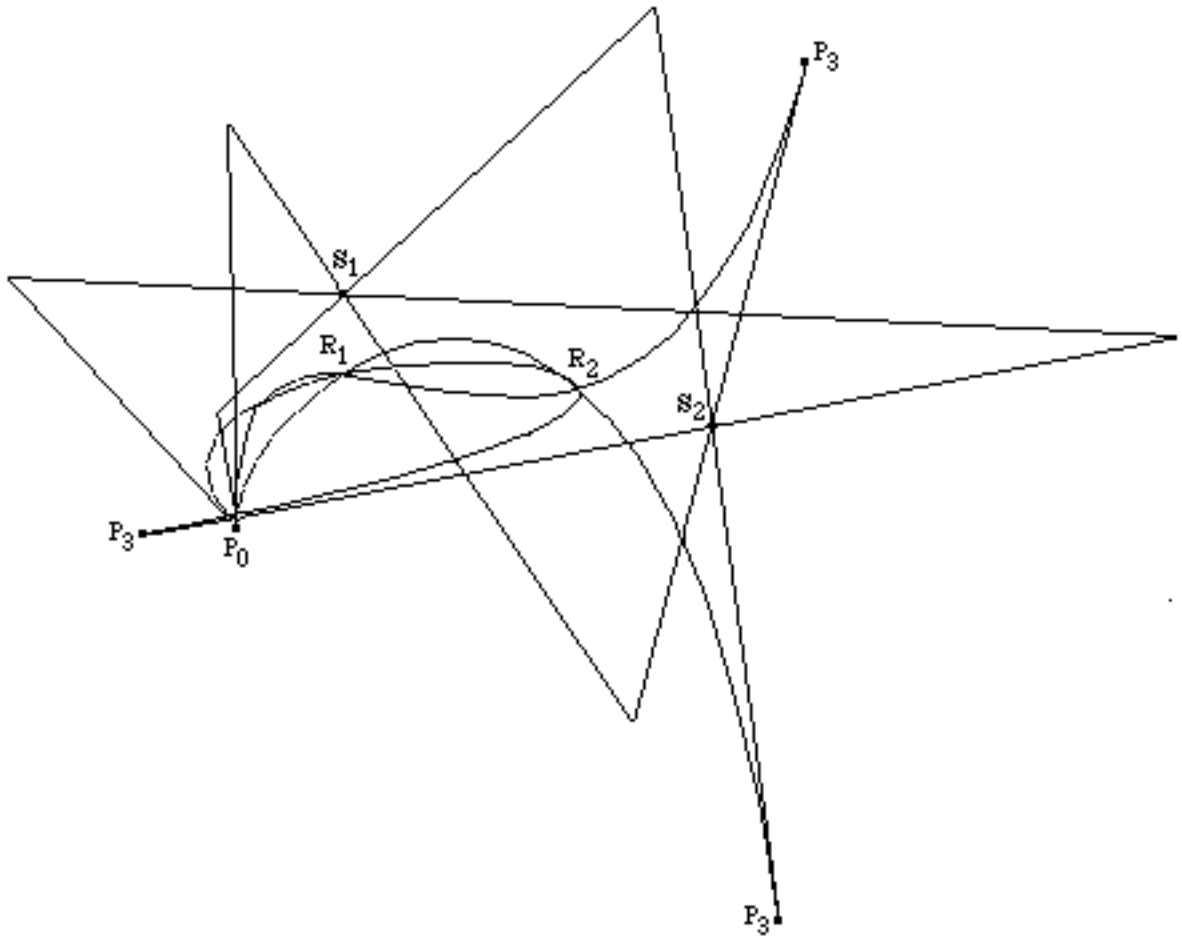


Figure 1 : Trois points-fixes sur une ligne polygonale.

Démonstration

Nous avons trois cas.

$P_0P_1 = \quad 3R_1 - 3/2\ R_2 - 11/6\ P_0 + \ 1/3\ P_3$

$P_1P_2 = -9/2\ R_1 + 9/2\ R_2 + \ 7/6\ P_0 - \ 7/6\ P_3$

$P_2P_3 = \ 3/2\ R_1 - \ 3\ R_2 - \ 1/3\ P_0 + 11/6\ P_3$

1 cas : P_0 déplacé

La famille de droites (P_2P_3) admet $S_2 = P_3$ comme point fixe.

La famille de droites (P_1P_2) admet $S_1 = 3/14\ R_1 - 12/7\ R_2 + 1/2\ P_3$ comme point fixe.

La famille de droites (P_0P_1) admet $S_0 = -\ 15/11\ R_1 + 9/11\ R_2 - 2/11\ P_3$ comme point fixe.

2 cas : R_1 déplacé

La famille de droites (P_2P_3) admet $S_2 = P_3$ comme point fixe.

La famille de droites (P_1P_2) admet $S_1 = -\ 3/2\ R_2 + 1/18\ P_0 + 4/9\ P_3$ comme point fixe.

La famille de droites (P_0P_1) admet $S_0 = P_0$ comme point fixe.

3 cas : R_2 déplacé

La famille de droites (P_2P_3) admet $S_2 = P_3$ comme point fixe.

La famille de droites (P_1P_2) admet $S_1 = -\ 3/2\ R_1 + 4/9\ P_0 + 1/18\ P_3$ comme point fixe.

La famille de droites (P_0P_1) admet $S_0 = P_0$ comme point fixe.

4 cas : P_3 déplacé

La famille de droites (P_2P_3) admet $S_2 = -\ 9/11\ R_1 + 18/11\ R_2 + 2/11\ P_0$ comme point fixe.

La famille de droites (P_1P_2) admet $S_1 = 12/7\ R_1 - 3/14\ R_2 - 1/2\ P_0$ comme point fixe.

La famille de droites (P_0P_1) admet $S_0 = P_0$ comme point fixe.

Remarques

Quelques points particuliers sont à signaler lorsque P_3 se déplace.

1) Les points P_0 et P_1 sont confondus, alors P_3 vérifie l'équation :

$$P_3 = -\ 9R_1 + 9/2\ R_2 + 11/2\ P_0$$

2) Les points P_1 et P_2 (= S_1) sont confondus alors P_3 est aussi le point S_2.

3) Si les points P_3 et P_2 sont confondus, alors P_3 a pour coordonnées :

$$P_3 = -\ 27/7\ R_1 + 27/7\ R_2 + P_0$$

4) Lorsque P_3 et P_0 sont confondus, les coordonnées deux points de contrôle P_1 et P_2 sont :

$$P_1 = 3\ R_1 - 3/2\ R_2 - 1/2\ P_0$$

$$P_2 = -3/2\ R_1 + 3\ R_2 - 1/2\ P_0$$

3. Caractérisation géométrique

3.1 Cas Général

Nous construisons un système de coordonnées en prenant les points-fixes de la courbe de Bézier. Ils définissent les cotés d'un triangle, car ils sont non alignés et tous différents. Nous définissons donc un système avec les coordonnées suivantes pour les points fixes : $P_0 = (0,0)$, $R_1 = (0, b)$, $R_2 = (a, b)$ où a et b sont des réels non nuls, et $P_3=(x,y)$ avec x et y des réels (voir Figure 2).

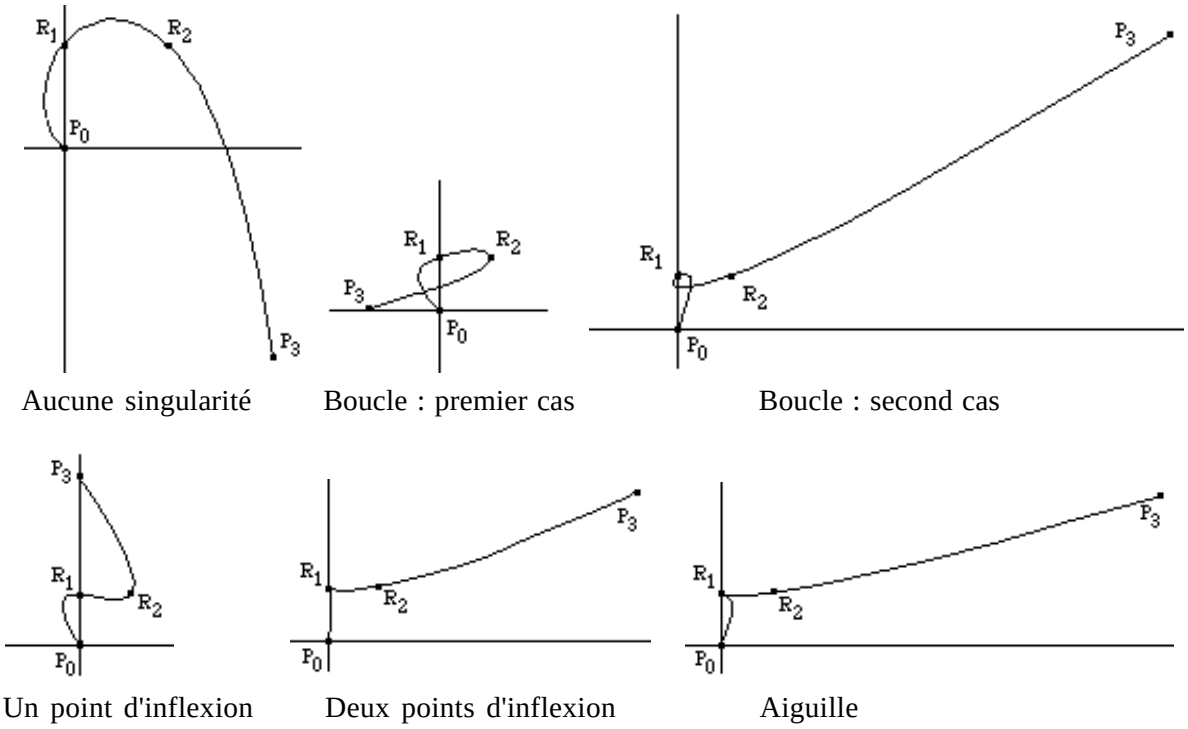


Figure 2 : Courbes de Bézier cubiques dans sa forme canonique.

Nous savons que toute équation du second degré représente une conique et que la forme de la conique dépend de la valeur de trois invariants [1]. De plus l'expression implicite de cette conique est déterminée par ces trois invariants. Soit (E) l'expression générale d'une courbe quadratique alors les invariants sont définis par :

$$(E) : aX^2+bY^2+ 2hXY+2gX+2fY+c = 0$$

$$D = abc + 2fgh - af^2 - bg^2 - ch^2$$

$$I = a + b$$

$$J = ab - h^2$$

et

conique	D	J	D/I
ellipse	$\neq 0$	sign +	sign -
hyperbole	$\neq 0$	sign -	/
parabole	$\neq 0$	0	/

Tableau 1 : Ce tableau résume la situation pour les coniques utilisées par la suite.

Substituons maintenant dans l'équation(10) les coordonnées des points fixés P_0, R_1 et R_2 afin de déterminer les composants des fonctions X(t) et Y(t) :

$$X(t) = (-9/2\ a + X)\ t + 9\ (2\ a - 1/2\ X)\ t^2 + 9/2(-3\ a + X)\ t^3$$

$$Y(t) = (\ 9/2\ b + Y)\ t - 9/2\ (b + Y)\ \ t^2 + 9/2\ Y\ t^3 \qquad (11)$$

Cette équation peut être utilisée pour calculer les valeurs caractéristiques p, q, et r définies dans l'équation (3) :

$$p = 3^5\ \ (\ \ bX + aY - 3\ ab\ )$$

$$q = 3^4/2\ (\ 3\ bX + aY - 9\ ab\ )$$

$$r = 3^2/2\ (\ 7\ bX + aY - 27\ ab\ ) \qquad (12)$$

$$G(X,Y) = q^2 - 2pr$$

$$G(X,Y) = 3^7/2^2\ (\ -bX^2 - 14abXY - aY^2 + 30ab^2X + 66a^2bY - 81a^2b^2\ ) \qquad (13)$$

G(X,Y) est l'expression générale d'une conique. Les invariants de G ont comme valeur :

$D = 3^4\ a^4\ b^4\ C_D > 0$, avec $C_D > 0$ est un réel

$$I = -\ 3^7/4\ (\ a^2 + b^2\ )$$

$$J = -\ 3^{19}/4^2\ a^2\ b^2$$

Ainsi G(X,Y)=0 est une équation d'hyperbole. Choisissons une valeur pour a et b, par exemple a=b=1, alors l'équation (13) devient :

$$G(X,Y) = 3^7/2^2\ (\ -X^2 - 14XY - Y^2 + 30X + 66Y - 81\ )$$

ayant pour centre de l'hyperbole le point (9/2, 3/2)

comme direction de l'axe de principal : 45°

et l'équation simplifiée dans le repère central : $3{,}375X^2 - 4{,}5Y^2 = 15{,}187$

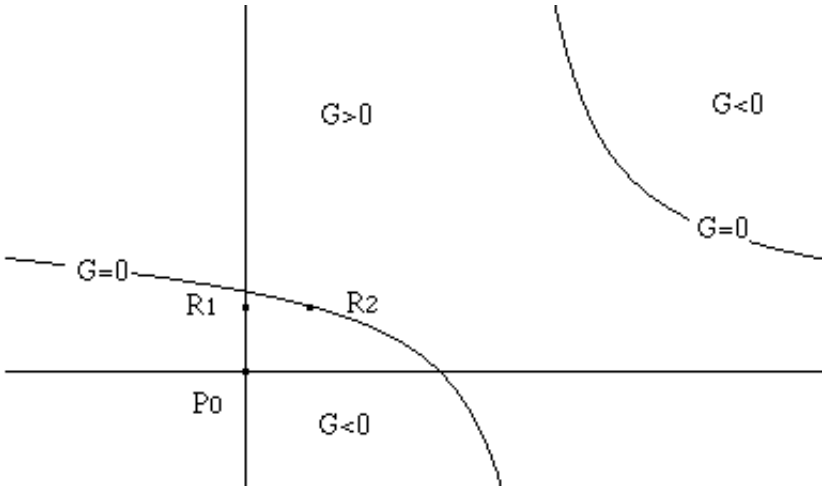


Figure 3 : Signe de G.

Deux cas, $p \neq 0$ et $p = 0$, doivent être discutés séparément.

3.1.1 $p \neq 0$

(1) Le segment de courbe admet une boucle pour $t \in [0,1]$ si et seulement si

$$< \frac{q + \varepsilon\sqrt{-3G}}{p} < 1$$

$$G < 0$$

• si $p > 0$, alors cette équation est équivalent à

$$q + \sqrt{-3G}$$

$$q - \sqrt{-3G}$$

$$G < 0$$

Après rationalisation

$$p^2 + 4q^2 - 2pq - 6pr > 0$$

$$2q^2 - 3pr > 0$$

$$p > q > 0$$

$$G < 0$$

Substituons (12) dans l'inégalitié ci-dessus, on obtient

$$X^2 + 2XY + 3Y^2 - 3X - 3Y > 0$$

$$X^2 - XY - X + 6Y > 0$$

$$3X + Y - 9 > 0$$

$$X + Y - 3 > 0$$

$$G < 0$$

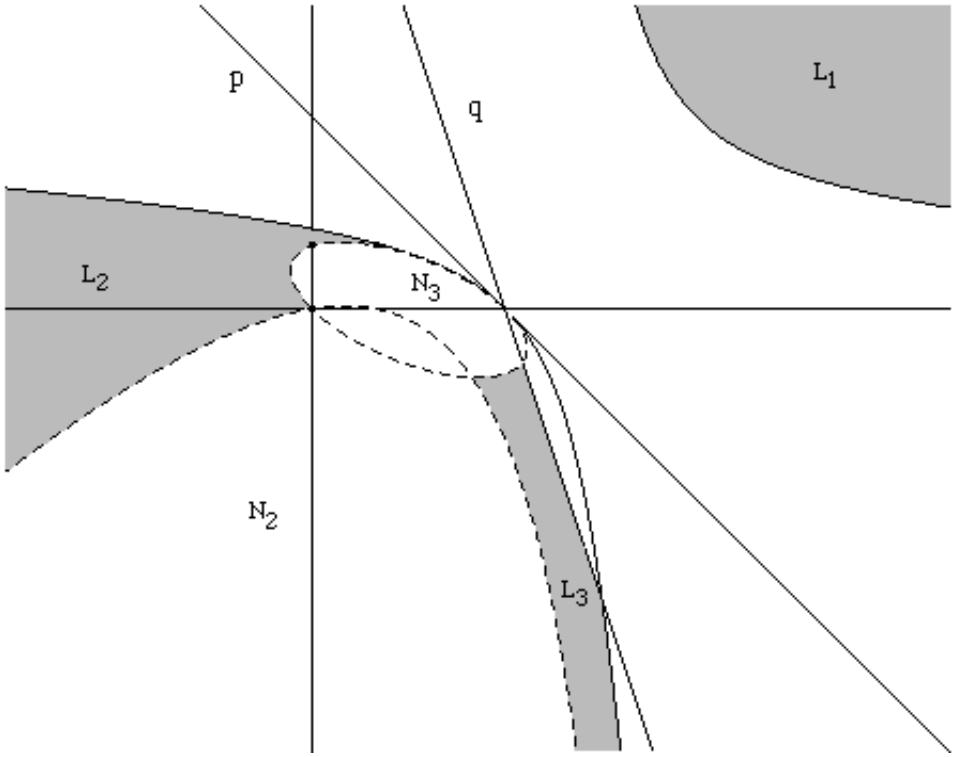


Figure 4 : Region de boucle (en gris).

H(X,Y) : $X^2 + 2XY + 3Y^2 - 3X - 3Y = 0$ -> une ellipse

avec pour centre de la courbe le point (3/2, 0)

direction de l'axe de principal : -22,5°

équation simplifiée dans le repère central :

$0{,}659X^2 + 3{,}841Y^2 = 2{,}5312$

H(X,Y) : $X^2 - XY - X + 6Y = 0$ -> une hyperbole

avec pour centre de la courbe le point (6, 11)

direction de l'axe de principal : 67,5°

équation simplifiée dans le repère central :

$24{,}8531X^2 - 144{,}85Y^2 = 3600$

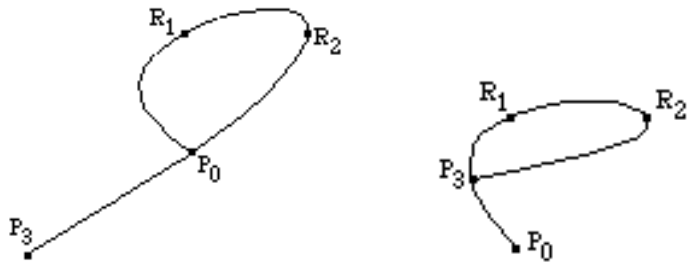


Figure 5 : Deux courbes ayant une boucle pour t=0 et t=1. Nous obtenons une ellipse et une hyperbole.

On conclue immédiatement qu'il existe une boucle sur la courbe segment de courbe si $P_3 \in L_1$, par contre si $P_3 \in N_1$ ou $P_3 \in N_2$ alors la courbe n'admet aucun point singulier ou d'inflexion (la région n'apparaît passur la figure 6, mais sur le dessin final en fin de l'article).

• Pour p<0, on obtient de façon analogue

$$X^2 + 2XY + 3Y^2 - 3X - 3Y > 0$$
$$X^2 - XY - X + 6Y > 0$$
$$3X + Y - 9 < 0$$
$$X + Y - 3 < 0$$
$$G < 0$$

Alors il existe une boucle sur le segment de courbe si $P_3 \in L_2$ ou L_3 et aucune boucle si $P_3 \in N_3$ (en fait pour cette partie du plan la courbe n'admet pas point singulier ou d'inflection).

(2) Le segment de courbe possède une aiguille quand $t \in [0,1]$ si

$$0 < \frac{p}{q} < 1$$
$$G = 0$$

• Si p>0, cette équation peut s'écrire comme

$$p - q > 0$$
$$p > q > 0$$
$$G = 0$$

Après rationalisation:

$$3X + 5Y - 9 > 0$$
$$X + Y - 3 > 0$$
$$3X + 5Y - 9 > 0$$
$$G = 0$$

en utilisant (12). Ainsi la courbe possède une aiguille pour $P_3 \in C_1$.

• Si p < 0, alors on a de façon similaire

$$3X + 5Y - 9 < 0$$
$$X + Y - 3 < 0$$
$$3X + 5Y - 9 < 0$$
$$G = 0$$

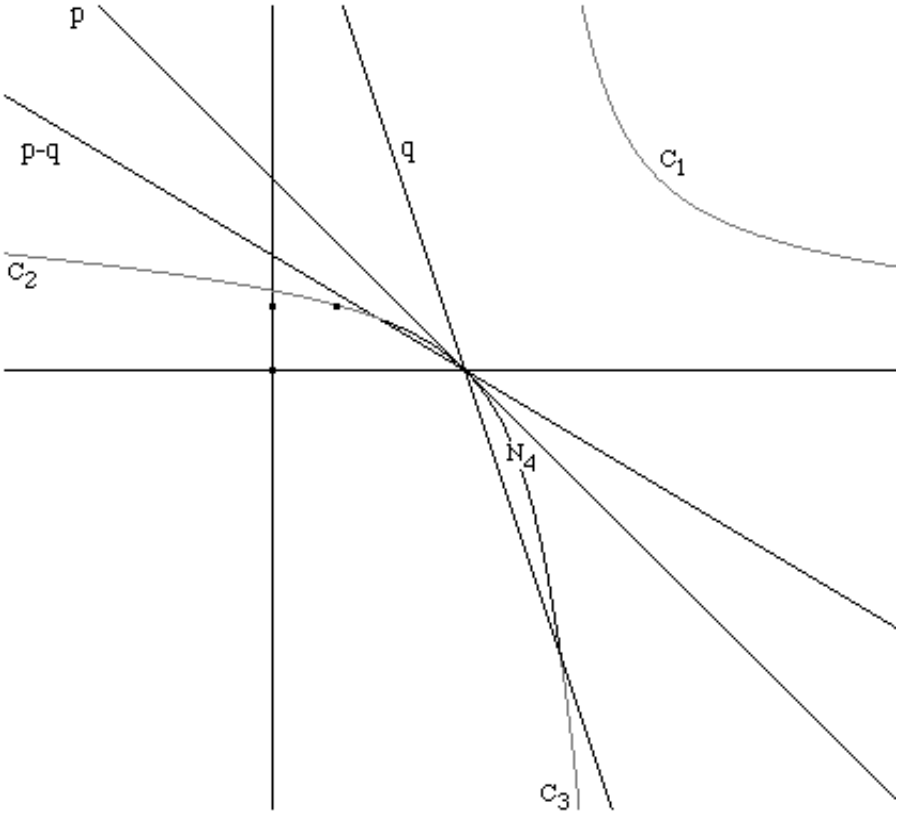


Figure 6 : Région des aiguilles (en gris).

Il existe une aiguille sur la courbe si $P_3 \in C_2$ ou C_3 et de là aucune aiguille n'apparaît pas pour $P_3 \in N_4$ (la courbe ne présente aucun point singulier ou d'inflexion sur cette partie d'hyperbole).

De (3),on peut évaluer la courbature de la courbe aux points extrémités :

$$k_0 = \frac{[Q'(0),Q''(0)]}{|Q'(0)|^3} \quad \text{et} \quad k_1 = \frac{[Q'(1),Q''(1)]}{|Q'(1)|^3}$$

soit $Q(t) = \alpha_0 + \alpha_1 t + \frac{1}{2}\alpha_2 t^2 + \frac{1}{2}\alpha_3 t^3$

alors $Q'(1) = \alpha_1$ $Q''(1) = \alpha_2$ et $Q'(1) = \alpha_1 t + \alpha_2 + \frac{1}{2}\alpha_3$ $Q''(1) = \alpha_2 + \alpha_3$

Ainsi l'inégalité $k_0 k_1 < 0$ est équivalente à

$$r \times (r - q + \frac{1}{2} p) < 0$$

En utilisant les substitutions :

$7X + Y - 27 > 0$ $\qquad$ $7X + Y - 27 < 0$

$7X + 19Y - 27 < 0$ $\qquad$ $7X + 19Y - 27 > 0$

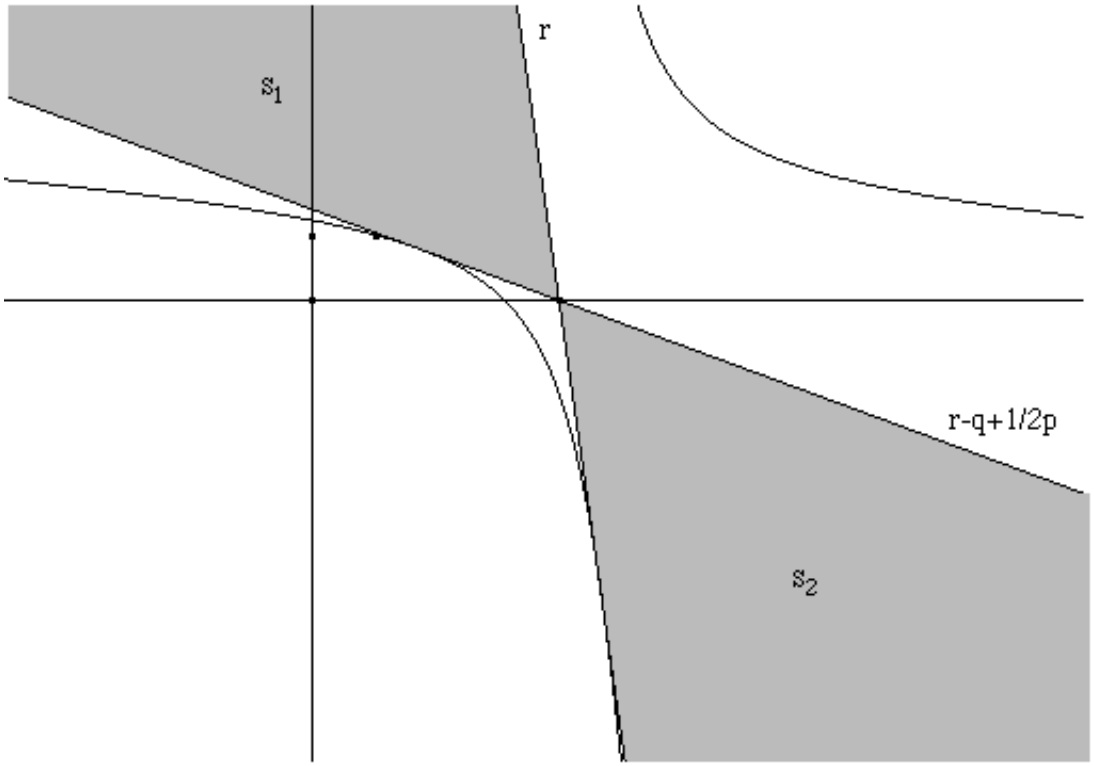


Figure 7 : Région d'un point d'inflexion (en gris).

Alors le segment de courbe possède un point d'inflexion si et seulement P_3 appartient à l'une des deux régions S_1 et S_2.

La courbe possède deux points d'inflexion si et seulement si

$$0 < \frac{q + \varepsilon \sqrt{G}}{p} < 1$$

$$G > 0$$

• si p>0, alors cette équation est équivalent à

$$p - q > \sqrt{G}$$

$$q > \sqrt{G}$$

$$G > 0$$

Après rationalisation

$$p - 2q + 2r > 0$$

$$r > 0$$
$$p > 0$$
$$G > 0$$

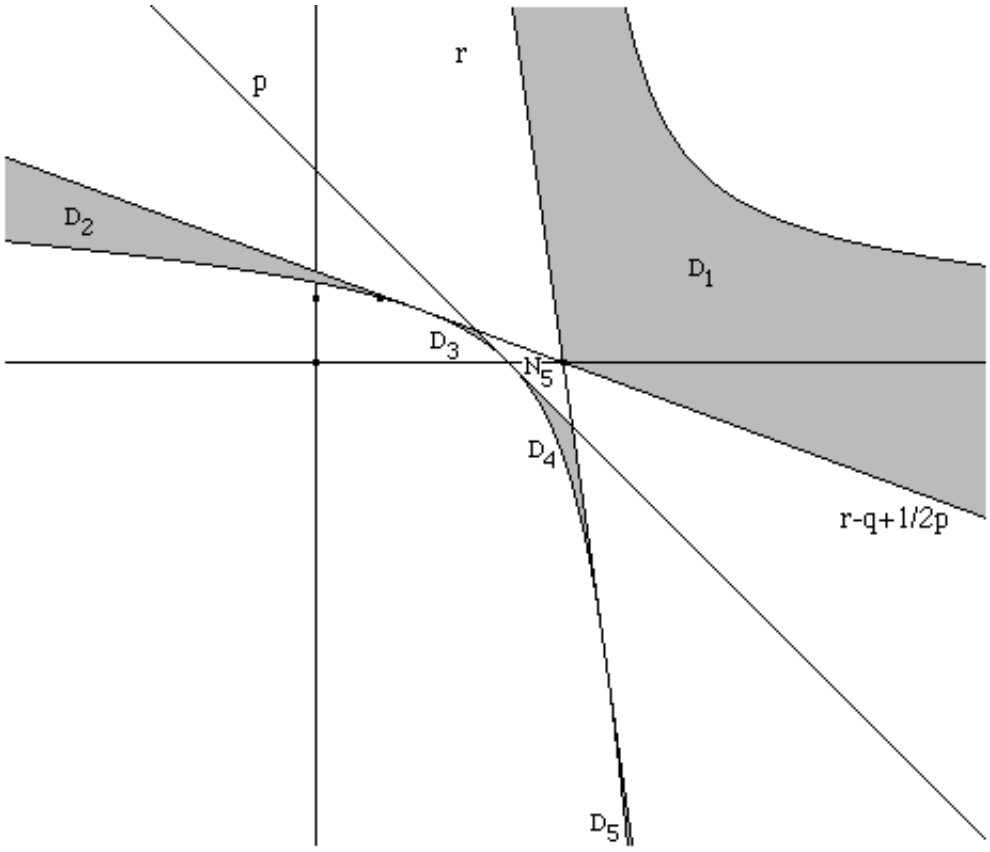


Figure 8 : Région de deux points d'inflexion (en gris).

Substituant (12) dans l'inégalités précédente, on obtient

$$7X + 19Y - 27 > 0$$
$$7X + Y - 27 > 0$$
$$X + Y - 3 > 0$$
$$G > 0$$

Donc la courbe possède deux points d'inflexion si $P_3 \in D_1$.

• Pour $p<0$, on a de façon similaire

$$7X + 19Y - 27 < 0$$
$$7X + Y - 27 < 0$$
$$X + Y - 3 < 0$$
$$G > 0$$

Alors la courbe possède deux points d'inflexion si $P_3 \in D_2$ ou D_3 ou D_4 ou D_5. Par contre pour $P_3 \in N_5$ la courbe ne possède pas de point singulier ou d'inflexion.

Il apparaît de l'étude fait que la courbe de Bézier cubique, lorsque P_3 se déplace, n'admet aucun point singulier, ni de point d'inflexion pour les régions notées N_1, N_2, N_3, N_4 et N_5.

3.1.2 p = 0

Dans ce cas, la courbe ne possède pas de singularité mais contient un point d'inflexion associé à la valeur du paramètre t :

$$t = \frac{r}{q}$$

Alors la courbe déterminée par le paramètre $t \in [0,1]$ possède un point d'inflexion si

$$0 < \frac{r}{q} < 1$$
$$p = 0$$

• Si q>0, () peut s'écrire comme

$$q - r > 0$$
$$q > r > 0$$
$$p = 0$$

Après rationalisation

$$10X + 4Y - 27 > 0$$
$$7X + Y - 27 > 0$$
$$3X + Y - 9 > 0$$
$$p = 0$$

On conclut qu'il existe un point d'inflexion si $P_3 \in S_1$.

• Pour q<0, on obtiebt de façon analogue

$$10X + 4Y - 27 < 0$$
$$7X + Y - 27 < 0$$
$$3X + Y - 9 < 0$$
$$p = 0$$

Alor la courbe possède un point d'inflexion $P_3 \in S_2$.

En résumé, le théorème concernant la distribution des points de singularité et des points d'inflection du segment de courbe est obtenu :

Théorème

$$\text{Si } P3 \in \begin{cases} L_1 \cup L_2 \cup L_3, \\ C_1 \cup C_2 \cup C_3, \\ S_1 \cup S_2, \\ D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_5, \\ N_1 \cup N_2 \cup N_3 \cup N_4 \cup N_5. \end{cases}$$

$$\text{alors la courbe de Bézier cubique possède } \begin{cases} \text{une boucle,} \\ \text{une aiguille,} \\ \text{un point d'inflexion,} \\ \text{deux points d'inflexion,} \\ \text{aucun point singulier ou d'inflexion.} \end{cases}$$

3.2 Cas de dégénérescence

3.2.1 P_0, R_1 et R_2 alignés et différents

Soit le repère orthnormal ayant pour origine le point $P_0 = (0,0)$. On définit les points suivants :

$R_1 = (\alpha,0)$ avec $\alpha > 0$, $R_2 = (\beta,0)$ avec $\beta \neq 0$ et α

et un point $P_3 = (X,Y)$ du plan

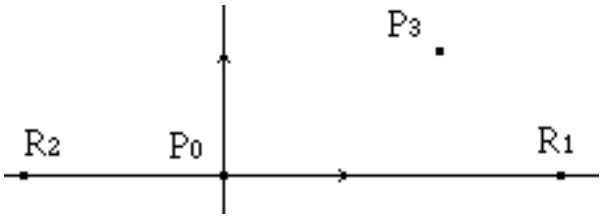


Figure 9 : P_0, R_1 et R_2 sont alignés.

Par l'équation (5) on obtient:

$X(t) = (9\alpha - 9/2\beta + X)t + (-45/2\alpha + 18\beta - 9/2 X)t^2 + (27/2\alpha - 27/2\beta + 9/2 X)t^3$

$Y(t) = Y t - 9/2 Y t^2 + 9/2 Y t^3$

$p = 3^5 (-2\alpha + \beta) Y$

$q = 3^4 (- 2\alpha + \beta/2) Y$

$r = 3^2 (- 4\alpha + \beta/2) Y$

Alors $G = 3^7 Y^2 (- 4\alpha^2 + 4\alpha\beta - \beta^2/4)$

Donc le signe de G est équivalent au signe de: $- 4\alpha^2 + 4\alpha\beta - \beta^2/4$. Cette équation dépend uniquement de β, parce que $\alpha > 0$ par hypothèse. Comme le discriminant est strictement positif, il existe deux racines distinctes β_1 et β_2. Deux cas apparaissent : si $P_3 \in P_0R_1$ (i.e. Y = 0) la courbe de Bézier est une courbe dégénérée (un segment de droite); sinon par une analyse similaire à celle du paragraphe précédent on obtient:

p = 0		$\beta = 2\alpha$	la courbe a un point d'inflexion
$p \neq 0$	$G < 0$	$4\alpha < \beta < \beta_2$ $-7/8\,\alpha < \beta < 0$	la courbe a une boucle
	$G = 0$	$\beta = \beta_1$ ou β_2	la courbe a une aiguille
	$G > 0$	$\beta > \beta_2$ and $\beta < \beta_1$	il existe deux point d'inflexion
sinon			la courbe n'a pas de point particulier

3.2.2 R1 = R2 = R

Soit le repère orthnormal ayant pour origine le point $P_0 = (0,0)$. On définit les points suivants :

$R = (\alpha,0)$ avec $\alpha > 0$,

et un point $P_3 = (X,Y)$ du plan

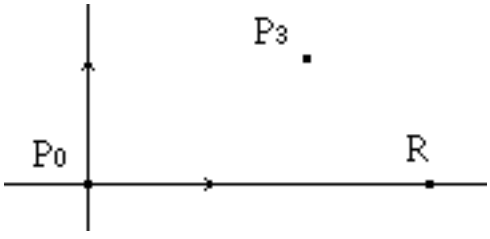


Figure 10 : R_1 et R_2 sont confondus.

Par l'équation (5) on obtient:

$X(t) = (9/2\alpha + X)t - 9/2 (\alpha + X)t^2 + 9/2 X t^3$

$Y(t) = Yt - 9/2Yt^2 + 9/2Yt^3$

$p = -3^5 \alpha Y$

$q = -3^5/2 \alpha Y$

$r = -7*3^2/2 \alpha Y$

Alors $G(X,Y) = -3^7/2^2 \alpha^2 Y^2$.

On a deux cas. Premier cas : si $P_3 \notin P_0R$ (i.e. $Y \neq 0$) alors $G < 0$ et $0 < t_0 < 1$ pour tout point P_3. Donc la courbe possède une boucle pour tous les points P_3 élément de la droite (P_0R). Deuxième cas : $P_3 \in P_0R$. Par définition tous les points de contrôle sont alignés. La courbe Bézier est une courbe dégénérée, un segment de droite définie par deux des quatre points de contrôle de la courbe.

3.2.3 $P_0 = R_1$ ou R_2

Nous avons deux cas: le point P_3 est ou n'est pas un élément de la droite (P_0R_2), si le point de contrôle P_0 est aussi le point R_1. La courbe de Bézier possède une boucle pour tout point P_3 qui n'est pas un élément de la droite (P_0R_1), parce que la courbe possède un point double, et par définition c'est une caractérisation de la boucle. Pour un point P_3 sur la droite (P_0R_2), tous les points de contrôle sont des éléments de cette droite (P_0R_2). On a une courbe dégénérée, un segment de droite d'extrémité deux des quatre points de contrôle de notre courbe.

Légende des couleurs de la figure 11 :

- uneboucle
- une aiguille
- un point d'inflexion
- deux points d'inflexion
- aucune singularité ou de point d'inflexion

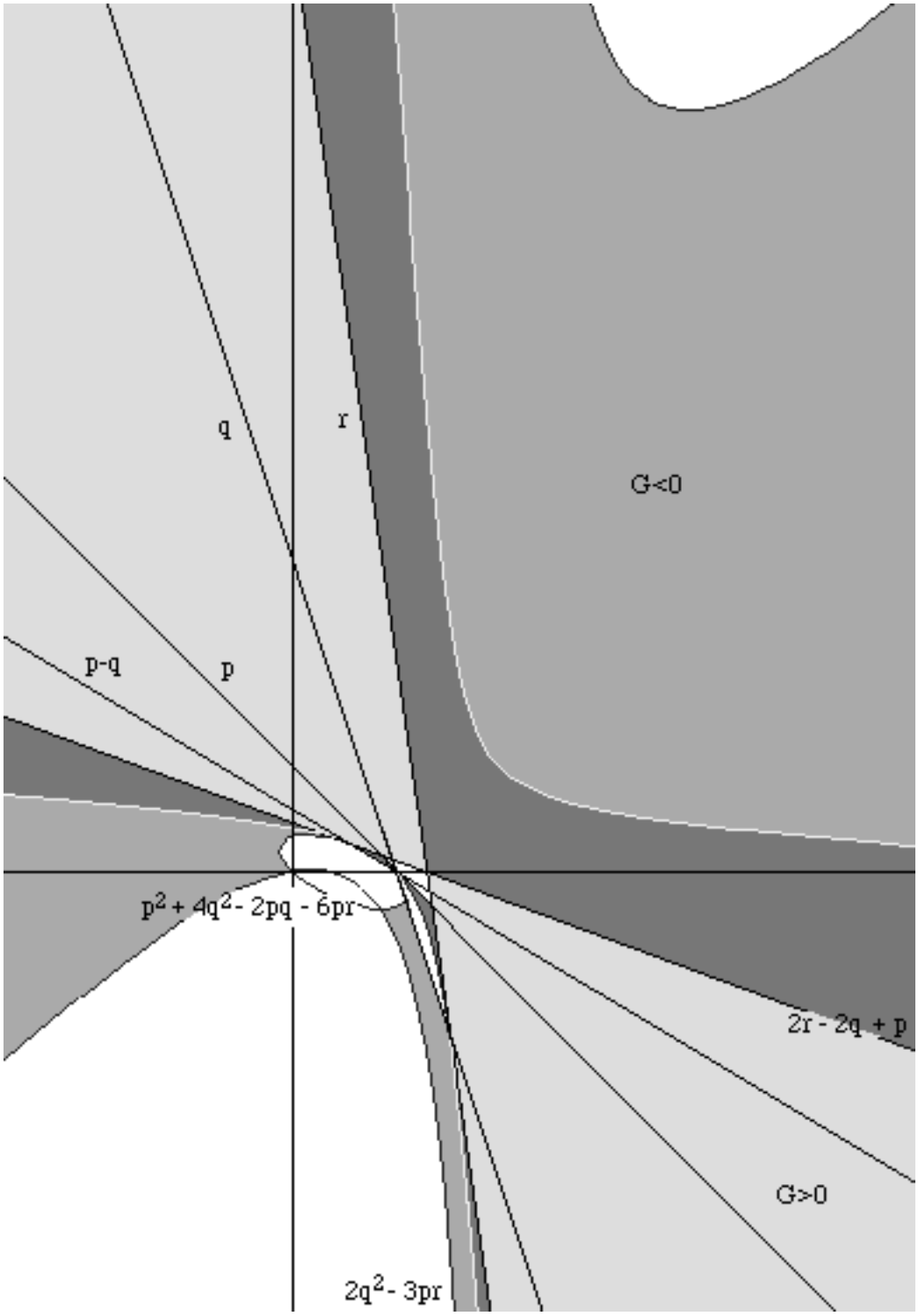


figure 11 : Nature de la courbe de Bézier suivant la position du point P_3.

Logiciels

Cabri-géomètre II, Jean-Marie Laborde et Franck Bellemain, [6] et [8].

Cabri-graphes 3.10, Jérôme Bordier, Yves Carbonneaux, Jean-Marie Laborde, Mourad Madani : logiciel développé au Laboratoire de Structures Discrètes et de Didactique (LSD2), au sein de l'Institut de Mathématiques Appliquées de Grenoble (IMAG), "Un outil pour la recherche et l'enseignement de la théorie des graphes". Il permet de créer de façon interactive des graphes, de le manipuler interactivement et de calculer ses propiétées [6] et [7].

Coniques 4.0, Jacques-Olivier Haenni : ce logiciel calcule, à partir des paramétres de l'expression générale d'une cônique, la forme et les invariants associés à cette côniques.

Bibliographie

[1] Spain Barry (1957), Analytical Conics, Pergamon Press.

[2] Stone Maureen C., DeRose Tony D. (1989), A Geometric Characterization of Parametric Cubic Curves, ACM Transactions on Graphics, vol. 8, n°3, July, pp. 147-163.

[3] Su Bu-Qing, Liu Ding-Yuan (1983), An affine invariant and its Application in Computational Geometric, Scientia Sinica (Series A), vol. 24, n°3, March, pp. 259-267.

[4] Su Bu-Qing, Liu Ding-Yuan (1989), Computational Geometry, Curve and Surface Modeling, Acadamic Press.

[5] Wang C. Y. (1981), Shape classification of the parametric cubic curve and parametric B-Spline cubic curve, Computer-Aided Design, vol. 13, n°4, July, pp. 199-206.

[6] CABRI-Géomètre (démonstration) est accessible par ftp anonyme sur le site ftp.imag.fr, /pub/Cabri/ .

[7] Carbonneaux Y (1998), Cabri 4.0 Manuel Utilisateur, Laboratoire Leibniz, université Joseph Fourier, Grenoble I.

[8] Laborde J-M. & Bellemain Franck (1994), Cabri-géomètre II, version 1.0 MS-DOS et Macintosh, Texas Intruments.

Table des matières