THESE DE DOCTORAT
DE L'UNIVERSITE DE BORDEAUX 1

Présentée par

Monsieur Maurin NADAL

Assistance à l'utilisateur novice dans le cadre du dessin de graphe à l'aide de méthodes d'apprentissage

Thèse soutenue à Talence le 16 décembre 2013 devant le jury composé de :

Madame Pascale KUNTZ-COSEREC	PR - Ecole Polytech. de l'Université de Nantes	Rapporteur
Monsieur Gilles VENTURINI	PR - Ecole Polytech. de l'Université de Tours	Rapporteur
Monsieur Alexandre DEMEURE	MCF - Université Joseph Fourrier Grenoble	Examinateur
Monsieur Sofian MAABOUT	MCF - Université Bordeaux 1	Examinateur
Monsieur Guy MELANÇON	PR - Université Bordeaux 1	Directeur de thèse

Rêver un impossible rêve
Porter le chagrin des départs
Brûler d´une possible fièvre
Partir où personne ne part

Jacques Brel, *La Quête*

Remerciements

Cette thèse n'aurait jamais pu atteindre son état final sans la participation d'un grand nombre de personnes et de structures. Je tiens à les remercier pour l'aide qu'ils m'ont apportée et pour les services dont j'ai pu bénéficier tout au long de ces trois ans.

Je tiens tout d'abord à citer les différentes structures avec lesquelles j'ai interagi, c'est à dire l'Université de Bordeaux 1, le centre Inria Sud-Ouest, l'Ecole Doctorale de Mathématique et d'Informatique, et surtout le LaBRI. J'ai pu grâce à ces structures travailler dans des conditions matérielles très confortables, mais surtout, j'ai pu rencontrer de nombreuses personnes et progresser dans mes recherches d'une manière inestimable. Je tiens aussi à remercier chaleureusement tous les personnels administratifs de ces structures sans lesquels rien ne fonctionnerait aussi bien.

Je tiens aussi à remercier l'Université de Bordeaux 1, l'Institut Polytechnique de Bordeaux, et l'ENSEIRB-Matmeca. C'est grâce à ces établissements que j'ai pu m'orienter avec succès vers la thèse à travers les trois années précédant cette dernière. Mais c'est aussi dans ces établissements que j'ai pu réaliser mes premiers enseignements dans de très bonnes conditions grâce à l'accompagnement mis en place par l'équipe enseignante, et je garde un très bon souvenir de cette expérience.

Mais toutes ces structures ne seraient pas grand chose sans les hommes et les femmes qui les animent et les font vivre.

Je suis aussi très reconnaissant envers tous les membres du LaBRI que j'ai pu rencontrer au fil des ans. En particulier ceux des équipes MaBioVis, et Gravite, qui m'ont accueilli. J'ai beaucoup apprécier la disponibilité de chacun, lorsque j'avais des questions pour eux, ce qui est arrivé très régulièrement. Je ne pourrais pas lister tous les noms, et certains m'échapperont sûrement, mais je tiens citer ici Patrick Mary, Sofian Maabout, David Auber, Romain Bourqui, Maylis Delest et Bruno Pinaud. Sans oublier les autres doctorants de l'équipe et les ingénieurs, François Queyroi, Benjamin Renoust, Antoine Lambert, Jonathan Dubois et Morgan Mathiaut, et tous les autres.

Plusieurs professeurs de l'ENSEIRB-Matmeca ont aussi une importance particulière dans mon parcours, que ce soit durant les 3 ans de formations qui ont précédé ma thèse, ou dans le cadre d'enseignements pour ma mission complémentaire. De la même manière, je ne pourrais ici citer les noms de toutes les personnes concernées, mais je tiens à remercier particulier Denis Lapoire, Frédéric Herbreteau et Georges Eyrolles et tous les professeurs de la section Informatique.

Ces années au LaBRI aurait aussi été très différentes sans la présence des autres doctorants. Je ne pourrais pas tous les citer, mais j'ai une pensée particulière pour Allyx, Pierre, Laetitia, Anna, Lorijn et tous les autres.

Toute ma gratitude va aussi vers mes relecteurs, Madame Pascale Kuntz-Cosperec et Monsieur Gilles Venturini, pour le temps qu'ils ont consacré à la lecture de mon manuscrit et l'évaluation du travail de thèse réalisé durant ces trois ans. Je tiens aussi à remercier mes examinateurs, Monsieur Alxandre Demeure et Monsieur Sofian Maabout pour leur temps et leur disponibilité.

Bien évidement, je dois aussi un grand merci à mes amis et ma famille, et pour n'en nommer qu'un, Laurent Dutertre. Une thèse correspond à une longue période d'engagement autour d'un sujet. Et pour ma part, c'est grâce au soutient de toutes ces personnes que j'ai pu réussir à me consacrer avec enthousiasme durant ces trois ans au sujet qui a été le mien. J'ajouterai aussi une

pensée particulière pour ma soeur Marine, pour l'aide inestimable qu'elle m'a apportée lors de la relecture finale.

Enfin, il me reste une dernière personne à remercier, sans qui cette thèse n'aurait tout simplement jamais vu le jour. Il s'agit bien évidement de Guy Melançon, qui a été mon directeur de thèse durant ces dernières années. J'ai besoin de conditions de travail relativement particulière pour pouvoir être productif, et Guy, à travers sa confiance et ses conseils, m'a donné les moyens d'être dans de telles conditions. J'ai aussi pu beaucoup progresser grâce à ses conseils et ses retours, durant l'écriture de ce manuscrit comme pendant la rédaction d'articles. J'ai cependant mis de côté un seul de ces conseils, celui de partir de Bordeaux pour trouver une thèse encore plus proche du domaine de l'apprentissage. Je n'ai pas le moindre regret d'avoir omis ce conseil car je pense que c'est en très grande partie grâce à lui que cette thèse s'est aussi bien passée et que j'ai pu autant apprécier ces trois ans.

Résumé

La motivation première de cette thèse peut être formulée dans ces termes : « Comment assister un utilisateur novice pour l'aider à obtenir un dessin de son graphe qui soit adapté à ses besoins ? ». À l'heure d'un accroissement constant du volume des données produites par notre société et l'exposition grandissante du grand public à ces données, cette réflexion est indispensable.
Pour l'instant, les méthodes de dessins, extrêmement nombreuses mais relativement peu accessibles, nécessitent une grande expertise tant dans le choix de la méthode que de son paramétrage pour obtenir un dessin de bonne qualité. De ce fait, les utilisateurs novices, qui n'ont généralement pas accès à une telle expertise, utilisent par défaut les algorithmes les plus connus même si ceux-ci ne correspondent pas parfaitement à leurs attentes. *In fine*, cela nuit à la qualité générale des dessins de graphes utilisés.
Pour répondre à cette problématique, la solution proposée dans le cadre de cette thèse consiste à mettre en place un système interactif proposant à l'utilisateur différents dessins pour un même graphe afin qu'il obtienne un résultat qui réponde correctement à ses besoins.
Pour sa partie dédiée à la visualisation, notre système repose sur une version modifiée de l'algorithme de dessin par modèle de force GEM [32] modifié afin d'être paramétrable beaucoup plus finement. Nous avons nommé cette version GaGEM. Le principe général de la modification consiste à définir un jeu de paramètres pour chaque sommet au lieu de définir ces paramètres pour l'ensemble du graphe. Cette modification permet d'obtenir des dessins très variés pour un même graphe. Cependant, cela induit aussi une explosion du nombre de paramètres ce qui rend la méthode impossible à utiliser manuellement.
Nous avons choisi d'utiliser un algorithme génétique pour réaliser ce paramétrage. L'objectif de notre AG étant de trouver un ensemble de conditions chacune associée à un jeu de paramètre. Il est possible à partir de cela de d'associer à chaque sommet des paramètres pour pouvoir dessiner le graphe à l'aide de GaGEM. Ce choix de l'encodage du génome a été guidé par notre volonté d'avoir des résultats réutilisables d'une exécution à l'autre. Cette problématique de la réutilisabilité a été au centre de nombreuses réflexions lors de la conception du système d'AG mis en place.
Ce système est centré sur une description facilement modifiable de la structure de l'algorithme génétique (manière de générer la population, définition de la fonction de fitness, fin d'algorithme, etc.). Cette structuration prend une forme modulable afin de pouvoir ajouter facilement de nouveaux modules inspirés de travaux réalisés par la communauté des AG. Par exemple, des modules dédiés à la recherche guidée par la curiosité ont été mis en place au sein du système.
Notre système a aussi la possibilité d'exécuter des algorithmes génétiques interactifs afin de permettre l'utilisation du système directement depuis un logiciel de dessin de graphe.

Table des matières

Chapitre 1

Introduction : L'humain, l'information et la visualisation

L'être humain est constamment amené à prendre de multiples décisions au quotidien. Elles peuvent prendre la forme d'une sélection, parmi deux ou plusieurs possibilités, ou de la création de toutes pièces d'une solution. Cela se ramène en général à l'identification de la meilleure alternative selon un ensemble de valeurs propres au décideur. Descartes rapproche cet acte du choix aux concepts de libre-arbitre et d'entendement. En effet, dans une de ses lettres [23], il avance l'idée qu'il est nécessaire de bien comprendre les tenants et aboutissants d'une décision, concept qu'il nomme entendement, afin de pouvoir exprimer pleinement son libre-arbitre lors de la prise d'une décision. En effet, le décideur n'est réellement libre de prendre une décision que si la part d'aléatoire dans ce choix est restreinte à son minimum. Un choix aléatoire n'étant pas considéré comme l'expression du libre-arbitre du décideur.

Il est donc nécessaire d'avoir un maximum d'informations concernant l'impact d'une décision pour choisir librement. La question devient alors : Quelle est la meilleure manière de permettre à un être humain d'acquérir toutes ces informations ? C'est dans le cadre de cette problématique générale que prend place le travail réalisé durant cette thèse. En effet, la visualisation de graphes, et plus généralement la visualisation de données, correspond à une des réponses proposées pour ce problème de l'acquisition d'une grande quantité d'informations. Ce premier chapitre présente une rapide mise en perspective allant de cette problématique générale jusqu'au sujet précis de cette thèse, qui est la génération de multiples dessins d'un même graphe, afin d'améliorer la qualité de la représentation des données pour l'utilisateur. Cette perspective présentera les avantages de cette approche ainsi que les contraintes, d'ordre naturel ou informatique, qui ont défini le cadre de ce travail.

1.1 L'être humain, entité physique douée et limitée

Le cerveau humain présente des capacités étonnantes pour assimiler et mémoriser une grande quantité d'informations. Pour l'instant, les mécanismes biologiques sous-jacents ne sont pas encore entièrement compris et expliqués (malgré de nombreux progrès réalisés par les neurosciences). Toutefois, de nombreux travaux ont tenté d'identifier ces processus et leurs limites en étudiant le comportement et la réaction d'êtres vivants dans certaines situations du quotidien [86, 48, 30]. Ces études de plus haut niveau ont permis d'apporter beaucoup d'éléments pour mieux comprendre les mécanismes du processus de prise de décision. De plus, comme nous allons le voir, certaines contraintes d'ordre biologique ont aussi un impact évident sur ce dernier.

1.1.1 L'être humain, un centre de décision, limites physiques et cognitives

Une étape préliminaire à la prise de décision consiste au rassemblement des données disponibles. Ces données peuvent prendre une multitude de formes différentes. La première, et généralement la plus importante, consiste en l'expérience personnelle du décideur. La mémoire, celle de toutes les expériences passées et des décisions déjà prises, fournit un contexte d'évaluation et d'estimation indispensable. Cependant, de nombreuses informations doivent venir de l'extérieur. Dans ce cas, plusieurs limites liées à la nature de l'être humain sont à prendre en compte.

La première concerne la manière d'acquérir ces informations. Les seules interfaces actuellement connues entre le cerveau et le monde extérieur sont les différents sens. Chacun d'eux permet en effet d'apporter des données caractérisant l'état courant de l'environnement. Ces données varient par leur contenu, mais aussi par leur forme. Par exemple, l'odorat permet de ressentir plusieurs odeurs en même temps, et avec de l'entraînement de les identifier individuellement. Mais il ne permet de ressentir que les odeurs ayant atteint notre nez, sa portée est donc très faible. Pour la vision, la situation est assez différente, car la portée est bien plus étendue et il est possible de se focaliser sur une petite part de l'environnement. Nous verrons dans un paragraphe suivant que la vue est un très bon moyen pour permettre une acquisition efficace d'une grande quantité de données.

La deuxième contrainte provient des limites des capacités de mémorisation. En effet, en général, un être humain ne sera pas capable de mémoriser l'ensemble des données qui lui sont présentées en un unique passage. Il est donc important de pouvoir revenir sur certaines informations afin de les assimiler correctement. Cela implique une forte interaction entre l'utilisateur et les informations.

La dernière contrainte a été mise en avant par des études réalisées sur le processus de prise de décision. En effet, il a été remarqué que la qualité de la décision diminue lorsque l'utilisateur est soumis à une trop grande quantité d'information [30]. Cet effet est fortement accentué lorsque la décision doit être prise en un temps limité [48]. Une solution permettant de diminuer cette dégradation consiste à synthétiser au mieux l'information avant de la fournir à l'utilisateur. L'idée derrière ce point consiste à tenter de minimiser la charge cognitive de l'utilisateur.

Comme nous allons le voir dans le prochain paragraphe, le fait de représenter graphiquement les données sous la forme de schémas et de graphiques permet de répondre partiellement à cette contrainte.

1.1.2 La vision, le sens le plus développé pour l'acquisition d'information

Des cinq sens disponibles chez l'être humain, la vue semble le plus approprié pour transmettre des informations abstraites au cerveau. Ce choix se justifie de deux manières. Tout d'abord, au niveau de contraintes physiques liées à ce sens. La vue permet une consultation indépendante du temps grâce à une détermination physique du point focal (contrairement à l'audition par exemple, où l'on ne peut entendre que les sons qui arrivent jusqu'à notre oreille). De ce fait, il est possible de consulter une ou plusieurs images aisément et de revenir au besoin sur certains points sans avoir à parcourir à nouveau l'ensemble des données. De plus, contrairement au toucher et au goût, aucun contact n'est nécessaire pour acquérir les données.

De ce fait, la vue est le sens qui allie la plus grande portée avec une précision importante. Pour ces raisons, c'est celui auquel a été consacré la plus grande partie du cortex chez les primates et chez l'homme (presque la moitié chez les primates selon Van Essen *et al.* [91]). Les

multiples couches de neurones permettent de reconnaître et d'identifier rapidement une grande variété d'éléments à travers leurs caractéristiques graphiques. De plus, une part importante du cortex visuel appartient au néocortex, ce qui permet une forte spécialisation grâce à la plasticité de cette aire cérébrale. Par exemple, la lecture devient de plus en plus rapide lorsqu'elle est pratiquée régulièrement.

Enfin, la question du support du média est aussi importante dans le cadre de cette problématique. En effet, il est important de pouvoir stocker, manipuler et consulter facilement les médias porteurs d'informations en cas de besoin. Les représentations graphiques (écrits et dessins) ont été pendant longtemps le seul moyen de conserver des informations durant une longue période. L'informatique a révolutionné ce domaine en permettant le stockage et l'accès d'un grand nombre de documents de tout type à l'aide d'un terminal unique. Cependant, la vue reste le sens le plus utilisé lors de leur consultation.

1.2 La visualisation de l'information

C'est donc assez naturellement que la représentation graphique des informations s'est développée. Les paragraphes suivants présentent un très rapide historique du domaine ainsi qu'un aperçu de différents types de visualisation qui ont été mis au point. Elle se focalisera ensuite sur le dessin de graphes, qui correspond au domaine dans lequel cette thèse prend place.

1.2.1 Représentation symbolique des données

Les représentations graphiques ont beaucoup évolué au fil du temps. Le cas des cartes géographiques est très intéressant. Il s'agit d'une représentation la plus réaliste possible de la géographie tout en utilisant une échelle très petite. Cela a conduit à l'introduction de symboles pour représenter les éléments importants. En effet, dans la mesure où il était impossible de les représenter à l'échelle, des dessins simplifiés ont d'abord été utilisés avant d'être progressivement remplacés par des symboles abstraits. Cela permet de faire facilement ressortir les informations importantes et facilite aussi la réalisation de la carte.

Des symboliques plus poussées ont parfois été aussi utilisées plus ponctuellement au sein d'une carte. Un exemple régulièrement cité est la carte de la campagne de Russie réalisée par C. Minard en 1869 sur laquelle est représenté le trajet de l'armée napoléonienne (voir figure 1.1). La largeur du trait indique le nombre d'hommes composant l'armée. De plus, une information liée à la température durant la retraite est ajoutée en bas de l'image. Cette image est très intéressante, car c'est la première fois que des informations non géographiques sont associées à une carte. Par chance, la campagne de Russie s'est réalisée d'ouest en est, ce qui facilite la lecture d'un tel schéma puisque cela correspond au sens de lecture occidental (gauche-droite). Même si ce détail peut paraître insignifiant, cela réduit la charge cognitive associée à la lecture de cette carte.

L'ensemble des applications du domaine de la visualisation de données correspond à une extension du principe appliqué à cette carte. En effet, l'objectif consiste en général à mettre en relation plusieurs données différentes afin de permettre à l'utilisateur d'assimiler les informations qu'elles contiennent. Au niveau des cartes, cela se traduit par diverses évolutions, soit en terme d'éléments symboliques ajoutés à la carte (et explicités au sein d'une légende), ou d'une déformation de la carte afin d'ajouter de l'information ou de mettre en valeur certaines caractéristiques. La figure 1.2a présente une carte déformée de manière à ce que la superficie de chaque pays

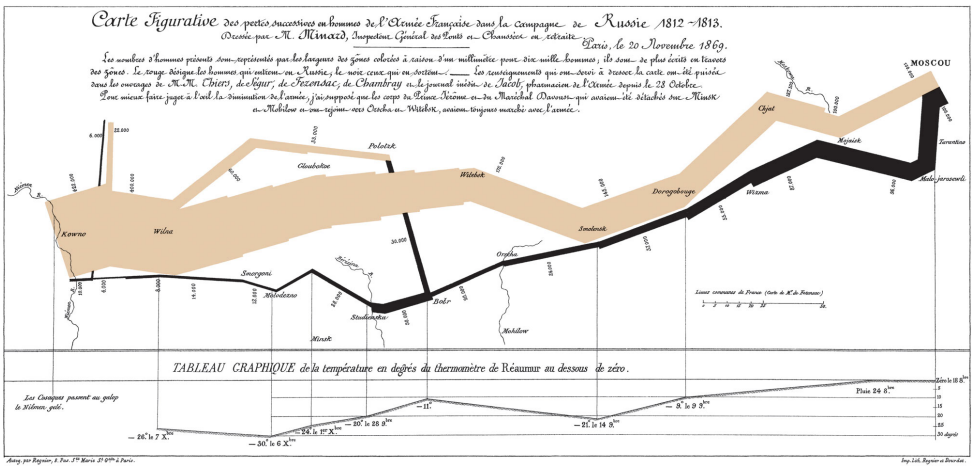


FIGURE 1.1 – Une carte de la campagne de Russie de Napoléon présentant la taille de l'armée en fonction de sa position

soit proportionnelle au nombre de papiers en sciences « dures » publiés par les chercheurs y travaillant. On remarque que les États-Unis, l'Europe et le Japon ont une forte domination sur les publications, alors que l'Afrique et le Moyen-Orient occupent un espace négligeable. La figure 1.2b présente une carte du métro de Rome. Les cartes de transports en commun sont en général déformées afin qu'il soit plus facile de suivre une ligne et d'identifier les correspondances.

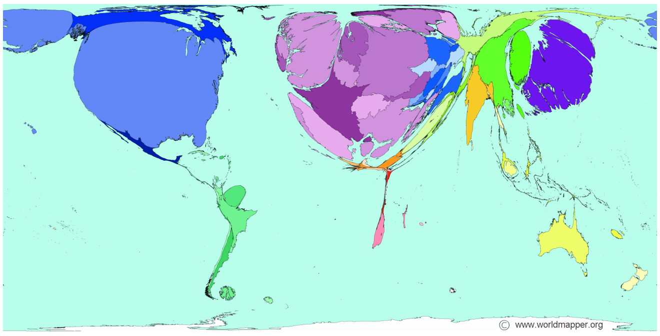
1.2.2 Abstraction et représentation

Une des caractéristiques communes de toutes ces cartes est qu'elles se basent toutes sur une réalité physique. D'une manière plus générale, il est possible de dire que tous les éléments qui y sont représentés sont géolocalisés, c'est à dire que la position des éléments dans le dessin correspond à leur position dans la réalité. Toutefois, la représentation graphique d'informations ne se limite pas à ce seul type de données. Il est en effet possible de représenter une très grande variété d'informations à l'aide d'outils visuels.

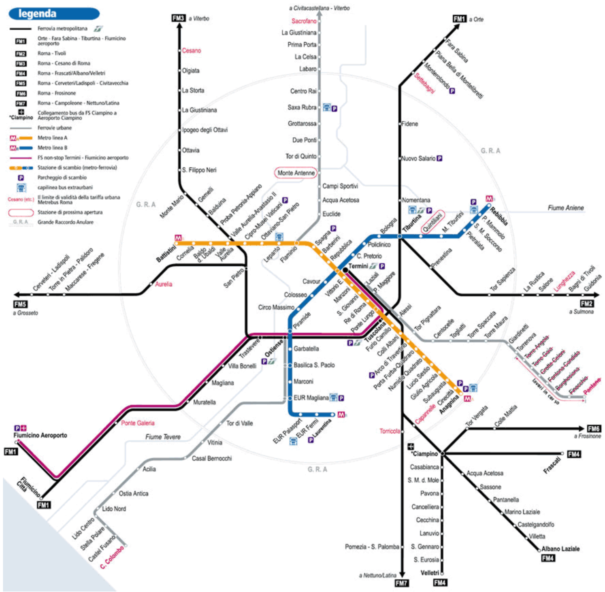
Le domaine de la visualisation de l'information est relativement jeune. Toutefois, la communauté scientifique ayant été très active, un grand nombre de publications sont disponibles. Plusieurs livres offrent un survol de différents aspects de ce domaine. On pourra conseiller la lecture du livre de Ware [93] qui se concentre plus sur une approche physiologique, en particulier au niveau de la perception des images par l'être humain. L'ouvrage de R. Spence [87] quant à lui se concentre sur les différents types de visualisations statiques et dynamiques qui ont été mises au point par la communauté.

Cependant, malgré ce développement important des techniques de visualisation, ces dernières sont loin d'être adoptées par le grand public. Les travaux de L. Grammel [44, 43] montrent les difficultés rencontrées par les novices pour construire des représentations graphiques intéressantes. Ces dernières étant fortement amplifiées lorsqu'il s'agit d'utiliser des représentations peu courantes.

Cette situation est toutefois en train d'évoluer, avec la diffusion au grand public d'une quantité toujours plus importante de données, en particulier grâce au mouvement « Open Data ». Cette mise à disposition d'informations s'accompagne d'outils qui permettent de les manipuler et les présenter. De plus, on observe aussi le recours de plus en plus fréquent à différents types



(a) Une carte du monde déformée en fonction du nombre de papiers de recherches publiés dans chaque pays.
Source : www.worldmapper.org



(b) Une carte du métro de Rome

d'infographies au sein de la presse. Ces deux points jouent un rôle majeur dans la prise de conscience de la puissance d'une visualisation adaptée au sein du grand public et dans la diffusion des techniques correspondantes.

1.2.3 Supériorité d'une représentation graphique adaptée

Il arrive régulièrement que des données soient présentées de manière brute, sous la forme d'un tableau présentant les différents éléments en ligne avec chacune de leurs caractéristiques en colonne. Ces représentations ont toutefois deux inconvénients majeurs. Tout d'abord, il est nécessaire de parcourir l'ensemble du tableau pour pouvoir avoir une idée globale de ce qu'il contient. De plus, toutes les données doivent être homogènes (c'est-à-dire présenter les mêmes types de caractéristiques) pour pouvoir être rassemblées au sein d'un même tableau.

La représentation graphique des données permet d'éviter ces deux inconvénients. En effet, le cerveau peut faire preuve d'une grande efficacité pour analyser rapidement une image et pour s'en faire une idée globale. Car les processus cognitifs utilisés lors d'une analyse graphique sont globalement moins lourds que ceux nécessaires pour la lecture et la comparaison d'un grand nombre de valeurs.

Les graphiques offrent aussi un second avantage grâce à la liberté qu'ils offrent à leur concepteur pour mettre efficacement en exergue certaines caractéristiques des données. En effet, contrairement aux tableaux où seul l'ordonnancement permet généralement d'influencer l'interprétation des données, les graphiques offrent une multitude d'outils pour permettre la mise en valeur de certains éléments. Ces outils sont assez variés et correspondent aux caractéristiques graphiques auxquelles l'être humain réagit instinctivement, comme la couleur, la forme, la position, la taille, les regroupements [93].

1.2.4 Variété des métaphores visuelles

De nombreuses métaphores visuelles ont été mises au point afin de présenter efficacement différents types de données. Quelques-unes vont être introduites ci-dessous. Cette liste est loin d'être exhaustive, mais elle permettra toutefois au lecteur de se faire une idée des différentes manières de transmettre des informations à travers une image.

Vocabulaire utilisé Chaque type de représentation permet d'afficher des données sous forme graphique. Nous allons ici définir rapidement quelques termes afférents à ces données.

- **Entité**, le terme « item » issu de l'anglais est aussi souvent utilisé. Cela correspond à un élément de base des données, chaque entité étant caractérisée par une ou plusieurs valeurs. Par exemple, si les données portent sur les élèves d'une classe, les entités correspondent aux différents élèves. Les différentes valeurs caractérisant chaque élève sont les notes obtenues dans chaque matière.
- **Valeur**, les caractéristiques de chaque entité sont les valeurs. Ce sont elles qui sont utilisées lors d'une représentation graphique des données. Ces valeurs peuvent avoir différents types. Les plus courantes sont les valeurs numériques (entières ou non). Toutefois, elles peuvent aussi être de type textuel (une chaîne de caractères), ou correspondre à un élément choisi parmi une liste prédéfinie. Dans l'exemple des données portant sur une classe, une caractéristique textuelle de chaque élève pourrait être son nom. Une caractéristique

choisie parmi une liste pourrait indiquer si l'élève est droitier ou gaucher (les listes peuvent toutefois être composées d'un nombre quelconque d'éléments).
- **Série**, une série correspond à un ensemble d'entités. Il arrive régulièrement qu'une représentation graphique permette de comparer plusieurs séries distinctes. Dans le cadre de l'exemple sur les élèves, les séries pourraient correspondre aux différentes classes.

Quelques métaphores visuelles

De très nombreuses métaphores visuelles permettent de présenter des informations. Chacune présente des intérêts différents, comme le type des données pouvant être représentées ou le fait de faciliter certaines tâches utilisateurs. Voici une liste (très peu exhaustive) de métaphores courantes :

- **Les histogrammes** représentent chaque item d'un ensemble avec une barre (en générale verticale). Les barres sont alignées selon une ligne de base (généralement le bas de la zone de dessin) et leur hauteur correspond à une valeur caractéristique de l'item qu'elle représente. Les histogrammes permettent d'identifier facilement les valeurs extrêmes et d'avoir une idée de la valeur moyenne. La figure 1.2 montre un histogramme présentant le nombre de lignes téléphoniques pour 100 habitants par pays. Les couleurs y sont utilisées pour indiquer le continent de chaque pays.

- **Les courbes** représentent une série ordonnée de points à l'aide d'une ligne reliant chacun de ces points. Cela permet par exemple de représenter facilement l'évolution d'une valeur dans le temps. Il est aussi possible d'afficher plusieurs courbes sur un même graphe afin de comparer différentes valeurs. La figure 1.3 présente par exemple deux courbes représentant les valeurs A et B. De plus, les zones comprises entre les deux courbes ont été colorées afin de pouvoir comparer ces deux surfaces plus facilement.

- **Les nuages de points** permettent de représenter un ensemble d'items selon deux valeurs caractéristiques. Chaque item est représenté par un symbole placé dans le plan selon ces deux valeurs. Ils sont entre autre utilisés pour identifier des regroupements d'items ou des corrélations entre certaines valeurs.

- **Les diagrammes en secteur**, ou plus communément appelés « camemberts » sont utile pour décrire une répartition. Ils se présentent sous la forme d'un disque découpé en plusieurs secteurs. Chaque item de l'ensemble est représenté par un secteur dont l'angle correspond à la valeur de l'item divisé par la somme de toutes les valeurs de l'ensemble.

- **Les aires empilées** sont moins courante et correspondent à un mélange entre les courbes et les diagrammes en secteur. Elles permettent de représenter l'évolution d'une répartition dans le temps. La figure 1.4 présente l'évolution des parts de marché des ventes de téléphones portables entre 2009 et 2011. Les répartitions sont affichées verticalement, et la part de marché d'une entreprise à une date donnée correspond à la hauteur de la zone de sa couleur à cette date.

De nombreuses autres métaphores, inspirées de ces dernières ou créées de toutes pièces ont aussi été mises au point. Pour plus d'informations sur ces différentes représentations et leur possibilité, l'ouvrage de R. Spence fournit un très bonne base [87].

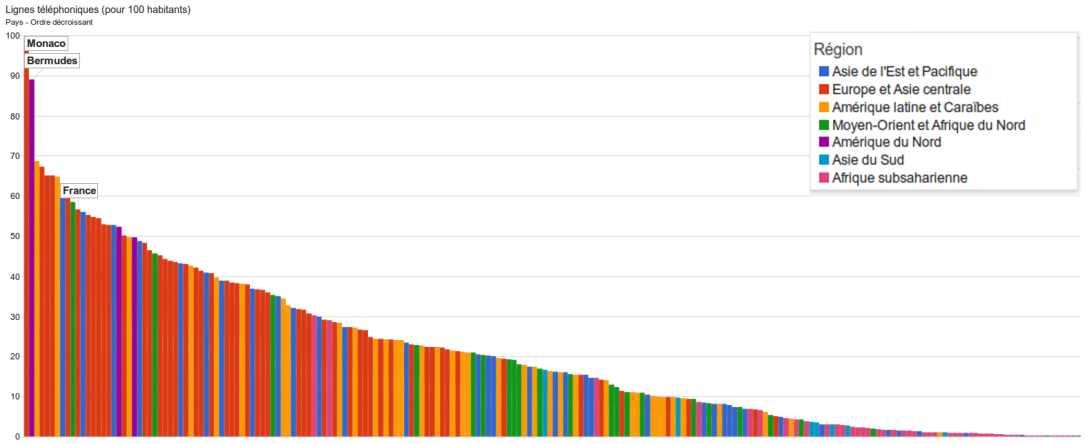


FIGURE 1.2 – Un graphique présentant le nombre de lignes téléphoniques pour 100 habitants. Il est intéressant de remarquer que les 4 premiers pays sont des paradis fiscaux. Source : Google public data explorer, Banque mondiale

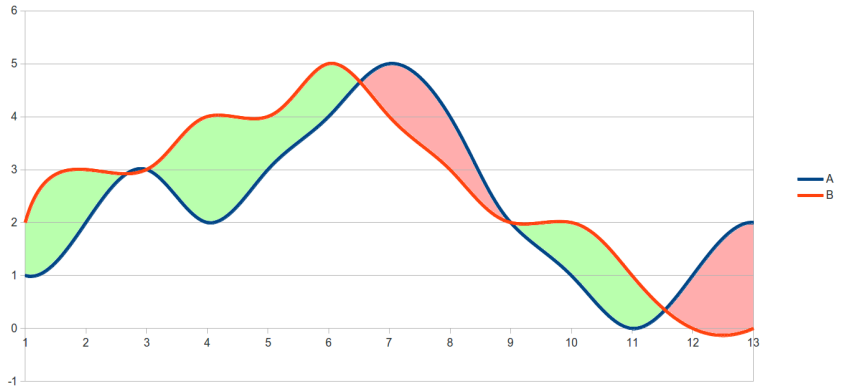


FIGURE 1.3 – Deux courbes présentant deux séries de données homogènes. L'espace compris entre les deux courbes est coloré en fonction de la courbe ayant la plus haute ordonnée (valeur sur le 2^e axe) à cette abscisse (valeur sur le 1^{er} axe).

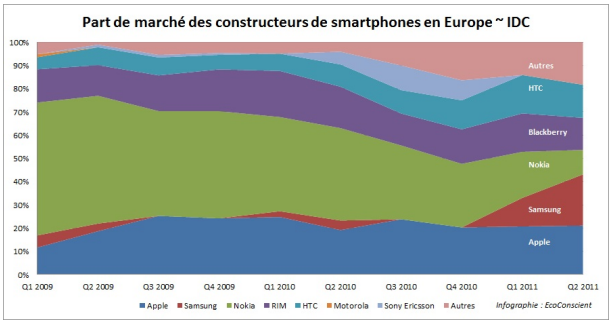


FIGURE 1.4 – La répartition des parts de marché des ventes de téléphone par constructeur entre 2009 et 2011

1.2.5 Le dessin de graphe

Le travail de cette thèse s'est quant à lui concentré sur une autre métaphore visuelle, relativement proche des nuages de points : la représentation de graphe à l'aide de diagramme nœud-lien.

Les graphes permettent de représenter des données relationnelles. Ces dernières correspondent à un ensemble d'éléments liés entre eux par une ou plusieurs relations. Les relations peuvent être porteuses d'informations supplémentaires, de plus, elles peuvent être orientées. Une définition plus formelle des graphes et de leurs représentations sera fournie dans le chapitre 2.

La métaphore visuelle la plus courante pour représenter un graphe est le diagramme nœud-lien. Il consiste à représenter sur un plan chaque entité par un rond (ou une autre forme), qui sera relié aux entités avec lesquelles elle est en relation par un arc (ou une flèche lorsque la relation est orientée).

Les tâches qui peuvent être réalisées avec ce type de représentation sont très nombreuses et ne seront pas toutes listées ici. Un travail de taxinomie de ces différentes tâches a été réalisé par Lee *et al.* en 2006 [56].Quelques unes seront aussi présentées au cours de ce manuscrit afin de décrire les différentes problématiques à envisager lors d'un travail dans ce domaine. En effet comme nous le verrons, l'évaluation de la qualité d'un dessin dépend beaucoup de la tâche réalisée par l'utilisateur.

La figure 1.5 présente un graphe des matchs de football américain entre les différentes universités américaines. Il est intéressant de voir qu'une grande partie des matchs sont réalisés au sein de fédérations locales (qui correspondent aux groupes de sommets fortement interconnectés). De plus, certaines universités (représentées sur la droite) ne font aucun match.

Un grand nombre de caractéristiques peuvent être représentées graphiquement. Il est possible de jouer sur la taille, la forme et la couleur des sommets. Il est possible d'ajouter des étiquettes ou de réaliser des mises en valeur de certains sommets. Il est aussi possible d'utiliser le type d'arête (pointillé, trait simple, double trait), son épaisseur, sa couleur, et le type de flèche pour ajouter de l'information à un graphe.

Le problème de la position des sommets est plus complexe. Il est possible d'utiliser deux des caractéristiques de chaque entité pour définir une position (lors d'un dessin en deux dimensions), toutefois le résultat est généralement assez peu lisible. De nombreux algorithmes permettent justement de calculer une position intéressante pour chaque sommet. Une bibliographie abondante est disponible sur le sujet, les ouvrages de Battista *et al.* [6] et de Kaufmann et Wagner [53] offrent une bonne présentation du domaine. C'est aussi sur cette problématique que se base le travail de cette thèse.

1.3 Le dessin de graphe et l'utilisateur

1.3.1 Une communauté de plus en plus ambitieuse

La représentation d'un graphe pose de nombreux problèmes. Comme nous venons de le dire, le positionnement des sommets représente l'un des plus importants. Le concept de graphe est bien antérieur à l'informatique, de ce fait, les premiers dessins étaient réalisés manuellement. Un des premiers problèmes modélisés à l'aide d'un graphe est celui des « Sept ponts de Königsberg » dont Euler a proposé une solution en 1759 dans son ouvrage « Solutio problematis ad geometriam situs pertinentis »[31].

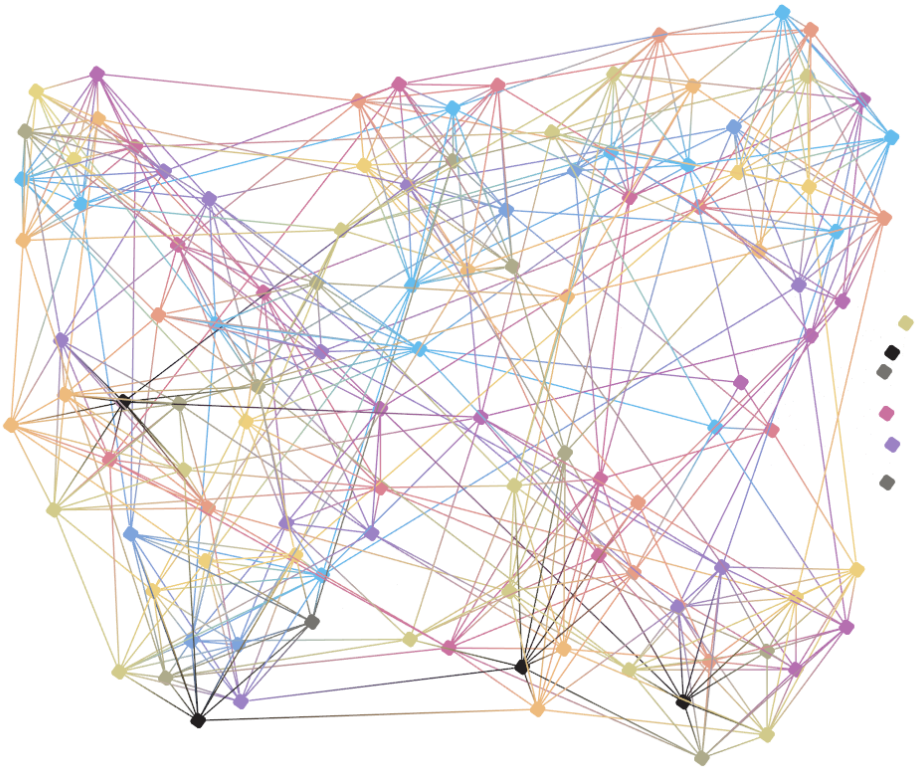


FIGURE 1.5 – Les rencontres de football américain entre les différentes universités américaines

La première approche algorithmique du problème de positionnement des sommets a été proposée par Tutte [89]. Ces travaux en ont inspiré de nombreux autres par la suite. La première problématique importante qui a été rencontrée est venue de la grande variété des graphes. En effet, certains graphes peuvent être facilement dessinés à l'aide d'algorithmes. Les arbres par exemple peuvent être représentés aisément en deux dimensions. La figure 1.6 présente par exemple trois dispositions des sommets pour le même arbre. D'autres classes de graphes peuvent être représentées selon un processus calculatoire simple, comme les grilles ou les cycles.

De nombreux algorithmes ont été donc mis au point pour différentes classes présentant des particularités permettant de les dessiner facilement. Cependant, un grand nombre de graphes n'appartiennent à aucune de ces classes. Le dessin de ces graphes soulève de nombreuses questions, qui comme nous le verrons par la suite, admettent pour certaines de toutes aussi nombreuses réponses. La première question concerne l'objectif d'un dessin : l'utilisateur veut-il explorer le graphe, identifier des motifs récurrents ou essayer de saisir une dynamique globale qui ressort de l'ensemble des données ? Chacun de ces objectifs implique une définition propre de ce qui correspond à un bon dessin pour le graphe étudié [78, 79]. De plus, de nombreuses considérations esthétiques doivent aussi être prises en compte, car souvent, pour qu'un dessin de graphe soit réellement « bon », il faut aussi qu'il soit « beau » [94, 77].

Toutefois, une contrainte doit être respectée quelque soit l'objectif : le dessin du graphe se doit d'être lisible, c'est à dire qu'il doit être possible d'extraire de l'information de l'image. Il est possible d'utiliser le graphe pour tenter de répondre à des questions sur les données (cf. la taxonomie des tâches qu'il est possible de faire avec un graphe [56]). La prise en compte de cette contrainte permet de définir certaines caractéristiques graphiques importantes. Les données relationnelles sont composées d'un ensemble d'entités et de relations entre ces entités. Il est donc

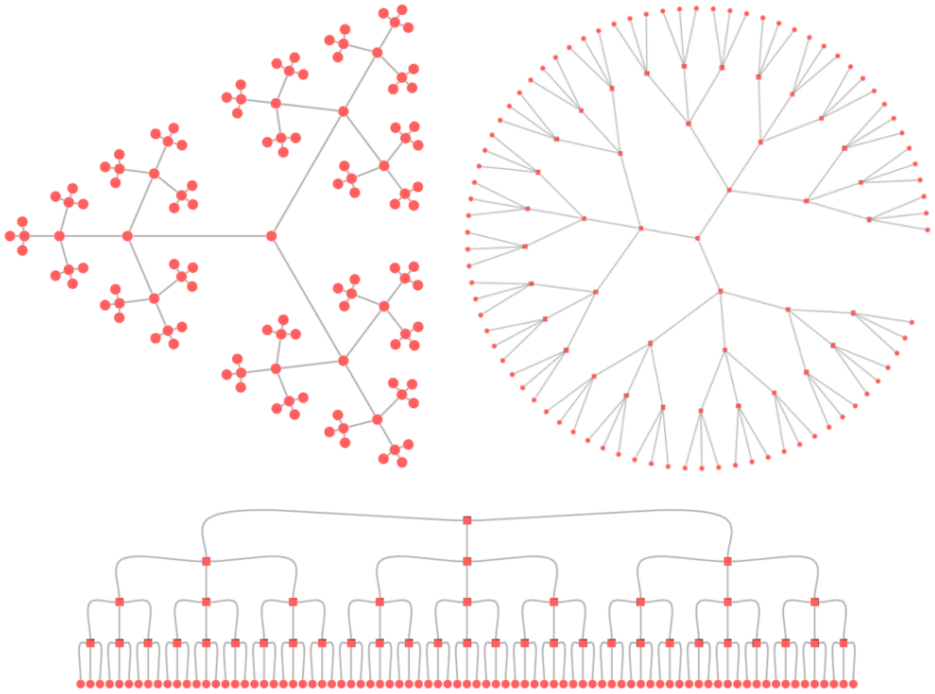


FIGURE 1.6 – 3 dessins d'un même arbre réalisés avec une méthode calculatoire.

important que les entités puissent être distinguées. De ce fait, il est préférable de ne pas superposer des sommets. Au niveau des relations, il faut pouvoir déterminer facilement le voisinage des sommets. Le voisinage d'un sommet correspondant à l'ensemble des sommets auxquels il est directement relié. Pour cela, il faut pouvoir suivre toutes les arêtes partant de ce sommet. Cela se traduit de différentes manières. Tout d'abord, il faut si possible placer tous les sommets voisins relativement proches les uns des autres. Bien évidemment, cela n'est pas toujours possible. Ensuite, les croisements d'arêtes nuisent à leur bonne distinction, particulièrement les arêtes se croisant avec un angle plat. De la même manière, les arêtes traversant un sommet doivent être évitées, car elles peuvent être prises pour deux arêtes partant du sommet concerné. Enfin, les arêtes en ligne droite sont généralement les plus lisibles, cependant elles rendent souvent très complexe le respect des contraintes précédentes. Enfin, on pourra citer le cas de certains dessins de graphes dans lesquels toutes les arêtes sont soit verticales, soit horizontales.

De nombreux autres critères permettant d'estimer la qualité d'un dessin ont été mis au point. Un grand nombre seront présentés par la suite dans la partie concernant l'évaluation de la qualité d'un dessin (2.4.2). De nombreux algorithmes se sont basés sur ces différentes caractéristiques pour définir une fonction d'énergie. L'algorithme reposant ensuite sur un système permettant de trouver un dessin qui minimise (ou maximise selon la définition de l'algorithme) cette fonction. Une branche importante de ces algorithmes repose sur un système par modèle de force pour rechercher une disposition optimale. Ces systèmes correspondent à des modèles physiques dans lesquels chaque sommet représente une masse et chaque arête un ressort. Différentes forces sont ensuite ajoutées au modèle (comme une force de répulsion entre chaque sommet pour éviter les superpositions) puis le système est simulé durant un certain temps (si possible jusqu'à sa stabilisation). Les positions finales des masses fournissent alors la disposition des sommets du graphe. Un grand nombre d'algorithmes utilisant ce principe ont été mis au point, chacun définissant sa propre fonction d'énergie, la manière dont les sommets sont ajoutés au modèle, les différentes forces et leur manière d'agir.

1.3.2 L'utilisateur novice et l'InfoViz

Comme nous venons de le voir, il existe un très grand nombre d'outils permettant de dessiner des graphes. De plus, la plupart de ces méthodes permettent en général d'être paramétrées afin de pouvoir s'adapter plus finement aux données ou au type de résultat attendu. Cependant, cette richesse au niveau du choix de la méthode de dessin comme au niveau des réglages qu'il est possible de faire rend leur usage délicat pour un public non-expert. Or le domaine de la visualisation de données étant encore relativement jeune, il existe très peu d'experts en dehors de ceux travaillant justement à l'élaboration de nouvelles méthodes de dessins.

La première solution à ce problème consiste à mettre au point des partenariats entre différents domaines de recherche. Cela se produit déjà très régulièrement dans de nombreux domaines. Toutefois, pour permettre une diffusion réelle de représentations graphiques pour de grandes masses de données, il est nécessaire que des outils simples d'accès soient disponibles auprès du grand public. Ce mouvement est en marche. En effet, de plus en plus d'outils de visualisation interactive sont mis au point et diffusés auprès du grand public. Cette diffusion est devenue nécessaire suite à l'ouverture au grand public de nombreuses données publiques. Pour n'en citer qu'un, le site Google Public Data Explorer permet de consulter des données fournies par la Banque mondiale ou l'O.N.U. à l'aide de quatre types de représentations graphiques.

Toutefois, les travaux de Grammel *et al.* [42, 43, 44] ont montré que les utilisateurs novices avaient du mal à construire des représentations graphiques de leurs données. Même en réduisant au maximum toute difficulté liée à l'utilisation de l'outil informatique, les utilisateurs n'arrivent pas à choisir efficacement quelles données doivent être représentées, et encore moins lorsqu'ils utilisent des graphiques auxquels ils ne sont pas habitués. Cela est relativement logique puisque l'utilisateur novice ne sait en général pas à priori à quoi va ressembler le résultat qu'il souhaite obtenir.

Il est donc nécessaire de continuer à rendre les outils existants plus accessibles aux utilisateurs novices. En effet, avec l'évolution de la société vers un monde où les données occupent une place de plus en plus importante, tant par leur volume que par leur omniprésence, il est vital qu'un maximum de personnes puissent facilement interagir avec ces données. Sans cela, une fracture risque d'apparaître entre les personnes capables d'explorer et d'exploiter les « banques de données » et les autres. Il est donc nécessaire de développer des outils permettant cette interaction entre des utilisateurs novices et de grandes quantités de données.

1.3.3 Suprématie des algorithmes les plus connus

Le même problème se pose dans le domaine du dessin de graphes lors du choix de la méthode de placement des sommets. La solution la plus souvent envisagée consiste à utiliser les algorithmes les plus célèbres. En effet, ces algorithmes sont généralement disponibles dans la plupart des logiciels de dessins de graphes, de plus, leurs noms étant relativement connus, les utilisateurs auront plus facilement tendance à les sélectionner au sein d'une grande liste. Au niveau du paramétrage, les différents paramètres étant généralement difficiles à régler, les valeurs par défaut sont généralement utilisées. Cela conduit à une homogénéisation des dessins obtenus. De plus, cette situation provoque une stagnation de la qualité des dessins de graphes utilisés dans des cas réels, alors qu'il existe de nouvelles méthodes de dessins plus efficaces. Ce problème a été mis en avant par Ulrich Brandes dans un keynote de 2011 [15].

Du point de vue du public, cette situation nuit à l'adoption des systèmes de visualisation de graphe. En effet, si un dessin n'est pas adapté à la tâche qui doit être réalisée, s'il ne permet pas de répondre aux questions de l'utilisateur, alors ce dessin est inutile. Or, si tous les dessins qui peuvent être facilement obtenus se ressemblent et ne permettent pas de s'adapter aux besoins de l'utilisateur, ce dernier risque de ne pas être satisfait. Il est donc nécessaire de travailler à la mise au point de systèmes permettant de mieux prendre en compte ces besoins spécifiques et de guider l'utilisateur vers l'obtention d'une « bonne » représentation de ses données.

1.3.4 Des systèmes de dessins interactifs

Une solution envisageable consiste à proposer des dessins à l'utilisateur et de lui demander dans quelle mesure ces représentations lui permettent d'exploiter ses données. Cela revient en général à déterminer si une image lui permet de répondre aux questions qu'il se pose. Cette approche présente plusieurs intérêts.

En effet, plusieurs études comme les travaux de Nascimento et Eades [72] ou ceux de Woolley et Stanley [98], plus orientées vers des créations esthétiques et fonctionnelles, ont montré que l'être humain est relativement doué pour déterminer des états intermédiaires intéressants et qui permettent d'atteindre un résultat satisfaisant rapidement. Le système PicBreeder par exemple [84] met en place un système collaboratif de création d'images. Le système génère un ensemble d'image et demande ensuite à l'utilisateur de lui désigner celles qui lui semblent les plus intéressantes. Un nouvel ensemble de candidats est ensuite généré à partir du choix de l'utilisateur.

Ces systèmes s'appuient en général sur un générateur de candidats, des images par exemple dans le cas de Pic Breeder, pour ensuite demander à l'utilisateur de les évaluer (un par un ou en réalisant une sélection parmi un ensemble de candidats). Ensuite, un nouveau jeu de candidats est généré à partir du choix de l'utilisateur, et le processus se réitère jusqu'à ce que l'utilisateur soit satisfait par le résultat qu'il obtient.

Dans le cadre du dessin de graphe, le travail de Biedl *et al.* [11] en 1998 se centre justement sur cette problématique. La question qu'il se pose revient à demander : "Comment dessiner quelque chose de joli sans avoir à définir « joli » ?" La solution proposée consiste à proposer à l'utilisateur de nombreux dessins du graphe pour qu'il puisse choisir celui qui lui convient le mieux. Un tel système dépend énormément du générateur de dessins associé. Le projet de Biedl *et al.* se base pour cela sur le système GLIDES qui permet de définir un certain nombre de contraintes puis qui tente de réaliser un dessin vérifiant au mieux ces dernières. SMILE, le système mis au point par Biedl, génère des ensembles de contraintes qu'il soumet ensuite à GLIDES pour réaliser les dessins. Ces derniers sont ensuite présentés à l'utilisateur qui choisit ceux qu'il préfère.

De telles méthodes sont toutefois soumises à une contrainte majeure : la fatigue de l'utilisateur. Ce terme désigne le fait que l'utilisateur ne peut réaliser qu'un nombre restreint d'évaluations à la suite, cette quantité diminuant lorsque la complexité de l'évaluation augmente. Cela implique de réduire au maximum les évaluations réalisées par l'utilisateur. Ce problème tient une place importante dans le survey réalisé par Takagi des système évolutif interactifs [88].

Une deuxième contrainte des systèmes interactifs concerne les délais de réponse du système. Deux cas de figure d'évaluation sont envisageables, dans le premier cas, le système est capable de générer un nouveau jeu de candidats très rapidement et le soumet immédiatement à l'utilisateur. Dans ce cas, l'utilisateur reste de manière continue devant l'application et réalise successivement

un grand nombre d'évaluations. Ce cas d'utilisation n'est plus envisageable si le système met un temps non négligeable à générer un ensemble de candidats intéressants, de l'ordre de la dizaine de minutes par exemple. Dans ce cas, l'utilisateur ne peut faire qu'une évaluation toutes les dix minutes et perd une grande quantité de temps à attendre les nouveaux candidats. Dans ce cas, il est préférable de prendre plus de temps pour essayer de générer un ensemble de candidats plus grand et plus diversifié afin de permettre à l'utilisateur d'alterner entre l'activité d'évaluation et une autre activité.

1.4 Solution proposée

Cette thèse a eu pour principal objectif de mettre au point un système interactif de dessin de graphes. Plusieurs contributions importantes peuvent être identifiées. Cette partie en fait une rapide présentation à travers une description du projet qui a été mis en œuvre.

1.4.1 Un graphe -> plusieurs dessins

Première contribution : Modification d'un algorithme par modèle de force

Comme nous venons de le voir, le premier objectif d'un système interactif de dessin de graphes consiste à pouvoir générer plusieurs dessins pour un seul graphe. La solution envisagée dans le cadre de cette thèse utilise une version modifiée d'un algorithme par modèle de force, GEM [32], transformé de façon à permettre la définition des paramètres pour chacun des sommets. Cette version modifiée, nommée GaGEM, permet d'obtenir des dessins très différents pour un même graphe Cependant, elle nécessite la définition d'un très grand nombre de paramètres dont l'impact peut être difficilement mesurable. Ce choix, et quelques alternatives que nous avons envisagées sont plus amplement documentés dans le chapitre suivant, qui est consacré au dessin de graphes.

Deuxième contribution : Mesure de similarité

Le second problème qui se pose dans le cadre des systèmes interactifs concerne la similarité des différents candidats. L'idée consiste à ne pas soumettre deux candidats identiques ou trop similaires à l'utilisateur. Pour cela, il est nécessaire de pouvoir mesurer la similarité de deux dessins. De plus dans un second temps, cela permet aussi de ne pas trop s'éloigner des candidats appréciés par l'utilisateur.

Le choix d'utiliser GEM pour générer les différents dessins candidats impose aussi une contrainte à l'évaluation de la similarité. En effet, GEM est un algorithme non déterministe. De ce fait, une petite modification d'un paramètre peut produire un résultat similaire à une homothétie du plan près. Pour rappel les homothéties du plan sont les translations, les rotations, les symétries, les effets d'échelles et les compositions de ces transformations. Or toutes ces transformations n'ont que peu d'impact sur la lecture du graphe par l'utilisateur car elles correspondent aux outils de manipulation de base des dessins de graphes (qui sont donc disponibles dans toutes les applications de visualisation de graphe). Il est donc préférable de minimiser l'impact de ces transformations sur la mesure de similarité, puisqu'elles ne changent que très peu la perception du graphe par l'utilisateur.

Plusieurs techniques venant du domaine médical ou des statistiques permettent de comparer deux ensembles de points. L'analyse procrustéenne permet par exemple de rechercher l'homothétie (translation, mise à l'échelle, rotation et si besoin symétrie) à appliquer à un ensemble de points pour qu'il se superpose au mieux à un deuxième ensemble de points de même taille. Cette technique est décrite dans les travaux de Goodall publié en 1991 [41]. Cependant, cette technique est plus adaptée à des ensembles de points appartenant à une surface (et permet alors de comparer deux surfaces), ce qui ne correspond pas au contexte des graphes.

Une autre méthode, utilisée par SMILE, le système de Biedl *et al.*, consiste à trouver le couplage entre les deux ensembles de points (chaque couple contenant un point de chaque ensemble) qui permet de minimiser la somme des distances entre les deux points de chaque couple. La similarité correspond alors à la moyenne de toutes ces distances. Cette solution ne nous convient pas non plus, car nous ne souhaitons pas considérer le problème du couplage dans notre évaluation de la similarité (pour nous les sommets sont tous identifiés précisément), et cette technique ne s'adapte pas bien aux homothéties du plan qui peuvent être engendrées par GaGEM (elle est par exemple extrêmement sensible aux translations).

Une nouvelle méthode, basée sur les triangles semblables, a donc été mise au point. Son principe repose sur le fait que si tous les triangles qu'il est possible de construire avec les points de l'ensemble (même s'ils ne sont pas reliés par des arêtes), sont semblables sur les deux dessins, alors les dessins sont globalement semblables.

Cette technique permet d'obtenir une mesure de similarité indépendante de toute homothétie du plan. De plus, sa complexité en $O(n^2)$ est plus intéressante que celles des méthodes issues de l'analyse procrustéenne qui sont généralement en $O(n^3)$. Enfin, elle permet de fournir une information pour chaque sommet et peut aussi être paramétrée afin de mesurer une similarité plus locale ou plus globale. Cette mesure de similarité est plus formellement présentée dans la partie 2.5.

1.4.2 Explorer efficacement un espace vaste, les algorithme génétique

L'espace des dessins de graphes est immensément grand. Le fait d'utiliser GaGEM comme intermédiaire permet de changer l'espace exploré. Cette transformation n'a pas d'impact important sur la taille de l'espace (qui reste très grand), mais permet une exploration plus facile (ce point sera amplement détaillé par la suite). Toutefois, ces espaces sont de toute façon bien trop grands pour être explorés manuellement. De plus, cela serait bien trop complexe pour un utilisateur novice, qui reste toujours la cible principale de notre système.

De nombreuses méta-heuristiques permettent d'explorer un grand espace. Nous avons choisi d'utiliser les algorithmes génétiques, car ces derniers ont déjà été souvent utilisés dans le cadre du dessin de graphes et qu'ils ont des propriétés très intéressantes pour une utilisation interactive. Un des objectifs annexes de cette thèse a donc été la mise au point d'une infrastructure pour exécuter de manière très flexible un algorithme génétique. Cette dernière contient plusieurs améliorations que nous avons apportées aux algorithmes génétiques.

Troisième contribution : Favoriser la réutilisation des résultats intermédiaires

Une des contraintes liées à l'aspect interactif du système est qu'il doit pouvoir répondre rapidement à une requête de l'utilisateur. Nous avons donc essayé de mettre en œuvre un système

permettant de réutiliser le plus souvent possible les résultats intermédiaires, qui correspondent aux génomes dans le cadre des algorithmes génétiques. Pour cela, lors de chaque évaluation d'un génome, des données caractérisant le contexte d'évaluation ainsi que le comportement de l'individu auquel ce génome a été attribué sont associés à chaque génome. Une définition complète des génomes et de leur rôle sera fournie dans le chapitre dédié à l'utilisation des algorithmes génétiques dans le cadre du dessin de graphe. De manière simplifiée, chaque génome contient les informations nécessaires à GaGEM pour pouvoir produire un dessin à partir d'un graphe.

Un des objectifs a donc été de déterminer quel sont les génomes « intéressants » pour ensuite pouvoir les stocker dans des populations annexes (conservation courte durée) ou dans une base de données (conservation longue durée) pour pouvoir être réutilisés par la suite.

Notre contribution consiste dans la mise en place des informations annexes, ainsi que des modalités de stockage et de réutilisation au sein de l'algorithme génétique. Cette exploitation se fait bien évidemment dans le cadre du dessin de graphes, cependant, l'infrastructure permet d'appliquer ces principes à n'importe quel cas d'utilisation.

Quatrième contribution : Systèmes d'auto-évaluation

Cette contribution est moins importante que les autres et correspond à un regroupement d'utilisations avancées des données stockées. Elles permettent d'obtenir de nombreuses informations sur la manière dont est faite l'exploration de l'espace. Le premier système à avoir été mis en place est une exploration guidée par la curiosité. Ce type d'exploration («Open-endness exploration), récemment mis en lumière par les travaux de Lehman et Stanley [57, 60, 59, 58, 98] permet d'obtenir des résultats très intéressants rapidement. Le principe consiste à évaluer les nouveaux candidats en fonction de la nouveauté de leur comportement, ce qui stimule de manière continue le système à explorer de nouvelles parties de l'espace.

Enfin, un dernier système, qui a surtout été utilisé lors de la phase de test de l'infrastructure et lors de l'initialisation de la base de données, permet de tenter de recopier un dessin fourni par l'utilisateur à l'aide de notre version modifiée de GEM. Ce système a aussi permis de valider l'utilisation de cette méthode de dessin en montrant qu'il est possible de générer plusieurs dessins très différents pour un même graphe.

1.4.3 Mise en place d'une infrastructure d'utilisation

Cinquième contribution : Mise en place d'un système interactif

Comme nous l'avons vu, l'objectif final de cette thèse consiste à mettre en place un système d'assistance à un utilisateur novice pour le dessin de graphes. Pour répondre à cette problématique, notre système permet à un utilisateur expert de définir différentes tâches d'exploration de l'espace des dessins de graphes. Ces dernières peuvent ensuite être utilisées par des utilisateurs novices pour réaliser des explorations de l'espace des dessins d'un graphe.

Le chapitre 4 se concentre justement sur l'utilisation de notre système dans le cadre du dessin de graphe. Quatre exemples de tâches sont fournis à la fin de ce chapitre. Deux d'entre elles peuvent être utilisées dans le cadre d'une interaction entre le système et un utilisateur novice, et les deux autres sont des tâches « internes », qui correspondent à une utilisation plus classique d'un algorithme génétique (sans composante interactive).

Sixième contribution : Solution logicielle

Les algorithmes génétiques sont des méta-heuristiques relativement lourdes et nécessitant des calculs importants. De plus, l'algorithme génétique mis en œuvre dans notre système est très hautement configurable. Cette configuration est de ce fait relativement compliquée si elle n'est pas assistée. Pour répondre à ces problématiques, un logiciel a été conçu afin de permettre de gérer facilement de nombreuses exécutions de l'algorithme génétique à distance (sur un ensemble de machines dédiées) ainsi que la configuration de ces exécutions.

Le chapitre 5 présente les différentes composantes de ce système, ainsi que les principales méthodes d'interactions avec les utilisateurs.

Chapitre 2

La visualisation de graphe

2.1 Les graphes

Les graphes permettent de représenter facilement une grande diversité de données. Comme cela a été dit au paragraphe 1.2.5, ils se basent sur des modèles de données relationnelles. Ce chapitre va être introduit par une définition formelle des graphes. Quelques problèmes généralement résolus à l'aide des graphes seront ensuite présentés afin de permettre au lecteur de se faire une idée de différents cadres d'utilisation des graphes. Ensuite, la problématique de leur représentation graphique sera plus amplement abordée, à l'aide d'une présentation des différentes méthodes de dessin de mesures de qualité d'un dessin. Enfin, la dernière partie du chapitre sera consacrée aux avancées réalisées durant cette thèse spécifiquement liées au dessin de graphe.

2.1.1 Modèle mathématique

Un graphe, dans sa version la plus simple, correspond à un couple (V,E) où V correspond à l'ensemble des sommets, et $E \subset V \times V$, l'ensemble des arêtes. Chaque arête relie deux sommets, et est donc caractérisé par une paire $\{a,b\}$ de deux sommets, $\{a,b\} \in \mathcal{P}_2(V)$. Cela correspond alors au cas des graphes simples non orientés.

Une deuxième variante du modèle consiste à considérer une orientation des arêtes, qui sont alors nommées « arcs ». L'ensemble des arcs est donc constitué de couple (a,b) avec $(a,b) \in V^2$. On parle alors de graphes simples orientés.

Un graphe est dit simple lorsqu'il existe au plus une arête entre deux sommets (au plus deux arcs, un dans chaque sens dans le cas orienté). Dans le cas contraire la liste des arêtes devient un multi-ensemble de paires de sommets (de couples dans le cas orienté).

Une autre caractéristique des graphes correspond à la possibilité d'avoir des boucles, qui correspondent à des arêtes dont les deux extrémités sont incidentes au même sommet.

Enfin, un graphe est dit connexe lorsqu'il existe un chemin (une liste d'arêtes deux à deux adjacentes) entre chaque paire de sommets du graphe. De manière plus simple, cela implique que le graphe ne forme qu'un seul bloc.

De plus, chaque sommet et chaque arête peuvent être étiquetés par des données supplémentaires. Deux types de données courants sont par exemple : un nom associé à chacun des sommets ou une valeur numérique associée à chacune des arêtes.

Les graphes considérés dans le cadre de cette thèse sont des graphes simples non orientés connexes. De plus, les données associées ne sont pas utilisées dans les différentes méthodes mises en place afin de pouvoir traiter un maximum de graphe. Le fait de ne considérer que les graphes connexes ne réduit pas la portée de la méthode, car il est toujours possible de dessiner indépendamment chaque partie connexe, puis d'assembler ensuite les différents dessins obtenus.

De très nombreuses propriétés peuvent être définies ou calculées pour les différents sommets et arêtes. Quelques-unes seront décrites dans la partie 2.4. La plus importante et la plus simple est le degré, qui correspond au nombre d'arêtes incidentes à un sommet. Dans le cas d'un graphe simple, cela correspond aussi au nombre de sommets voisins (relié directement par une arête). Dans le cas des graphes orientés, le degré entrant (resp. sortant) correspond au nombre d'arêtes entrantes (resp. sortantes) du sommet. Les ensembles de sommets induits sont alors nommés les prédécesseurs (resp. les successeurs).

2.1.2 Nombreux problèmes abstraits et réels

Les graphes sont utilisés pour résoudre de nombreux problèmes. Certains sont relativement abstraits et directement issus des recherches réalisées sur la théorie des graphes. D'autres sont au contraire très concrets et permettent de répondre à des problématiques directement issues du monde réel.

Nous nous intéresserons ici uniquement aux problèmes pour lesquels le positionnement des sommets est important. En effet, c'est sur ce point que ce concentre notre système. De ce fait, tous les problèmes liés à d'autres aspects de la représentation des graphes, comme les problèmes de colorations ne seront pas considérés.

Recherche du plus court chemin

La recherche du plus court chemin entre deux sommets est une tâche très courante. Un exemple réel consiste à chercher un trajet en transport en commun entre deux points d'une ville. Une carte de transport en commun étant généralement modélisée sous forme d'un graphe dans lequel chaque station est un sommet et lorsque deux stations sont reliées par au moins une ligne, elles sont liées par une arête. La solution généralement utilisée consiste à explorer le graphe à partir d'un des sommets jusqu'à la rencontre du deuxième sommet. Cette tâche et ses dérivées occupent une place importante dans les taxonomies des tâches sur le graphes [56]. De plus, plusieurs études évaluant l'efficacité des diagrammes noeud-lien font appel à cette dernière [37, 77, 75, 50]

Détection de communautés

La détection de communautés est un problème plus récent, qui est apparu avec l'étude de grands graphes représentant des données d'interactions humaines. Ils consistent à déterminer des groupes de sommets représentant une communauté. L'objectif est alors de déterminer une partition du graphe respectant ces différentes communautés. Il est ensuite possible d'identifier certains sommets ayant un rôle particulier au sein de ces communautés.

Une problématique sous-jacente à cette dernière consiste à représenter correctement ces graphes « sociaux » afin que le lecteur puisse rapidement identifier les communautés et les sommets qui les composent.

Les travaux de Newmann et Girvan [73, 38] portent en grande partie la détection et l'évaluation de communautés. D'un autre côté, Huang et Eades ont publiés un papier en 2000 portant sur la représentation de graphes en cluster [27].

Reconnaissance de motifs

La dernière problématique présentée ici consiste en une approche relativement différente de l'utilisation des graphes. Il s'agit de la reconnaissance de motifs dans des graphes dessinés en respectant certaines contraintes. Cette tâche est très courante dans le domaine de la bioinformatique. Les graphes y sont en autre utilisés pour représenter les voies métaboliques. Ces dernières représentent les interactions chimiques entre les différentes protéines qui se déroulent au sein de cellules vivantes.

Cette utilisation ne trouve pas une réponse strictement calculatoire, mais permet à l'aide d'une représentation correcte à un être humain de gagner énormément en efficacité. La partie informatique consiste alors surtout à représenter les graphes d'une manière qui respecte les besoins des biologistes, ainsi qu'à mettre en valeur certaines configurations spécifiques. Plusieurs méthodes de dessins de graphes se concentrent spécifiquement sur ce type de graphes et tenter de dessiner ces graphes en respectant les contraintes des biologistes [14, 7]. D'autres applications de dessins de graphes utilisent aussi la détection de motifs dans des domaines plus proches de la vie courante, comme la détection de fraudes bancaires ou téléphoniques [20].

2.1.3 Classification des graphes

De nombreuses catégories peuvent être définies au sein des graphes. Certaines sont directement issues d'une problématique et des données qu'ils représentent, comme les graphes de voie métaboliques cités précédemment. D'autres sont liées à des propriétés mathématiques vérifiées par le graphe. Quelques-unes de ces catégories vont être présentées dans les paragraphes suivants.

Les arbres

Les arbres sont des graphes connexes et acycliques. Un graphe est connexe lorsqu'il existe au moins un chemin entre chaque paire de sommets du graphe. Un graphe est acyclique lorsqu'il n'existe aucun cycle dans le graphe, c'est à dire qu'il n'existe aucun chemin partant d'un sommet et revenant sur ce même sommet en utilisant au maximum une fois chaque arête.

Les arbres présentent de nombreuses propriétés très intéressantes, en termes de structure comme de représentation. Par exemple, le fait qu'il n'existe qu'un unique chemin entre chaque paire de sommets (corollaire direct de l'absence de cycle) permet de beaucoup simplifier la recherche du plus court chemin. De plus, la présence d'une racine (qui peut au besoin être choisie arbitrairement) permet de nombreux étiquetages des sommets permettant des parcours et des recherches très rapides.

Comme nous le verrons par la suite, de nombreuses méthodes permettent de dessiner facilement des arbres de manière lisible.

Les graphes acycliques orientés (ou DAG tiré de l'anglais) sont relativement proche des arbres. Comme leur nom l'indique, ce sont des graphes orientés ne comportant aucun cycles. Il est donc possible d'en ordonner les sommets et de les représenter selon un axe (de gauche à droite ou

de haut en bas par exemple). De manière simplifiés, ces graphes ressemblent à des arbres dans lesquels les sommets peuvent avoir plusieurs parents.

Les cliques

Les cliques, ou graphes complets, sont des graphes simples pour lesquels toutes les arêtes possibles sont présentes, soit $E = V \times V$. Ces graphes sont notés K_n avec n le nombre de sommet du graphe. Ces graphes sont relativement rare dans la réalité, par contre, un problème souvent considéré est celui de la plus grande clique présente dans un graphe. C'est à dire, le plus grand ensemble de sommets du graphe qui soient tous reliés entre eux.

La notion de cluster correspond à une extension des cliques. Il s'agit d'un groupe de sommets fortement interconnectés entre eux. Nous verrons dans la partie 2.4 une méthode calculatoire permettant d'identifier les sommets pouvant appartenir à un tel cluster. Il est toutefois important de noter que la définition des clusters est bien moins stricte que celle des cliques. La recherche de communauté dans un graphe revient généralement à identifier les différents clusters.

Graphes bipartis

Les graphes bipartis sont des graphes pour lesquels il est possible de définir une partition en 2 parties de l'ensemble des sommets (soit $V = A \cup B$ et $A \cap B = \emptyset$) tel que toutes les arêtes du graphe relient toujours deux sommets de parties différentes. Il n'y a donc aucune arête au sein de chaque partie.

Les graphes bipartis complets (chaque sommet est relié à tous les sommets de l'autre partie) sont notés $K_{a,b}$ avec a et b la taille de chacune des partitions.

Graphes planaires

Les graphes planaires sont les graphes qui peuvent être dessinés sur un plan sans le moindre croisement d'arêtes. De nombreuses recherches ont été menées sur cette classe de graphe, dont en particulier un théorème permettant de déterminer de manière calculatoire si un graphe est planaire ou non.

De nombreux graphes issus de données réelles sont planaires. Par exemple, la représentation d'une carte par un graphe pour lequel deux pays sont reliés par une arête lorsqu'ils partagent une frontière aboutit à un graphe planaire. Une propriété intéressante des graphes planaires est qu'ils sont toujours 4-coloriables, c'est-à-dire qu'il possible de colorier tout graphe planaire avec uniquement 4 couleurs différentes [82].

Classification des graphes réels

Cependant, hormis pour les graphes planaires, assez peu de graphes issus de données réelles rentrent dans une des catégories définies précédemment. La plupart des graphes réels ne présentent aucune régularité particulière. Ces graphes sont de plus relativement difficiles à modéliser. Toutefois, deux « catégories » non exclusives ont été définies pour les graphes réels. Il s'agit des graphes sans échelle et des graphes petit-mondes.

La caractéristique « sans échelle » est liée à la distribution des degrés des différents sommets. Si cette distribution est proche d'une loi puissance, alors le graphe est dit sans échelle.

Les graphes petit-mondes sont définis par deux caractéristiques, tout d'abord, le diamètre du graphe (la distance entre les deux sommets les plus éloignés du graphe) doit être faible et le coefficient de clustering du graphe doit être élevé. Ce dernier coefficient sera précisément défini dans la partie 2.4, il correspond globalement à la tendance des sommets du graphe à former des clusters (cf. paragraphe 2.1.3). Les graphes représentant des interactions sociales sont souvent petit-mondes.

Ces deux classes de graphes réelles ont toutefois des définitions relativement vagues. Pour la première, l'expression « proche d'une loi de puissance » n'est pas précise, le terme proche n'étant pas clairement défini. De la même manière, en ce qui concerne les graphes petit monde, l'interprétation de ce qu'est un petit diamètre peut varier d'un auteur à l'autre.

2.1.4 Formalisation du problème de dessin

Ces différentes classifications représentent un aspect important de la théorie des graphes. Cependant, leur importance est moindre dans le cadre du problème de leur représentation. En effet, dans notre cas, le principal problème lié à la manipulation des graphes concerne la visualisation des données qu'ils contiennent. Comment réaliser une image permettant de prendre connaissance efficacement de l'information contenu dans un graphe ?

Plusieurs métaphores visuelles sont utilisables pour représenter un graphe. La plus courante, et celle sur laquelle se focalise le travail de cette thèse, est la représentation noeud-lien. Dans cette dernière, chaque sommet correspond à un noeud, représenté par un symbole (généralement un cercle ou un carré) et chaque arête est représentée par une ligne entre les deux noeuds qu'elle relie. Dans le cas d'un graphe orienté, les liens sont des flèches orientées dans le sens de l'arc.

Lors de la réalisation d'une vue noeud-lien, la première problématique est celle du placement des sommets. En effet, l'efficacité de la représentation est directement liée au placement des différents éléments.

Une disposition des sommets est nommée « layout ». Mathématiquement, elle est caractérisée par une fonction $l : V \rightarrow \mathbb{R}^2$ qui à chacun des sommets associe une position. On note que la définition proposée correspond à un dessin en deux dimensions, en trois dimension l'espace d'arrivée serait $\mathbb{R}^3$. De manière générale, l'espace d'arrivée correspond à l'espace des coordonnées du dessin. La question du bornage et de la discrétisation de l'espace peut aussi être considérée (certains algorithmes travaillent en plaçant les sommets sur les intersections d'une grille). Dans notre cas, l'espace de dessin est un plan simple en deux dimensions qui peut être considéré comme non borné (les coordonnées correspondent à un couple de nombres flottants).

Lors de l'affichage d'un graphe, c'est toujours une fenêtre ajustée au mieux au layout utilisé qui est affichée. De ce fait, les problèmes de centrage et d'échelle n'ont pas d'impact significatif sur l'efficacité d'une représentation.

La question de la disposition des arêtes peut aussi être considérée. La solution la plus simple consiste à tracer une ligne droite allant d'un sommet vers le deuxième. Cependant, d'autres solutions peuvent être envisagées, permettant dans les cas les plus complexes de faire suivre des chemins particuliers à chaque arête spécifiquement. Toutefois, les problèmes attenants aux arêtes ne seront pas plus abordés dans le cadre de cette thèse.

Comme nous le verrons par la suite, la création d'un layout pour un graphe peut être un problème extrêmement complexe. Tant que le graphe reste petit (moins d'une vingtaine de sommets), le layout peut être créé manuellement. Mais, dès que le graphe considéré devient plus grand, réaliser cette opération manuellement devient extrêmement long. Il est donc nécessaire d'automatiser ce processus d'une manière efficace. Nous verrons dans les parties suivantes que plusieurs solutions ont été envisagées pour permettre cette automatisation.

Au niveau du vocabulaire, il peut être intéressant de différencier les termes de « layout » et de « dessin » d'un graphe. Le layout correspond à un positionnement des sommets du graphe. Un dessin utilise un positionnement mais ajoute à cela de nombreux autres éléments, comme l'apparence des sommets, celle des arêtes, la trajectoire des arêtes, l'utilisation de couleurs. C'est le layout associé à toutes ces informations qui constitue le dessin.

Cependant, le travail réalisé dans le cadre de cette thèse se concentre uniquement sur la construction de layout, et pas du tout sur les autres caractéristiques graphiques d'un dessin de graphe. De ce fait, dans la suite de ce manuscrit, les termes de layouts et de dessins seront tous les deux utilisés pour désigner uniquement le layout. De la même manière, le fait de dessiner un sommet correspondra au fait de lui fournir une position dans un layout.

2.1.5 Caractères graphiques et cerveau humain

Le fait que notre système s'occupe uniquement de la position implique une hypothèse relativement forte, qui est que la position des sommets porte suffisamment d'information pour permettre ou non à l'utilisateur de répondre aux questions qu'il se pose.

Il existe en effet une multitude de manière d'ajouter de l'information à un dessin de graphe. Les plus simples concernent la représentation des sommets. Il est possible de modifier la forme utilisée, sa taille et sa couleur. De la même manière, pour les arêtes, le type de trait, son épaisseur et sa couleur peuvent être modifiés.

Laisser de côté tous ces caractères s'ils étaient porteur de la majorité de l'information conduirait à une voie sans issue. Cependant, ce domaine a été très étudié, et il est établit que la position des sommets est apporte la majeure partie de l'information portée par un dessin. En particulier, les travaux réalisé par Bertin [10, 9] établissent une hiérarchie des différentes marques visuelles par rapport à leur impact sur l'humain. Ces travaux montrent que la position a un des impacts les plus forts dans la grande majorité des contextes. D'un autre côté, les travaux réalisés par Purchase [94, 77] montrent que certaines mesure de qualité d'un dessin ont un impact cognitif important. Or ces métriques (comme les croisement d'arêtes ou l'uniformité de la longueur des arêtes) sont directement liés à la position des sommets.

Afin de donner une idée aux lecteurs de l'impact des différentes marque graphiques, la figure 2.1 présente 3 dessins comparant chacun deux marques graphiques. Les sommets sont groupés 2 à 2 selon une première marque, et l'autre regroupement possible est fait avec l'autre marque. L'objectif est de voir que sont les groupes qui apparaissent en premier en regardant l'image.

Dans le premier cas, les deux marques graphiques utilisées sont la position et une zone entourée de pointillé. Ces deux manières de faire sont presque équivalentes en terme d'impact, avec toutefois une légère domination par le positionnement dans certaines situations.

Le deuxième cas compare le positionnement et la couleur. Ici, la couleur a beaucoup moins d'impact que la position, et la séparation haut/bas est bien plus forte que la rouge/bleu.

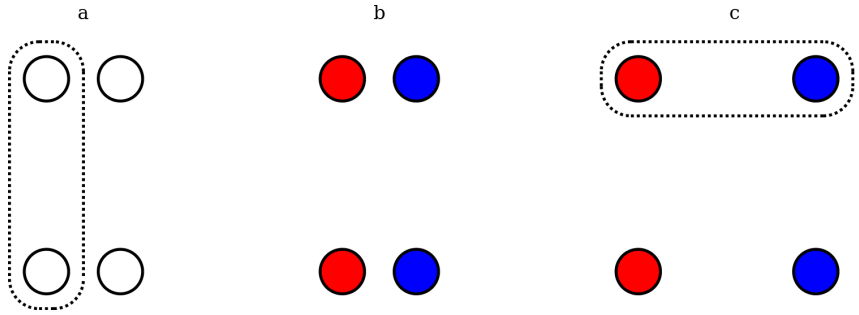


FIGURE 2.1 – Trois comparaisons de caractères graphiques utilisés pour réaliser des groupements d'éléments

La troisième comparaison concerne une zone pointillée et la couleur. Ici, c'est aussi la zone pointillé qui « bat » la couleur en termes d'efficacité de regroupement.

2.1.6 Dessins classiques pour certaines classes de graphes

Comme nous l'avons vu, le dessin d'un graphe est une tâche complexe pour de multiples raisons. Cependant, certains graphes sont relativement aisés à représenter. Il s'agit de ceux présentant des régularités particulières. Par exemple, les différentes classes vues dans la partie 2.1.3 admettent souvent une ou plusieurs méthodes de dessin efficaces.

Par exemple, pour les arbres, plusieurs méthodes permettent de les représenter facilement. En effet, le fait de pouvoir déterminer une racine de l'arbre, et de pouvoir dessiner l'ensemble des sommets en s'éloignant progressivement de la racine. Cela peut être fait de l'intérieur vers l'extérieur, ou d'un côté vers l'autre. Il est alors possible d'utiliser des méthodes calculatoires pour obtenir directement les positions des différents sommets. La figure 1.6 (p. 11) présente un arbre dessiné avec trois de ces méthodes.

De la même manière, les graphes très simples comme les cycles ou les chemins peuvent eux aussi être dessinés facilement, sous forme d'un « U » ou d'une ligne pour les chemins, et sous la forme d'un cercle pour les cycles. Plus généralement, les graphes représentant une figure géométrique régulière (comme une grille, ou une triangulation) peuvent être représentés selon cette même figure.

Le cas des graphes planaires est lui aussi très intéressant. En effet, la propriété principale de ces derniers est qu'ils peuvent être représentés sans croisement d'arête. Les méthodes de dessins se sont donc majoritairement concentrées sur ce point précis [85, 74]. Les principales différences viennent de la manière de répartir les différents sommets dans le plan. La figure 2.2 montre deux dessins d'un même graphe planaire.

Cas des graphes géo-localisés

Certains graphes réels présentent une spécificité leur permettant d'être facilement dessinés. Il s'agit des graphes géo-localisés. Dans ces derniers, les différents sommets représentent des éléments géographiques (des villes par exemple) et sont donc associés à une paire de coordonnées (la longitude et la latitude). Ces dernières peuvent être utilisées pour dessiner le graphe. Les dessins peuvent alors être réalisés sans le moindre calcul. Lorsque cela est possible, il est intéressant de fournir le fond de carte correspondant afin de fournir des repères de lecture.

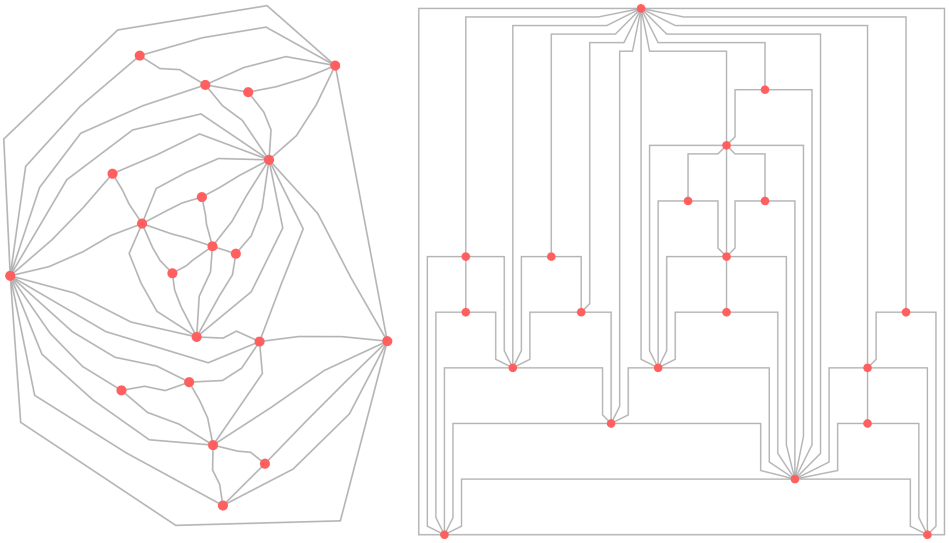


FIGURE 2.2 – Deux manières de dessiner un même graphe planaire

2.2 Algorithmes par modèle de force

Toutes les méthodes que nous avons vues précédemment sont relativement simples d'un point de vue calculatoire. En effet, pour dessiner un cycle en cercle par exemple, le centre et le rayon du cercle sont choisis arbitrairement, puis un premier sommet du cycle est choisi et placé au sommet du cercle. Cette position correspond à l'angle 0, puis les autres sommets sont placés successivement autour du cercle en respectant un écart constant. De même, pour les arbres, de nombreuses méthodes permettent de calculer directement la position de chaque sommet. Les travaux de Reingold et Tilford en 1981 [81], suivi de ceux de Walker en 1990 [92] et de Eades en 1991 [26] présentent chacun une manière de dessiner de tels arbres en allant de plus en plus rapidement et en proposant différentes occupations de l'espace.

Toutefois, de telles méthodes reposent sur les très fortes régularités des classes de graphes qu'elles permettent de dessiner. Or la plupart des graphes réels ne proposent aucune régularité de ce type. Il est donc nécessaire d'utiliser une autre approche du problème. Plusieurs solutions ont été explorées par la communauté. L'une d'entre elle, et qui occupe une place prépondérante, consiste à appliquer des séries de petits mouvements à chaque sommet jusqu'à l'obtention du dessin final. Les systèmes par modèle de force correspondent justement à ce type de solution.

Ces systèmes simulent un modèle physique dans lequel chaque sommet est représenté par une masse et chaque arête par un ressort. De plus, une force répulsive est ajoutée entre chaque paire de sommets. Le système est simulé par pas successifs. Durant chaque pas, toutes les forces qui s'appliquent sur un sommet sont ajoutées, puis ce dernier est déplacé proportionnellement à la force résultante. Ce processus est appliqué à chaque sommet à chaque pas de simulation. De nombreux algorithmes s'appuient sur ce type de processus pour construire progressivement le dessin d'un graphe. Ils varient dans leur manière de placer initialement les sommets, dans les forces qu'ils ajoutent au système et dans la manière de conduire et de terminer la simulation. Il est généralement possible de définir une fonction d'énergie sur le système (c'est à dire qu'à chaque état du système, il est possible de calculer le niveau d'énergie correspondant), l'objectif de l'algorithme étant alors de minimiser cette dernière.

2.2.1 Présentation du principe

Ce paragraphe présente de manière schématique le fonctionnement d'un algorithme par modèle de force. La suite du travail de cette thèse repose en grande partie sur ce mécanisme.

L'algorithme 1 présente la procédure du calcul de l'impulsion qui doit être appliquée à un sommet. Le principe général des algorithmes par modèles de forces consiste ensuite à appliquer ces impulsions à chaque sommet régulièrement afin d'améliorer progressivement l'état du système.

La figure 2.3 présente plusieurs étapes lors du déroulement d'un algorithme par modèle de force. L'algorithme en question est GEM, qui sera présenté plus longuement dans les paragraphes suivants.

2.2.2 État de l'art

Le premier algorithme posant les bases de ce principe a été proposé par Tutte en 1963 [89]. Ce dernier a proposé une méthode pour dessiner les graphes planaires. La première étape consiste à placer l'ensemble des sommets de la face extérieure selon un polygone. L'algorithme recherche ensuite une position des sommets intérieurs qui permet d'obtenir un équilibre stable de l'ensemble. Cette idée a été étendue ensuite par Eades en 1984 [28] et par Fruchterman et Reingold en 1991 [34]. Ils ont généralisé le principe proposé par Tutte pour pouvoir dessiner tout type de graphe. Ce sont ces deux travaux qui les premiers ont défini le principe des algorithmes par modèle de force tel que présenté dans les paragraphes précédents.

Une autre branche a été ouverte par Kamada et Kawai en 1989 [52]. Ils se basent sur un système similaire dans lequel les différents sommets sont reliés par des ressorts. Ensuite, ils définissent une fonction d'énergie caractérisant l'état du graphe. Fonction formée de deux composante, la première correspondant aux différents ressort qui souhaient avoir une longueur égale à leur longueur à vide, et une deuxième qui indique si la distance euclidienne (donc calculée à partir du layout) entre chaque paire de sommets du graphe est proportionnel à la distance topologique (le nombre d'arête du plus court chemin entre ces deux sommets). Une fois cette fonction d'énergie définie, ils utilisent une méthode calculatoire permettant de trouver un maximum local de cette fonction d'énergie. Cette méthode est aussi connue sous le nom de « Stress majorization ».

Les travaux qui ont suivi ont généralement repris un de ces principes de base en lui ajoutant des éléments permettant d'obtenir des résultats plus rapides, de meilleure qualité, ou sur des graphes plus grands. Par exemple, le travail de Frick *et al.* en 1995 [32] sur le moteur GEM. Ce dernier ajoute plusieurs éléments aux algorithmes par modèle de force. Tout d'abord, le placement des premiers sommets est réalisé progressivement en fonction du nombre de voisins de chaque sommet et des sommets déjà placés. De plus, il essaye de limiter au mieux les mouvements répétitifs comme les oscillations ou les rotations. Enfin, un paramètre de température est ajouté au graphe afin de déterminer la fin de l'algorithme. Le travail de cette thèse se base en grande partie sur cet algorithme, qui sera donc décrit plus précisément au paragraphe 2.3.4.

L'algorithme GRIP, mis au point par Gajer et Kobourov en 2002 [35], utilise plusieurs techniques de dessin inspirées des algorithmes précédents alternativement. De plus, les sommets sont ajoutés très progressivement tout en améliorant le dessin entre chaque ajout. Ces deux nouveautés permettent d'accélérer fortement le processus de dessin tout en obtenant un résultat de bonne qualité sur de plus grands graphes.

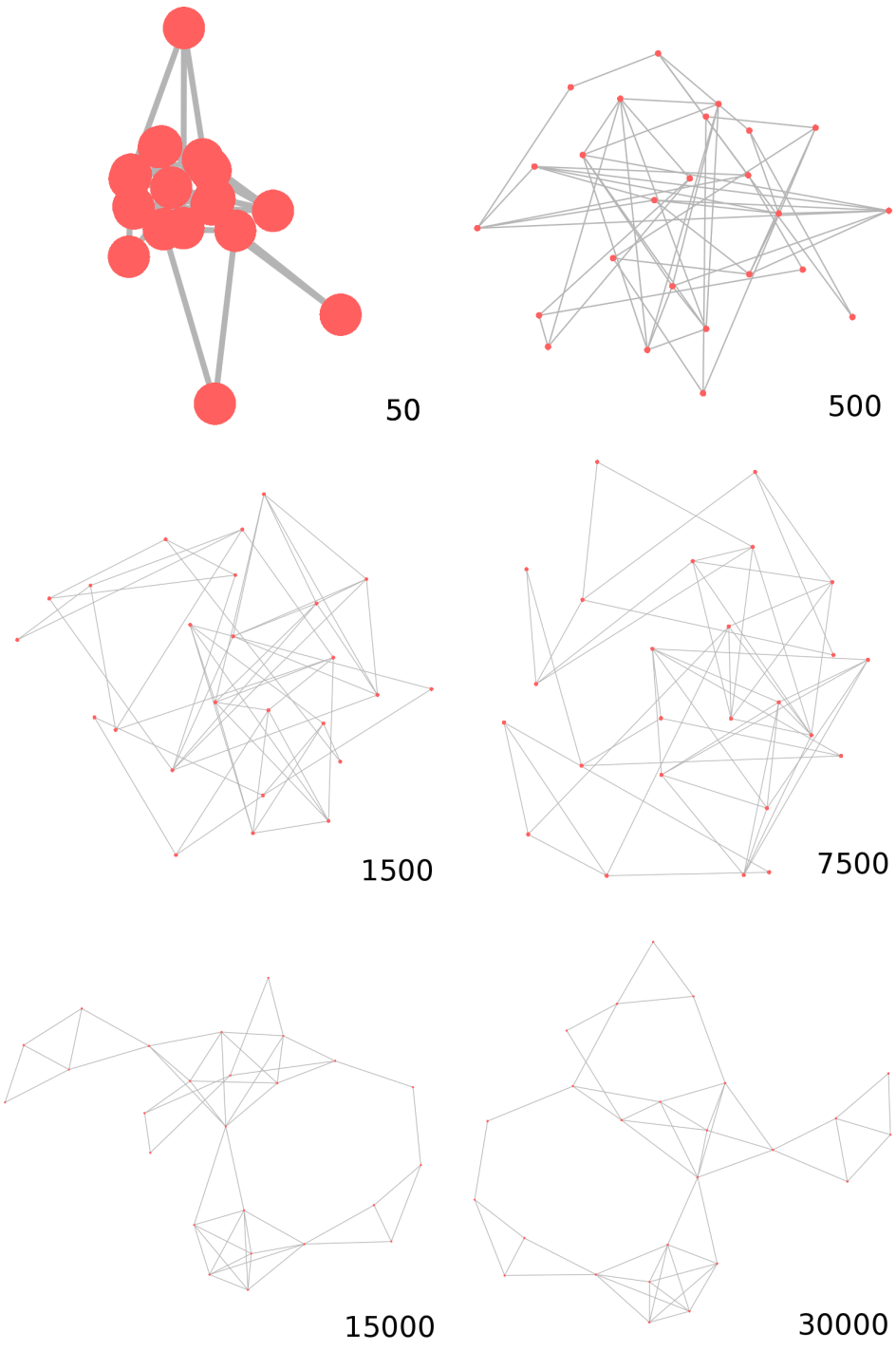


FIGURE 2.3 – Plusieurs étapes durant le déroulement de GEM,
un algorithme par modèle de force.
Le nombre associé à chaque image indique le nombre d'itération qui ont été effectuées.

Algorithme 1: calculerImpulsion(Sommet **s**)

Argument : Sommet **s** : Le sommet pour lequel la force à appliquer doit être calculée
Données : Tableau de particule **particules** : Les particules représentant les différents sommets dans le système physique
Données : Graphe **graphe** : Le graphe à dessiner
Données : Position **centre** : L'iso-barycentre du layout
Constante : Scalaire **agitation** : Le coefficient de la force aléatoire
Constante : Scalaire **gravité** : Le coefficient de la force de gravité
Constante : Scalaire **longueurArête** : La longueur idéale des arêtes
Constante : Scalaire **attractionMaximale** : La borne supérieur de la force d'attraction
Résultat : Impulsion **résultante** : L'impulsion devant être appliquée au sommet
début

```
p ← particules[s]
résultante ← Impulsion nulle
// La force aléatoire
résultante ← résultante + générerForceAléatoire(agitation)
/* Permet de générer une force dont chaque composante est comprise entre
   -agitation et agitation                                              */
// La force de gravité
résultante ← résultante + (centre - p.position)*gravité*p.masse
/* La force de gravité est proportionnelle à la distance entre
   l'isobarycentre et la particule, orientée vers le centre             */
// Les forces de répulsion
pour tous les Sommet autre : Sommets de graphe faire
    si autre ≠ s alors
        delta ← p.position - particules[autre].position
        n ← (delta.norme())²
        si n > 0 alors
            résultante ← résultante + delta * longueurArête / n
// Les forces d'attraction
pour tous les Sommet voisin : Voisins de s faire
    delta ← particules[voisin].position - p.position
    n ← delta.norme() / p.masse
    si n > attractionMaximale alors
        n ← attractionMaximale
        // Permet de borner la norme de l'attraction
    résultante ← résultante + delta * n / (longueurArête² + 1)
retourner résultante
```

Les ouvrages de Kaufmann et Wagner et de Battista *et al.* [6, 53] offrent de nombreux éléments concernant la multitudes d'algorithmes qui existent pour dessiner les graphes. Une part importante est dédiée aux algorithmes par modèles de force et leurs dérivés.

Problème du passage à l'échelle

Cette problématique de la taille du graphe est très importante, en effet, les premiers algorithmes ne pouvaient traiter que des graphes d'au plus une trentaine de sommets. Cela était bien évidemment lié aux capacités des ordinateurs à cette époque, toutefois, les algorithmes par modèles de forces comme définis ci-dessus ont une complexité en $O(n^2)$ (pour pouvoir calculer les interactions entre chaque paire de sommets) associés à une constante très grande (liée au temps de simulation). Dans certains cas, le temps de simulation étant lié à la taille du graphe, la complexité de l'algorithme peut être en $O(n^3)$ voir en $O(n^4)$ pour certains algorithmes (GEM utilise par exemple une simulation en $O(n^2)$ étapes).

Ces algorithmes ne permettent donc que de traiter des graphes de taille restreinte. Or de très nombreux graphes réels sont très grands (plusieurs milliers voir plusieurs centaines de milliers de sommets). Une nouvelle variante permet de dessiner ces graphes, les algorithmes multi-niveaux. Le principe de ces derniers consiste à construire une hiérarchie des sommets du graphe. Ensuite, chaque niveau de la hiérarchie est dessiné en partant du sommet de cette dernière. Les niveaux suivants sont ensuite ajoutés progressivement au dessin. Lors de chaque ajout, les nouveaux sommets ne sont dessinés que par rapport à leurs voisins dans la hiérarchie. De ce fait, cela limite fortement la quantité de calcul réalisé lors de ces phases de dessins.

Un algorithme de ce type, FM^3, a été mis au point par Hachul et Jünger en 2005 [47]. Cet algorithme introduit une très forte composante multi-niveaux et s'appuie sur les forces liées aux potentiels des champs électriques pour placer les sommets. Ce type d'algorithme permet le dessin de graphes très grands, avec une complexité en $O(n\dot{l}og(n))$.

Respect de contraintes et de dynamiques locales

Une deuxième limite de ces algorithmes vient de l'homogénéité du traitement qu'ils appliquent au graphe. En effet, l'ensemble du graphe est toujours traité de la même manière, sans prise en compte de caractéristiques locales particulières. De plus, il n'est pas possible de leur faire respecter certaines contraintes d'alignement ou d'angles particulières.

Deux méthodes tentent de répondre partiellement à ces deux problématiques. Toutefois, ces approches s'éloignement quelque peu des algorithmes par modèle de force. La première piste, TopoLayout, a été présentée par Archambault *et al.* en 2007 [4]. Le principe consiste à identifier certaines structures connues au sein du graphe afin de les dessiner avec des algorithmes adaptés, comme ceux présentés dans la partie 2.1.6. Ces algorithmes présentent le grand avantage d'être extrêmement rapide en plus de dessiner très bien ces graphes. Un des points délicats consiste en l'assemblage des différents dessins ainsi obtenus.

Une autre approche proposée par Dwyer *et al.*, à travers les systèmes DIG-CoLa [24] et IPSep-CoLa [25], consiste à définir des ensembles de contraintes qu'un système de résolution linéaire tente de respecter au mieux. Cette approche est assez différente puisqu'aucune force n'intervient dans le calcul du dessin, et le layout est obtenu directement une fois l'application du solveur linéaire terminé.

2.3 Un graphe, plusieurs dessins

Toutes ces méthodes permettent d'obtenir en général un dessin pour un graphe. De très nombreux progrès ont été accomplis pour permettre le traitement de plus grands graphes, plus rapidement, et le résultat obtenu est en général de bien meilleure qualité. Toutefois, ces algorithmes sont trop nombreux, et parfois difficilement accessibles. Même pour un expert travaillant spécifiquement sur le dessin de graphe, il serait très difficile de connaître tous les algorithmes existants et leurs spécificités. Quand il s'agit d'un utilisateur novice, il est évident qu'il est impossible d'attendre cela de lui. Cela rend difficile le choix d'une méthode lors du dessin d'un graphe.

Comme cela a été dit dans l'introduction et a été décrié par Brandes dans un keynote [15], ce choix se restreint généralement à un des algorithmes les plus connus. Cela nuit fortement à la qualité des dessins obtenus, car comme le dit Brandes, ces algorithmes connus sont maintenant largement surclassés par les nouvelles méthodes.

De plus, le fait de toujours utiliser un nombre restreint de méthodes de dessins réduit la variété des dessins obtenus, ce qui induit une diminution de l'adaptation des dessins aux besoins de l'utilisateur. Cette question de l'adaptation aux besoins est primordiale, car les graphes ne sont utiles comme outils de modélisation des données que si ils permettent de répondre efficacement aux questions de l'utilisateur final.

Le travail de cette thèse tente justement de répondre à cette problématique. La solution présentée dans l'introduction consiste à générer plusieurs dessins d'un même graphe pour l'utilisateur qui choisit ensuite celui qui lui convient le mieux. Cela implique de pouvoir dessiner un même graphe de plusieurs manières différentes. La partie suivante aborde les différentes solutions qui ont été envisagées durant cette thèse.

2.3.1 Paramétrage des algorithmes

Une première solution pour générer plusieurs dessins consiste à utiliser un unique algorithme en jouant sur ses paramètres. En effet, la plupart des algorithmes par modèles de force, et plus généralement l'ensemble des algorithmes de dessin, admettent un grand nombre de paramètres. Ces derniers permettent de contrôler le comportement de l'algorithme en jouant sur la durée de la simulation ou sur les coefficients des différentes forces.

Toutefois, cette voie n'est pas aussi efficace qu'elle y parait. En effet, les paramètres n'ont généralement pas une influence très importante sur l'aspect du dessin lorsque les changements sont légers, et des éloignements trop grands des valeurs par défaut « cassent » complètement le dessin. La figure 2.4 présente 4 dessins d'un même graphe à l'aide de l'algorithme GEM, que nous utiliserons aussi par la suite. Le premier est obtenu avec l'ensemble des paramètres par défaut. Pour les 3 autres, des paramètres ont été plus ou moins modifiés, le deuxième reste relativement proche du résultat de référence, les deux autres s'en éloignent d'une manière très peu intéressantes.

2.3.2 Première solution : Ensemble d'algorithmes

La première solution envisagée dans le cadre de cette thèse a été d'utiliser un ensemble d'algorithmes pour générer les dessins. L'idée consiste à générer des dessins avec plusieurs algorithmes

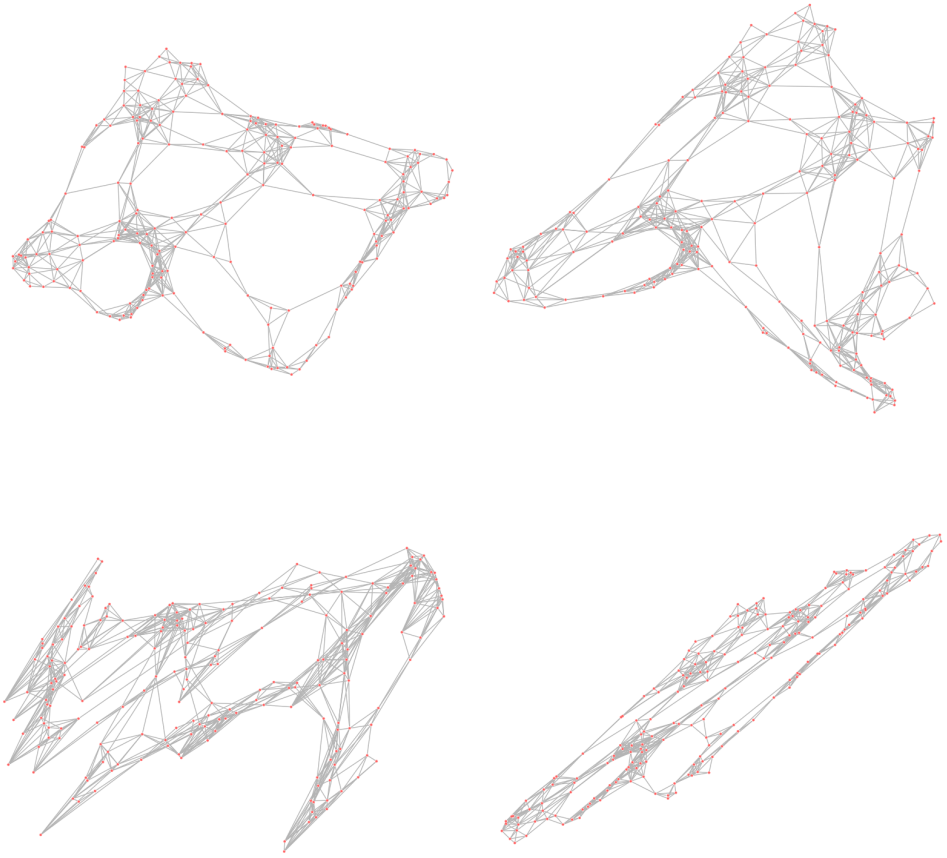


FIGURE 2.4 – 4 dessins pour le même graphe, obtenus avec GEM

différents lorsque l'utilisateur fournit un graphe. Ces différents dessins sont ensuite proposés à l'utilisateur qui choisit ceux qui lui conviennent le mieux.

Cependant nous avons rapidement abandonné cette piste car les résultats obtenus étaient très décevants.En effet, il n'était pas possible d'affiner et d'améliorer les dessins générés. Comme nous venons de le voir, les paramètres ne permettent pas vraiment de modifier de manière utile le comportement d'un algorithme. De plus, il n'était pas du tout possible de mélanger deux dessins obtenus avec des méthodes différentes. Ce système revenait finalement à exécuter linéairement l'ensemble des algorithmes disponibles sur l'ordinateur de l'utilisateur et de lui montrer les résultats obtenus. Même si cela permet à l'utilisateur d'avoir plusieurs exemples de dessins, cela ne permet tout de même pas d'avoir une réelle adaptation aux besoins de l'utilisateur.

2.3.3 Deuxième solution : Mise en place d'un algorithme multi-forces

L'abandon de cette première piste a conduit à la problématique suivante : Comment générer tous les dessins qu'il est possible d'obtenir pour un graphe ? Une première réponse, qui sera abordée plus loin consiste à générer aléatoirement une position pour chaque sommet. Cette approche sera approfondie dans le chapitre 4 dédié à l'utilisation d'algorithme génétique dans le cadre du dessin de graphe. Il est possible de borner l'espace de dessin en ne considérant pas les effets d'échelles (deux dessins identiques à l'échelle près sont considérés comme identiques). Cette solution permet de répondre trivialement au problème (car tous les dessins sont effectivement accessibles), mais l'espace de recherche est alors extrêmement vaste et difficile à explorer.

Nous avons plus orienté notre recherche sur les algorithmes de force, en réfléchissant aux problèmes en ces termes : Quelles sont les forces nécessaires à un algorithme par modèle de force pour pouvoir dessiner un graphe de toutes les manières possibles ? Les algorithmes par modèle de force ont mis en place les forces d'attraction et de répulsion. Certains instaurent aussi des forces globales, comme une gravité. D'autres ajoutent des contraintes d'alignement vertical ou horizontal ou de positionnement régulier. De toutes ces forces, lesquelles sont nécessaires pour pouvoir obtenir tous les dessins possibles ?

Cette idée a mené à l'implémentation d'un algorithme par modèle de force modulable, dans lequel de nouvelles forces pouvaient facilement être ajoutées en cas de besoin. Chaque sommet se voyait attribuer un ensemble de force qui agissait sur lui durant la simulation.

Toutefois, cette solution a dû être abandonnée pour raison technique et par manque de temps. En effet, cet algorithme a été mis en place en même temps que l'algorithme génétique qui sert de support au système interactif. Cette simultanéité a induit trop de problèmes techniques à régler en même temps. Nous avons donc choisi une solution plus simple modifiant un algorithme de force déjà existant. Cela limite possiblement la variété des dessins accessibles, mais a permis l'obtention de dessin intéressant bien plus rapidement. Toutefois, la réalisation d'un algorithme de force permettant de manière sure d'obtenir tous les dessins possibles reste un problème ouvert des plus intéressants.

2.3.4 Solution choisie : Utilisation et modification de GEM

La solution qui a donc été retenue dans le cadre de cette thèse a été d'utiliser un algorithme de force déjà existant, GEM, en lui apportant une modification qui permet d'obtenir de très nombreux dessins pour un même graphe. Cela nous a permis d'obtenir très rapidement des dessins

satisfaisant en consacrant nos efforts sur la mise en place du système interactif.

Cette partie va tout d'abord réaliser une présentation précise de l'algorithme GEM. Le deuxième paragraphe se concentrera sur la modification que nous lui avons apportée. La suite se concentre sur les possibilités offertes par cette évolution ainsi que ses limites.

Présentation de GEM

L'algorithme GEM, pour "Graph EMbedder", a été présenté par Frick en 1995 [32]. Il s'inspire principalement des travaux de Fruchterman et Reingold [34] en ajoutant plusieurs éléments à la définition basique des algorithmes par modèle de force. La première amélioration apportée vient des systèmes de recuit simulé. Le problème est le suivant : une recherche d'un optimum par descente de gradient peut relativement facilement se bloquer dans un extremum local. Le principe du recuit simulé est d'accepter tout changement qui améliore l'état du système et d'accepter de temps à autre un changement qui le détériore. La probabilité d'accepter un tel changement dépend d'abord de l'ampleur de la détérioration, mais aussi d'une température globale du système qui décroit peu à peu au cours de la simulation. Lorsque la température est nulle, un changement non améliorant n'est plus accepté.

De plus, ces systèmes physiques présentent couramment des instabilités périodiques qui n'ont finalement pas d'impact majeur sur le dessin obtenu au final. Ces instabilités sont généralement des rotations ou des oscillations. GEM introduit un contrôle de ces mouvements (en comparant chaque mouvement appliqué à un sommet aux précédents). Lorsqu'un tel mouvement est détecté son impact est limité et la température locale est diminuée.
Enfin, la fin de l'algorithme est déterminée soit par un nombre d'itérations maximal, soit en fonction de la température maximale du système. Lorsque cette dernière passe sous un certain seuil l'algorithme est terminé.

La dernière évolution introduite dans GEM concerne la masse des sommets. En effet, le mouvement de chaque sommet est pondéré par l'inverse de la masse du sommet (plus le sommet est lourd, plus il est difficile à déplacer). Cette masse est calculée au début de l'algorithme et correspond au degré de chaque sommet. Cela permet de déplacer plus doucement les sommets reliés à beaucoup d'autres afin d'éviter de « casser » une trop grande partie du dessin lors d'un déplacement.

Algorithme

GEM se déroule en deux étape, la première sert à insérer les sommets un par un dans le système. Ensuite, la deuxième déplace chaque sommet les un après les autres vers une position "plus" intéressante.

L'algorithme de calcul de la force appliqué à un sommet présenté page 29 est lui aussi utilisé par l'algorithme présenté ici. Certaines fonctions sont simplifiées (comme celle du déplacement d'un sommet). Ces passages simplifiés sont ceux qui n'ont subis aucune modification lors de l'intégration dans notre système. Pour en avoir une descriptions complète en pseudo-code, vous pouvez vous référer à l'article original de Frick [32].

L'algorithme complet est décrit en 4 blocs différent, le principal est décrit dans l'algorithme 2, la phase d'insertion dans l'algorithme 3, le déplacement des sommets dans l'algorithme 4. De

plus l'algorithme 1 de calcul de l'impulsion présenté page 29 est lui aussi utilisé.

Algorithme 2: GEM

Entrées : Graphe **graphe** : Le graphe à dessiner
Sorties : Layout **résultat** : Le résultat de l'algorithme
Constante : Entier **itérationMax** : Nombre d'itération maximum
Constante : Scalaire **températureArrêt** : Température signalant la fin de l'algorithme
début

```
    nombreSommets ← graphe.nombreDeSommets
    particules ← Tableau de nombreSommets particules vides
    // Les particules sont indéxées selon les sommets du graphe
    pour tous les Sommet s : Sommets de graphe faire
        p ← particules[s]
        p.position ← (0,0)
        p.masse ← 1 + graphe.degré(s) / 3
        p.in ← 0 // Paramètre utilisé lors de la phase d'insertion

    phaseInsertion() // Voir algorithme suivant

    // Phase d'arrangement
    nombreItération ← 0
    s ← graphe.premierSommet()
    tant que températureSystème() > températureArrêt ET nombreItération <
    itérationMax faire
        f ← calculerForce(s)
        déplacerSommet(s,f)
        s ← graphe.sommetSuivant(s)
        /* Cette fonction boucle, lorsque s correspond au dernier sommet du
           graphe, c'est le premier qui est renvoyé                          */
    pour tous les Sommet s : Sommets de graphe faire
        résultat.définirPosition(s, particules[s].position)
    retourner résultat
```

Algorithme 3: phaseInsertion

Entrées : Graphe **graphe** : Le graphe à dessiner
Entrées : Tableau de particule **particules** : Les particules représentant les différents sommets dans le modèle physique
Constante : Entier **itérationMaxInsertion** : Le nombre maximum d'itération de la phase d'insertion
Constante : Scalaire **températureFinInsertion** : La température signalant la fin de la phase d'insertion

```
début
    n ← heuristiqueCentreGraphe(graphe)
    // Le premier sommet placé est le centre du graphe
    premierSommet ← Vrai
    pour i allant de 1 à graphe.nombreSommet() faire
        si premierSommet alors
            premierSommet ← Faux
        sinon
            n ← Sommet tel que particules[n].in soit minimal
        particules[n].in = 1 ;
        pour tous les Sommet v : Ensemble des voisins de n faire
            si particules[v].in ≤ 0 alors
                particules[v].in ← particules[v].in - 1
                /* Les valeurs in des voisins pas encore placés de n sont
                   diminuées pour favoriser leur sélection par la suite      */
        particules[n].position ← Isobarycentre des voisins déjà placés de n
        nombreItération ← 0
        tant que particules[n].température > températureFinInsertion
        ET nombreItération < itérationMaxInsertion faire
            /* La position de la particule est ajustée jsuqu'à ce que sa
               température atteigne la température de fin d'insertion      */
            impulsion ← calculerImpulsion(n)
            déplacerSommet(n,impulsion)
            /* Les températures du système et de la particule sont mises à
               jour durant le déplacement du sommet                        */
            nombreItération ← nombreItération + 1
```

Algorithme 4: déplacementSommet(Sommet **s**, Impulsion **impulsion**)

Entrées : Sommet **s** : Le sommet à déplacer
Entrées : Impulsion **impulsion** : L'impulsion, non nulle, à appliquer au sommet
Données : Tableau de particule **particules** : Les particules représentant les sommets
début
  impulsion
  ← affaiblirRotationOscillation(**impulsion**,**particules**[s].impulsionPrécédente)
  `/* Cette fonction compare les deux impulsions pour déterminer si des phénomènes de rotations ou d'oscillation sont présents */`
  `// Elle n'a aucun effet lors du premier déplacement d'une particule`
  particules[s].température ← miseAJourTempérature(**s**, **impulsion**)
  `// La température globale du système est aussi mise à jour par cette fonction`
  miseAJourCentreSystème(**s**, **impulsion**)
  `// Mise à jour de l'isobarycentre du système`
  impulsion ← **impulsion** * **particules**[s].température
  particules[s].position ← **particules**[s].position + **impulsion**
  particules[s].impulsionPrécédente ← **impulsion**

Modification apportée

Toutefois, comme nous l'avons vu, GEM dans sa version originale ne permet pas de générer des dessins réellement différents pour un même graphe. Toutefois, des travaux ont montré qu'en modifiant certains paramètres localement dans le graphe, il était possible d'améliorer fortement la qualité des dessins. Par exemple, dans le cadre du dessin de graphes de communautés, des groupes de sommets sont connus pour former une communauté. Réduire la longueur idéale des arêtes au sein de chaque groupe et l'augmenter pour les arêtes entre deux groupes permet d'améliorer fortement la qualité du dessin obtenu au final en termes de visibilité des différentes communautés. Ce concept a entre autres été introduit par Eades et Huang en 2000 [27].

Nous avons poussé plus loin cette transformation en permettant de définir l'ensemble des paramètres pour chaque sommet séparément. Cette nouvelle version a été nommée GaGEM puisqu'elle est utilisée conjointement avec un algorithme génétique. Les paramètres concernés correspondent aux coefficients des différentes forces, la longueur idéale des arêtes et la masse de chaque sommet.

D'un point de vue algorithmique, l'impact est relativement faible, puisque seule la fonction **calculerImpulsion** a été modifiée. De plus, les différents paramètres définis localement ont été ajouté à la structure *Particule*. Les paramètres rajoutés sont les suivants :

- La masse du sommet
- La longueur idéale de ses arêtes
- La coefficient d'attractions qu'il applique à ses voisins
- Le coefficient d'attraction qu'il s'applique à lui même
- Le coefficient de répulsions pour les autres sommets
- Le coefficient de répulsion qu'il applique aux autres
- L'amplitude de sa force aléatoire
- Le coefficient de sa force de gravité

De plus, pour la plupart, ces paramètres peuvent prendre des valeurs négatives, ce qui per-

met de fortement influencer le comportement de l'algorithme. Le déroulement de cette nouvelle fonction est décrit dans l'algorithme 5.

Comportement de l'algorithme modifié

Il est difficile d'étudier simplement l'effet de cette modification. En effet, comme nous le verrons dans le paragraphe 2.3.4, la première conséquence de cette modification correspond à une très forte augmentation du nombre de paramètres. Nous n'avons donc pas trouver de tests simple à réaliser sur cette nouvelle méthode de dessin.

Cependant, comme nous allons le voir, l'utilisation de GaGEM avec un algorithme génétique permet de réaliser des tests de plus haut niveau. Une tâche très souvent considérée consiste à reproduire un layout fixé avec GaGEM. Cela correspond donc à trouver un ensemble de paramètres à définir sur chaque sommet afin d'obtenir un layout le plus proche possible de la cible. Le paragraphe suivant présente les premiers essais qui ont été réalisés avec GaGEM. La deuxième moitié du chapitre 4 est consacrée à la présentation de quatre utilisations différentes de notre système (et donc de GaGEM). Les deux premières étant consacrée à la duplication d'un layout, et les deux autres à la générations de nouveaux layouts pour un graphe.

Cependant, cette nouvelle méthode présente aussi certaines limites qui impactent fortement les capacités de notre systèmes. Le paragraphe 2.3.4 présentent justement les différent éléments qui posent aujourd'hui problème avec GaGEM. Une partie de la conclusion de ce manuscrit est aussi dédiée à un point de vue plus large sur l'impact qu'à l'utilisation de GaGEM sur notre système.

Algorithme 5: calculerImpulsionModifiée(Sommet **s**)

Données : Tableau de particule **particules** : Les particules représentant les différents sommets dans le système physique, contenant tous les paramètres locaux
Données : Graphe **graphe** : Le graphe à dessiner
Données : Position **centre** : l'iso-barycentre du layout
Constante : Scalaire **attractionMaximale** : La norme maximale de la force d'attraction
Entrées : Sommet **s** : Le sommet pour lequel l'impulsion à appliquer doit être calculée
Sorties : Impulsion **résultante** : L'impulsion devant être appliquée au sommet
début

- **p** ← **particules**[s]
- **résultante** ← Impulsion nulle
- `// La force aléatoire`
- **résultante** ← **résultante** + générerForceAléatoire(**p**.agitation)
- `// L'amplitude de la force dépend maintenant de la valeur locale p.agitation`
- `// La force de gravité`
- **résultante** ← **résultante** + (**centre** - **p**.position)*__p__.gravité*__p__.masse
- `/* La masse du sommet ne dépend plus du degré du sommet, mais est fournie comme paramètre du sommet, de même que pour le coefficient de gravité */`
- `// Les forces de répulsion`
- **pour tous les** *Sommet* ***autre*** *: Sommets de* ***graphe*** **faire**
  - **si** ***autre*** $\neq$ ***s*** **alors**
    - **delta** ← **p**.position - **particules**[**autre**].position
    - **n** ← (**delta**.norme())^2
    - **si** ***n*** $>$ *0* **alors**
      - **résultante** ← **résultante** + **delta** * $\dfrac{\mathbf{p}.\text{répulsionPersonnelle} + \mathbf{particules}[\mathbf{autre}].\text{répulsionAutre}}{2 * \mathbf{n} * \mathbf{p}.\text{masse}}$
      - `/* Le coefficient de répulsion correspond à la moyenne entre la répulsion personnel de p et la répulsion externe de la deuxième particule. */`
      - `/* De plus, la masse est maintenant prise en compte, de la même manière que pour les forces d'attraction */`
- `// Les forces d'attraction`
- **pour tous les** *Sommet* ***voisin*** *: Voisins de* ***s*** **faire**
  - **delta** ← **particules**[**voisin**].position - **p**.position
  - **n** ← $\dfrac{\mathbf{delta}.\text{norme}()}{\mathbf{p}.\text{masse}}$
  - **si** ***n*** $>$ ***AttractionMaximale*** **alors**
    - n ← **AttractionMaximale**
    - `// L'attraction maximale reste une constante globale`
  - **résultante** ← **résultante** + delta * $\dfrac{(\mathbf{p}.\text{attractionPersonnelle} + \mathbf{particules}[\mathbf{voisin}].\text{attractionAutre}) * \mathbf{n}}{\mathbf{p}.\text{longeurArête}^2 + 1}$
  - `/* De la même manière, le coefficient d'attraction correspond à la moyenne des deux coefficients. */`
  - `// Ici, la masse est prise en compte via n`
- **retourner** ***résultante***

Premiers résultats obtenus

La première tâche qui a du être réalisée une fois le système mis en place a été la validation de la version modifiée de GEM. Il fallait poru cela répondre à la question : « Cet algorithme est-il capable de dessiner un même graphe de manières très différentes ? ».

Le premier essai a consisté à utiliser des mesures classiques de qualités de dessins de graphes (qui seront décrites dans le paragraphe 2.4.2). Toutefois, les dessins obtenus ainsi n'étaient pas très intéressants, soit parce qu'ils étaient inutilisables, soit parce qu'ils ressemblaient beaucoup aux dessins obtenus à l'aide d'algorithmes par modèle de force classique (qui s'appuient justement sur ces mesures de qualité).

Pour éviter ce problème, un processus de recopie a été mis en place. L'objectif du système n'était plus de générer de nouveaux dessins pour un graphe, mais de recopier un dessin qui lui est fourni en même temps que le graphe. L'intérêt de cette méthode consiste à fournir un dessin très éloigné de celui que produirait un algorithme par modèle de force (soit obtenu par une méthode différente, soit dessiné manuellement) pour voir si le système arrive à reproduire ce dessin en utilisant uniquement GaGEM.

Un premier essai a été réalisé avec un graphe représentant le mot « Hello ». Ce graphe, dont un dessin est fourni par la figure 2.5, présente plusieurs intérêts. Tout d'abord, il ne contient que peu de sommet et d'arêtes (24 sommets et 24 arêtes), ce qui permet une exécution rapide de GaGEM. De plus, comme ce graphe représente les lettres d'un mot, il est aisé d'identifier très rapidement les candidats les meilleurs au sein d'une interface affichant tous ceux obtenus. Le cerveau étant très habitué à la forme des lettres, cela permet de gagner beaucoup de temps lors des premières vérifications. Le dernier élément intéressant de cette image concerne les différentes contraintes contenues dans ce type de dessin. Tout d'abord, les lettres sont alignées, de ce fait, l'image est plus large que haute, or les algorithmes par modèles de force tentent généralement de regrouper les points (et donc d'obtenir un contour rond ou carré). De plus, certaines arêtes sont parallèles ou présentent des angles droits entre elles.

La figure 2.6 présente un layout obtenu avec le système. Il s'agit d'un des plus proche de l'objectif en terme de similarité moyenne de ses sommets. Cette dernière étant ici égale à 0,122.

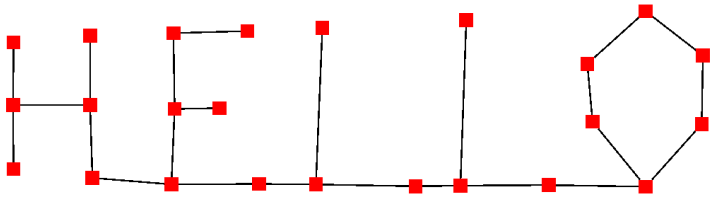


FIGURE 2.5 – Le premier dessin utilisé comme objectif du système

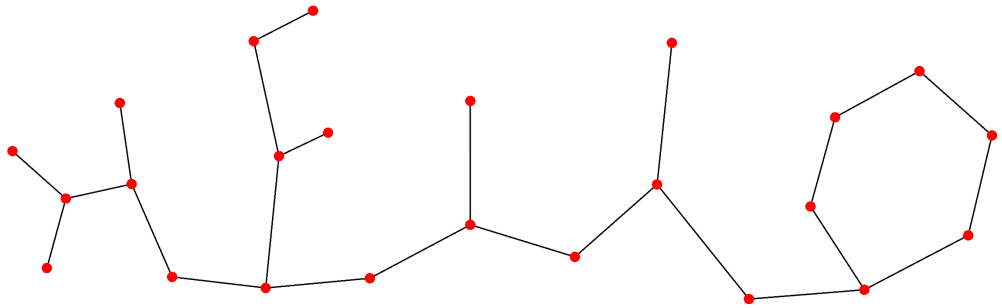


FIGURE 2.6 – Un des layouts les plus proche de l'objectif présenté dans la figure 2.5

Génération des paramètres

La première conséquence de cette modification de l'algorithme correspond à l'explosion du nombre de paramètres à définir . En effet, il y a maintenant 8 paramètres par sommets. Pour un graphe de 50 sommets, cela fait 400 paramètres. Même pour un utilisateur expert, cette tâche est bien trop lourde pour être réalisée manuellement. De plus, l'objectif final du système est de réaliser une exploration de l'espace des dessins du graphe. Il ne s'agit donc pas de trouver un jeu de paramètres correct, mais de pouvoir en générer des ensembles successifs, si possible en prenant en compte les retours de l'utilisateur.

Nous avons décidé d'utiliser un algorithme génétique pour réaliser cette tâche de génération des candidats. Le chapitre 3 est consacré à la description de l'algorithme génétique qui a été mis en place durant cette thèse, ensuite le chapitre 4 est consacré à l'utilisation de cet algorithme dans le cadre du dessin de graphe, et en particulier de manière conjointe avec cette version modifiée de GEM.

Un élément primordial des algorithmes génétiques concerne l'évaluation. Il est nécessaire d'attribuer un score à chaque candidat. La première possibilité consiste à afficher le dessin obtenu avec un candidat et l'afficher à l'utilisateur pour qu'il l'évalue. Cependant, cette méthode ne permet que d'évaluer qu'un nombre très limité d'éléments. Une problématique centrale de l'utilisation d'un algorithme génétique a donc été la mise en place de méthodes d'évaluation automatique afin de limiter au maximum celles réalisées directement par l'utilisateur. Une large place est dédiée à ces méthodes et aux outils attenants dans la suite de ce manuscrit.

Limites de la méthode

Cette méthode de dessin présente toutefois plusieurs inconvénients qui limitent sa portée. En effet, l'objectif fixé était d'avoir une méthode permettant de générer tous les dessins possibles pour un graphe. Or, suite aux différents tests que nous avons pu réaliser, cela ne semble à priori pas être le cas. En effet, les forces disponibles dans GEM : l'attraction, la répulsion, la gravité et le mouvement aléatoire ne permettent pas toutes les actions qui peuvent être nécessaires pour atteindre certains dessins. Cela constitue toutefois une limite relative dans notre cas puisqu'un grand nombre de dessins peuvent toutefois être atteints. Certaines contraintes ne pourront pas être exactement vérifiées, mais un dessin relativement proche pourra être atteint.

La deuxième limite vient du temps d'exécution de cet algorithme. Comme nous l'avons vu, sa complexité est en $O(n^2)$ associé à une constante généralement très grande. De ce fait, lorsque le graphe devient grand, l'exécution de l'algorithme peut devenir vite très longue (plusieurs dizaines de secondes pour un graphe de 1500 sommets et 15000 arêtes). Cela rend cette méthode inutilisable dans le cadre de la génération de candidats à l'aide d'un algorithme génétique (ou toute autre méta-heuristique d'exploration) qui nécessite l'évaluation de plusieurs centaines voir plusieurs milliers de candidats. Ce problème de passage à l'échelle a été résolu dans le cadre du dessin de graphe à l'aide des algorithmes multi-échelles.

La possibilité d'ajouter une composante multi-échelle a été un temps envisagée pour répondre à cette problématique. De plus, cela aurait sûrement permis d'atteindre de plus nombreux dessins en ajoutant une couche d'abstraction supplémentaire. Cependant, cette modification a été trop complexe à mettre en œuvre dans le temps dédié à cette thèse.

Les deux derniers points problématiques de cette méthode sont plus associés à son utilisation conjointe avec les algorithmes génétiques et seront de ce fait abordés bien plus précisément dans les chapitres suivants. Le premier concerne la répartition des paramètres et leur réutilisabilité dans le cadre de l'algorithme génétique. En effet, comme nous le verrons dans le chapitre suivant, l'algorithme génétique qui a été mis en place est très fortement axé vers l'utilisation de résultats obtenus lors d'exécutions précédentes. Nous avons du résoudre pour cela un problème important lié à la variation du nombre de sommets et de la topologie d'un graphe à l'autre (la topologie correspond ici à l'ensemble des arêtes et à la manière qu'elles ont de relier les différents sommets). En effet, la plupart des algorithmes génétiques utilisent un génome définissant directement une position pour chacun des sommets. Cela empêche toute réutilisation avec un graphe ayant un nombre de sommets différents et ne permet pas du tout de prendre en compte la topologie du graphe.

La solution que nous avons choisi consiste a utiliser des génomes composés d'un ensemble de règles, chacune associée à un jeu de paramètres. Lors de l'utilisation d'un génome pour dessiner un graphe, chaque sommet est associé au jeu de paramètres de la règle qui lui correspond le mieux. Cela permet d'avoir un nombre de règles indépendant du nombre de sommet du graphe dessiné. De plus, l'évaluation de la règle la plus adaptée à un sommet se base sur la topologie du graphe, ce qui permet de la prendre en compte dans le processus. Cette structure de génome sera plus amplement détaillée dans le chapitre 4.

Le deuxième problème posé par GEM pour une utilisation avec un algorithme génétique concerne l'aspect aléatoire du résultat. En effet, GEM, et de ce fait GaGEM, est muni d'une composante aléatoire importante. Plusieurs actions de l'algorithme font appel à la génération de nombre aléatoire (cela est explicite dans le cadre de la force aléatoire qui est appliquée à chaque sommet) ce qui induit une variabilité assez forte du résultat obtenu. Toutefois, pour obtenir un résultat identique en fournissant à deux exécutions les mêmes paramètres, la méthode de génération aléatoire est initialisée afin de fournir toujours la même suite de nombres, ce qui permet d'obtenir un comportement « déterministe » de la part de l'algorithme. Toutefois, ce caractère aléatoire induit une grande variabilité du résultat obtenu par rapport suite à une petite modification d'un paramètre. Comme nous le verrons par la suite, cela nuit à l'efficacité des algorithmes génétiques, et en particulier à celle de l'opérateur génétique de mutation.

2.4 Mesures sur un graphe

La dernière partie de ce chapitre concerne différentes mesures qu'il est possible d'effectuer sur un graphe. Ces mesures seront fortement utilisées par l'algorithme génétique pour caractériser les différents sommets et les dessins obtenus. Dans le domaine du dessin de graphe, ces mesures sont généralement appelées « métriques ». La plupart de ces métriques sont très connues dans la communauté du dessin de graphe et de la théorie de graphe plus généralement. Certaines ont été adaptées afin de fournir une valeur pour chaque sommet (mais la plupart fournissent ce type de résultat nativement). Toutes ces métriques sont calculées dans le cas de graphes simples et non orientés.

Le premier ensemble de métriques présentées sont dites topologique. En effet, elles sont calculées uniquement à l'aide de la définition formelle du graphe ((V,E) soit les ensembles des sommets et des arêtes). De ce fait, elles sont indépendantes de tout dessin. Ensuite sera présenté un ensemble de métriques graphiques, issues pour la plupart de de qualité d'un dessin de graphe. Ces dernières sont pour leur part calculées en fonction d'un dessin du graphe.

La dernière métrique présentée est un peu particulière, puisqu'elle permet de mesure la similarité de deux dessins d'un même graphe. Cette mesure est la deuxième contribution importante de cette thèse, et sera donc abordée en détail, elle occupe aussi une place centrale dans l'algorithme génétique.

2.4.1 Des métriques topologiques

La figure 2.7 présente une application de métriques suivantes à un graphe de manière visuelle.

Le degré

Le degré, qui a déjà été défini plus tôt (voir partie 2.1.1), est la métrique topologique la plus simple. Il associe à chaque sommet le nombre de voisins qu'il a. Cela revient aussi à compter le nombre d'arêtes incidentes à chaque sommet (les boucles n'étant comptées qu'une seule fois). Contrairement aux métriques suivantes, aucun calcul particulier n'est effectué pour obtenir cette métrique. Cette métrique prend des valeurs entières entre 0 (pour un sommet isolé) et $+\infty$.

Le coefficient de clustering

Comme nous l'avons vu dans le paragraphe 2.1.3, un cluster est un groupe de sommets très fortement connecté entre eux et avec peu de connexion avec le reste du graphe. Le but du coefficient de clustering est de mesurer si un sommet appartient à un tel cluster ou non. La moyenne de ce coefficient sur l'ensemble du graphe indique si le graphe est globalement constitué de cluster ou non.

La méthode de calcul consiste à considérer pour chaque sommet s la part de ses voisins qui

sont connectés entre eux.

Algorithme 6: Calcul du coefficient de clustering

Données : Graphe **graphe** : Le graphe
Données : Sommet **s** : Le sommet pour lequel le coefficient de clustering est calculé
Résultat : Scalaire **c** : Le coefficient de clustering de **s**
début
  nombrePaire $\leftarrow 0$;
  c $\leftarrow 0$;
  pour tous les *Paire de sommet (**a**, **b**) : Paires issues de **graphe**.voisins(**s**)* **faire**
    si *graphe.existeArête(**a**, **b**)* **alors**
      c $\leftarrow$ **c** $+ 1$;
    nombrePaire $\leftarrow$ **nombrePaire** $+ 1$;
  si *nombrePaire* > 0 **alors**
    c $\leftarrow \frac{\textbf{c}}{\textbf{nombrePaire}}$;
  retourner *c*;

Cette métrique a donc une complexité « brute » en $0(n^3)$. Cette borne est atteinte lors du calcul du coefficient de clustering d'une clique. Cependant, dans la majorité des cas, le temps de calcul est presque linéaire par rapport au nombre de sommets. Par exemple, avec l'hypothèse d'un degré borné, la complexité passe en $0(n * d^2)$.

La centralité (Betweenness centrality)

La centralité, plus couramment nommée « betweenness centrality », le terme anglais dédié, permet de déterminer si un sommet est sur de nombreux plus court chemin ou pas. Pour cela, tous les plus courts chemins entre toutes les paires de sommets du graphe sont calculés. Des compteurs sont placés sur chaque sommet et sont incrémentés lorsqu'un sommet est sur un plus court chemin. Le résultat de la mesure correspond à la valeur de chacun de ses compteurs divisés par le nombre de plus courts chemins sur l'ensemble du graphe.

Avec n le nombre de sommet du graphe, il y a $\frac{n(n-1)}{2}$ paires de sommets dans le graphe. Si le graphe est connexe, il y a donc au moins $\frac{n(n-1)}{2}$ plus courts chemins. Il peut y en avoir plus, car deux sommets peuvent être reliés par plusieurs chemins différents de même longueur.

D'un point de vue algorithmique, cette métrique s'appuie sur un parcours en largeur du graphe. Ce parcours est modifié afin de permettre d'arriver sur un sommet par plusieurs chemin de même longueur. Cela permet de compter le nombre de plus court chemin et d'obtenir la liste des prédécesseurs de chaque sommets durant ce parcours. Il suffit d'utiliser un marquage des sommets explorés à l'aide d'un nombre entier correspondant à la longueur du chemin entre la source du parcours et le sommet courant (et -1 pour un sommet pas encore exploré). Cette méthode est appelée **parcoursLargeurMultiChemin** et renvoit un triplet correspondant à l'ordre de visite des sommets, le nombre de plus courts les reliant à la source et leurs listes de prédécesseurs respectifs.

Le calcul de la centralité se déroule ensuite de la manière suivante :

Algorithme 7: Calcul de la centralité

Argument : Graphe **graphe** : Le graphe sur lequel est calculé la centralité
Résultat : Tableau de scalaires **résultat** : La valeur de la centralité pour chaque sommet
début
  résultat.initialiserValeurs(0)
  accumulateur ← Tableau de scalaires indéxés par les sommets
  pour tous les *Sommet* ***s*** *: Sommets de* ***graphe*** **faire**
    (**ordreParcoursInverse**, **nombreChemins**, **prédécesseurs**)
    ↪ ← parcoursLargeurMultiChemin(**graphe**, s)
    `// s est toujours le dernier sommet de ordreParcoursInverse`
    accumulateur.initialiserValeurs(*0*)
    pour tous les *Sommet* ***v*** *:* ***ordreParcoursInverse*** **faire**
      pour tous les *Sommet* ***pred*** *:* ***prédécesseurs[v]*** **faire**
        accumulateur[**pred**] ← **accumulateur**[**pred**] +
        ↪ $\frac{\textbf{nombreChemins}[\textbf{pred}]}{\textbf{nombreChemins}[\textbf{v}]}$ * (1 + **accumulateur**[**v**])
        `/* On ajoute à ses prédécesseurs la prise en compte des chemins aboutissants au sommet v */`
      si $v \neq s$ **alors**
        résultat[**v**] ← **résultat**[**v**] + **accumulateur**[**v**]
        `/* Tous les chemins aboutissant ou passant par v et partant de s ont été pris en compte, on l'ajoute donc au résultat */`
  retourner ***résultat***

Le coût d'un parcours en profondeur étant linéaire en fonction de la taille du graphe, la complexité de cette métrique est en $O(n^2)$ toujours avec n le nombre de sommet du graphe. Brandes a fourni une approximation plus précise de la compléxité en $O(nm)$ dans son article publié en 2001 [16]. Cette métrique prend des valeurs entre 0 et 1. Au niveau de son interprétation, plus la betweenness centrality d'un sommet est haute, plus ce sommet aura une place importante sur les grandes routes du graphe. Dans le cas d'un graphe en cluster, cela correspondra généralement aux sommets « ponts » qui relient les différents clusters entre eux. À l'opposé les sommets ayant une centralité basse sont relativement isolés ou peu importants et leur suppression n'aura que peu d'impact sur la structure du graphe.

Le degré de proximité

La dernière métrique topologique utilisée dans le cadre de cette thèse est le degré de proximité, nommé généralement closeness centrality, dans sa version proposée par Dangalchev en 2006 [21]. L'objectif est d'obtenir une estimation de la distance moyenne de chaque noeud au reste du graphe. Pour cela, pour chaque sommet s, la somme des distances à tous les autres sommets est effectuée, puis le résultat est divisé par le nombre de sommet afin de normaliser le résultat. La version proposée par Dangalchev permet une meilleure adaptation aux graphes qui manquent de connectivité. Pour cela il remplace le terme de la somme par $2^{-d(s,u)}$ avec $d(s,u)$ la distance entre les sommets s et u. Ainsi, les sommets non connectés, et qui sont donc séparés par une distance infinie ne sont pas problématiques. Il propose une normalisation plus complexe de cette mesure de proximité (au lieu de simplement divisé le résultat par le nombre de sommets du graphe). Toutefois, cette dernière n'est pas utilisée dans le cadre de cette thèse.

Le calcul de cette métrique se fait très simplement en réalisant un parcours en largeur depuis chaque sommet. Cela permet d'obtenir la distance entre un sommet et les autres sommets du graphe. La somme est réalisée au fur et à mesure du parcours. La complexité de cette métrique est donc en $O(n^2)$

2.4.2 Et graphiques

Contrairement aux précédentes, les métriques suivantes sont calculées à partir du layout d'un graphe. Il s'agit donc de métriques portant sur la représentation graphique d'un graphe. Les quatre premières sont directement issues de mesures de la qualité d'un dessin de graphe établies par la communauté. Les suivantes correspondent plus à des indicateurs graphiques, qui ne sont généralement ni maximisé ni minimisé pour un « bon » dessin. Cependant, elles permettent d'obtenir des informations sur un dessin.

Nombre de croisements d'arêtes

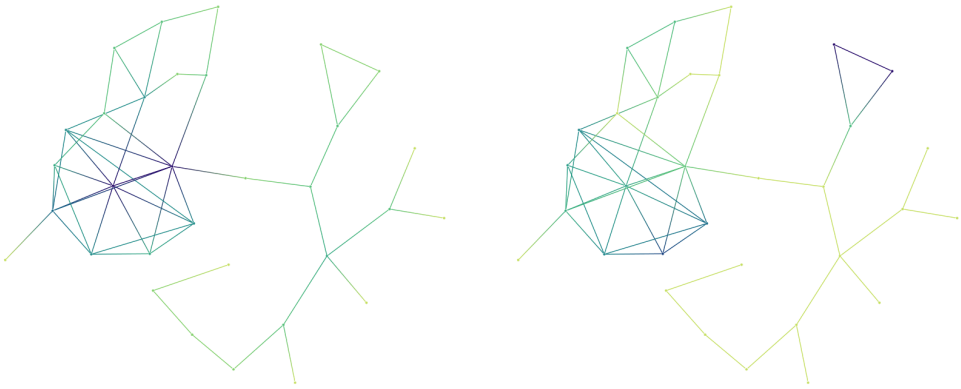
La première mesure de qualité présentée est le nombre de croisements d'arêtes. Il a été montré que les croisements d'arêtes nuisent à la lisibilité d'un graphe[76, 77]. De nombreux algorithmes de dessins tentent donc de réduire au minimum ce nombre de croisements. Selon un processus déterministe lorsque cela est possible (dans le cas d'un graphe planaire par exemple) ou en utilisant ce critère pour évaluer l'état d'un système.

Le décompte des croisements d'arête se fait de manière relativement simple en énumérant les paires d'arêtes et en comptant pour chacune d'elle le nombre de croisements qu'elles admettent avec d'autres arêtes. Afin d'obtenir une valeur pour chaque sommet, c'est la somme des nombres de croisements de chaque arête incidente à ce sommet qui est utilisée. La complexité de cette métrique est en $O(|E|^2)$ avec E l'ensemble des arêtes. Dans le cas d'un graphe complet (où toutes les arêtes possibles sont présente), on a $E = V^2$ donc $|E| = n^2$ avec n le nombre de sommets. La complexité de la métrique devient alors $O(n^4)$. Il est cependant possible de réaliser une recherche locale des croisements afin de réduire fortement le nombre de paire d'arêtes tester.

Conservation de la distance

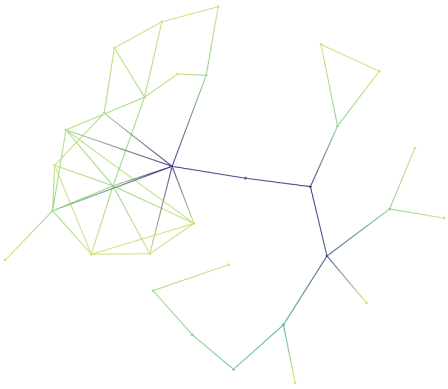
Cette métrique s'inspire de la fonction d'optimisation de l'algorithme par modèle de force proposée par Kamada et Kawai [52]. L'objectif de cet algorithme est d'avoir un rapport constant entre la distance euclidienne entre 2 sommets dans le dessin et leur distance topologique dans le graphe (les sommets non connectés n'étant pas pris en compte). Dans le paragraphe suivant, ce rapport entre les deux distances sera nommé ratio.

Pour réaliser cela, l'écart-type de ce ratio est calculé pour l'ensemble du graphe. Afin d'avoir une valeur pour chaque sommet, un écart-type local est utilisé. Cela revient à calculer pour un sommet s l'écart-type du ratio entre chaque paire formé s et un autre sommet du graphe par rapport à la moyenne du ratio sur l'ensemble du graphe. Cette valeur est ensuite divisée par la valeur moyenne du ratio sur l'ensemble du graphe afin de normaliser cet écart-type. Ce type d'écart-type local sera utilisé couramment par la suite pour obtenir une valeur locale à un sommet d'un écart-type normalement calculé sur l'ensemble du graphe.

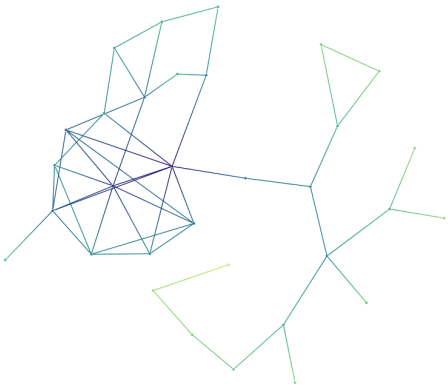


(a) Le degré

(b) Le coefficient de clustering



(c) La centralité



(d) Le degré de proximité

(e) L'échelle utilisée pour l'ensemble des images

FIGURE 2.7 – Les 4 métriques topologiques appliqués à un graphe.

De manière plus formelle, en nommant r le ratio, soit $r(u,v) = \frac{d_e(u,v)}{d_t(u,v)}$ avec d_e la distance euclidienne dans le dessin et d_t la distance topologique dans le graphe, $\overline{r}$ la valeur moyenne de r sur l'ensemble du graphe, l'écart-type local est obtenu selon la formule :

$$dist\_cons(s) = \frac{\sqrt{\frac{\sum_{u \in V, u \neq s} (r(s) - \overline{r})^2}{|V| - 1}}}{\overline{r}}$$

D'un point de vue algorithmique, un premier passage sur les paires de sommet permet de calculer la valeur moyenne de r. Ensuite, pour chaque sommet, la liste des sommets du graphe est égrenée afin de calculer l'écart-type local du ratio entre ce sommet et les autres. Cette métrique a donc une complexité en $O(n^2)$.

Longueur des arêtes

Deux métriques permettent d'obtenir des informations sur la longueur des arêtes. La première, la plus simple, consiste à simplement calculer la longueur moyenne des arêtes incidentes à un sommet. Cette métrique fournit plus une information sur le graphe plutôt qu'une mesure de sa qualité. En effet, la moyenne ne fournit finalement pas énormément d'information sur le rendu visuel.

La deuxième métrique basée sur la longueur des arêtes correspond à l'écart-type de cette longueur dans le graphe. En effet, il est généralement plus simple de lire un graphe lorsque l'ensemble des arêtes ont la même longueur. Comme pour la conservation de la distance, c'est un écart-type local qui est calculé. Ce calcul se déroule de la même manière en utilisant la longueur des arêtes à la place du ratio. De plus, cet écart-type est maintenant calculé uniquement sur les arêtes incidentes au sommet.

Résolution angulaire

La dernière mesure de qualité connue utilisée est la résolution angulaire. Le but de cette métrique est d'estimer la répartition des arêtes autour d'un sommet. En effet, un graphe est généralement plus lisible s'il est facile de distinguer les différentes arêtes qui partent d'un sommet. De ce fait, il faut éviter qu'elles soient regroupées d'un même côté. Plus généralement, il est intéressant d'éviter que deux arêtes fassent un angle trop aigu lorsque cela est possible.

Cette métrique se calcule sommet par sommet. Pour un sommet s, l'ensemble des arêtes incidentes sont triées selon l'angle qu'elles forment avec une arête de base choisie arbitrairement. Cela permet d'obtenir les angles entre 2 arêtes consécutives. La métrique correspond à l'écart-type de cette série d'angles (il s'agit ici d'un écart-type classique). La moyenne étant forcément égale à $\frac{2*\Pi}{d(s)}$ avec $d(s)$ le degré de s, puisque la sommes des angles consécutifs forme toujours un disque complet.

Si cet écart-type est grand, cela signifie que les angles varient beaucoup autour de l'angle moyen, et donc que la répartition des arêtes autour du sommet est mauvaise. Au contraire s'il est nul, la répartition est parfaite.

Densité graphique

Cette métrique, comme les suivantes, n'est pas issue d'une mesure de qualité connue. Elle a été créée dans le cadre de cette thèse sans que cela ne puisse être considéré comme une contribution. En effet, elle est très simple et présente de nombreuses lacunes, mais elle permet toutefois d'apporter des informations intéressantes sur le dessin.

L'objectif de cette métrique est d'estimer la densité des sommets autour d'un sommet considéré. Elle calcule simplement pour chaque sommet la somme suivante :

$$d(s) = \frac{\sum_{v \in V, v \neq s} \frac{1}{d(s,v)^2}}{|V| - 1}$$

Avec s un sommet du graphe, et $d(s,v)$ la distance euclidienne entre les positions des sommets s et v. Cela permet de différencier les zones chargées des zones plus vides du graphes. Au niveau de la mesure de la qualité du dessin obtenu, il est intéressant que les sommets soient homogènement répartis, il est donc intéressant que l'écart-type de cette valeur sur l'ensemble du graphe soit faible.

Excentricité graphique

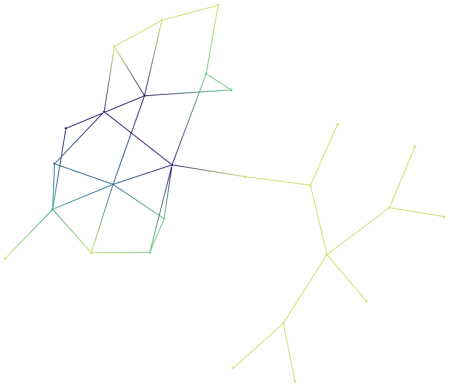
Toute comme la précédente, cette métrique a été construite dans le cadre de cette thèse. Il s'agit simplement de calculer la distance entre le centre du dessin (l'isobarycentre de tous les sommets) et chaque sommet. Cette valeur est divisée par la distance moyenne afin d'éviter tout problème d'échelle.

2.5 Une nouvelle méthode de comparaison de layout

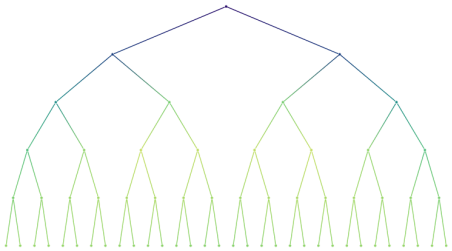
La dernière mesure présentée dans le cadre de ce chapitre correspond à la deuxième contribution majeure de cette thèse. Le but de cette mesure est d'évaluer la similarité de deux dessins d'un même graphe. Cette mesure est énormément utilisée dans le cadre de notre système interactif. En effet, l'objectif est de pouvoir proposer des dessins différents à l'utilisateur. Il est donc important de pouvoir déterminer si deux dessins se ressemblent ou pas. La définition complète de cette métrique est fournie au paragraphe 2.5.1.

La contrainte majeure que devait vérifier la méthode est que les différentes homothéties du plan ne doivent pas être prises en compte. Pour rappel, les homothéties du plan sont les transformations qui conservent les rapports des distances entre les éléments. Cela correspond aux rotations, aux translations, aux symétries, aux effets d'échelles et à leurs compositions. En effet, d'un point de vue humain, le fait d'appliquer une quelconque de ces transformations ne change en général pas le ressentit lors de la lecture du graphe. Il est donc important de ne pas les prendre en compte lors du calcul de la similarité. Cela est encore plus important dans le cadre de l'utilisation de GaGEM, car ce dernier peut produire deux layouts très proche à une homothétie prêt après une modification même faible d'un paramètre.

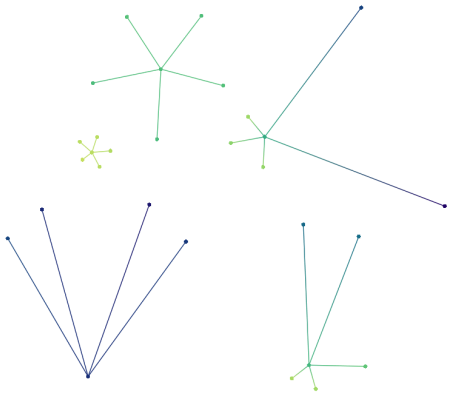
Un autre élément important concerne le fait qu'il n'est pas nécessaire de déterminer une association point à point des deux ensembles à comparer. Dans notre cas, chaque sommet est associé à un identifiant unique, et les deux ensembles correspondent à deux layouts du même graphe. De ce fait, chaque sommet a exactement une position dans chaque layout. Il n'est donc



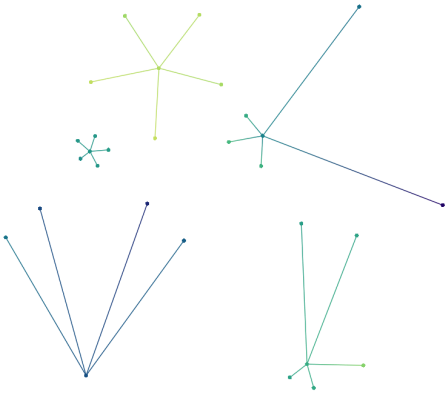
(a) Le nombre de croisements d'arêtes



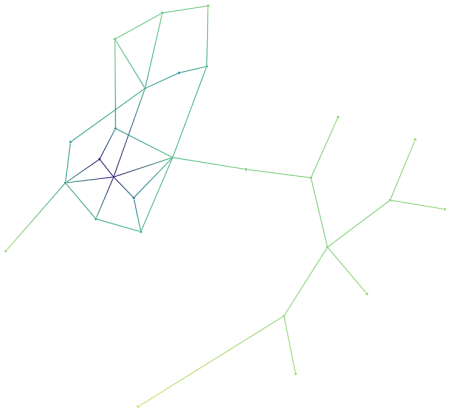
(b) La conservation de la distance, ou Stress Majorization



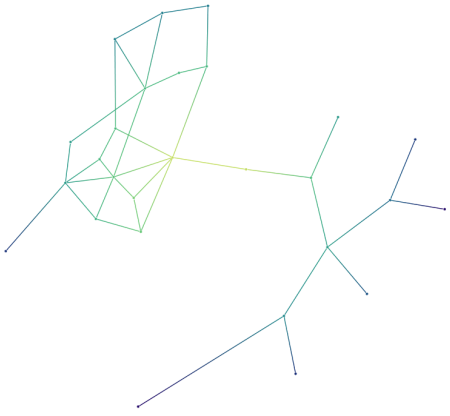
(c) La longueur moyenne des arêtes



(d) L'homogénéité de la longueur des arêtes



(e) La densité graphique



(f) L'excentricité graphique

FIGURE 2.8 – Les 6 métriques graphiques appliqués à différents graphes.

pas nécessaire de trouver pour chaque point un point correspondant dans l'autre layout.

De nombreuses méthodes permettent de réaliser des comparaison de ce type. Bridgeman a réalisé en 2002 [18] un survey sur un ensemble de méthodes basées strictement sur la comparaison de layout de graphes. Cependant la plupart de ces méthodes se basent sur des classements des sommets pour ensuite comparer ces classements. Une autre approche du problème consiste à considérer un layout comme un observation d'une distribution de probabilités. Cela permet ensuite d'utiliser les outils de comparaison développé dans ce domaine. Enfin, les algorithmes issues de la méthode "Iterative Corresponding Point" (ICP) pourraient se révéler intéressant. Cependant, il serait possiblement nécessaire d'adapter la méthode pour permettre une convergence de l'algorithme même lorsque les deux layouts sont très différents.

Cependant, ces différentes méthodes ne répondaient pas parfaitement à nos besoins, nous avons donc essayé cette nouvelle métrique dans sa version la plus simple. Les résultats obtenus étant intéressant et cette approche semblant relativement nouvelle par rapport à celles existant au sein de la communauté. Nous avons donc décidé de pousser plus loin l'analyse des différentes propriétés de cette métrique.

2.5.1 Principe et méthode calculatoire

Cette méthode se base sur le principe des triangles semblables. Deux triangles, dont les côtés sont notés a,b,c et a',b',c' sont semblables si et seulement si $\frac{a}{a'} = \frac{b}{b'} = \frac{c}{c'}$ (en supposant le nommage des côtés adéquat). Cela implique également qu'il existe une homothétie du plan qui permet d'envoyer le premier triangle sur le deuxième. Le principe de la méthode consiste à dire que deux layouts sont semblables lorsque tous les triangles qu'il est possible de réaliser avec les sommets du graphe sont semblables dans les deux layouts. La figure 2.9 présente par exemple deux layouts et 4 des 10 triangles qu'il est possible de faire à partir des sommets ($\binom{3}{n}$ triangles possibles pour un graphe à n sommets). Il est important de noter qu'il n'est pas nécessaire que des arêtes soient présentes pour réaliser un triangle.

Quelques notations vont être nécessaires pour la suite de ce paragraphe. Notons l_1 et l_2 deux layouts d'unu même graphe G. Soient d_1 et d_2 les distances euclidiennes entre les sommets dans l_1 et l_2, et d_G la distance topologique dans G (le nombre d'arête du plus court chemin entre les deux sommets). Toutes ces distances prennent deux sommets comme arguments. Le rapport entre les distances d_1 et d_2 sera noté r, donc $r(u,v) = \frac{d_1(u,v)}{d_2(u,v)}$ avec u et v deux sommets de G, on a donc $r : V \times V \to \mathbb{R}^+$. De plus, il sera admis que les layouts ne permettent pas la superposition des sommets (afin d'éviter que $d_2(u,v)$ ne soit nul).

Si tous les triangles qu'il est possible d'obtenir sont semblables dans les deux layouts, alors le ratio r est constant pour l'ensemble des paires de sommets du graphe.

En effet, avec U et V deux sommets du graphe, il est alors possible de faire $n-2$ triangles qui utilisent ces deux sommets. Les troisièmes points de ces triangles seront notés W_1,W_2,$\ldots$ W_{n-2}. On a alors :

$$\begin{aligned} &\forall k \in [1..n-2] : r(U,V) = r(U,W_k) = r(V,W_k) \\ \Rightarrow &r(U,V) = r(U,W_1) = \ldots = r(U,W_{n-2}) = r(V,W_1) = \ldots = r(V,W_{n-2}) \end{aligned}$$

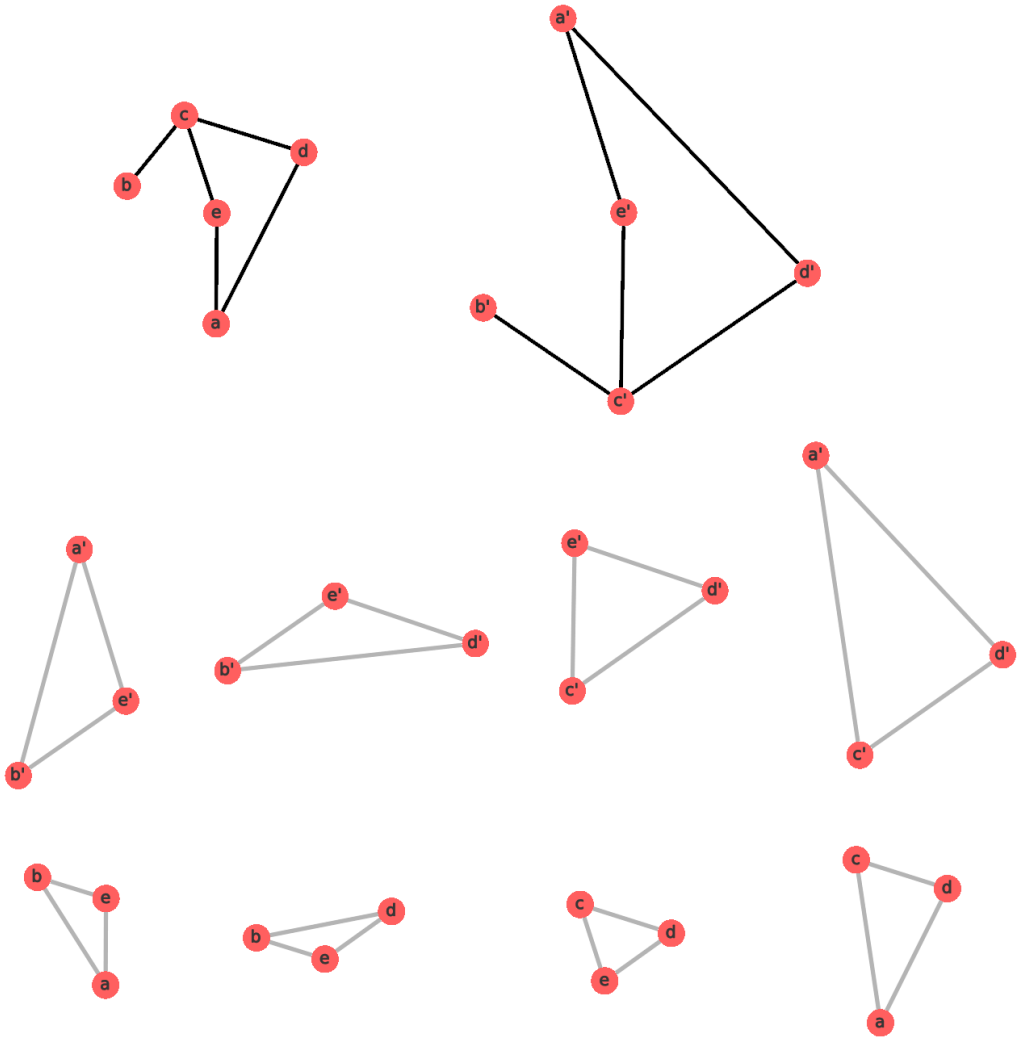


FIGURE 2.9 – Deux layouts semblables d'un graphe à 5 sommets et 4 triangles qu'il est possible de faire à partir de ces sommets

Il est maintenant possible de faire varier le sommet V. En effet, tout triangle contenant U contiendra un des côtés de un des triangles qui viennent d'être traités (un des $r(U,W_k)$). On obtient ainsi la propriété suivante :

$$\exists \alpha \in \mathbb{R}^+ tq. \forall (U,V) \in V^2 : r(U,V) = \alpha$$

Il devient alors possible de comparer les deux layouts en étudiant l'écart-type de ce rapport sur l'ensemble du graphe. Afin d'éviter tout problème d'échelle, il est intéressant de normaliser cet écart-type par la moyenne du rapport (sinon un layout dans lequel tous les points seraient très proches serait considéré comme similaire à tous les autres).

C'est dans le cadre de cette métrique que la version locale de l'écart-type a été développée. En effet, nous étions intéressés par l'obtention d'un score pour chaque sommet. Pour rappel, l'écart-type local normalisé se calcule de la manière suivante :

$$\phi_{loc}(r,u) = \frac{\sqrt{\frac{\sum_{v \in V, v \neq u} (r(u,v) - \overline{r})^2}{|V|-1}}}{\overline{r}}$$

Avec $\overline{r}$ la valeur moyenne sur l'ensemble du graphe (et non pas sur l'ensemble des paires de sommets contenant u).

Cet écart-type est défini sur $\mathbb{R}^+$, et deux layouts sont similaires si cette valeur est nulle. Cela pose donc un petit problème de vocabulaire (cette mesure devant plutôt être appelée dissimilarité de ce fait). La notation suivante a toutefois été utilisée :

$$sim(u) = \phi_{loc}(r,u)$$

D'un point de vue algorithmique, le calcul de tous les écarts-types locaux peut être effectué en deux passages sur l'ensemble des paires de sommets. Le premier passage permettant de calculer la valeur moyenne du ratio et le deuxième permettant de calculer les écarts-types locaux. Cela permet donc d'obtenir une complexité en $O(n^2)$ avec n le nombre de sommet du graphe.

Le retour de l'écart-type local à l'écart-type global peut être fait en réalisant la moyenne de toutes les variances locales (les écarts-types locaux mis au carré) puis en considérant la racine carrée de cette moyenne. Cependant, comme nous le verrons dans le chapitre suivant, la moyenne des écarts-types locaux a souvent été utilisée comme estimateur de l'écart-type global.

Paramétrage et similarité locale

Il est possible d'ajouter un coefficient à chaque terme de l'écart-type afin d'obtenir une similarité qui prend en compte la topologie du graphe. Pour cela, la distance topologique (notée d_g) est utilisée. Le coefficient associé à chaque ratio $r(u,v)$ peut être une puissance positive ou négative de $d_G(u,v)$. Dans le cas d'une distance positive, les sommets éloignés du sommet considéré auront plus d'impact sur le résultat, dans le cas contraire, ce seront les sommets proches du graphe qui seront prédominants. C'est généralement ce deuxième cas qui présente le plus d'intérêt, car il permet de s'assurer du respect de motifs locaux sans considérer leur disposition globale.

Il est aussi possible d'utiliser une fonction seuil, qui permet par exemple de prendre en compte uniquement les sommets à une distance inférieure à 5. Cette alternative permet un respect strict

de motifs de certaines tailles, toutefois, elle a été moins utilisée dans le cadre des essais qui ont été effectués.

La partie 2.5.2 présente une série de résultat comparant les résultats obtenus avec différents coefficients.

Dissymétrie de la mesure

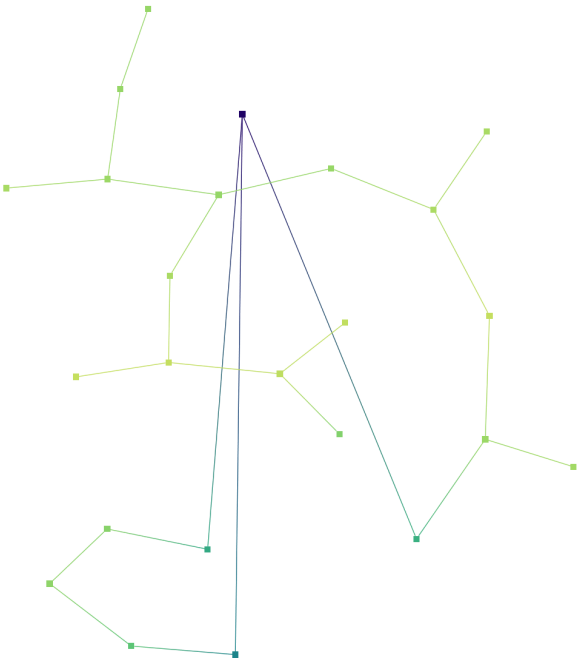
Cette mesure présente la particularité de n'être pas symétrique, ce qui à priori peut semble surprenant pour une mesure de similarité. La figure 2.10 présente explicitement cette dissymétrie. Elle présente deux layouts l_1 et l_2 d'un même graphe, tous deux identiques sauf pour un sommet, noté u qui a été fortement déplacé dans l_1. Les sommets du graphe dessinés avec l_1 sont colorés en fonction de la similarité $s(l_1,l_2)$. Les sommets du deuxièmes sont colorés en fonction de $s(l_2,l_1)$. Ce changement inverse fractions du ratio r, les distance dans l_1 sont au numérateur pour le premier cas, et au dénominateur dans le second.

Il est intéressant d'identifier en quoi consiste précisément cette dissymétrie. Dans les deux cas, le sommet déplacé, noté u, est celui qui est toujours le plus dissimilaire. En effet, presque tous ses ratios aux autres sommets ont été modifiés. L'écart-type de ces ratios au ratio moyen est donc toujours importants. 3 groupes de sommets peuvent ensuite être identifiés dans le graphe, ceux du bas, dont u est est plus loin dans l_1 que dans l_2, ceux du centre, dont u ne s'est ni éloigné ni rapproché. En effet, les sommets au centre sont relativement proche de la médiatrice du segment formé par les deux positions de u (avant et après sont déplacement). Enfin, les sommets du haut du graphe, dont u est plus proche dans l_1 que dans l_2.

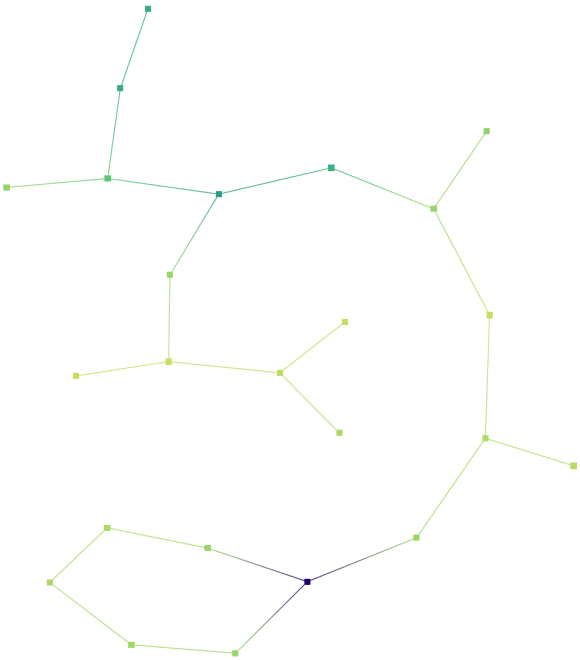
En comparant les deux colorations, on observe que dans le premier cas, ce sont les sommets du bas qui sont le plus impactés, et dans le deuxième cas ce sont les sommets du haut. Les sommets au centre sont toujours les plus similaires, car leur distance à u n'a presque pas changé. Dans les deux cas, les sommets les plus impactés sont avec lesquels u a un ratio grand. En effet, tous les sommets proche de u dans l_2 ont vu u s'éloigner dans l_1. De ce fait, en considérant que v est un de ces sommets, le ratio $r(u,v) = \dfrac{d_1(u,v)}{d_2(u,v)}$ est largement supérieur à 1. De ce fait, lors du calcul de la similarité, le ratio moyen sera presque à 1 (seul 1 sommet s'est déplacé, donc pour tous les couples de sommets ne contenant pas u, les ratios sont tous égaux à 1). De ce fait, les écarts à la moyenne de tous ces couples seront presque nuls. Il est possible d'approximer la similarité de v par $\sqrt{\dfrac{(r(u,v)-1)^2}{|V|}} = \dfrac{|r(u,v)-1|}{\sqrt{|V|}}$.

Cette approximation permet de comprend pourquoi ce sont toujours les grands ratio qui impactent le plus l'écart-type. En effet, Si un ratio est inférieur à 1, l'écart à la moyenne (et donc à 1 dans l'approximation) sera toujours compris entre 0 et 1, ce qui n'est pas le cas si le ratio est supérieur à 1. En supposant que $r_{1\to2}(u,v) = \dfrac{d_1(u,v)}{d_2(u,v)} = 3$, c'est à dire que u est 3 fois plus loin de v dans l_1 que dans l_2, alors $s_{1\to2}(v) = \dfrac{2}{\sqrt{V}}$ et $s_{2\to1} = \dfrac{1-\frac{1}{3}}{\sqrt{|V|}} = \dfrac{2}{3*\sqrt{|V|}}$. Si au contraire u s'était rapproché de v dans l_1 par rapport à l_2, c'est la deuxième similarité qui aurait été la plus grande.

Cet effet provient de l'utilisation d'un écart-type arithmétique avec une grandeur géométrique. En effet, l'écart-type dans sa définition classique considère une distance à la moyenne en util-



(a) Le layout l_1 coloré selon la similarité $s(l_1,l_2)$



(b) Le layout l_2 coloré selon la similarité $s(l_2,l_1)$

FIGURE 2.10 – Une mise en exergue de la dissymétrie de la mesure

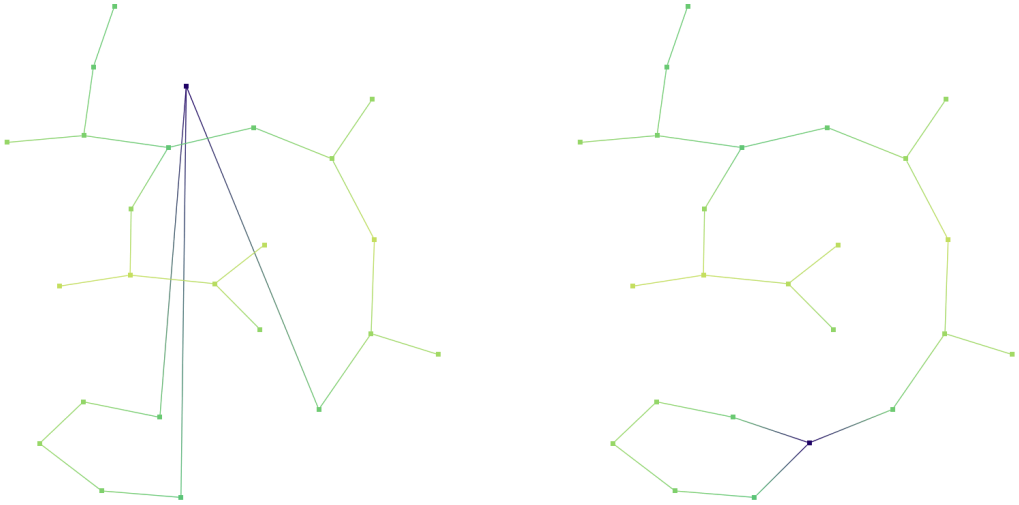


FIGURE 2.11 – Les similarité des deux layouts en utilisant un écart-type géométrique
Cette fois, les similarités sont bien identiques dans les deux cas

isant une différence et une mise au carré pour obtenir une "distance". De ce fait, deux valeurs de la séries auront le même impact sur l'écart-type si elles sont à égales distance moyenne. En notant $\overline{x}$ cette moyenne, on peut avoir $x_1 = \overline{x} + \alpha$ et $x_2 = \overline{x} - \alpha$. Or dans notre cas, puisque nous considérons une fraction, c'est l'inverse de cette fraction qui est obtenu lors du calcul de la similarité réciproque, ce qui ne permet pas d'obtenir un effet identique sur l'écart-type.

Une solution à ce problème consiste à utiliser un écart-type géométrique qui suit la formule suivante :

$$\phi_g(r) = \exp\left(\sqrt{\frac{\sum_{x \in E} (\ln(r(x)) - \ln(\overline{r_g}))^2}{|E|}}\right)$$

Avec $\overline{r_g}$ la moyenne géométrique du ratio dont l'écart-type est calculé. L'utilisation de cet écart-type permet d'annuler cet effet de dissimilarité puisque les variations autour de la moyenne sont maintenant correctement prises en comptes. De plus il n'est plus nécessaire de normaliser l'écart-type obtenu par le ratio moyen. En effet, l'écart-type géométrique est déjà une grandeur sans unité. En effet, si une unité abstraite est associé au rapport entre les deux dimensions, cette unité est conservée lors du calcul d'un écart-type arithmétique (qui utilise une différence à la moyenne) mais elle disparaît lors du calcul de l'écart-type géométrique (suite à l'utilisation d'une fraction entre la valeur et la moyenne). La formule de la similarité pour le sommet u devient alors :

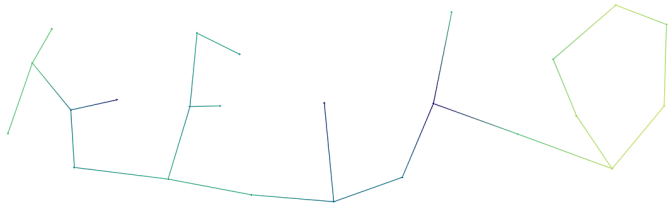
$$sim_{log}(u) = \exp\left(\sqrt{\frac{\sum_{v \in V, v \neq u} (ln(r(u,v)) - ln(\overline{r_g}))^2}{|V| - 1}}\right)$$

La figure 2.11 présente la similarité des deux layouts avec l'écart-type géométrique.

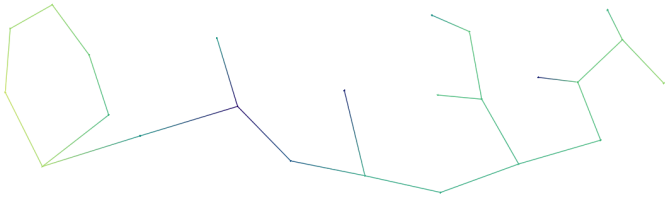
2.5.2 Résultats

Cette métrique a fournit des résultats très satisfaisant. D'un point de vue qualitatif, l'ordre fournit par la similarité à un objectif correspond à l'ordre ressenti. Les figure 2.12 et 2.13 présente un ensemble de dessins du graphe hello de moins en moins ressemblants, l'objectif visé étant repris à la fin de la liste. Quatre valeurs sont associées à chaque dessin, la moyenne des écart-types locaux, qui est utilisée comme estimateur de la similarité globale du graphe, la similarité globale du graphe, la valeur du sommet le plus similaire (min) et celui du moins similaire (max). Ces deux dernières valeurs permettent d'interpréter la coloration des sommets. En effet, cette dernière s'adapter à la plage des valeurs du dessin, le jaune pâle correspond donc au minimum et le bleu foncé au maximum. Il est intéressant de noter que l'estimation de la similarité globale est régulièrement inférieure à la similarité exacte, cependant, la variation est relativement faible et l'ordre des dessins est conservés. Cela ne peut toutefois être immédiatement généralisé. En effet, aucune étude théorique de l'efficacité de cet estimateur n'a pour l'instant été réalisée.

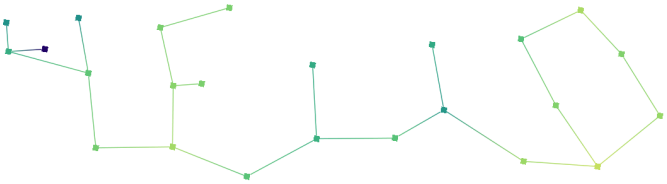
Le 5^e dessin proposé semble relativement proche de l'objectif, il a cependant un écart-type maximal relativement élevé, ce qui augmente fortement l'écart-type globale. En effet, on peut observer que deux sommets sont extrêmement proches l'un de l'autre. De ce fait, leur ratio sera très différent de celui dans le layout objectif, ce qui explique le maximum élevé. Pour des raison d'affichages, la plupart des dessins ont subi une rotation afin d'homogénéiser le format des images. Cependant, on peut tout de même observer l'insensibilité de la mesure aux autres homothétie. Par exemple, le deuxième dessin 2.13 présente une symétrie par rapport à l'objectif, le 3^e a été dessiné à une échelle très différente de tous les autres (éléments observables à la taille des sommets du graphes, qui sont très visibles, ce qui signifie que les arêtes sont « petites »). Globalement, les meilleurs graphes sont ceux qui ont réussi à placer l'ensemble des sommets dans la bonne zone. En effet, dès qu'un sommet est mal placé par rapport au reste, sa similarité est mauvaise. Par exemple, dans le 3^e dessin, le sommet en bas à gauche du H se retrouve au dessus de la barre centrale et obtient alors une très mauvaise similarité, même si le reste du dessin semble très intéressant.



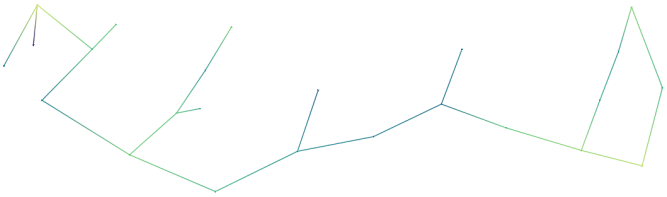
(a) Moyenne = 0.1525 - Ecart-type = 0.1601
Min = 0.0705 - Max = 0.2623



(b) Moyenne = 0.1676 - Ecart-type = 0.175
Min = 0.0822 - Max = 0.2963

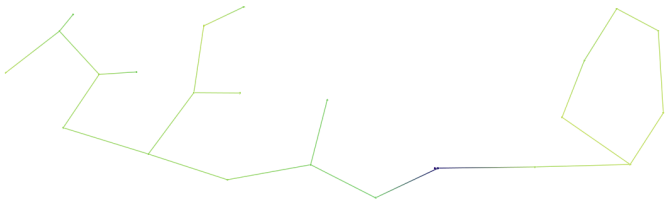


(c) Moyenne = 0.1751 - Ecart-type = 0.188
Min = 0.0718 - Max = 0.4397

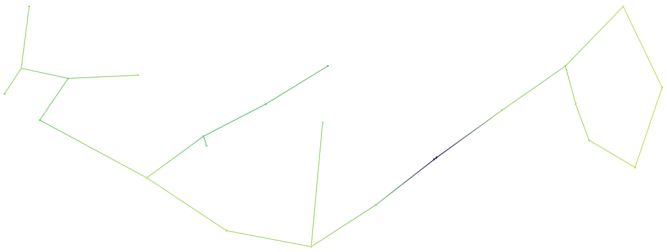


(d) Moyenne = 0.2409 - Ecart-type = 0.2484
Min = 0.1177 - Max = 0.4064

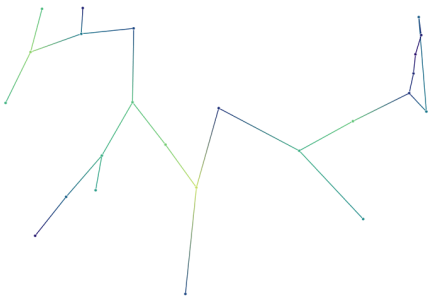
FIGURE 2.12 – Les 4 premiers résultats, pour lesquels il est aisé de lire le mot « Hello ».



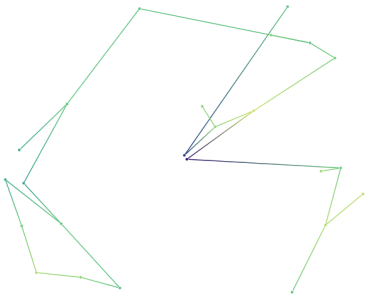
(a) Moyenne = 0.2774 - Ecart-type = 0.3388
Min = 0.0977 - Max = 1.3081



(b) Moyenne = 0.3774 - Ecart-type = 0.4189
Min = 0.1626 - Max = 1.3327



(c) Moyenne = 0.4008 - Ecart-type = 0.4134
Min = 0.1356 - Max = 0.5756



(d) Moyenne = 0.6728 - Ecart-type = 0.6904
Min = 0.3326 - Max = 1.5344

FIGURE 2.13 – Les quatre résultats suivants, les deux derniers étant très peu ressemblant à l'objectif.

Réactions aux transformations locales

Un autre intérêt de cette mesure de similarité concerne l'utilisation du coefficient présenté dans la partie 2.5.1. Ce dernier permet de prendre en compte la topologie du graphe lors du calcul de la similarité. Ce coefficient, noté p, peut prendre une valeur allant de -10 à 10, et prend une valeur de 0 dans la version normale de la métrique. La formule de la métrique de similarité devient alors :

$$sim^p_{log}(u) = \exp\left(\sqrt{\frac{\sum\limits_{v\in V, v\neq u} d_g(u,v)^p (ln(r(u,v)) - ln(\overline{r_g}))^2}{\sum\limits_{v\in V, v\neq u} d_g(u,v)^p}}\right)$$

Nous allons voir qu'en faisant varier p, il est possible de mettre en valeur des éléments très différents du graphe. Cette partie va présenter deux cas d'application sur deux graphes différents. Le premier est le graphe Hello dans lequel le dernier « L » et le « O » ont subi une rotation qui les a fait venir en dessous des autres lettres. Cette modification est présentée dans la figure 2.14. Le deuxième graphe provient d'un extrait de la base de donnée cinématographique IMDB. Chaque sommet représente un acteur, et chaque arête indique que les deux acteurs concernés ont joués dans un même film. Le graphe se base sur une dizaine de films. Deux modifications ont été effectuées sur ce graphe, la première consiste uniquement en un réaménagement de certains clusters localement (la position de certains sommets au sein du cluster à été changée). La deuxième modification consiste en une modification de la position globale de deux des clusters. La figure 2.15 présente 2 layouts, celui de gauche correspondant à la modification locale et celui de droite à la modification globale.

Deux cas d'études vont être présentés, un sur chaque graphe. Chaque cas d'étude se présentera sous la forme de trois images correspondant aux similarités locales (coefficient mis à -7), classique (coefficient à 0) et globale (coefficient à 7). De plus, un graphique présentant l'évolution du comportement de la mesure en fonction du coefficient est aussi fourni. Ce graphique comporte 4 courbes :

- La courbe jaune, nommée **Minimum**, qui correspond à la similarité du sommet de plus petite similarité pour chaque valeur de p.
- La courbe bleu, nommée **Maximum**, qui correspond à la similarité du sommet de plus grande similarité.
- La courbe rouge, nommée **Moyenne**, qui correspond à la moyenne des similarité de chaque sommet (ce qui correspond donc à l'estimateur de la similarité d'un layout, comme dit plus haut).
- La courbe verte, nommée **Ecart-type**, qui correspond à l'écart-type réel du ratio sur l'ensemble du graphe. Cette dernière courbe permet d'avoir une idée de l'erreur induite par l'usage de la moyenne comme estimation.

Toutes les similarités appliquées dans ces exemples utilisent un écart-type géométrique afin d'éviter tout problème de dissymétrie. Dans chaque groupe de 3 images, celle du centre est dessinée avec un des deux layouts comparés et les deux autres utilisent le deuxième. Cela permet de facilement comparer les deux layouts. L'utilisation pour l'affichage de l'un ou l'autre des layouts n'a aucune influence sur la similarité et la couleur des sommets.

Premier cas : Graphe « Hello » Ce premier cas montre la capacité de la mesure à identifier précisément le lieu autour duquel se place la modification. En effet, dans la figure 2.16a, les deux

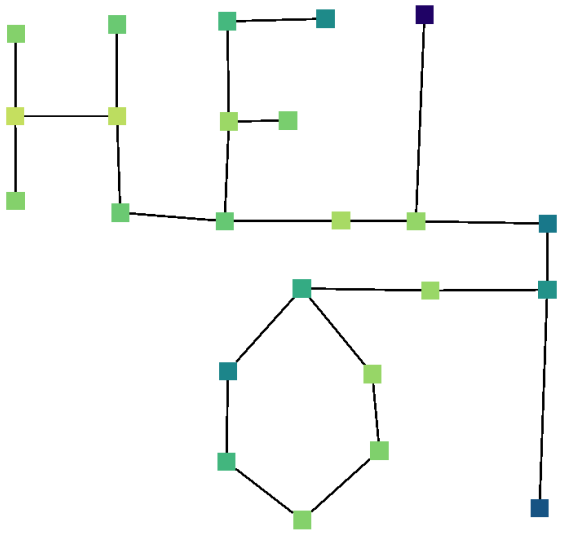


FIGURE 2.14 – Le graphe « hello » une fois modifié

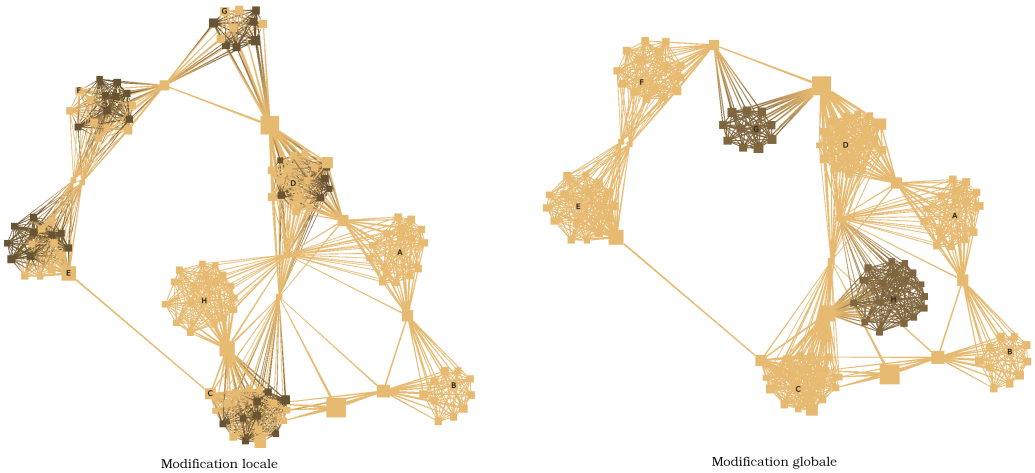
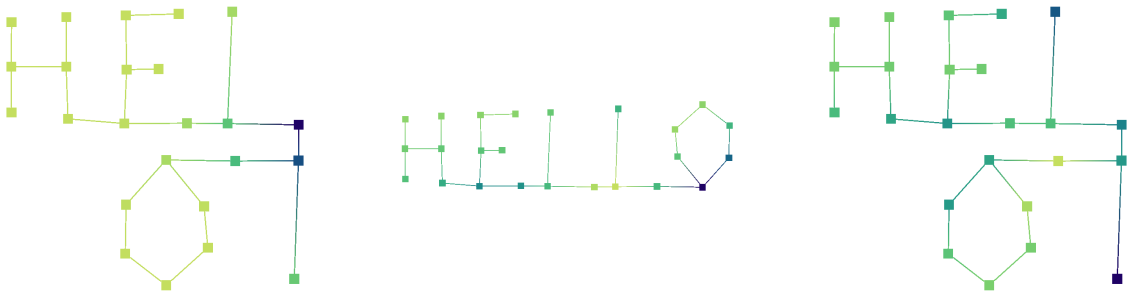
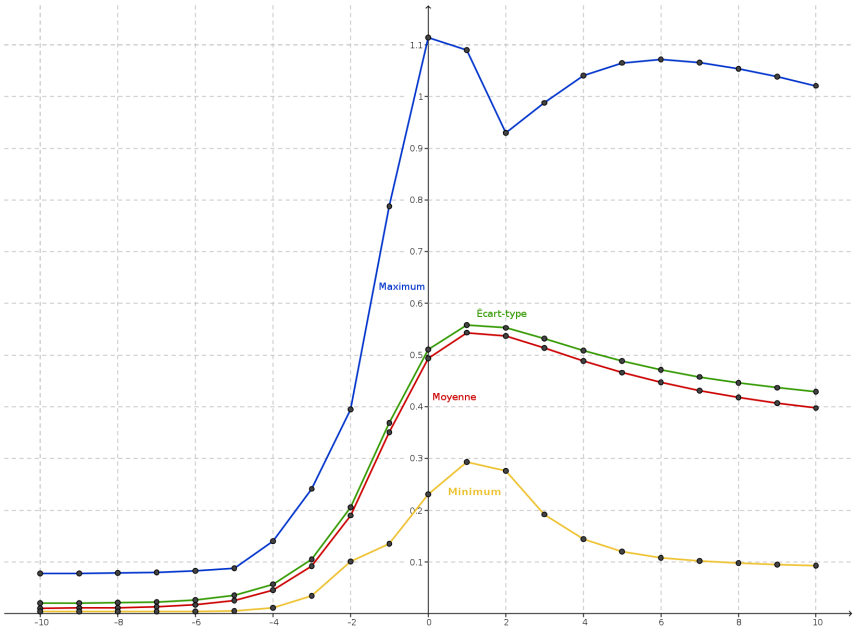


FIGURE 2.15 – Les 2 layouts sur lesquels se base le 2e cas
Les sommets les plus sombres sont ceux qui ont été déplacés



(a) -7 (b) 0 (c) 7



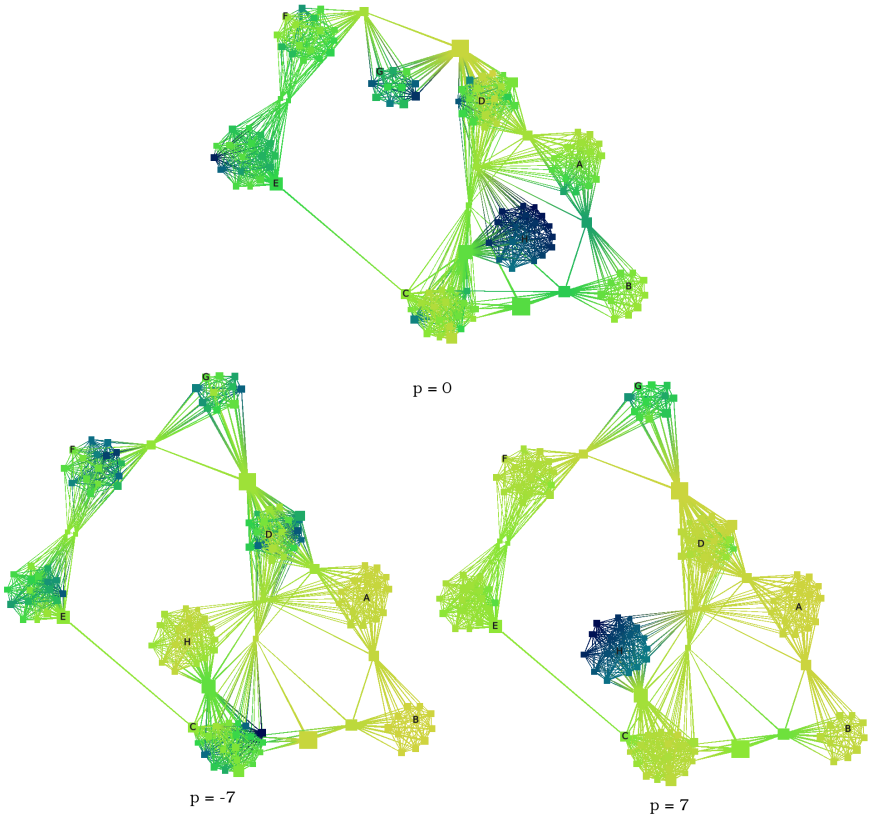
(d) La variation de la similarité en fonction du coefficient appliqué

FIGURE 2.16 – Le comportement de la similarité sur une modification apportée au graphe « Hello »

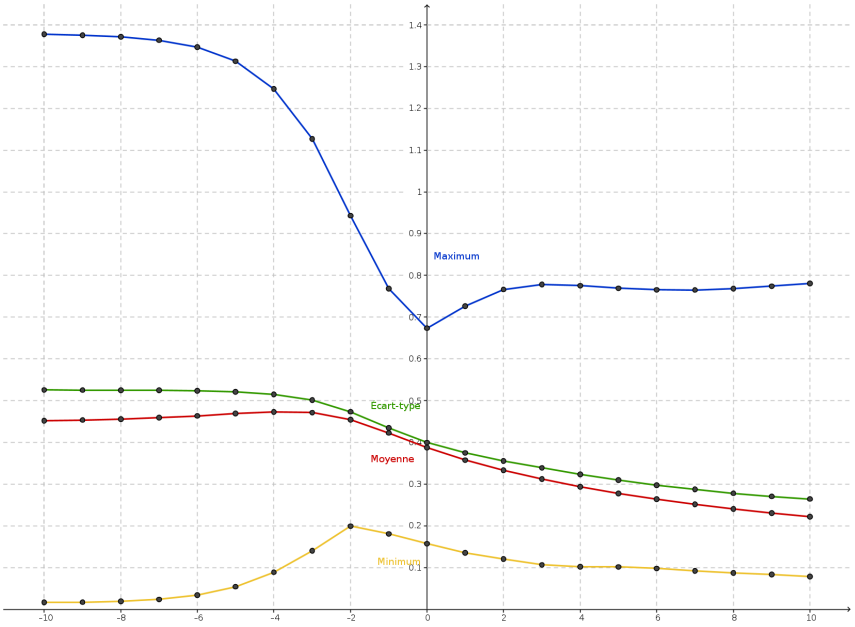
sommets autour desquelles s'est fait la rotation sont les seuls à se démarquer fortement. Les sommets les entourant étant bien moins impactés, et ceux à distances deux étant strictement similaires. Plus le coefficient est négatif, plus cet effet s'amplifie. Par contre, un coefficient positif apporte assez peu d'information, l'ensemble des sommets ayant été déplacés par rapport à au moins la moitié des autres.

Deuxième cas : Modifications locale et globale d'un graphe en cluster Ce deuxième cas utilise le graphe en cluster. Ici, la métrique de similarité permet de mettre en valeur des ensembles de sommet très différents en fonction du coefficient utilisé. Lorsque p est négatif, ce sont les clusters ayant subi un modification locale qui sont mis en avant. Avec une valeur positive, seuls les deux clusters déplacés globalement se démarquent des autres. Le résultat de la similarité classique indique quant à lui que la plupart des sommets ont subi une modification sans permettre d'avoir d'informations plus précises sur cette modification.

La modulation du coefficient de puissance appliqué à la distance entre les deux sommets per-



(a) Les sommets colorés en fonction de la similarité



(b) La variation de la similarité en fonction du coefficient appliqué

FIGURE 2.17 – Le comportement de la similarité sur des modifications locale et globale sur un graphe en cluster

met donc d'obtenir de nombreuses informations sur la similarité entre deux dessins, et surtout leur manière d'être similaire. D'un point de vue calculatoire, l'ajout du coefficient ne change pas la complexité globale. Il faut au préalable calculé l'ensemble des distances topologiques entre tous les sommets du graphe. Cette opération admet une complexité en $O(n^2)$, ce qui correspond à la complexité de la mesure de similarité classique. De plus, dans le cas de la comparaison de plusieurs dessins d'un même graphe, il est possible de ne calculer les distances topologiques entre les sommets qu'une seule fois car ces derniers ne dépendent aucunement d'un dessin. Cela permet de presque retrouver le temps de calcul de la similarité classique.

Temps de calcul

Le tableau suivant présente les temps de calculs de la similarité sur mon ordinateur personnel. La métrique n'est pour l'instant pas parallélisée, elle utilise donc un seul coeur cadencé à 1.87GHz. Les temps de calculs sont donnés en millisecondes. Dans le cas d'un coefficient différent de 0, les distances sont recalculées à chaque fois ce qui explique l'augmentation du temps de calcul. L'ensemble de ces résultats ont été obtenus à l'aide du logiciel Tulip

Il est à noter que le nombre d'arête n'influe pas sur le temps de calcul. En effet, ces dernières ont un impact négligeable sur le temps de calcul des distances topologiques et elles ne sont pas prises en compte lors du calcul de la similarité.

Nombre de sommets	Similarité classique	Similarité avec coefficient
50	5ms	14ms
200	55ms	120ms
500	240ms	650ms
1000	880ms	2537ms
10000	64,7s	317,3s

On remarque une forte augmentation sur la dernière ligne (supérieur à celle attendue), cela s'explique par la quantité de mémoire alloué lors du calcul des distances. En effet, toutes les distances entre chaque paire de sommet doivent être stockées pour être accessibles par la suite. Pour cela, une map est actuellement utilisée, ce qui n'est optimal ni en temps d'accès ni en mémoire occupée. Cependant, nous n'avons pour l'instant jamais eu besoin de calculer des similarités pour des graphes aussi grand, ce point n'a pas été encore optimisé dans le plugin Tulip. Pour information, le calcul d'une similarité avec coefficient sur un graphe de 10000 sommets consomme actuellement 3,3Gio de mémoire vive, contre une consommation de 1 Mio pour le calcul classique (sans coefficient).

2.6 Le dessin de graphe, un cas d'utilisation complet

Nous avons vu à travers ce chapitre différents aspects du domaine du dessin de graphe. D'un côté, la première contribution présentée, GaGEM, permet la génération d'un layout à partir d'un

graphe et de paramètres pour chacun de ces sommets. De l'autre côté, de nombreuses métriques permettent de caractériser les graphes (à travers les métriques topologiques) et des layouts (à travers les métriques graphiques).

Dans le cadre de la solution proposée dans cette thèse, l'ensemble de ces éléments vont être utilisés par un système basé sur algorithme génétique pour produire différents layouts pour un même graphe. Ce système utilisera l'ensemble des métriques topologiques pour caractériser un graphe fournit en entrée. Il fournira ensuite à GaGEM des jeux de paramètres pour les différents sommets, qui produira alors un layout pour ce graphe à partir de chaque ensemble de paramètres. Ces différents layouts seront ensuite évalués à l'aide des métriques graphiques. La métrique de similarité permettra quant à elle de comparer différents résultats, ce qui sera des plus utile.

Le chapitre suivant va se concentrer sur une présentation générique des algorithmes génétiques. Les bases de fonctionnement de notre système seront présentés en même temps. En effet, ce dernier a été prévu pour être utilisé pour de très nombreux cas d'utilisation et est donc indépendant du dessin de graphe.

Le chapitre 4 présente quant à lui l'utilisation des algorithmes génétiques dans le cadre précis du dessin de graphe. C'est là que seront utilisés l'ensemble des outils liés au graphes qui viennent d'être présenté et leur rôle précis dans notre système.

Chapitre 3

Algorithme génétique

3.1 Introduction

Les algorithmes génétiques sont généralement présentés comme une méta-heuristique d'optimisation. Heuristique d'optimisation, car leur objectif est de rechercher un extremum d'une fonction donnée, cette recherche pouvant aboutir à un extremum local. Méta car ils se basent fortement sur des tirages aléatoires pour le guidage de cette recherche. Toutefois, cette caractérisation reste assez large et de nombreuses méthodes très différentes existent pour réaliser de telles optimisations. Les algorithmes génétiques s'inspirent du processus évolutionnaire des génomes naturels, desquels ils tirent leur nom.

Cette classification des AG en tant qu'outils d'optimisation ne rend toutefois pas compte de l'ensemble de leurs possibilités. En effet, ils peuvent aussi être utilisés pour réaliser des explorations ouvertes de vastes espaces. Leur objectif étant alors d'identifier des ensembles de points intéressants, et si possible de construire une base de connaissances de ces points. En effet, dans le cadre de problèmes généraux, une solution peut présenter un intérêt très variable en fonction des besoins de l'utilisateur. De ce fait, l'espace des solutions peut ne pas présenter un extremum unique qui serait l'objectif de toute recherche. Dans le cadre de telles recherches, il est aussi régulièrement possible d'utiliser des résultats issus d'explorations précédentes afin de faciliter et d'accélérer l'exploration de l'espace.

C'est dans cette optique qu'a été développé le système d'AG mis en place dans le cadre de cette thèse. De plus, comme nous le verrons par la suite, ce type de système s'adapte particulièrement bien à une utilisation interactive des AG. Cette approche est aussi relativement bien adaptée au dessin de graphe, car le concept d'instances similaires nombreuses correspond très bien au cas du dessin de graphes. En effet, de nombreux graphes différents auront à être dessinés. Cependant, ces différent graphes peuvent présenter des points communs, et donc partager une même manières d'être dessinés. . De ce fait, il peut être très intéressant de pouvoir réutiliser les résultats obtenus lors d'exécutions précédentes. Toutefois, l'utilisation des AG dans le cadre du dessin de graphe sera uniquement abordée à partir du chapitre suivant.

3.2 Principe de base

Le principe de base des algorithmes génétique est relativement simple. Son objectif est de trouver une solution optimale à un problème donné. Le problème est généralement caractérisé par un algorithme, qui prend en entrée un certains nombre de paramètres, et qui fournit en résultat un nombre qui correspond au score du jeux de paramètres utilisé. Le principe de l'AG

sera de travailler sur un ensemble de jeux de paramètres, chaque jeu étant appelé un « génome ». Cet ensemble est tout d'abord initialisé aléatoirement, puis amélioré progressivement par un processus d'évaluation des génomes de la population, de sélection des meilleurs individus et de génération d'un nouvel ensemble de génomes. Chaque itération d'un cycle est nommé génération de l'AG. La génération des nouveaux individus est réalisés généralement à l'aide de deux opérateur génétiques : la mutation et le croisement. La mutation consiste à modifier aléatoirement quelques valeurs dans un génomes, le croisement à la combinaison de deux génomes (nommé le père et la mère) en prenant chaque paramètre dans l'un ou l'autre des génomes.

Le bloc de pseudo-code suivant présente ce principe de fonctionnement de manière plus formelle :

Algorithme 8: Déroulement simplifié d'un algorithme génétique

```
Entrées : Fonction d'évaluation évaluateur : Méthode qui permet d'associer un score à
          chaque génome
Entrées : Entier générationMax : Nombre de génération à effectuer
Entrées : Entier taillePopulation : Taille de la population
Résultat : Génome résultat : Meilleur génome trouvé par l'AG
début
    population ← Tableau de génomes aléatoires de taille taillePopulation
    scores ← Tableau de taillePopulation scalaires
    résultat ← Génome vide
    meilleurScore ← −∞
    génération ← 0
    répéter
        generation ← génération + 1
        évaluateur.évaluerPopulation(population, scores)
        /* Cette fonction évalue chaque génome de la population et place le
           score correspondant dans le tableau scores                        */
        meilleursGénomes ← SélectionMeilleursGénomes(population,scores)
        /* De nombreuses méthodes de sélections sont possibles, et seront
           décrites plus tard                                                */
        si scores.meilleurScore() > meilleurScore alors
            meilleurScore ← scores.meilleurScore()
            résultat ← population[scores.indexMeilleurScore()]
        population ← GénérerNouveauxGénomes(meilleursGénomes)
        /* Cette fonction réalise une combinaison de croisements et de
           mutations sur les génomes contenus dans meilleursGénomes          */
    jusqu'à génération ≥ générationMax
    retourner résultat
```

De manière générale, les génomes s'expriment sous la forme d'une chaine de caractères ou d'un tableau de nombres. Cela permet d'avoir une expression simple des opérateurs génétiques. Dans le cas d'un tableau de nombres, la mutation consiste simplement à choisir aléatoirement un des nombres et de le modifier. Pour le croisement, le principe consiste généralement à choisir un point de pivot, et de prendre les éléments avant le pivot chez le père et ceux après le pivot chez la mère. Un opérateur d'élitisme, qui conserve les meilleurs génomes tels quels est souvent régulièrement utilisé (car il permet d'éviter toute diminution du meilleur score de la population).

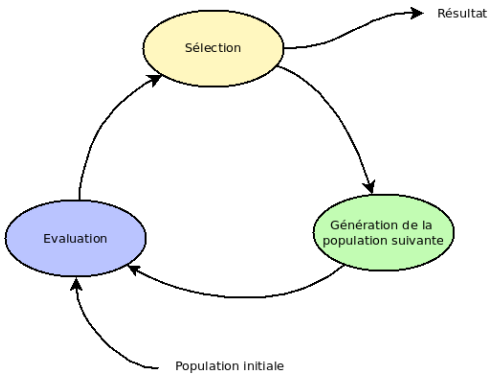


FIGURE 3.1 – Un schéma simplifié du fonctionnement d'un algorithme génétique

La fonction **évaluateur** est souvent décomposé en deux modules. Le premier consiste en un algorithme qui utilise les données du génome comme paramètres d'entrée et qui fournit un résultat complexe (par exemple un layout dans un dessin de graphe). Le deuxième module est alors une fonction de score qui attribue un score scalaire au résultat fournit par l'algorithme. Pour poursuivre l'analogie avec le dessin de graphe, cette fonction de score peut se ramener à la mesure d'un critère esthétique sur le dessin, le nombre de croisement d'arêtes par exemple. Le score attribué à chaque génome est nommé « fitness » du génome.

Pour la sélection des meilleurs génomes, de nombreuses méthodes sont possibles. Une des plus courante est la sélection par roue de la fortune. Le principe consiste à associer à chaque génome une probabilité de sélection égale à $\frac{score\ du\ genome}{somme\ des\ scores}$. De manière imagée, cela revient à fournir à chaque génome un angle proportionnel à son score sur une roue de la fortune et de sélectionner le génome pointé après avoir lancée la roue. D'autres méthodes comme les sélections à bases de tournois sont aussi régulièrement utilisées.

Ce schéma reste très général, et le déroulement d'un AG dépend énormément de la manières dont ces différents points sont paramétrés. La sélection des génomes et les modalités de génération des nouvelles populations détermine complètement la manière d'explorer l'espace. Comme nous le verrons par la suite, la structure de l'AG a une très forte influence sur le résultat, et la structure « optimale » dépend énormément de l'objectif final de l'utilisateur.

La figure 3.1 présente un schéma simplifié du fonctionnement d'un algorithme génétique.

3.2.1 État de l'art

Les premiers travaux qui ont abouti au concept d'algorithme génétique ont été menés par un groupe de chercheurs dirigés par J. H. Holland à partir des années 1960. Le groupe travaillait sur le concept de programme adaptatif et sur le principe du recuit simulé, qui permet d'améliorer une solution à un problème donné en lui faisant subir des modifications successives et en conservant celles qui améliorent la solution. Une mutation qui diminue le score a toutefois une chance d'être acceptée afin de faciliter la sortie de maximum locaux. Le passage du recuit simulé aux algorithmes génétiques s'est fait en utilisant une population de solutions au lieu d'un seul individu et en ajoutant le principe du croisement qui permet d'obtenir un nouveau candidat en mélangeant deux autres bons candidats. Une étape intermédiaire a été l'utilisation d'une population large pour le recuit simulé et d'une sélection des meilleurs candidats comme base des suivants.

La première présentation formelle des algorithmes génétiques apparaît dans le livre de J H Holland de 1975 [49], dans lequel le principe de base de l'utilisation de génomes et de gènes et des opérateurs génétiques au sein d'un système adaptatif est mis en place. Holland pousse très loin l'analogie avec les systèmes d'évolutions naturels et présente de nombreuses idées issues de cette analogie. Durant les années suivantes, plusieurs travaux ont permis d'étayer et de compléter le modèle de base, en particulier les thèses de De Jong en 1975 [22] et de Goldberg en 1983 [39] réalisées sous la direction de Holland qui ont apporté de nombreux éléments et les travaux de Grefenstette en 1986 [45] qui portent sur le paramétrage des différents modules constituants les AG. Toutefois, c'est après la publication d'un livre de Goldberg en 1989 [40] reprenant l'ensemble des éléments mis en place autour des AG que ces derniers vont se diffuser plus largement dans la communauté.

À partir de cette date, de très nombreuses applications des algorithmes génétiques à des domaines très différents vont voir le jour. Dans le cas du dessin de graphe, les premières applications ont été présentées par Groves et Michalewicz en 1990 [46] et Markus en 1991 [63]. La bibliographie de ce domaine d'application sera cependant abordée plus en détail dans le chapitre suivant, dédié à l'utilisation des algorithmes génétiques dans le cas précis du dessin de graphe.

De nombreuses améliorations ont par la suite été apportées aux modèles des AG. Toutes ne pourront bien évidement pas être cités, et le lecteur pourra se tourner vers de nombreux livres écrits sur le domaine, comme ceux de Michalewicz dont la 3e édition est parue en 1996 [69] ou de Melanie, paru la même année [68].

Les paragraphes suivants vont chacun présenter une amélioration des AG proposée par la communauté. Elles ont chacune eu une influence particulière sur le système mis au point durant cette thèse.

3.3 Améliorations proposées par la communauté

3.3.1 Initialisation contextuelle

L'initialisation contextuelle d'un algorithme génétique (plus connue sous le nom de case-based initialization) consiste à initialiser la population de départ à l'aide de résultats obtenus auparavant. Le principe consiste à enregistrer les meilleurs résultats et de leur associer des informations caractérisant la situation dans laquelle ils ont bien fonctionné. Par la suite, il est possible de sélectionner un ensemble de génomes qui sont susceptibles d'être intéressants pour une nouvelle instance du problème. Cette sélection se faisant en fonction du contexte de la nouvelle instance.

Ce principe provient d'un rapprochement des algorithmes génétiques avec des concepts tirés du raisonnement par cas (case-based reasonning), dont Aamodt et Plaza ont proposé un intéressant survol en 1994 [1]. Le principe consiste à dire qu'une solution adaptée dans un contexte peut être intéressante aussi dans des contextes similaires. On entend par contexte d'une évaluation une caractérisation de l'environnement dans lequel se déroule cette évaluation. Par exemple, dans le cadre du dessin de graphe, le contexte correspond au graphe qui est dessiné. L'idée revient alors à dire que des paramètres permettant de bien dessiner un graphe permettront probablement d'obtenir de bon résultats pour des graphes similaires.De ce fait, il est possible d'utiliser ce principe lors de la génération de la première initiale d'un AG. C'est l'idée proposée par Ramsey et Grefenstette en 1993 [80]. Cette idée sera par la suite reprise et améliorée par Louis *et al.*, qui proposeront d'abord une application concrète du principe en 1997 [61] puis une généralisation en

permettant l'injection d'anciens résultats au fur et à mesure de l'algorithme génétique en 2004 [62].

Ce principe a aussi mené à une réflexion sur l'hybridation des populations. En effet, injecter ainsi des génomes venant d'autres exécutions peut parfois aboutir à des résultats décevants, car les nouveaux génomes ainsi injectés sont trop différents pour fonctionner correctement avec ceux déjà présents. Il est donc nécessaire de contrôler ces mélanges afin de ne pas casser la dynamique d'exploration de l'AG. Par exemple, il est intéressant que les génomes injectés aient été obtenus après un temps de calcul similaire à celui de l'AG courant. Plus simplement, lors d'une injection qui se déroule après la 10^e génération de l'AG, il est déconseillé d'injecter des génomes obtenus après 100000 générations. Cela permet d'éviter d'injecter des génomes trop spécialisés dès le début, ce qui provoquerait une convergence prématurée vers ces derniers. Ces idées ont été développées plus en détail dans un article de Ashlock *et al.* en 2008 [5]. Le principe proposé consiste à injecter régulièrement des génomes obtenus précédement dans l'algorithme, tout en veillant à ce que la complexité des génomes injectés soit cohérente avec l'état courant de l'AG.

Des problématiques similaires avaient été approchées auparavant dans le cadre des algorithmes génétiques en îles dans lesquels plusieurs sous population servent de base à l'algorithme génétique et s'échangent des individus ponctuellement. La question a été abordée plus en détail dans le papier de Whitley *et al.* en 1997 [96].

Il est intéressant de noter que l'ensemble des idées de cette partie sont presque directement issues de l'observation du comportement des algorithmes génétiques naturels, dans lesquels la taille de la population et le fait de mélanger des populations « indépendantes » permet d'obtenir des résultats très intéressants (et généralement une spéciation plus forte et plus rapide).

3.3.2 Dispersion contrôlée de la population

Une des grandes problématiques des algorithmes génétiques concerne la convergence de la population, et en particulier une convergence prématurée. Lors d'une convergence extrême, l'ensemble des génomes d'une population sont identique et de ce fait, les croisements n'ont plus aucun effet. Cela peut se produire très rapidement lorsque la sélection est très stricte (par exemple en ne sélectionnant que les 5 meilleurs génomes) et que les mutations sont trop peu courantes ou trop faibles. Ce type d'AG se bloque alors très facilement dans des maximums locaux. C'est une problématique au centre de nombreux papier sur le domaine [3, 97, 67].

Une des solutions consiste à maintenir une répartition uniforme des fitness dans la population. Pour cela, lorsqu'un nouveau génome est évalué, il n'est ajouté à la population qu'en remplacement de celui qui a obtenu le score le plus proche de lui (et de préférence un score inférieur). Cela permet d'empêcher strictement toute convergence trop forte de la population. Une limite de cette approche concerne le fait que deux génomes qui obtiennent exactement le même score peuvent quand même être très différents. De ce fait, en empêchant toute convergence, cette méthode limite aussi d'une certaine manière la diversité de la population.

Cette idée s'inspire du principe de crowding [36], qu'il étend en se focalisant sur la sélection et la suppression de génomes qu'ils adaptent afin de toujours préserver une bonne diversité.

3.3.3 Algorithmes génétiques interactifs

Une évolution importante des AG a été l'ajout d'une composante interactive dans leur comportement. Cela se fait en général en demandant à l'utilisateur de réaliser lui-même l'évaluation des différents candidats. Ce changement ouvre de nombreuses possibilités autant au niveau de l'efficacité de l'exploration que de la variété des espaces qu'il est possible d'explorer [95, 88]. En effet, l'être humain est capable d'évaluer relativement facilement toute sorte de candidats pour lesquels il serait difficile de réaliser une évaluation automatique. De plus, il se révèle particulièrement doué pour déterminer des objectifs intermédiaires intéressants et permettant d'avancer rapidement vers un résultat final [98].

Le domaine pour lequel ces AG ont été les plus utilisés est sûrement l'aide à la créativité. Que ce soit dans un cadre strictement informatique, avec l'assistance à la réalisation d'interface graphique [64], ou des domaines plus liés à la vie réelle, comme la mode [54]. Les algorithmes génétiques sont en effet relativement efficaces pour générer des « idées » de design en grand nombre. Ces dernières sont ensuite sélectionnées par l'utilisateur qui peut par là même orienter la suite de l'exploration.

Une autre utilisation des algorithmes génétiques a été la recherche d'images ou d'éléments qu'il est difficile d'indexer textuellement. Un article publié par Cho et Lee en 2002 [19] propose un système de recherche d'images au sein d'une base de données à l'aide d'un AG interactif. De manière analogue, le système PicBreeder [84], proposé par Secretan *et al.*, permet quant à lui de générer des images en utilisant un type d'algorithmes différents des AG (NEAT, un réseau de neurones évolutif), mais qui s'appuie sur le même type d'interaction que le ferait un AG.

Plusieurs autres systèmes interactifs dédiés spécifiquement au dessin de graphes seront présentés dans le chapitre suivant. Cependant, tous les systèmes interactifs partagent une même contrainte : la fatigue de l'utilisateur. En effet, les êtres humains sont relativement limités en termes de temps et de concentration qu'ils peuvent dédier à une tâche d'évaluation (en opposition avec une machine qui peut à priori le faire indéfiniment). Or dans le cadre d'un algorithme génétique, le nombre d'évaluations devant être réalisées est très important (il peut atteindre plusieurs millions dans certains cas). Le nombre d'évaluation nécessaires étant obtenu en multipliant simplement le nombre de générations par la taille de la population principale. Il est donc vital d'adapter les AG à cette contrainte humaine. La première solution consiste à limiter fortement le nombre d'évaluations en réduisant la taille de la population et le nombre de générations, mais cela impacte fortement la part de l'espace explorée. Une autre possibilité consiste à demander à l'utilisateur d'évaluer d'un petit échantillon des candidats et en utilisant une évaluation automatique pour les autres.

3.3.4 La curiosité

La curiosité est dans la nature un des moteurs les plus forts de l'apprentissage. L'identification et l'exploration volontaire de situations inconnues permet d'accélérer énormément la découverte d'un domaine. C'est en grande partie autour de ce principe que s'est basé le travail de Stanley et Lehman [60, 59, 58]. Le principe de leur idée consiste à évaluer les nouveaux candidats d'un système évolutif en fonction de la nouveauté qu'ils apportent au système. Cela mène à l'apparition de comportements complexes qui permettent généralement d'aboutir à des solutions intéressantes.

D'une autre manière, Goldberg a initialement donné une très forte importance à l'exploration de nouvelles solutions dans sa description initiale des algorithmes génétiques. Il définit en effet un opérateur de croisement bien plus complexe que celui généralement considéré. Son idée de base était en effet de construire progressivement des groupes de gènes fonctionnant efficacement ensemble. L'objectif de l'opérateur de croisement étant alors de préserver ces groupes tout en essayant progressivement d'en faire apparaître de plus complexes. Toutefois, cette idée n'a pas été retenue dans l'archétype des algorithmes génétiques à cause des problèmes de mises en œuvre qu'elle soulève et de la complexité du processus de sélection.

Ces deux manières de simuler une « curiosité », bien que très différente dans leur modalité d'application tente d'atteindre un même but qui est la lutte pour un maintien d'une diversité optimale tout en explorant régulièrement de nouvelles zones de l'espace. Cette idée à eu un rôle central dans la structuration du système que nous avons mis en place.

3.4 Modifications apportées à l'algorithme génétique

Comme nous venons de le voir, les algorithmes génétiques correspondent finalement à un modèle de base pouvant être enrichi de multiples manières. Dans son livre « Genetic Algorithm + Data Structure = Evolution Programs » [69] Michalewicz défend fermement l'idée que l'association de ce principe de base avec des données riches et structurées peut ouvrir la voie à des applications extrêmement efficaces.

Une problématique centrale, déjà approchée par d'autres chercheurs, mais relativement rare lors de l'utilisation et la conception d'algorithmes génétiques était le fait que cet algorithme serait utilisé sur de très nombreuses instances du problème. En effet, les AG sont généralement utilisés pour trouver une solution optimale à un problème donné. Nous voulions concevoir un AG qui puisse quant à lui fournir de nombreuses solutions à de nombreux problèmes très similaires (plus précisément le dessin de graphes différents associés à des besoins utilisateurs eux aussi variables). Cette variation autour d'un même problème permet un partage relativement important des résultats obtenus, mais aussi de fortes variations dans la manière de concevoir et de structurer l'algorithme. Enfin, nous souhaitions aussi pouvoir intégrer facilement des idées d'améliorations proposées par la communauté scientifique.

Cet usage des AG prend tout son sens dans le cadre de la problématique de cette thèse, assister un utilisateur novice dans le cadre. En effet, l'objectif ne sera pas simplement d'optimiser au maximum une fonction donnée mais plutôt d'explorer des zones interessantes de l'espace des dessins de graphes. Dans le cadre d'une requête donnée d'un utilisateur, il s'agira de l'espace des dessins pour un graphe donné. Mais en considérant toutes les requêtes qui pourront être faites à notre système, l'espace exploré devient celui des dessins possibles pour un grand ensemble de graphes. Afin de limiter le temps nécessaire à chaque exploration, il nous semble de plus indispensable de pouvoir partager les résultats obtenus entre chaque exploration. Comme nous le verrons, cette limitation du temps d'exécution est nécessaire dans le cadre d'AG interactifs.

Cependant, afin de correctement s'adapter aux besoins des utilisateurs, toutes ces exécutions ne doivent pas se dérouler de la même manière. Pour deux besoins différents, une manière d'explorer sera possible très adaptés au premier (adaptées dans le sens de fournir des résultats satisfaisant rapidement), alors qu'elle ne sera que très peu utile dans le cadre du deuxième besoin. Dans le cadre des AG, le « guidage » d'une exploration est déterminé par la structure de

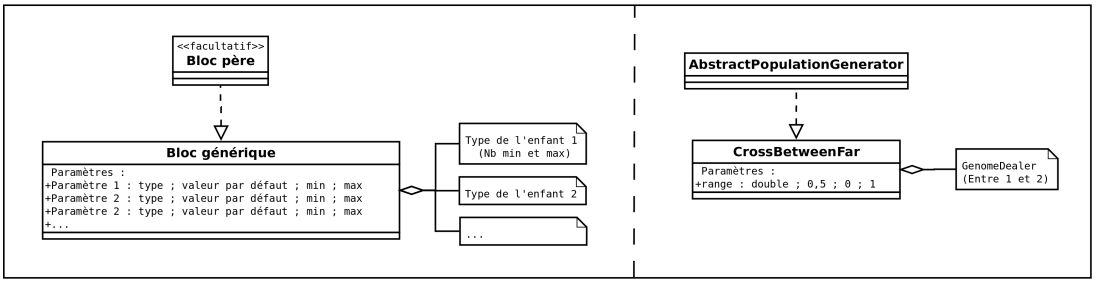


FIGURE 3.2 – La structure générique des blocs utilisés par notre système et un exemple issu d'un bloc réel

l'AG. Cette structure correspond globalement à la manière de déterminer comment sont générés les populations au cours de l'algorithme. En effet, à chaque génération une nouvelle population de génome est générée puis évaluée. Dans le cadre classique des AG, cette génération est réalisée à l'aide des deux opérateurs génétiques (croisement et mutation), et la sélection des génomes utilisés par ces opérateurs se fait au sein de la génération précédente. Mais chacun de ces processus peut se faire de très nombreuses manières, surtout lorsque le génome utilisé correspond à une structure de données complexe. De plus, le fait de vouloir réutiliser des résultats intermédiaires entre les différentes exécutions augmente considérablement la complexité que peut atteindre une structure.

Plusieurs bibliothèques d'AG sont déjà disponibles sur internet. Cependant aucune d'entre elles ne nous a permis de répondre satisfaisant à cette volonté de pouvoir définir précisément la structures de nos AG, tout en permettant des interactions simples avec un utilisateur novice. Nous avons donc décidé de mettre au point un nouveau système qui permettrait de répondre précisément à ces contraintes.

Afin d'obtenir une structure malléable et fortement configurable, nos structures d'algorithme génétique se construisent par bloc. Un bloc est défini par son nom, ses paramètres et les enfants qu'il peut avoir. Il est aussi possible d'ajouter un lien d'héritage entre deux blocs et de définir un bloc comme abstrait (ne pouvant être utilisé tel quel). Les blocs abstraits ne sont utilisés que pour décrire les enfants autorisés d'un bloc donné, et les emplacements correspondants devront être remplis par des blocs héritant du bloc abstrait en question. Comme nous le verrons aussi par la suite, ce concept de bloc transparaît directement dans la mise en œuvre de notre système. Chaque bloc correspond généralement à une classe vérifiant certaines caractéristiques communes. Cela permet en particulier de facilement définir de nouveaux types de blocs. La figure 3.2 présente schématiquement la structure d'un bloc générique, et montre un exemple de bloc réel, **CrossBetweenFar** qui représente l'opérateur de croisement classique. Ce dernier a un paramètre (range) dont l'utilité sera vue plus tard, et accepte 1 ou 2 **GenomeDealer** comme enfant. S'il n'a que un fournisseur de génomes, il réalise un croisement entre 2 génomes de ce fournisseurs, s'il en a deux, tous les croisement sont effectués entre un génome de chaque fournisseur (reproduction sexuée).

Les éléments les plus complexes sont les enfants d'un bloc. Ces derniers peuvent être obligatoire ou non et être limités en nombre (par un maximum, et/ou un minimum définis pour chaque type d'enfant).
Le bloc de base de toute structure d'AG est un bloc **geneticAlgorithm**, qui accepte les enfants suivants :

– Un bloc **abstractEvaluator**, qui correspond au module d'évaluation (permettant d'inter-

prêter les génomes et de fournir un score pour chacun d'eux)

- Un bloc d'initialisation de la population (qui permet de générer la première population de l'AG).

- Au moins un bloc **step**, qui permet de déterminer le fonctionnement de l'algorithme génétique durant une étape. Le bloc **step** permet de définir comment la population de la génération suivante est générée, des paramètres d'évaluation, une condition de fin d'étape et des éléments liés à la gestion des populations annexes (qui seront abordées plus précisément dans la partie 3.4.2).

- Des blocs de populations annexes. Pour ces derniers aucune contrainte sur le nombre n'est présente. Ces blocs et leur utilité seront abordés plus en détail dans la partie consacrée 3.4.2.

Description XML de l'AG

En terme de représentation, nous avons choisi d'utiliser les fichiers XML comme support des structures d'AG. En effet, la syntaxe XML inclut un grand nombre de fonctionnalités que nous utilisons pour définir notre structure. Le premier étant bien évidement la structure hiérarchique qui lie les différents éléments.
Un grand intérêt du XML est qu'il permet d'être facilement lu et modifié manuellement tout en autorisant aussi des générations et des manipulations automatiques. De plus, un grand nombre de parsers sont disponibles dans de très nombreux langages de programmation.

Cependant, un problème important de notre système est le grand nombre de blocs différents et leurs paramètres. Il est en effet très délicat de se rappeler du nom exact de chaque bloc et de ses différents paramètres. Actuellement, 57 blocs sont liés au noyau de l'algorithme génétique (indépendants de tous cas d'utilisation) et 21 sont dédiés au cas d'utilisation spécifique du dessin de graphe. Un format de description des blocs a donc été mise au point afin de permettre le renseignement de leurs différentes caractéristiques (nom, nombre d'enfants, nombre de paramètres, type, valeurs minimale, maximale et par défaut, héritage).

Nous avons ensuite mis au point un éditeur XML qui s'appuie sur toutes ces informations pour permettre de construire des structures d'AG facilement. Cela a été réalisé rapidement dans le cadre de ce projet, car la réalisation de structure complexe était une tâche particulièrement lourde lorsqu'elle était réalisée manuellement. Cet éditeur se base sur deux fichiers pour récupérer l'ensemble des blocs disponible, le premier étant dédiés aux blocs liés au coeur de l'algorithme génétique, et le deuxième ceux spécifiques au cas d'utilisation du dessin de graphes. Quelques détails concernant cet éditeur sont fournis dans le chapitre 5 dédié à une rapide description des solutions logicielles mises en œuvres pour répondre aux différents besoins identifiés au cours de cette thèse.

Extensibilité de l'AG

Un élément important lors de la conception de ce système a été le fait que de nouvelles méthodes associées aux algorithmes génétiques apparaissent régulièrement. Chacune s'adaptant plus ou moins bien à un cas d'utilisation donné. De ce fait, il n'est pas possible de mettre au point un système qui dès le départ contiendrait toutes les techniques d'exploration pouvant être utilisées pour tous les cas d'application du système. De ce fait, nous avons fait en sorte que tout type de

module puisse facilement être ajouté par la suite par un nouvel utilisateur du système.

Ainsi, la plupart des blocs (et en particulier, tous les blocs abstraits) peuvent et ont été conçus afin de permettre à un utilisateur expert de les étendre afin d'ajouter des comportements ou des fonctionnements à l'AG. De plus, cette modularité a été poussée à un point très avancé, via en particulier l'ajout de modules génériques qui sont régulièrement appelés par l'AG à différents moments. Aucune tâche particulière n'est prévue pour ces modules, leur seul objectif est de permettre à l'utilisateur d'ajouter le traitement qu'il souhaite effectuer. De plus, d'autres modules génériques sont associés à une tâche précise, par exemple le bloc **abstractTerminator** permet de définir comment se termine une étape. Il laisse toute liberté à l'utilisateur quant à la manière de déterminer cela, toutefois, son unique but consistera à déterminer quand une étape doit se terminer. Des effets de bords sont bien évidemment possibles lors de cette vérification, ce qui permet aussi de réaliser une autre action que le seul choix de terminer ou non l'étape en cours.

Cette capacité d'évolution associée au mécanisme des populations annexes (qui sera présenté dans les paragraphes suivants) offre une immense liberté quant à la manière de diriger et d'orienter l'exploration réalisée par l'algorithme génétique. De plus, le caractère extensible de cette structure permet d'ajouter facilement de nouveaux mécanismes développés par la communauté.

Définition des scores : Les critères

De très nombreux modules nécessitent de manipuler des valeurs numériques. Que ce soit dans la manière de définir le score d'un génome, un seuil utilisé par un filtre, ou une manière de choisir un génome, de très nombreux paramètres prennent une forme numérique. Dans certains cas, ces paramètres sont de simples constantes définies pour l'ensemble de la durée de l'AG. Pour d'autres paramètres une telle valeur constante est inenvisageable. Pour la définition du score d'un génome par exemple, un paramètre constant serait strictement inutile. Dans ces cas là, il est nécessaire de pouvoir facilement manipuler des valeurs numériques exprimés sous la formes de fonction.

Un module de base a été défini afin de répondre à cette problématique : l'**AbstractCriterion**. Ce module abstrait permet de fournir une valeur calculée à partir du résultat de l'évaluation d'un génome. Plusieurs opérateurs offrant une grande liberté d'expression (opérations classiques, puissance, minimum, maximum et moyenne) ainsi que des valeurs constantes ou aléatoires sont fournis. De ce fait, il suffit pour chaque cas d'utilisation du système de mettre en œuvre des modules servant de feuilles à l'arbre d'opération. Des modules plus complexes, les filtres, sont aussi disponible. Ils fonctionnent soit comme des diodes (transmission du signal sur lequel s'applique le filtre sur une certaine bande), ou comme des transistors (transmission d'un autre signal).

La première utilisation de ces critères est la définition des différents scores manipulés par l'AG. Les exemples d'exécutions fournis dans le chapitre suivant montrent de nombreux usages des différents opérateurs disponibles.

Suivis des exécutions

Ces possibilités de configuration de l'AG par bloc offrent d'innombrables possibilités à l'utilisateur. Cependant, le paramétrage précis de l'algorithme devient relativement délicat, surtout dans le cadre de l'utilisation de blocs pouvant avoir des interactions complexes entre eux. Nous avons pour cela mis en place un grand nombre d'élément de surveillance du déroulement de l'algorithme via un système de monitoring des exécutions. Deux types d'informations peuvent

être logguées durant une exécution : des valeurs numériques, qui permettent de suivre l'évolution d'un paramètre précis, et des chaînes de caractères, qui sont surtout utilisées pour afficher des messages d'information ou d'avertissement.
L'envoi de tels messages est extrêmement facile au sein du code (une simple ligne de code permet l'envoi des informations au serveur de manière automatisée, l'information apparaissant ensuite directement sous la forme d'un graphique dans l'application de monitoring). Cela permet à tout utilisateur souhaitant ajouter un bloc de suivre comment se comporte son nouveau bloc.

Cette dernière fonctionnalité ne change pas le comportement de l'AG, mais modifie fortement la manière d'appréhender sa configuration. En effet, il est possible de comparer facilement les exécutions découlant de deux paramétrages légèrement différents puis de consulter les synthèse de ces exécutions (en termes quantitatifs grâce au suivi numérique, et qualitatifs par les messages d'avertissement) afin de se faire rapidement une idée du paramétrage optimal. De plus, toutes les synthèses des exécutions sont conservées sous forme compressées, ce qui permet de nombreuses comparaisons a posteriori. Toutes les images de suivis d'exécutions présentées dans ce manuscrit sont directement issues de ce système de surveillance.

3.4.1 Définition d'étapes multiples

Un autre élément important mis en place dans le système concerne la multiplicité des étapes. Pour rappel, les étapes servent à définir la manière de générer la population de travail, des paramètres d'évolutions, et les différentes modalités de sélection des génomes.

L'utilisation d'étapes successives va permettre de définir une méthode d'exploration progressive. Un schéma classique sera par exemple de faire une première étape de longueur moyenne réalisant une exploration assez ouverte afin de trouver de multiples pistes de solutions, puis une étape de synthèse de ces solutions. Cette étape utilisera par exemple massivement des croisements sur des génomes sauvegardés lors de la première étape. Enfin, une dernière étape permettra d'optimiser localement les meilleurs solutions trouvées lors de la seconde étape. Des exemples précis d'utilisation de ce mécanisme seront proposé à la fin du chapitre suivant, dans la partie consacrée aux exemples de tâches utilisateur 4.3.1.

3.4.2 Réutilisabilité des génomes : Les populations annexes

Notre deuxième priorité lors de la conception de notre système d'algorithme génétique a été la mise en place d'un système de réutilisation des génomes. L'idée étant que dans la mesure où l'AG est utilisé dans le cadre de nombreuses instances similaires, certains résultats intéressants de certaines instances peuvent aussi être intéressants pour d'autres instances proches (en terme de contexte ou de résultat souhaité). C'est sur ce principe que se basent les recherches liées aux *Case-based reasoning*, et l'application de ce principe aux algorithmes génétiques. Plusieurs éléments ont été aussi ajoutés afin de permettre une bien plus grande liberté d'action. La possibilité de réutiliser au sein de la même exécution des génomes évalués précédemment a aussi été ajoutée. Les paragraphes suivants sont consacrés aux différentes contributions liées à la réutilisation des génomes.

Cet aspect de notre système se base sur le principe des populations annexes. Cet ajout au principe des AG représente la troisième contribution de cette thèse, et la plus importante, en particulier en ce qui concerne les AG. L'idée consiste à mettre en place des ensembles de génomes existants en parallèle de la population principale de l'AG. Les génomes sont ajoutés de plusieurs manières à la population et ont toujours été évalués avant leur ajout à la population annexe.

Une fois ajoutés, ils ne changent jamais et n'ont donc pas besoin d'être à nouveau évalués.

Plusieurs opérations sont envisageables avec ces populations annexes, qui peuvent être regroupées en trois catégories :

- L'ajout d'un génome à une population
- La sélection et l'utilisation d'un génome d'une population
- La suppression d'un génome d'une population

De plus, une population annexe peut ou non être transférée dans la base de données à la fin d'une exécution de l'AG. Cela permet de regrouper au sein d'une ou plusieurs populations les génomes que l'on souhaite réutiliser lors d'exécutions ultérieures. Les populations non transférées servent donc uniquement à améliorer le comportement d'une exécution.

D'un point de vue de « haut niveau », les populations annexes permettent globalement d'augmenter énormément la taille de la base de génomes sur laquelle travaille l'algorithme génétique, en général dans l'objectif d'augmenter la diversité des génomes manipulés par le système.

Le paragraphe suivant va tout d'abord présenter les principes de contexte et de comportement, deux informations ajoutées à chaque génome afin de faciliter leur réutilisation ultérieure. En effet, si les seules informations concernant un génome sont son score et les données qui composent le génome, il est alors très difficile de réaliser une sélection correcte des génomes susceptibles d'être efficace dans une autre situation.

Contexte et comportement

Deux informations sont toujours attachées à chaque génome lors d'une évaluation :

- Le contexte d'utilisation, qui correspond à la description de l'environnement dans lequel le génome va être évalué. Dans le cas des graphes, cela correspond par exemple à une caractérisation topologique du graphe (le contexte précis utilisé dans le cadre du dessin de graphe sera décrit plus en détail dans le prochain chapitre). L'objectif du contexte est d'essayer de regrouper un maximum de données sur les paramètres de l'environnement qui ont une influence sur l'évaluation du génome.

- Le comportement, qui tente de décrire comment le génome a réagi dans l'environnement. Cela ne correspond pas forcément au score dans la mesure où ces différentes informations ne donnent pas toujours une information sur le fait que le génome soit bon ou pas. Le comportement doit permettre d'avoir une idée sur l'action du génome, à des fins de sélection ultérieure ou pour tenter de maintenir une diversité dans la population. En effet, deux génomes ayant été évalués dans un même contexte doivent être différents s'ils ont eu un comportement différent. Un corollaire de cette propriété est qu'un génome évalué deux fois dans le même contexte doit avoir le même comportement (et de ce fait, la deuxième évaluation est strictement inutile). Cette règle est toutefois à relativiser dans le cas de l'évaluation simultanée de plusieurs génomes, qui peuvent alors avoir une influence les uns sur les autres. Dans le cadre du dessin de graphes, le comportement correspond à un ensemble de métriques esthétiques caractérisant le layout obtenu finalement. Dans un autre cas d'utilisation, comme l'apprentissage d'une démarche pour un robot bipède, là ou le score est égal à la distance parcourue par le robot, le comportement pourra correspondre au nombre de contact avec le sols, la surface et la durée de chacun de ces contacts, ou tout

autre caractérisation de l'évaluation. Le contexte doit permettre de pouvoir différencier deux génomes différents qui auraient pourtant obtenu un score similaire.

Ces termes ont été empruntés à des travaux précédents sur les AG. Celui de contexte vient du raisonnement par cas (« Case-based reasonning »), et celui de comportement vient des travaux de Lehman et Stanley sur la recherche guidée par la nouveauté [59].

Ces deux caractérisations doivent être les plus larges possible, afin de permettre des sélections ultérieures non prévues initialement. Cependant, il est préférable d'éviter d'enregistrer des données trop fortement corrélées afin de ne pas faire exploser la mémoire utilisée pour le stockage de ces informations. En effet, quelques soient les modalités de stockage (XML ou SQL) ces informations sont enregistrées avec chaque génome. Il est donc important de limiter la taille de ces informations afin de permettre l'archivage d'un grand nombre de génomes.

Ajout de génomes à une population

Deux manières permettent d'ajouter des génomes à une population :

- Les **genomeSelector** (nom exact du module abstrait correspondant dans l'AG) sont utilisés pour dire si un génome qui vient d'être évalué afin de l'ajouter à une population annexe. Au niveau de la mise en œuvre, les **genomeSelector** doivent implémenter une fonction prenant en argument un génome et renvoyant un booléen. Les **genomeFilter**, un type de bloc qui étend le concept de **genomeSelector**, permettent quant à eux de prendre en compte l'état actuel d'une population pour déterminer si un génome doit ou non être sélectionné (comme l'indique leur nom, les modules héritant de **genomeFilter** ont aussi pour but de filtrer une population, ce point sera cependant abordé plus loin).

- Les **abstractSavedPopulationInitializer** qui permettent d'initialiser une population avant le début de l'algorithme génétique, soit en récupérant un ensemble de génomes depuis la base de données, soit depuis un fichier XML, un format de stockage souvent utilisé pour charger un échantillon pré-construit de la base de données.

Plusieurs modules de sélections ont été mis au point, certains s'inspirant directement de principes mis en place par la communauté, comme par exemple une recherche guidée par la curiosité ou des évaluations multi-objectifs. Le principe de fonctionnement de tels modules sera abordé plus tard, une fois les différents aspects de l'utilisation des populations annexes décrits entièrement.

Filtrage des génomes obsolètes

Afin de ne pas conserver dans les populations annexes des génomes inintéressants durant trop longtemps (par exemple des génomes enregistrés durant le tout début de l'AG), il est important de filtrer ces populations.

Ce sont les blocs héritant du module **AbstractPopulationFilter** qui sont dédiés à cette tâche (comme nous l'avons vu, ils peuvent aussi être utilisés pour sélectionner des génomes intéressants). Le principe consiste à faire un passage régulier sur chaque population afin de vérifier que chaque génome doit effectivement être conservé ou non dans la population. La régularité de ces vérifications dépend du niveau associé au filtre :

- 0 pour un filtrage à chaque fin de génération

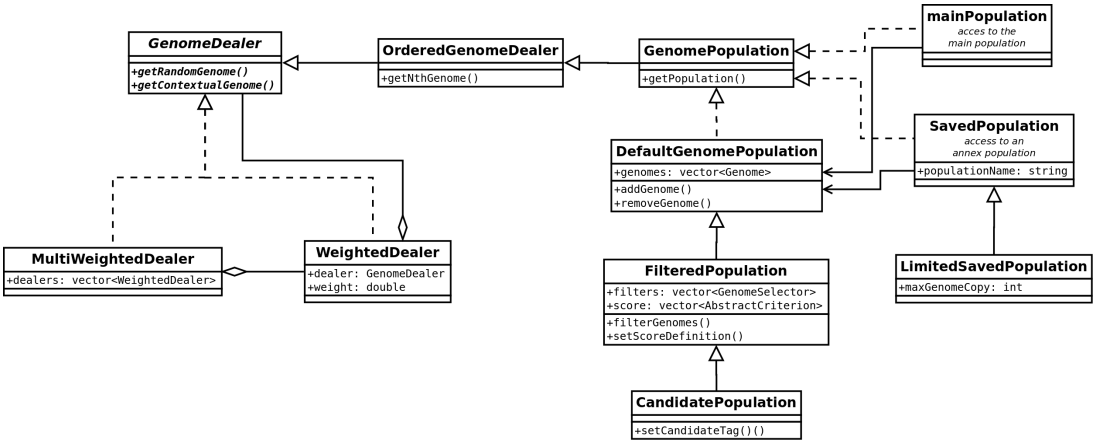


FIGURE 3.3 – L'arbre d'héritage du module **GenomeDealer**

– 1 pour un filtrage à chaque fin d'étape
– 2 pour un filtrage à la fin de l'AG.

Ce dernier niveau de filtrage est surtout dédié au nettoyage d'une population juste avant son enregistrement sur la base de données. Cependant, un usage très détourné des filtres permet d'obtenir un effet de bord (comme l'enregistrement au format XML ou l'affichage de certaines données sur la sortie standard) sans pour autant avoir d'impact sur la population (le filtre répond toujours qu'il faut conserver le génome et réalise simultanément son effet de bord). Ces filtres particuliers sont généralement appliqués uniquement à la fin de l'AG afin de ne pas répéter inutilement leur action. Un autre type filtre permet par exemple de transférer un génome vers une autre population, soit pour le sauver dans la base de données (par exemple pour sauver les 10 meilleurs génomes de chaque population), ou comme nous le verrons plus tard, pour ajouter un génome à la liste des candidats à soumettre à l'utilisateur dans le cadre de systèmes interactifs.

Sélection et utilisation des génomes au sein de l'AG

Les populations annexes, et plus particulièrement celles qui ne sont pas transférées sur la base de données, sont utiles que si elles ont un effet sur le déroulement de l'AG. Pour cela, les génomes qu'elles contiennent doivent d'une manière ou d'une autre être réinjectés dans la population principale de l'AG. La population principale est celle qui contient les génomes qui doivent être évalués, et dont le contenu change donc complètement à chaque génération.

Une grande partie des modules sont strictement dédiés à cette tâche. Le plus simple est le **GenomeDealer** dont 9 modules héritent plus ou moins directement, 4 d'entre eux étant abstraits. Le bloc abstrait **GenomePopulation** correspond à la base de la hiérarchie des population annexes. Chacun apportant des fonctionnalités supplémentaires lors de la sélection des génomes. L'image 3.3 présente les différents modules de cette structure d'héritage.

Ces différents modules sont utilisés dans le cadre de la génération de la population de chaque étape. En effet, il est nécessaire de définir pour chaque étape comment la population de la génération suivante doit être produite. Pour cela, les différents opérateurs génétiques sont utilisés, chacun d'eux utilisant généralement un ou plusieurs fournisseurs de génome pour fonctionner. Par exemple, il est possible de choisir de générer 10% de la population en réalisant un croisement. Il faudra ensuite définir comment sont choisis les génomes qui serviront comme parents lors

du croisement. De nombreux opérateurs génétiques ont ainsi été définis, et chacun d'eux peut généralement être utilisé d'une multitude de manière en fonction des fournisseurs de génomes qui lui sont associés.

Cette hiérarchie dans les modules de sélection permet de mettre en place plusieurs types de sélections de plus en plus complexes. Le module de base propose deux types de sélection :

- Une sélection aléatoire, cette sélection peut tout de même être paramétrée afin de favoriser les génomes ayant obtenus un « bon score ». Pour cela, le principe de roue de la fortune présenté plus haut est utilisé. Il permet d'associer une chance de sélection à chaque génome proportionnelle à son score.

- Une sélection contextuelle, pour laquelle le contexte dans lequel le génome sera évalué est fourni en même temps. Cela permet par exemple d'essayer de trouver des génomes relativement bien adaptés à un contexte donné rapidement. Par contre, ce type de sélection réduit les chances de tomber sur une solution extrêmement nouvelle. Il est donc toujours important de garder un équilibre entre une exploration « guidée » et une exploration « en aveugle ».

Il est important de garder à l'esprit que cette dernière sélection contextuelle n'induit aucune contrainte sur le type de sélection effectuée. Un type de sélection pas encore mise en œuvre, mais qui pourrait toutefois se révéler intéressant serait par exemple de sélectionner des génomes n'ayant encore jamais été essayés dans un tel contexte afin de voir comment ils se comportent.

Le niveau de sélection suivant consiste en une sélection ordonnée. Il s'agira par exemple de récupérer les k meilleurs génomes de la population. Cela permet par exemple de mettre en place un élitisme fort. Les modules qui permettent une telle sélection héritent de **OrdonnedGenomeDealer**. Pour l'instant, les seuls modules héritant de ce derniers blocs sont les populations annexes. En effet, tous les modules héritant de **GenomePopulation** correspondent à une population annexe, ou à un module permettant d'accéder à une telle population.

La mise en oeuvre la plus basique de ces populations est la **DefaultGenomePopulation**. Elle ne contient qu'une structure permettant de stocker des génomes, de les trier, et d'en sélectionner un aléatoirement. Il est donc possible d'ajouter ou d'enlever un génome de la population. Ce sont ces populations qui sont utilisées par l'AG lorsqu'il doit en créer une automatiquement.

Le module **filteredPopulation** est le suivant dans la ligne d'héritage. C'est lui qui permet d'ajouter des filtres et des modules avancés pour obtenir une gestion plus fine de la population. Il permet aussi de déterminer comment se déroule une sélection contextuelle à l'aide des index de population. Le dernier module, les **candidatePopulation** permettent de regrouper les génomes à soumettre à l'utilisateur dans le cadre de système interactifs.

Index de population

Comme nous l'avons vu, il existe de nombreuses manières de sélectionner des génomes pour les fournir à un opérateur génétique. Afin de ne pas avoir à redéfinir une population par type de sélection, cette tâche a été dédiée aux modules d'index. Ces derniers sont informés lorsqu'un génome est ajouté ou supprimé de la population, et lors d'une sélection contextuelle d'un génome, la population délègue cette tâche à l'index.

Il arrive qu'un index, en plus de réaliser les sélections, soit utilisé pour fournir une évaluation.

Cela est particulièrement courant dans des explorations guidées par la curiosité. Dans ces cas là, le score correspond à la distance entre le comportement d'un nouveau génome et celui du plus proche actuellement enregistré. Ces cas sont prévus en permettant des accès transverses entre différents modules afin d'éviter de faire certains calculs plusieurs fois.

3.4.3 Exemples d'utilisation des populations

Répartition uniforme du score

Un premier module mis en oeuvre au sein de l'algorithme est le **scoreSpreaderFilter**. Il peut être utilisé pour filtrer une population annexe ou pour sélectionner des génomes à enregistrer dans une population. Son objectif est de limiter la taille d'une population tout en faisant en sorte que ses génomes soit suffisamment différent les uns des autres. Cette différence est évaluée selon une valeur fournie par un textbfAbstractCriterion, ce qui laisse donc toute liberté à l'utilisateur quant à sa définition.
La sélection se déroule donc de la manière suivante :

Algorithme 9: Méthode de sélection afin de maintenir une répartition uniforme des génomes selon le score

Données : Critère d'évaluation **critères** : Les critères d'évaluation du filtre
Données : Tableau de score **ancienScores** : Les scores des génomes déjà stockés dans la population
Données : Scalaire **delta** : Différence minimale entre 2 génomes
Données : Nombre entier **tailleMax** : Nombre maximum de génomes pouvant être stockés dans la population
Entrées : Génome **g** : Le génome à filtrer
Sorties : Booléen **sélection** : Un booléen indiquant si le génome doit être conservé
début
  $\mathbf{s_g} \leftarrow$ calculerScore(**critères**, **g**)
  $\mathbf{s_{moins}} \leftarrow$ scorePlusHautInferieur(**ancienScores**, $\mathbf{s_g}$)
  $\mathbf{s_{plus}} \leftarrow$ scorePlusBasSuperieur(**ancienScores**, $\mathbf{s_g}$)
  selection $\leftarrow$ Faux
  si s_g - $s_{moins} \geq$ ***delta*** * s_g *ET* s_{plus} - $s_g \geq$ ***delta*** * s_{plus} **alors**
    ajouterScore(**ancienScores**, s, $\mathbf{s_g}$)
    si *taille(**ancienScores**) + 1* > ***tailleMax*** **alors**
      Supprimer le moins bon génome connu
    fin
    sélection $\leftarrow$ Vrai
  fin
  retourner ***sélection***
fin

Ce filtre permet donc d'éviter l'enregistrement de génomes trop similaires au sein de la population. Deux éléments sont toutefois à considérer sur son effet sur l'AG. Le premier concerne le phénomène qui se produit lorsque la population devient pleine (nombre de génome max atteint) après une longue évolution. Dans ce cas, toutes les « cases » disponibles ont déjà été remplies. De ce fait, si l'évolution a été très progressive et qu'aucun saut n'a été fait, il n'est plus possible d'ajouter un nouveau génome à la population sans une amélioration d'au moins *delta*. La population devient alors très statique. Cependant, en début d'évolution, lorsque les améliorations

sont relativement rapprochées et qu'elles ont un impact important sur le score, de nombreux espaces sont disponibles entre les génomes successifs, et la population reste dynamique. Il est donc préférable d'utiliser ce filtre en début d'algorithme.

Cette effet de blocage d'une population peut être atténué en permettant à tout nouveau génome d'être conservé et en supprimant alors à chaque fois le génome anciennement conservé ayant obtenu le score le plus proche. Diverses méthodes de sélections sur ce principes sont étudiées plus en détails dans [51].

Comme nous le verrons dans la partie 4.3.1 dédiés à une utilisation basique de l'algorithme génétique pour une recherche rapide à partir de rien, ce filtre donne de très bon résultats lors de ces courtes phases d'initialisation. De plus, ce filtre peut être utile pour ne garder qu'un échantillon d'une population en fin d'étape.

Index de population : Kd-Tree

Le deuxième module présenté ici est le $Kd-tree$, un index de population très configurable qui permet d'organiser les génomes d'une population selon différentes dimensions. Il fournit ensuite des fonctionnalité de recherche du voisinage d'un point, soit pour réaliser une sélection contextuelle d'un génome, soit pour fournir une évaluation (dans le cas d'une recherche guidée par la curiosité).

Ce module tire directement son nom de la structure de données sur laquelle il s'appuie, le kd-tree. Il s'agit d'un type d'arbre proposé par Bentley et Friedman [33, 8]. L'objectif consiste à indexer des points dans un espace à k dimensions afin de permettre une recherche rapide des voisins d'un point (le nombre de voisins recherchés pouvant varier). De plus, il permet d'indexer des ensembles de points de densité variable sans que cela rende trop longue l'exploration d'un voisinage.

Dans le cadre du module, les différentes dimensions sont définies à l'aide d'**AbstractCriterion**, ce qui permet de facilement essayer différents type d'indexation. De plus, cela permet de regrouper certaines données au sein d'une unique dimension.

Recherche guidée par la curiosité

Plusieurs modules ont été mis en place pour permettre une recherche guidée par la curiosité. Le principe de cette recherche consiste à déterminer le score d'un génome en fonction de la nouveauté qu'il apporte au système. Pour cela, le comportement de ce génome est comparé à tous ceux des génomes évalués précédemment. La distance à son plus proche voisin (ou à ses plus proches voisins selon le paramétrage) correspond à son score.

De très nombreuses variations sont proposées autour de ce principe. Il arrive souvent qu'un score plus classique soit associé à ce critère de curiosité afin de guider un minimum l'exploration. Cela permet en particulier d'éviter que la curiosité entraîne le système vers des parties très peu intéressante de l'espace. Cependant, certains travaux mettent en avant l'avantage d'une recherche dépourvue de tout objectif concret. Un des papiers de Lehman et Stanley [59] est justement basé sur ce principe d'abandon de tout objectif concret au profit d'une curiosité stricte.

Cependant, ce type de recherche pose de nombreux problème en terme de stockage, de comparaison et d'indexation des comportements des anciens génomes. En effet, ce nombre peut très facilement grandir très vite, et si la comparaison est coûteuse, le parcours complet de cette archive peut devenir très long. De ce fait, lorsqu'aucune méthodes d'indexation n'est disponible, il est nécessaire de réaliser des aménagements particuliers pour envisager ce type de recherche.

Ces contraintes sont particulièrement fortes dans le cadre du dessin de graphe. Les méthodes envisagées pour permettre un guidage par la curiosité seront abordées en détails dans la partie 4.3.2 du chapitre suivant.

Chapitre 4

Dessin de graphe et algorithme génétique

Les deux chapitres précédents ont chacun présenté de manière indépendante la problématique du dessin de graphe et le principe des algorithmes génétiques. Ce chapitre va quant à lui mélanger ces deux sujets afin de présenter quelles possibilités offrent les algorithmes génétiques dans le cas de l'exploration des différents dessins qu'il est possible d'obtenir pour un même graphe.

Cette association a déjà été envisagée à de nombreuses reprises par des chercheurs. La première partie de ce chapitre se consacre justement à un état de l'art des applications d'algorithmes génétiques au problème du dessin de graphe. La partie suivante sera consacrée à une description précise de la manière dont nous avons réaliser ce mariage dans notre cas. La deuxième moitié du chapitre sera consacrée à 4 descriptions complètes de structures d'algorithmes génétiques, les deux premières liées à des recherches de solutions optimales pour recopier un layout, et les deux dernières utilisant des AG interactifs. Ces deux dernières structures correspondent à la réponse proposée à la problématique initiale de cette thèse, « Comment assister un utilisateur novice dans le cadre du dessin de graphe ».

4.1 État de l'art

Les premiers essais d'application d'un AG pour dessiner un graphe ont été réalisés par Grooves et Michalewicz en 1990 [46]. Leur idée consiste à placer N sommets dans les cases d'une matrice de taille $H \times W$. Chaque génome correspondait donc à une chaine numérique de longueur $H * W$. Une case vide était représentée par un 0, et les différents sommets étaient représentés par leurs identifiants (donc de 1 à N).

Leur structure imposait donc une contrainte sur les génomes, chaque nombre entre 1 et N doit être présent exactement une fois, le reste étant rempli uniquement de 0. De ce fait, leurs opérateurs génétiques ne devaient générer que des génomes vérifiant cette contrainte. Pour répondre à ce problème, ils ont mis au point de nombreuses manières de croiser ou de modifier des génomes (par exemple, une des mutations consiste à intervertir deux colonnes de la matrice). Au niveau de l'évaluation des génomes obtenus, deux critères esthétiques proposés par la communauté sont utilisés : le nombre de croisements d'arêtes et le nombre d'arêtes ne pointant pas vers le bas. Leurs essais ont été faits sur deux graphes dessinés sur une grille de 6×10.

Ils proposent aussi une deuxième méthode d'encodage des génomes dans l'article, en utilisant

cette fois une matrice de taille $2 \times N$ qui contient les coordonnées des N sommets du graphe. Dans cette deuxième version, le génome est beaucoup moins contraint, car tous les remplissages possibles de la matrice peuvent être interprétés afin de fournir un dessin. Toutefois, les graphes traités avec cette méthode restent relativement petits (dans le cadre de leurs travaux).

De nombreux autres travaux ont par la suite repris ce principe de génome encodant directement les positions des sommets [63, 65, 66, 17, 90, 2]. Cependant, de nombreux auteurs remarquent que le croisement tel qu'il est défini classiquement fonctionne relativement mal pour ce cas d'utilisation [17, 2, 29]. En effet, le fait de sélectionner la position d'une moitié des sommets dans un layout, et d'une autre moitié dans un autre peut fortement perturber les caractéristiques des layouts et le résultat peut être très décevant. La réponse généralement apportée à ce problème consiste à modifier les opérateurs de croisements, en réalisant une translation et une rotation d'un des deux layouts pour améliorer leur similarité puis en utilisant le voisinage d'un sommet pour déterminer les sommets dont les positions sont conservées chez le « père », les autres positions provenant de la « mère ». De manière similaire, Markus [63] propose de sélectionner un point dans chaque parent et de faire une exploration en largeur à partir de ces deux points, chaque zone d'influence des points déterminant de quel parent sont tirées les positions.

La problématique du temps de calcul étant particulièrement importante dans le cas des AG, des optimisations faisant appel à des algorithmes de force ont été associées à une recherche guidée par un AG. Branke [17] exécute une phase d'amélioration à l'aide d'un algorithme par modèle de force sur les meilleurs génomes de chaque génération afin d'améliorer la convergence de l'AG.

Cependant, ce type de génome présente à nos yeux un inconvénient majeur à nos yeux en termes de réutilisabilité. En effet, il est très difficile de réutiliser les résultats obtenus sur un graphe pour un autre graphe. Il faudrait pour cela qu'ils aient exactement le même nombre de sommets, mais aussi qu'ils partagent une topologie très similaire. De plus, même pour un graphe strictement identique, dont seul l'ordre des sommets aurait été modifié, le layout obtenu pourrait ne plus convenir. Or, il nous semblait indispensable de pouvoir nous appuyer sur une base de connaissances importante afin d'assister efficacement un utilisateur novice.

Des tentatives ont toutefois été réalisées pour tenter de définir des génomes différents. Par exemple, Masui, dans ses travaux de 1994 [66], utilise les AG afin d'extraire d'une série de bons et de mauvais exemples des fonctions d'évaluations de dessin. Il utilise pour cela des génomes exprimant des fonctions LISP fournissant une évaluation d'un layout. D'une autre manière, Kosak *et al.* proposent la même année [55] un système de description de layout par règle. Les génomes encodent alors une série de règles permettant de placer les sommets les uns par rapport aux autres. Ces deux travaux sont très intéressants, car ils permettent d'ajouter un fort niveau d'abstraction entre le génome et le résultat finalement obtenu.

4.1.1 Autres utilisations des AG

De nombreuses applications des AG ont aussi été réalisées dans le cadre de la représentation de certaines classes de graphes. Par exemple, dans le cas des graphes acycliques orientés (DAG), les propriétés d'ordonnancement des sommets permettent d'utiliser des génomes plus spécifiques. Les AG permettent alors généralement d'obtenir des résultats intéressants dans des délais très raisonnables. Cependant, ces applications ne correspondent pas à notre problématique, puisque nous recherchons pour notre part à traiter un maximum de graphe sans la moindre distinction.

4.1.2 Méthode d'évaluation

Même si ces différents projets n'utilisent pas la même manière que nous pour générer un layout, tous doivent tout de même évaluer ces derniers. De ce fait, il est aussi très intéressant de voir quelles modalités d'évaluations ont déjà été essayées, et quels résultats elles ont produits. La plupart des travaux réalisés reposent sur un ensemble de critères esthétiques fournis par la communauté de dessin de graphe. Certains, comme Zhang [99] utilisent une combinaison linéaire de 8 critères et laisse l'utilisateur ensuite choisir les différents coefficients. D'autres, comme Grooves [46] comparent l'amélioration simultanée de 2 critères avec une optimisation séquentielle. De manière générale, cette problématique est traitée dans l'ensemble des papiers cités dans cette partie.

En terme d'évaluation, le deuxième article de Masui [66] propose toutefois une approche très différente. L'utilisateur doit fournir au début de l'exécution de l'algorithme une série de bons et de mauvais exemples. L'AG tente ensuite de trouver une fonction d'évaluation qui correspond à cette sélection. Un candidat correspond donc à une fonction, et son score correspond à l'adéquation entre le résultat de cette fonction appliquée aux différents exemples, et les résultats attendus.

Dans notre cas, notre volonté a été de laisser une grande liberté lors de la définition de la méthode d'évaluation d'un génome. Ce système nous a permis de tester une grande variété de méthodes sans pour autant avoir à en choisir particulièrement une. Différents exemples seront donnés dans la dernière partie de ce chapitre.

4.2 Genetips : Algorithme génétique dédié au dessin de graphes

Dans le cadre de l'application du système présenté dans le chapitre précédent au dessin de graphe, nous avons réalisé certaines modifications par rapport aux différents essais réalisés précédemment par la communauté.

La première concerne les données encodées par nos génomes. Le fait de vouloir pouvoir réutiliser des génomes pour d'autres exécutions de l'algorithme génétique nous impose de fortes contraintes sur les génomes. Ensuite, nous avons vu dans la partie 3.4.2 que notre système nécessite d'ajouter un contexte d'évaluation et une description de son comportement à chaque génome. Le paragraphe 4.2.2 décrit justement les informations que nous avons décidé d'utiliser pour cela. Enfin, les différentes possibilités en terme d'évaluation mises en oeuvre pour le cas d'utilisation du dessin de graphe seront présentées.

De manière générale, ce chapitre présente une application des principes présentés au chapitre précédent dans le cas spécifique du dessin de graphe. D'un point de vue applicatif, cela correspond à la mise en place d'un logiciel, *Genetips*, qui s'appuie fortement sur une bibliothèque *genLib*. De nombreux modules au sein de *Genetips* correspondent à des mises en oeuvre de modules abstraits décrits dans *genLib*.

4.2.1 Données encodées par le génome

Comme nous l'avons vu, nous avons décidé d'ajouter une contrainte de réutilisabilité à notre génome. De ce fait, le modèle classique consistant à encoder directement les positions des sommets ne nous convenait pas. Cela impliquant une contrainte bien trop forte sur la topologie du graphe auquel ce génome peut être appliqué.

Cette contrainte mène donc rapidement à une autre : la topologie du graphe doit avoir une influence sur l'interprétation du génome. En effet, si ce n'est pas le cas, deux graphes de topologie différente devraient aboutir au même résultat. Ce phénomène est par exemple présent lorsque le génome encode directement la position des sommets. Quelques soient les arêtes du graphe, le dessin obtenu sera le même (à nombre de sommets identique).

De plus, nous souhaitions augmenter l'abstraction entre le génome et le résultat obtenu afin d'essayer d'améliorer l'efficacité du croisement. C'est dans ce cadre que la méthode *GaGEM* a été développée. L'objectif de l'AG n'était donc plus de fournir un ensemble de positions pour les sommets, mais des jeux de paramètres permettant de dessiner le graphe avec *GaGEM*.

Les premiers essais réalisés utilisaient un génome par sommet, chaque génome contenant un jeu de paramètre complet. Cependant, cela nous a posé de nombreux problèmes, en particulier liés au fait que le dessin d'un sommet (et donc le comportement d'un génome) dépendait trop des autres génomes utilisés pour dessiner le graphe (ceux des autres sommets). De ce fait, le score et la qualité d'un génome pouvaient énormément varier d'une évaluation à l'autre. De plus, la prise en compte de la topologie était relativement faible au sein du génome (puisque seuls les paramètres étaient présents).

Nous nous sommes donc orientés vers un autre type de génome. Le génome est constitué d'un ensembles de n gènes (un gène correspondant à un bloc de base du génome). Chaque gène contient une condition sur la topologie locale du graphe, et un jeu de paramètres (possiblement incomplet, certains paramètres pouvant ne pas être spécifiés). Les conditions correspondent à des opérations non linéaires se basant sur les valeurs locales des métriques topologiques (ces opérations sont décrites à l'aide des modules héritant des **AbstractCriterion**). Il est possible de voir chaque gène comme une règle qui associe à une certaine situation topologique une manière d'être dessinée. Le génome est donc constitué d'un ensemble de règles pouvant être appliquées aux différents sommets du graphe.

Voici un exemple de gène très simple :

- Condition du gène :
  $3 \times degre(v) + centralite(v) + \frac{min(degre(v), coefficient\_clustering(v))}{2}$
- Jeu de paramètres associé :
  - Masse : 10
  - Gravité : 1,5
  - Amplitude de la force aléatoire : Non-spécifié
  - Longueur d'arête : 10
  - Coefficient d'attraction personnel : 0,5
  - Coefficient d'attraction extérieur : 2
  - Coefficient de répulsion personnel : Non-spécifié
  - Coefficient ce répulsion extérieur : Non-spécifié

Lors de l'interprétation du génome, il va être nécessaire de générer un jeu de paramètres pour chaque sommet. Pour cela, les conditions de chaque gène sont calculées en fonction de la topologie locale d'un sommet. Les scores ainsi obtenus sont triés. Les gènes ayant obtenu les meilleurs score fournissent alors leur jeu de paramètres. Pour cela, tous les paramètres spécifiés dans le premier gène sont utilisés, puis le jeu de paramètres est complété par le deuxième gène et ainsi de suite jusqu'à ce que le jeu de paramètre soit complet. Cette procédure d'attribution

des paramètres est décrite par l'algorithme 10. Un exemple d'application de cet algorithme sur un génome et un graphe simples est présenté par l'image 4.1.

Une fois tous les sommets traités, GaGEM est utilisé pour calculer le layout résultant.

Algorithme 10: Procédure d'attribution de paramètres à chaque sommet à partir d'un génome

```
Entrées : Génome génome : Le génome utilisé pour dessiner le graphe
Entrées : Graphe graphe : Le graphe à dessiner
Sorties : Tableau de paramètres résultat : Les paramètres pour chacun des sommets
début
    pour tous les Sommet s : Sommets de graphe faire
        tableauScores ← Tableau de scalaires vide;
        pour tous les Gène gène : Gènes de génome faire
            s ← CalculerScore(gène.condition, s);
            /* Un score est associé à chaque condition afin de déterminer
               celle qui correspond le mieux au sommet                    */
            tableauScores.ajouterScore(s, gene.paramètres);
        Trier tableauScores par scores décroissants;
        paramS ← Jeu de paramètres vide;
        i ← 0;
        tant que paramS n'est pas plein faire
            Compléter paramS avec les paramètres de tableauScores[i];
            i ← i+1;
        résultat →ajouterParametres(s, paramS);
    retourner résultat;
```

Si certains paramètres ne sont pas définis pour un sommet, les paramètres par défaut sont utilisés. Cependant, ce cas est rarissime puisqu'il faut qu'aucun des gènes n'ait une valeur pour ce paramètre.

Une fois tous les paramètres attribués, l'algorithme *GaGEM* est exécuté afin d'obtenir un layout pour le graphe. Ce layout correspond au phénotype du génome, c'est-à-dire son interprétation dans « l'environnement » du dessin de graphe.

D'un point de vue plus symbolique, chaque génome décrit une manière de dessiner un sommet en fonction de caractéristiques topologiques. Par exemple, il serait possible de définir une règle s'appliquant aux sommets de fort degré, en leur associant une longueur d'arête idéale grande et une répulsion forte afin qu'ils aient de la place autour d'eux. Lors du croisement de deux génomes, ce sont ces règles qui sont partagées, de ce fait, le layout obtenu peut bien plus facilement hériter de certaines des caractéristiques de ses parents.

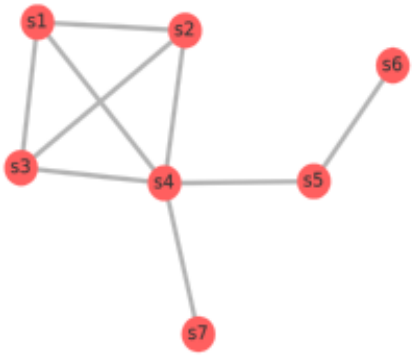
Toutefois, cette identification topologique présente un inconvénient dans le cadre de graphe dont l'ensemble des sommets présente au moins un isomorphisme non trivial. Pour rappel, cela signifie qu'il est possible d'échanger au moins deux sommets sans changer le graphe (ie, deux sommets sont strictement identiques d'un point de vue topologique).

Cela arrive par exemple lorsqu'un graphe présente une sous-clique dont un seul sommet est relié au reste du graphe. Il n'est alors pas possible de différencier avec des règles topologiques les différents sommets de la clique. De ce fait, il n'est pas possible de les différencier avec des conditions se basant sur des métriques topologiques.

Génome

Gène	A	B	C	D
Condition	3*deg+clu	-5*min(deg,cent)	deg-clu+cent	prox^2
Paramètres	Masse : 10 Gravité : 1 Shake : NA Long. Ar. : NA Attr. Per. : 1 Attr. Autres : 1 Rep. Per . : 2 Rep Autres : 3	Masse : 2 Gravité : -1 Shake : 1 Long. Ar. : 10 Attr. Per. : NA Attr. Autres : NA Rep. Per . : NA Rep Autres : NA	Masse : 5 Gravité : NA Shake : 2 Long. Ar. : 5 Attr. Per. : 2 Attr. Autres : 2 Rep. Per . : 1 Rep Autres : 1	Masse : 1 Gravité : 4 Shake : 0,5 Long. Ar. : NA Attr. Per. : 0,5 Attr. Autres : 0,5 Rep. Per . : NA Rep Autres : NA

Graphe et valeurs des métriques topologiques



	deg	clu	cent	prox
s1	3 (0,83)	1	3,125	0
s2	3 (0,83)	1	3,125	0
s3	3 (0,83)	1	3,125	0
s4	5 (2,18)	0,3	3,75	11
s5	2 (0,16)	0	3	5
s6	1 (-0,51)	0	2,25	0
s7	1 (-0,51)	0	2,625	0

Les données sont centrées réduites selon les valeurs contenues dans la base de données

deg	1	2	3	5
deg norm.	-0,51	0,16	0,83	2,18

Le score de chaque gène est calculé pour les différents sommets

	A	B	C	D
s1	3,51	-2,00	0,24	0,73
s2	3,51	-2,00	0,24	0,73
s3	3,51	-2,00	0,24	0,73
s4	6,17	-5,10	3,61	1,81
s5	-0,51	-0,81	1,44	0,02
s6	-2,54	2,57	0,02	0,73
s7	-2,54	2,57	0,39	0,73

Le jeu de paramètres d'un sommet est construit selon l'ordre des scores des gènes pour ce sommet :

s1 : A → D → C → B
s4 : A → C → D → B
s6 : B → D → C → A
...

Les paramètres attribués à s1 au final sont :

Masse : 10 (A)
Gravité : 1 (A)
Shake : 0,5 (D)
Long. Ar. : 5 (C)
Attr. Per. : 1 (A)
Attr. Autres : 1 (A)
Rep. Per . : 2 (A)
Rep Autres : 3 (A)

La lettre entre parenthèse indique le gène dont est issue la valeur

FIGURE 4.1

Les sommets $s1$,$s2$ et $s3$ du graphe de la figure 4.1 fournissent un bon exemple de sommets indentique au niveau des métriques topologiques. En effet, on observe que pour les 4 métriques topologiques, ces sommets prennent exactement les même valeurs. De ce fait, ils auront forcément les mêmes paramètres, puisque les scores de chaque gènes seront identiques.

Ce problème est présent dans le cas réel des graphes de voies métaboliques. Dans ce cas, certains sommets identiques topologiquement doivent être dessinés différemment en fonction de la molécule qu'ils représentent. Dans le cadre de l'AG, deux sommets sont identiques topologiquement si toutes leurs métriques sont égales (degré, centralité, proximité et coefficient de clustering). Dans ce cas, quelque soit la combinaison qui est fait de ces métriques, le résultat sera toujours forcément identique pour ces deux sommets. Pour pallier à ce problème, les conditions peuvent aussi se référer à l'identifiant du sommet (et donc, être utiles que pour un graphe précis), ou à des données textuelles contenues dans l'étiquette d'un sommet. Ces deux types de données ne peuvent donc pas être utilisées pour tous les graphes, mais elles permettent de mieux s'adapter à certaines familles de graphes.

4.2.2 Contexte et comportement d'un dessin de graphe

Comme nous l'avons vu dans le chapitre précédent, la réutilisation des génomes via les populations annexes et la base de données nécessite d'associer à chaque génome des informations supplémentaires, le contexte et le comportement lors de l'évaluation.

Dans le cas du dessin de graphe, le contexte correspond à la topologie du graphe. Pour cela, la définition du graphe ($\{V,E\}$ l'ensemble des sommets et des arêtes) est tout d'abord enregistrée. De plus, des données d'identification du fichier du graphe sont enregistrées afin de permettre un suivi des évolutions, cela correspond au nom du fichier et à sa date de dernière modification.

Toutefois, ces données sont relativement difficiles à utiliser pour définir des conditions. De ce fait, plusieurs métriques topologiques, celles présentées dans la partie 2.4 sont calculées sur chaque graphe, et leurs résultats sont stockés dans le contexte. Ce résultat est enregistré à l'aide des valeurs suivantes :

- La valeur moyenne de la métrique sur l'ensemble du graphe
- Le maximum, le minimum et l'écart-type
- Les 4 quintiles (1/5,2/5,3/5,4/5)
- Les valeurs pour chaque sommet

De plus, ces valeurs peuvent aussi être normalisées (moyenne à 0 et écart-type à 1) selon l'ensemble des valeurs contenues dans la base de données. Cela permet de comparer facilement des contextes de graphe différents selon ces différentes métriques. Quelques soient les métriques enregistrées (le comportement lui aussi prend la forme d'un ensemble de métriques), c'est sous cette forme qu'elles sont stockées.

Cependant, il n'est pas certain que nous n'ayons pas besoin d'ajouter de nouvelles informations au contexte. C'est majoritairement pour cette raison que la structure complète du graphe est conservée, car cela nous permettra par la suite d'ajouter tout type d'information en les recalculant au besoin. De plus, cela permet aussi d'utiliser des conditions liées à la topologie du graphe (par exemple, ai-je un voisin qui vérifie telle condition ?).

En ce qui concerne le comportement, le layout est stocké intégralement, de plus, l'ensemble des métriques graphiques présentées dans la partie 2.4.2 est lui aussi enregistré. La métrique de similarité est elle aussi calculée, dans sa version classique (pas de prise en compte de la distance topologique entre les sommets). Le deuxième layout utilisé pour la comparaison correspond soit à l'objectif dans le cas d'une optimisation de « copie » d'un layout, soit à un dessin de référence pour le graphe dessiné. Cependant, cette dernière métrique ne devrait pas persister à terme dans la définition du comportement.

4.2.3 Modalités d'évaluation

L'évaluation est toujours un point critique lors de la mise en place d'un AG. La manière de définir un score et de sélectionner les génomes est un des principaux rouages de l'AG. Si l'on considère que l'encodage du génome détermine ce qui peut être exprimé par l'AG, le score définit lui l'intérêt qui sera porté à un comportement. De plus, cela influe beaucoup sur la manière d'explorer l'espace de l'AG, des directions qu'il explore. D'une tâche à l'autre, un score sera ou non adapté, mais même au sein d'une même tâche, il aura beaucoup d'influence sur l'efficacité de l'algorithme. Il est donc important de pouvoir le paramétrer finement, et de pouvoir obtenir un retour précis quant à son efficacité.

Dans cette optique, le système que nous avons mis en place a été conçu pour laisser la plus grande liberté possible à l'utilisateur. Le score correspond à un **AbstractCriterion**, qui peut être défini pour chaque étape de l'algorithme. Comme nous l'avons vu dans la partie 3.4, de multiples opérateurs permettent d'exprimer une grande variété de combinaisons de critères. Pour le cas d'application du dessin de graphe, une feuille permettant d'accéder à n'importe quelle valeur d'une métrique du contexte ou du comportement (dans l'ensemble des valeurs décrit ci-dessus) a été mise au point. Un second type de feuille a été ajouté pour permettre le calcul de la similarité avec n'importe quel autre layout du même graphe.

Deux autres feuilles peuvent être utilisées dans ces arbres d'opérations. Elles sont toutes les deux liées à des critères de curiosité. La première utilise un KD-tree (celui utilisé pour indexer une population), et permet d'obtenir la distance aux plus proches voisins dans l'arbre, ou le score moyen des plus proches voisins. Le deuxième type de feuille concerne la similarité avec un ensemble d'autres layouts, il est pour l'instant spécifique au cas du dessin de graphe, mais une version générique devrait voir le jour.

Les différents critères esthétiques utilisés par les autres méthodes sont tous inclus dans la liste des métriques graphiques, il est donc possible de réaliser des évaluations très similaires à celles obtenues par les autres projets de recherche vus dans l'état de l'art de ce chapitre.

Toutes ces différentes options laissent une grande liberté de définition du score. Les prochains paragraphes sont justement consacrés à différents exemples. Le premier utilise un score très basique et les suivants feront appel à des techniques plus avancées reposant sur les différents points énoncés ci-dessus.

4.3 Exemples de structures d'optimisation

Les deux premiers exemples sont consacrés à des tâches de recopie d'un layout. Cet objectif a été très courant durant l'ensemble de cette thèse. Tout d'abord afin de valider les capacités de la

version modifiée de GEM, *GaGEM*. Les résultats obtenus étant relativement satisfaisant, cette tâche a permis d'évaluer les améliorations apportées à l'AG, en terme de vitesse ou de résultat finalement atteint.

La première tâche présentée est la tâche la plus basique utilisée pour obtenir de premiers résultats sur un nouveau graphe ou après une modification de l'AG. Elle se compose d'une unique étape, et utilise une seule population annexe. De plus, elle part toujours de 0, afin que les exécutions précédentes n'aient pas d'influence sur son déroulement.

La deuxième tâche correspond quant à elle à une version très optimisée de la structure de l'AG consacrée à une tâche de copie. Elle utilise plusieurs groupes de génomes issus de la base de données, mélange plusieurs objectifs au sein de plusieurs étapes et utilise des modules de gestion de la population avancés. La description de cette deuxième tâche se focalisera plus surcertains points précis et permettra de donner une idée des capacités d'un AG complexe visant un objectif précis.

4.3.1 Recherche basique

La structure présentée dans cet exemple correspond à la structure la plus basique considérée comme viable. Elle utilise une unique population annexe et une seule étape. Elle est comparée à deux autres structures n'utilisant pas de population annexe. Comme nous allons le voir, les résultats obtenus avec ces dernières structures sont moins intéressants que ceux obtenus avec une population annexe. Les figures 4.3 et 4.4 comparent les résultats de ces différentes structures.

La figure 4.2 fournit une représentation schématisée de la structure de l'algorithme génétique. Les éléments bleus sont ceux qui sont décrits dans le fichier XML. Ceux en jaunes correspondent aux actions automatiques de l'AG. L'utilisateur ne peut donc pas influencer leur comportement. Certains modules ont été numérotés afin de pouvoir les décrire plus facilement.

1. La première étape consiste à initialiser l'AG. Pour cela, le graphe qui doit être dessiné est chargé, et le contexte est construit (toutes les métriques topologiques sont calculées). Ce contexte sera ensuite fournit à l'ensemble des génomes. En effet, tous les génomes évalués par cet exécution de l'AG le seront sur le même graphe, et partageront donc un même contexte.

2. Le bloc d'initialisation de la population définit comment la première génération de génome est créée. Il est possible de le composer d'autant de blocs de génération de population que souhaité. Ici, seul un bloc de génération de génome de graphe (nommé **graphPopulationInitializer**) est utilisé. Il permet de créer des génomes aléatoires. Les différents paramètres permettent de contrôler la manière dont les gènes sont générés (le nombre d'opérateurs au sein de la condition par exemple). Le maximum de gène est ici fixé à 10, ce qui correspond à des génomes simples. Cela est intéressant pour une population initiale pour permettre à la complexité d'apparaître progressivement au cours de l'exécution.

3. Le bloc d'évaluation est défini par deux éléments dans la structure XML. Il faut tout d'abord définir le type d'évaluateur utilisé. Dans notre cas, seul un évaluateur est disponible, celui de dessin de graphe à l'aide de GaGEM. Il faut ensuite définir la manière de calculer le score de chaque génome. Ici, le score correspond à la similarité calculée entre le layout obtenu à partir d'un génome et le layout objectif (dans notre cas, le dessin représentant

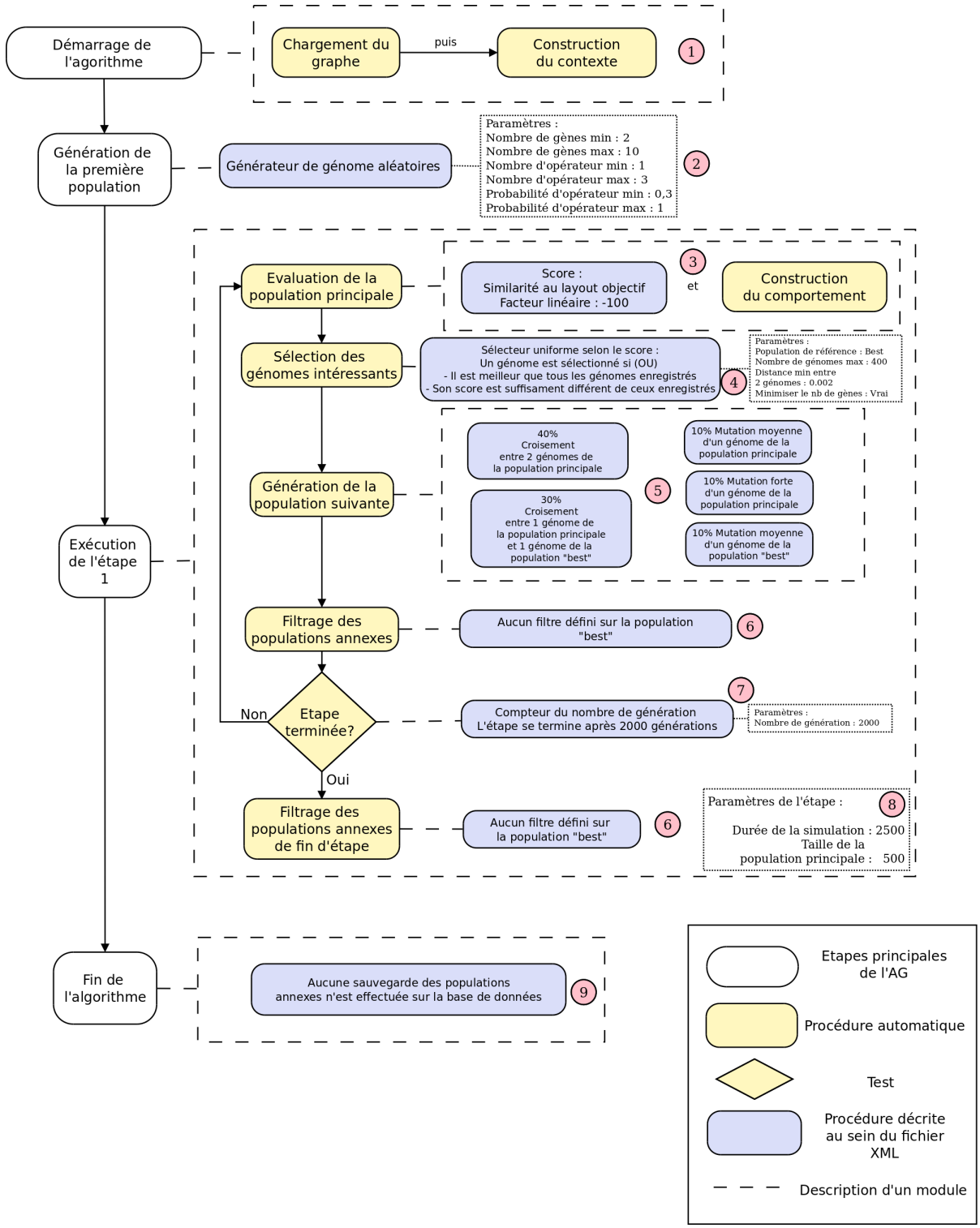


FIGURE 4.2 – La structure de l'AG utilisé pour une recherche basique

le mot « HELLO »). La similarité obtenue est multipliée par -100 afin d'obtenir un score croissant (l'AG tente toujours de maximiser le score).

4. Une fois évalué, les génomes sont testés pour voir s'ils doivent ou non être enregistrés dans une population. Dans cette structure, seule une population est définie, la population « best ». Le sélecteur utilisé correspond au bloc décrit dans la partie 3.4.3. Son principe consiste à ne sélectionner des génomes que si leur score est assez différent de ceux qui ont déjà été enregistrés. De plus, ce sélecteur s'occupe aussi de supprimer les génomes en trop dans la population lorsqu'il décide d'en ajouter un nouveau. C'est pour cela qu'il a un paramètre de nombre maximal de génomes. Le fait de minimiser le nombre de gène consiste à privilégier un génomes contenant moins de gènes qu'un autre à score équivalent.
5. Le bloc de génération de la population est un des points cruciaux de l'AG. C'est lui qui détermine quels sont les nouveaux génomes à évaluer, il définit donc en grande partie dans quelle direction se fait l'exploration de l'AG. Pour cela, il peut contenir différents sous blocs, décrivant chacun comment une partie de la population est générée. Ici 5 sous-blocs sont utilisés. Les deux premiers correspondent à l'opérateur génétique de croisement. La différence entre ces deux blocs tient uniquement aux génomes qu'ils utilisent pour réaliser leurs croisements. Le premier utilise 2 génomes de la population principale (celle qui vient d'être évaluée), et le deuxième utilise un génome de la population principale et un génome de la population « best ». Les trois blocs suivants correspondent à des mutations plus ou moins fortes sur des génomes venant soit de la population principale, soit de « best ».
6. Aucun filtre n'est défini sur la population « best ». En effet, le sélecteur utilisé supprime lui même les génomes lorsqu'ils ne sont plus utiles, il n'est pas nécessaire de filtrer d'une autre manière la population, ni en fin de génération, ni en fin d'étape.

7. Après chaque génération, l'AG interroge la condition de fin d'étape pour savoir s'il faut passer à l'étape suivante. Ici, la condition de fin utilisée est extrêmement simple : l'étape doit durer 2000 générations. Nous verrons dans les exécutions suivantes d'autres types de conditions.

8. Chaque étape peut avoir deux paramètres correspondants à la durée de l'évaluation (dans notre cas, cela correspond au nombre d'itérations minimales de GaGEM) et à la taille de la population principale. Cette taille est fixée ici à 500 génomes, ce qui signifie qu'à chaque génération, 500 génomes seront évalués.

9. Une fois toutes les étapes terminées, l'AG interroge les différentes populations pour savoir si elles doivent être enregistrées sur la base de données. Dans le cas de cette structure, aucun enregistrement n'est fait (les résultats étant obtenu dans un environnement indépendant de toute autre exécutions, enregistrer ces génomes n'aurait servi à rien).

Comparaisons avec d'autres structures

Cette structure va être comparée à deux autres encore plus simples et n'utilisant pas de population annexe. La première sera exactement similaire à celle déjà présentée, en supprimant uniquement la population annexe et les fournisseurs de génomes qui lui sont associés. Les deux types de mutations sont conservés, mais elles travaillent toutes les deux sur la population principale. De plus, un seul bloc de croisement est conservé, sa part dédiée est augmentée en proportion afin de conserver la même structure d'algorithme.

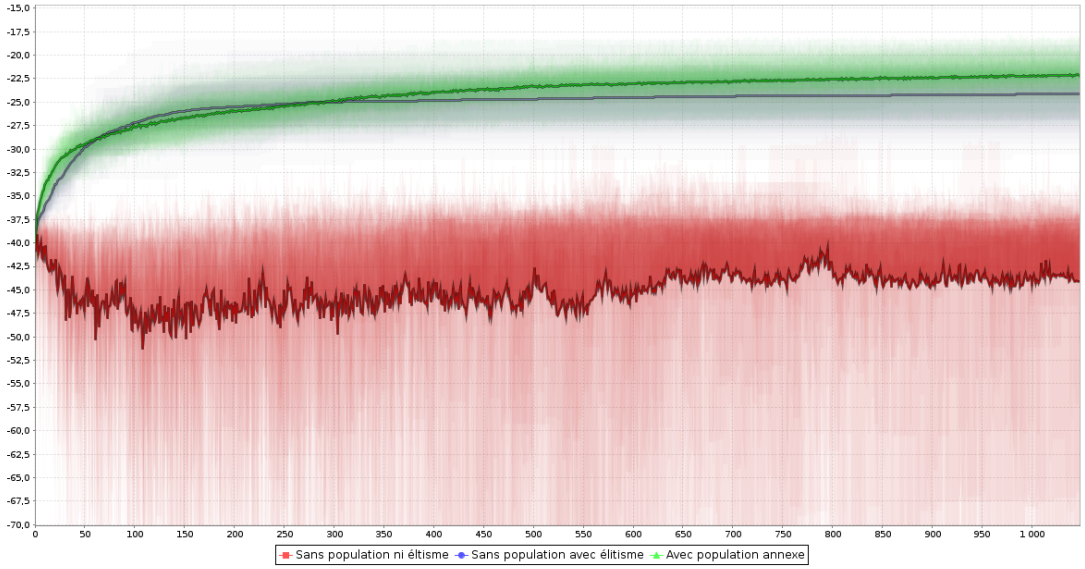


FIGURE 4.3 – Les scores maximaux à chaque génération regroupés selon la structure utilisée par l'exécution
50 exécutions par structure

La dernière structure sera identique à la deuxième avec l'ajout d'un module d'élitisme au générateur de population. Ce bloc permet de conserver à l'identique les meilleurs génomes de la population. En effet, comme on pourra le constater dans les résultats, il arrive que le score maximal diminue avec la 2^e structure. En effet, rien ne permet de conserver les meilleurs résultats, et il se peut qu'ils se perdent lors d'un croisement ou d'une mutation. Ce problème ne se produit pas avec les populations annexes, car un génome n'y est remplacé que lorsque de meilleurs sont trouvés. Ce bloc permet donc de stabiliser le score maximal. Cependant, les génomes ainsi copiés sont évalués plusieurs fois sans que cela présente le moindre intérêt (le score du génome étant déjà connu). Ce type de bloc est très courant dans les AG classiques, car comme le montrent les résultats, ils permettent d'obtenir en général de meilleurs scores (tout en favorisant cependant une convergence plus rapide).

Résultats obtenus

Pour comparer ces 3 structures, chacune d'elle a été utilisée pour 50 exécutions. Les figures 4.3 et 4.4 présentent respectivement les scores maximaux à chaque génération et le score moyen de la population principal. Les courbes tracées en dur correspondent à la valeur moyenne des 50 exécutions de chaque structure. Par exemple, pour la première image, chaque point d'une ligne opaque correspond à la valeur moyenne des meilleurs individus dans la population principale de chacune des exécutions. De plus, pour chaque exécution, l'aire entre la courbe de l'exécution et la valeur moyenne de la structure est colorée avec une forte transparence. Cela permet d'avoir une idée du faisceau généré par les 50 exécutions.

On peut voir sur la première figure que la courbe verte, qui correspond à la structure avec population annexe converge moins vite que celle avec élistime, mais qu'elle atteint un meilleur score en 1000 générations, et surtout que sa dérivée semble diminuer moins vite que celle de la structure avec élitisme. La courbe sans élitisme est quant à elle relativement désastreuse, mais ce résultat semble fortement influencé par quelques exécutions qui ont eu des résultats très mauvais et qui ont donc tiré la moyenne vers le bas. En effet, la zone au-dessus de la moyenne est

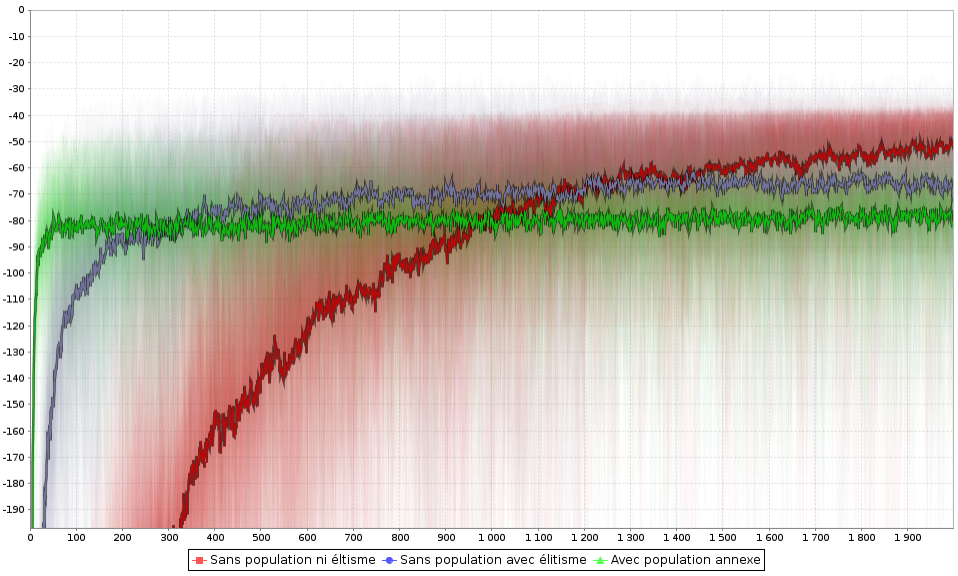


FIGURE 4.4 – Les scores moyens de la population principale à chaque génération regroupés par structure
50 exécutions par structure

beaucoup plus foncée que celle au dessous, ce qui montre que de plus nombreuses exécutions ont été au-dessus de la moyenne.

La deuxième figure porte quant à elle sur les valeurs du score moyen de la population principale. La courbe correspond donc à la moyenne des scores moyens. Ces courbes donnent des informations sur la convergence de la population. En effet, lorsque le score moyen est égal au score maximal, cela signifie en général que tous les génomes de la population sont identiques. Dans notre cas, cela ne peut pas arriver strictement grâce aux mutations, mais comme le montre la figure 4.4, les structures sont plus ou moins sujettes à ce phénomène.

En effet, on peut remarquer que même si la courbe de la structure sans population ni élitisme démarre bien plus bas que les autres (-900 à la génération 0 et -580 à la génération 50), elle augmente très régulièrement et finit relativement proche de son score maximal. De plus, on peut voir à l'aide du faisceau rouge que ce score maximal est atteint relativement rapidement pour un certain nombre d'exécutions.

Les deux autres courbes semblent plus prometteuses, car leur score moyen semble se stabiliser à une plus grande distance de leur score maximal (surtout dans la mesure où ce dernier est généralement plus élevé). La courbe verte qui correspond à la structure avec population est celle qui croît le plus vite. Cependant, elle se stabilise alors et ne se rapproche pas beaucoup plus du score maximal. De plus, on voit que le faisceau est large et très instable, ce qui indique que la population contient toujours des génomes bien plus mauvais que le meilleur (et donc un minimum différent). De plus, son score n'augmente pas trop, car des génomes ayant un score possiblement moins bon que le score maximum sont régulièrement injectés depuis la population « best » puisque cette dernière tente de maintenir une répartition uniforme selon le score de ses génomes. En effet, comme le montrent les paramètres, la distance minimale entre deux génome est de 0.002. De plus, 400 génomes sont conservés, donc la distance entre le moins bon de la population et le meilleur est d'au moins 0.8. Ce coefficient étant relatif au score, cela induit que le score du moins bon génome de la population est d'au plus -45 (le meilleur étant en moyenne

a -25). De plus, ce score maximal pour le moins bon génome n'est généralement pas atteint durant les 2000 générations que durent ces exécutions (il faut pour cela que tous les génomes de la population soient les plus proches possibles les uns des autres, ce qui est relativement difficile à obtenir). Il arrive donc très régulièrement que des génomes moins bons soient injectés dans la population principale, ce qui a donc pour effet de diminuer l'augmentation du score moyen (mais qui permet à priori une recherche plus large et moins centrée autour des quelques meilleurs génomes de la population principale). Il faut ajouter à cela l'obtention de croisement infructueux car utilisant des gènes qui ne pas compatibles (ce qui arrivent plus souvent si la population converge moins). Deux gène étant non compatible s'il ne fonctionne pas correctement ensemble, par exemple en tentant tous les deux de modifier les paramètres des mêmes sommets. Les génomes issus de tels croisement ont souvent de très mauvais scores, ce qui fait donc fortement diminuer la moyenne.

Ces résultats permettent donc de donner un premier aperçu des possibilités offertes par les populations annexes. Cependant, la structure reste extrêmement simple et n'exploite que très peu les capacités de notre système. La croissance très rapide de la valeur moyenne est par exemple assez peu intéressante. Elle est provoquée par l'injection en masse de quelques génomes lors du début de l'algorithme, lorsque la population « best » est encore presque vide, mais qu'elle est quand même utilisée pour générer 40 % des génomes de la population principale. De plus comme nous le verrons dans l'exemple suivant, cette structure ne se comporte pas très bien lors de très longues exécutions.

4.3.2 Structure complexe d'optimisation d'un objectif

La structure au centre de cet exemple est beaucoup plus évoluée que la précédente. Elle fait appel à la base de données afin d'utiliser les résultats des exécutions précédentes. De plus, une de ses étapes est basée sur une recherche guidée par la curiosité afin d'essayer de trouver de nouvelles directions d'exploration intéressantes. Enfin, elle s'appuie beaucoup sur les interactions entre les différentes populations.

La structure se compose de 3 étapes :

1. La première étape sert à faire la fusion des différents génomes extraits de la base de données. Pour cela, elle réalise de très nombreux croisements entre ces génomes et conserve ceux qui semblent intéressants.

2. La deuxième étape est celle durant laquelle l'exploration est guidée par la curiosité. L'objectif est d'étendre la population créée durant la première étape. Cette étape utilise beaucoup de mutations afin de permettre l'apparition de nouveaux comportements.

3. Enfin, une troisième étape tente d'affiner les résultats obtenus durant les deux premières étapes. Elle utilise des caractéristiques relativement classiques en terme de croisements et de mutations.

Ces trois étapes seront abordées chacune plus en détail dans le paragraphe 4.3.2. La figure 4.5 présentent une vue schématique de l'enchaînement de ces étapes.

Le but de cette structure est de montrer les possibilités qu'offre la réutilisation des résultats intermédiaires. Pour cela, elle va être comparée à la structure du premier exemple fortement

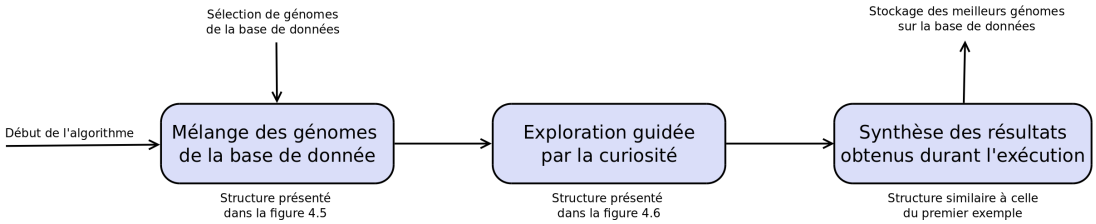


FIGURE 4.5 – Les 3 étapes qui composent cette structure d'optimisation

rallongée (100000 générations au lieu de 2000). La nouvelle structure quant à elle est beaucoup plus courte (8000 générations) et pourra donc être exécutée plus de 12 fois de suite pour un temps d'exécution équivalent. De plus, les exécutions sont réalisées sur 8 graphes différents afin de diversifier le contexte d'exécution.

Avant la présentation des étapes, le paragraphe suivant se concentre sur le problème de la recherche guidée par la curiosité dans le cadre du dessin de graphe. En effet, comme nous l'avons vu dans la partie consacrée à la mesure de similarité, le coût de la comparaison de deux contextes est relativement élevé. De plus, nous n'avons pas réussi à trouver de méthode d'indexation des layouts compatible avec la mesure de similarité. De ce fait, l'exploration guidée par la curiosité soulève plusieurs contraintes techniques.

Recherche guidée par la curiosité dans le cadre du dessin de graphe

Pour rappel, la recherche basée sur la curiosité proposée par Lehman et Stanley [60, 59, 58, 98] repose sur la comparaison entre le comportement d'un génome nouvellement évalué et ceux des génomes évalués précédemment, conservés au sein d'une archive. Le score du nouveau génome dépend alors de la distance entre le nouveau comportement et le comportement archivé le plus proche de lui.

La méthode utilisée pour comparer deux comportements est la métrique de similarité. Cela pose un problème majeur quant à la gestion de l'archive. En effet, la mesure de similarité implique de comparer le nouveau layout avec tous ceux obtenus jusque-là. Dès que l'archive commence à être importante, cela devient impossible. Pour pouvoir réaliser cela, il serait nécessaire de pouvoir indexer les différents génomes de l'archive afin de pouvoir obtenir rapidement les plus proches voisins d'un nouveau génome. Nous n'avons cependant pas trouvé de méthode correcte pour réaliser un tel index.

Une alternative au stockage de l'archive complète a toutefois été proposée Lehman et Stanley dans [59]. Ils soulèvent en effet le problème de la taille de l'archive et proposent simplement de ne garder que les k derniers individus testés dans l'archive. Ils ont pour cela réalisé des essais en fixant k à la taille de la population et ont alors obtenus des résultats que légèrement inférieur à ceux obtenus avec l'archive complète. Il existe cependant un risque que l'AG fasse alors des aller-retour entre deux points suffisamment distant.

Pour mettre en œuvre cette limitation de la taille de l'archive, un filtre a été mis en place. Il supprime à la fin de chaque générations les génomes les plus anciens de la population annexe lorsque la taille de cette dernière excède une certaine limite.

Un deuxième problème lié à l'exploration par la nouveauté se pose dans le cadre du dessin de

graphe. Il s'agit de l'aspect « ouvert » de l'espace des dessins. En effet, l'expérience de Lehman et Stanley se base sur l'exploration d'un labyrinthe par un robot. Le comportement correspond alors à la position finale du robot. Le labyrinthe étant fermé, le nombre de comportements ne permettant pas d'aboutir à l'arrivée du labyrinthe est fini. Or dans le cadre du dessin de graphe, il n'existe pas de telles frontières. De plus, les essais réalisés par Lehman en supprimant les murs extérieurs du labyrinthe ont donné lieux à des résultats très mauvais (la plupart des robots se perdant dans l'infinité du plan) [59]. Il est donc nécessaire de simuler des frontières à la zone explorable. Nous utilisons pour cela un score à base d'un filtre, qui ne prend en compte la nouveauté d'un génome que si sa similarité à l'objectif est au-dessus d'un certain seuil.

Structure de l'algorithme génétique

Première étape : Fusion des résultats obtenus précédemment Cette première étape a pour objectif de définir une population de base à partir de croisements multiples de génomes issus de la base de données. Ces génomes sont regroupés en 2 populations. La première, nommée *databaseGraph*, est composée de 100 génomes issus d'exécutions travaillant exactement sur le même graphe. La deuxième, nommée *databaseTask*, utilise 80 génomes tirés d'exécutions travaillant sur les 8 graphes utilisés pour cet exemple, à raison de 10 génomes par graphe.

La figure 4.6 présente la structure de cette étape et les modules qu'elle utilise.

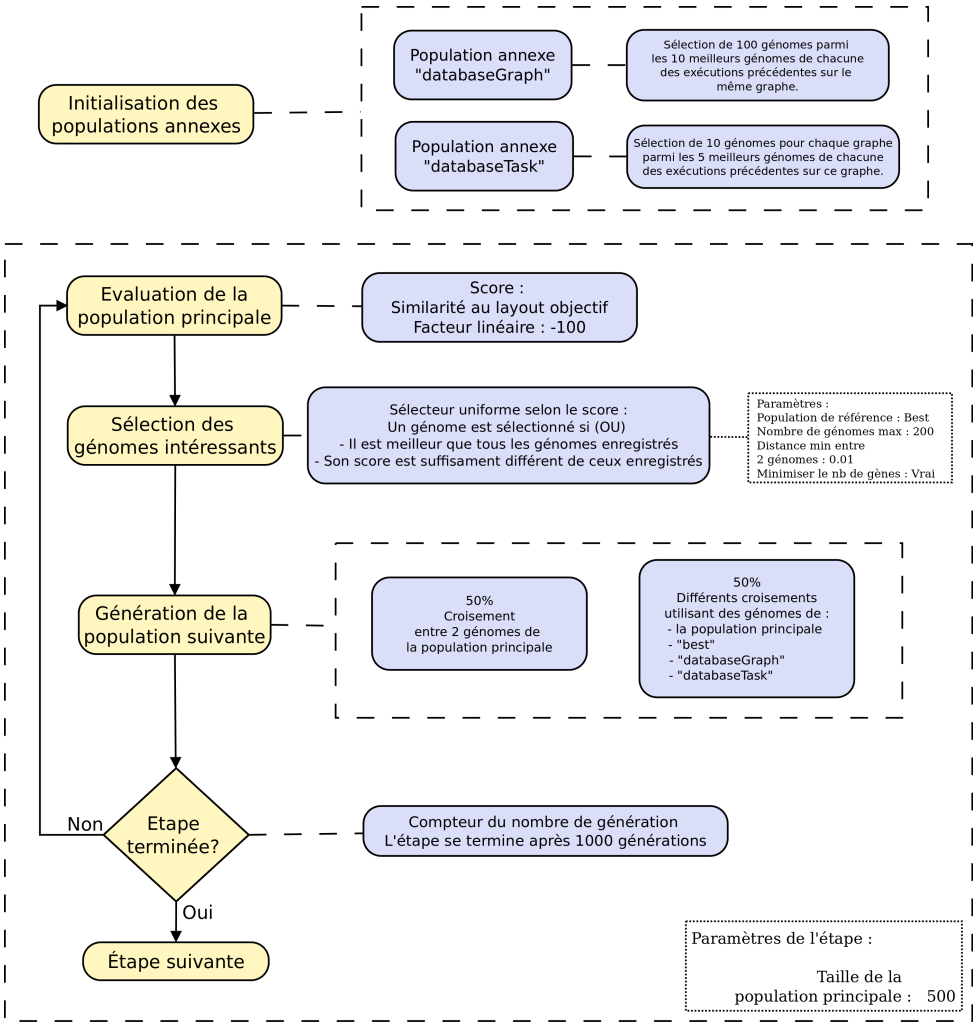


FIGURE 4.6 – Une représentation schématique de la première étape de la structure d'optimisation complexe

L'initialisation des deux populations annexes sont réalisées à l'aide d'un bloc **tulipParametrizedSQLPopulationInitializer**, qui permet de fournir une requête SQL fournissant une sélection de génomes à insérer dans la population. Ce module permet en plus d'utiliser certains mots clés comme *$graphName$* ou *$taskId$* afin de paramétrer dynamiquement la requête. Cette fonctionnalité est par exemple utilisée pour ne récupérer que des génomes issus d'exécutions travaillant sur le même graphe que l'exécution courante.

Les requêtes correspondent aux processus suivants :

Algorithme 11: Pseudo-code correspondant à la requête pour la population *databaseGraph*

```
début
    génomes ← Ensemble de génomes vide;
    pour Exécution exécution : Exécutions précédentes travaillant sur le même graphe
    faire
        Ajouter à génomes les 20 meilleurs génomes de exécution
    génomes ← Sélection aléatoire de 100 génomes de génomes;
    retourner génomes
```

Algorithme 12: Pseudo-code correspondant à la requête pour la population *databaseTask*

```
début
    génomes ← Ensemble vide;
    pour Graphe graphe : Graphes utilisés dans cet exemple faire
        génomesGraphe ← Ensemble vide;
        pour Exécution exécution : Exécutions précédentes travaillant sur graphe faire
            Ajouter à génomesGraphe les 5 meilleurs génomes de exécution
        Ajouter à génomes une sélection aléatoire de 10 génomes de génomesGraphe;
    retourner génomes
```

Une fois ces génomes récupérés, l'objectif de la première étape va être d'établir une population représentative des possibilités de ces génomes. Pour cela cette étape se base uniquement sur l'opérateur de croisement afin de réaliser une exploration de l'espace délimité par les différents génomes récupérés sur la base de données. Moraglio a en effet montré que l'utilisation d'un opérateur de croisement sans mutation revenait à réaliser une exploration de l'espace convexe défini par les points correspondants aux éléments de départs [70].

C'est justement ce type de recherche qui doit être ici réalisée pour « relier » les génomes obtenus dans la base de données. En effet, ces derniers proviennent d'exécutions différentes et peuvent de ce fait présenter de grandes différences les uns entre les autres. Ces différences rendent plus difficile l'obtention de génomes intéressants à l'aide de croisements, car les gènes provenant d'explorations différentes peuvent être incompatibles. L'objectif de l'étape est donc de trouver un ensemble de génomes un minimums intéressants obtenus à l'aide de croisements se basant sur ces deux populations. De plus, un nouveau paramètre de l'opérateur de croisement est utilisé. Il permet de contrôler la prévalence du premier parent sur le deuxième. Ce paramètre influe sur les chances qu'un gène vienne de la mère ou du père. La prévalence est donc comprise entre 0 et 1 (non inclus, les valeurs 0 et 1 correspondant à des génomes identiques à la mère ou au père). Sa valeur par défaut est de 0.5 afin de ne fournir aucun avantage à un des deux parents. Cependant dans le cadre de cette étape, des prévalences de 0.8 sont aussi utilisées pour certains croisements (la prévalence définie étant celle associée à la mère). Cela permet d'obtenir un croisement où le génome résultant est très proche de la mère et avec tout de même quelques gènes du père (environ 20 %). Ces croisements permettent d'obtenir des génomes qui fonctionnent correctement plus facilement, mais en contrepartie, ils ont aussi plus de chance d'être exactement similaires à leur mère.

Les populations annexes utilisées dans cette étape sont les deux dans lesquels les génomes de la base de données sont stockés, « databaseGraph » et « databaseTask », et une 3^{e} population, « best » qui fonctionne comme celle de la structure basique. Elle utilise un unique sélecteur qui tente de maintenir une répartition uniforme des génomes selon leur score dans la population. Cependant, la taille de cette population est ici fixée à 200 génomes au maximum.

Comme nous l'avons dit, cette étape n'utilise que des croisements comme opérateurs génétiques. Les populations sont générées selon les proportions suivantes :

- 50 % Croisement classique (prévalence à 0,5) entre 2 génomes de la population principale.
- 10 % Croisement avec prévalence à 0,8 entre 2 génomes de la population principale.
- 10 % Croisement avec prévalence à 0,8 entre 1 génome de la population principale et 1 génome venant d'une population annexe parmi « best », « databaseGraph » et « databaseTask »
- 10 % Croisement avec prévalence à 0,8 entre 2 génomes venant d'une des trois populations annexes.
- 10 % Croisement avec prévalence à 0,8 entre 2 génomes de « databaseGraph »
- 10 % Croisement avec prévalence à 0,8 entre 1 génome de la population principale et 1 génome de « databaseGraph »

L'objectif de cette étape est de remplir la population « best » avec un ensemble de génomes utilisant les gènes récupérés dans la base de données. Cette étape ne produit généralement pas de grand progrès au niveau du score. Comme nous allons le voir, c'est majoritairement durant la 3e et dernière étape que cette structure réalise les plus grandes améliorations du score. En effet, l'étape suivante est quant à elle un prolongement de cette première étape en utilisant cette fois une exploration guidée par la curiosité.

Deuxième étape : Recherche guidée par la curiosité Cette deuxième étape repose entièrement sur une recherche guidée par la curiosité. Comme nous l'avons vu dans le paragraphe 4.3.2, ce type de recherche pose plusieurs problèmes dans le cadre du dessin de graphe, tout particulièrement au niveau de la taille de l'archive utilisée pour enregistrer les comportements des génomes précédents et de l'aspect ouvert et sans limites de l'espace des dessins d'un graphe. La structure de cette étape est présentée schématiquement dans la figure 4.7.

La limitation de l'espace exploré est réalisée à l'aide de la métrique de similarité. Pour chaque layout produit, sa similarité avec le layout objectif est calculée et comparée avec une valeur seuil. Si la similarité est supérieure au seuil, le comportement est considéré comme n'apportant aucune nouveauté. Nous verrons plus tard comment ce seuil est défini.

Le problème de la taille de l'archive est plus complexe à résoudre. La solution utilisée consiste à utiliser 2 populations annexes. La première conserve 25 génomes ayant obtenus de bons scores de similarité par rapport à l'objectif (moins de 0,5), tout en ayant une similarité entre chacun d'au moins 0,3. Le but de cette population est de construire un ensemble de bons génomes qui approchent la solution depuis plusieurs directions. De plus, le sélectionneur associé à cette population va permettre de remplacer un génome par un autre très ressemblant, mais qui améliore la similarité à l'objectif. Cette population est nommée « novelty » et est transférée sur la base de données à la fin de l'AG.

Cependant, utiliser seulement une population ne suffit pas. En effet, si le score des nouveaux génomes évalués est directement leur distance au plus proche génome enregistré dans cette population, l'AG a tendance à rapidement converger vers un état où la population annexe est pleine et que tous les nouveaux génomes évalués sont le plus loin possible des génomes de la population annexes (mais pas suffisamment pour être enregistrés au sein de cette population), mais sans réussir à s'éloigner d'avantage sans nuire à leur similarité avec le layout objectif (et donc, passer au-dessus du seuil à partir duquel tout génome est considéré comme mauvais). Dans ce cas, l'algorithme reste dans une position d'équilibre et les génomes nouvellement évalués sont très

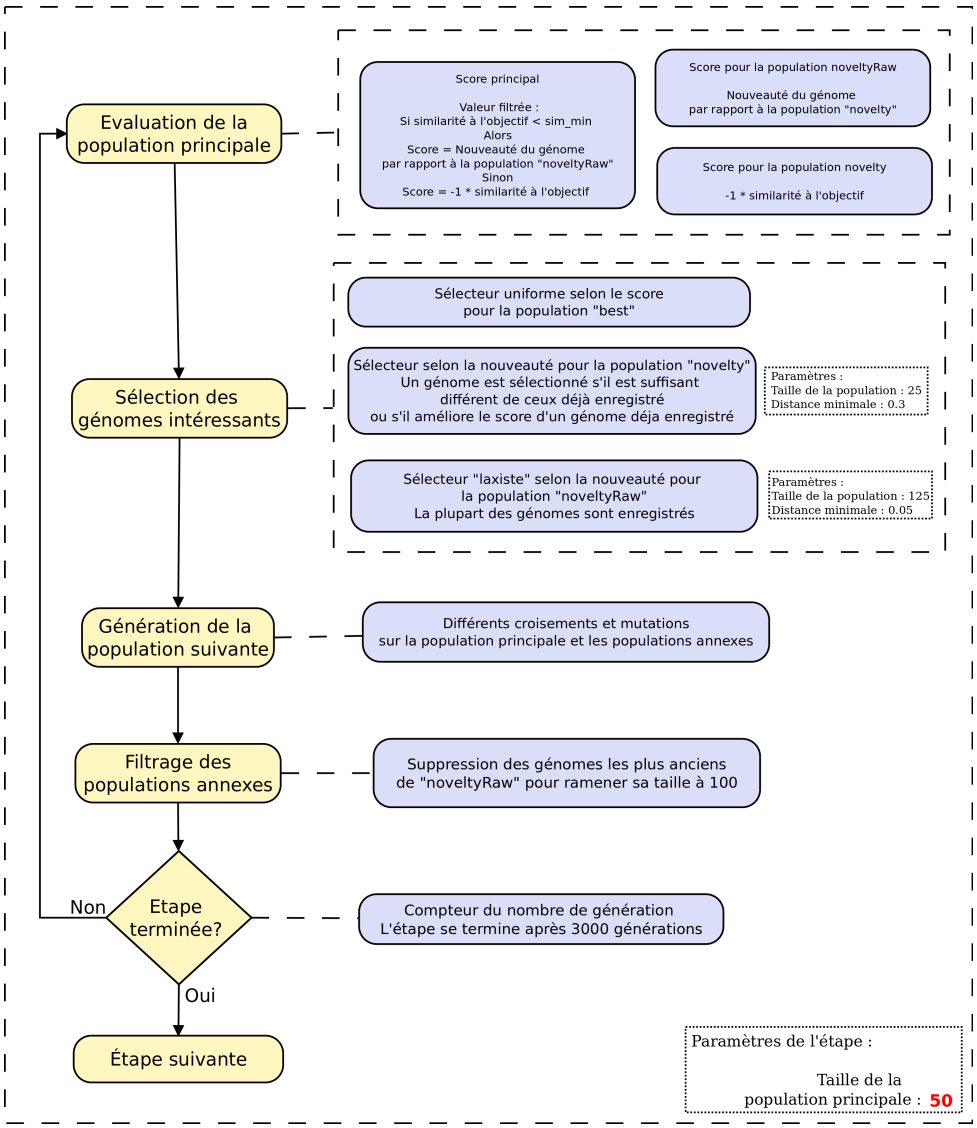


FIGURE 4.7 – Une représentation schématique de la deuxième étape de la structure d'optimisation complexe

similaires entre eux.

Pour éviter ce phénomène, une deuxième population annexe est ajoutée, qui est nommée « noveltyRaw ». Elle va servir d'archive dynamique en enregistrant les k derniers génomes évalués, même si ceux-ci semblent très peu intéressants, afin de toujours stimuler un mouvement à la population principale. Cette deuxième population utilise le même type de sélectionneur que la première, mais avec cette fois des seuils beaucoup plus laxistes (pour être enregistré, le seuil de similarité à atteindre est plus haut et la distance minimale entre 2 génomes plus faible). Ce sélectionneur permet d'enregistrer au maximum 125 génomes dans la population. De plus, un filtre est ajouté, qui ne garde que les 100 génomes les plus récents dans la population. Ainsi, lorsque la population est pleine, le sélectionneur selon la nouveauté peut quand même ajouter des génomes, mais à la fin de l'étape, les plus anciens seront supprimés afin de ramener la taille de la population à 100. Ce système permet d'éviter toute stagnation de la population principale. Cette deuxième population n'est pas enregistrée dans la base de données à la fin du GA, car les génomes qu'elle contient ne sont généralement pas des plus intéressants.

Le point suivant de cette étape concerne les fonctions de score utilisées. "Fonctions" au pluriel, car en plus de la fonction de score de l'étape, il est possible d'attribuer à chaque population annexe une définition de score propre. Ce score sert majoritairement à définir les chances de sélection de chaque génome lors d'une sélection aléatoire dans la population. Il y a donc 3 scores définis, celui de l'étape, et ceux des populations « novelty » et « noveltyRaw ».

Le score le plus simple est celui de la population « novelty », il s'agit de celui déjà utilisé dans la structure précédente, *ie.* la valeur de la métrique de similarité, multiplié par un coefficient linéaire de -1. Ainsi, ce score est croissant et s'approche peu à peu de 0 par valeurs négatives.

Le score de la deuxième population annexe, « noveltyRaw » est quelque peu différent. Il s'appuie sur un **graphNoveltyCriterion**, un module héritant de **AbstractCriterion**, qui permet de calculer la nouveauté apportée par un comportement par rapport à ceux stockés dans une certaine population. Ici, la population de référence est « novelty ». De plus, ce module permet aussi d'obtenir le score associé au génome dont le comportement est le plus proche. Deux coefficients linéaires sont définis pour chacune de ces deux valeurs (la nouveauté du comportement, et le score du génome le plus proche), dans notre cas, ces deux coefficients valent 1. Le score des génomes stockés dans « noveltyRaw » est donc égal à la somme de la nouveauté de son comportement par rapport à ceux stockés dans « novelty » et du score du plus proche génome stockés dans « novelty ».

Enfin, le dernier score est celui associé aux génomes évalués à chaque génération. L'objectif de cette étape est de toujours avoir une population changeante. Pour cela, un **graphNoveltyCriterion** basé sur la population « noveltyRaw » est utilisé. Cette population contient les contextes des 100 derniers génomes évalués, de ce fait, pour avoir un bon score, un génome est obligé d'avoir un comportement différent de tous ceux de la population. Toutefois, nous avons vu qu'il est nécessaire de restreindre l'espace de recherche pour éviter une divergence trop éloignée de l'objectif. Pour cela, le critère de nouveauté est utilisé conjointement avec un **filterCriterion**. Ce critérion se comporte comme un filtre (passe-bande, passe-bas ou passe-haut). Il prend deux critérions, c_1 et c_2, en entrée et fournit une valeur v_s suivant la règle suivante (cas du filtre passe-bas) :

$$v_s = \begin{cases} c_2 & Si \quad c_1 \leq l_b \\ 0 & Si \quad c_1 > l_b \end{cases}$$

Avec l_b la valeur seuil définie pour le filtre passe-bas. La métrique de similarité est utilisée comme valeur pour c_1, avec une valeur de seuil l_b égale à la meilleure similarité atteinte durant la première étape multipliée par 2. Cette valeur seuil est elle aussi définie à l'aide de critérions. Elle s'appuie en particulier sur le **populationStatisticCriterion** qui permet d'obtenir des informations sur les scores des génomes d'une population.

Le score est finalement défini comme la somme de ce filtre et de la métrique de similarité, un petit coefficient étant associé à cette dernière. Cette dernière composante permet de « guider » les génomes vers la valeur seuil lorsqu'il sont au-dessus.

Le dernier point important de cette étape concerne la taille de la population. En effet, cette dernière doit être inférieure à la taille de l'archive dynamique (soit la population « noveltyRaw » ici), de plus, chaque comportement de génome va être comparé à de nombreux autres (en général ceux de « novelty » et « noveltyRaw » lorsque sa similarité est suffisamment basse). Ces comparaisons sont très coûteuses ($O(n^2)$ avec n le nombre de sommets du graphe), il est donc nécessaire de limiter leur nombre. La taille de la population principale est donc fixée à 50 durant cette étape. Cela permet d'obtenir un temps de calcul par génération proche de celui de la première étape. De plus, cela permet d'avoir une archive couvrant au moins les génomes des 2 dernières générations.

Cette étape ne trouve que très rarement des génomes qui se rapprochent plus de l'objectif que ceux connus jusque là. Cela s'explique par la très faible pression élitiste appliqué au travers des sélections et des fonctions de score. En effet, son objectif est plus de trouver de nouveaux résultats corrects différents de ceux connus plutôt que de meilleurs résultats. Comme nous allons le voir tout de suite, c'est la 3$^{\text{e}}$ et dernière étape qui est en charge de cette amélioration.

Troisième étape : Obtention de nouveaux résultats Les deux premières étapes ont permis de constituer 2 populations « best » et « novelty » contenant toutes les deux des génomes à priori nouveaux et répartis autour de l'objectif (les différents génomes de « novelty » ont une distance entre eux supérieure à leur distance à l'objectif). Cette étape va dont utiliser ces différents génomes pour tenter de trouver des génomes améliorant effectivement leur similarité à l'objectif.

Aucune nouvelle population annexe n'est utilisée durant cette étape. Deux sélectionneurs permettent toutefois d'enregistrer des génomes dans « best » et « novelty » pour améliorer peu à peu ces populations. De ce fait, elle reprend globalement la structure de l'étape de la première structure, avec toutefois une modification au niveau du score. Afin d'augmenter la pression de sélection, au lieu d'utiliser un coefficient linéaire de -1 pour rendre croissante la métrique de similarité, cette étape utilise un coefficient de puissance de -1, et utilise donc l'inverse de la métrique de similarité comme score pour les génomes de la population principale. Cela change considérablement les chances de sélections des génomes de la population principale lorsqu'ils se rapprochent fortement de l'objectif.

Au niveau des opérateurs génétiques, cette étape utilise différents croisements et différentes mutations se basant sur la population principale et les deux populations annexes.

Les contraintes issues de la recherche guidée par la nouveauté (et l'utilisation massive de la métrique de similarité) ne sont ici plus d'actualités, la taille de la population principale est de nouveau définie à 500. De plus l'étape se termine au bout de 4000 générations, ce qui en fait l'étape la plus longue de cette structure.

Résultats obtenus

Pour l'obtention de résultats, cette nouvelle structure a été comparée à celle du premier exemple, modifiée afin de durer 100000 générations. L'objectif de cette comparaison est de montrer qu'il est beaucoup plus efficace de se baser sur des résultats obtenus durant des exécutions précédentes plutôt que de faire de très longues exécutions travaillant indépendamment de toutes les autres. Ces exécutions ont été réalisées sur 8 graphes différents, avec 5 longues exécutions ($5 * 100000 = 500000$ générations calculées) et 30 exécutions de la nouvelle structure ($30 * 8000 = 240000$ générations). Cependant, la nouvelle structure ne fonctionne que très mal si la base de données est initialement vide. Pour éviter cela, la base de données a été préalablement initialisée avec 10 exécutions sur chaque graphe d'une structure équivalente à celle du premier exemple ne durant que 1000 générations.

Les 8 graphes utilisés correspondent à 6 graphes issus du domaine de la bio-informatique et représentant des voies métaboliques. Ces graphes représentent des chaines de réactions chimiques se déroulant au sein d'un métabolisme vivant. Leurs représentations doivent respecter des règles strictes au niveau du placement des différents sommets afin que les biologistes puissent rapidement extraire des informations à la lecture de ces représentations visuelles. Les deux autres graphes sont le graphe HELLO utilisé dans le premier exemple et un deuxième graphe présentant deux cliques reliées par un pont dont les sommets ont été placés manuellement. Les différents sommets ont été placés afin de mettre en avant des contraintes d'alignement au sein des deux cliques. Ces différents graphes ainsi que les meilleurs résultats obtenus durant cette étape sont présentés dans les figures 4.11 et 4.12.

La figure 4.8 présente le résultat (score maximum à chaque génération) des 5 exécutions longues pour 3 des graphes utilisés. Cette image permet de se rendre en compte de l'évolution par palier du score, et surtout la relative stagnation au bout de 10000 générations du score pour les deux premiers graphes. Le 3^e graphe présenté, CC_50 est celui qui obtient le moins bon score au final, cependant, on observe que son score évolue beaucoup plus longtemps que les autres. Cela est lié à la complexité du graphe, et de la représentation qu'on en attend. En effet, le nombre placé à la fin du nom de chaque graphe indique le nombre de sommets du graphe. *CC* 50 est donc le plus grand des 3 présentés ici, et surtout, le layout objectif est relativement complexe et très éloigné du résultat produit par un algorithme par modèle de force sur ce graphe. Comme nous allons le voir, cette complexité rend difficile le traitement de ce graphe avec la structure présentée dans cet exemple.

Dans la suite de cette comparaison, seul le meilleur score atteint pour chaque graphe par l'ensemble des exécutions longues sera conservé (soit par exemple $-8{,}465$ pour *Cascade 23*). Ces scores vont maintenant être comparés à ceux obtenus au cours des exécutions basées sur la structure présentée précédemment.

Cependant, il n'est pas possible pour la nouvelle structure de superposer les résultats des différentes exécutions à génération équivalente, car chaque exécution profite des résultats obtenus par celles réalisées précédemment. De ce fait, les résultats vont être présentés sous la forme d'une courbe indiquant le meilleur score obtenu par la k-ième exécution sur un graphe, k variant de 1 à 30. Les résultats sont présentés dans les figures 4.9 et 4.10.

Deux lignes sont ajoutées à chaque graphique, une première correspondant au score maximal

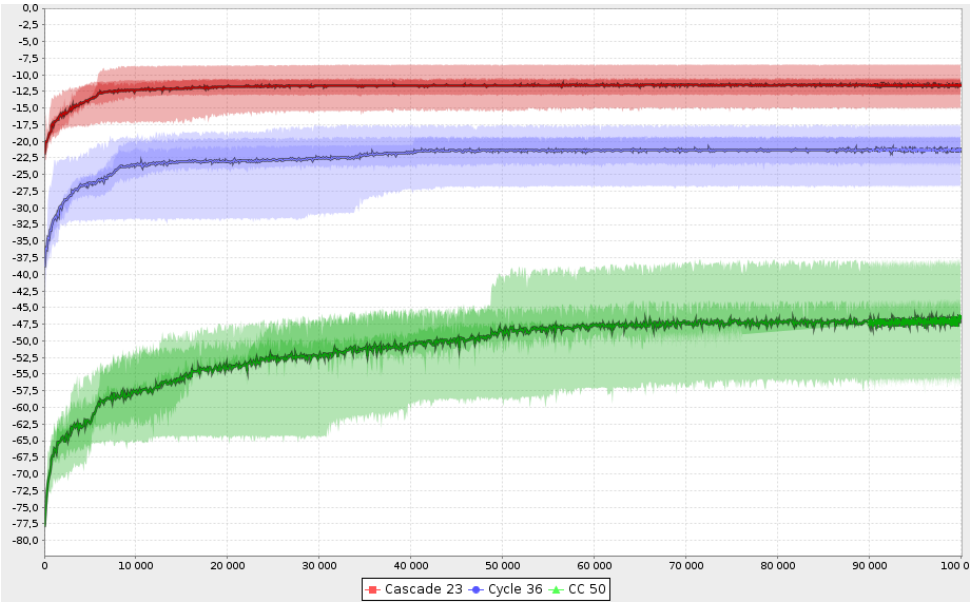


FIGURE 4.8 – Les résultats des exécutions longues sur 3 des graphes utilisés

obtenu durant les exécutions d'initialisation (en rouge), et une deuxième correspondant au score maximal atteint par les exécutions longues (en vert).

Ces résultats montrent que cette structure arrive en général à optimiser très rapidement les scores obtenus durant la phase d'initialisation. Cela s'observe particulièrement pour les graphes Cascade 23, CC 21, Hello et Double cluster. Ces graphes ont en commun le fait d'avoir relativement peu de sommets. Pour les autres, et en particulier CC 50 et Cascade 54, cette structure semble moins efficace. Pour ces deux derniers graphes, le score atteint au bout des 30 exécutions n'est toujours pas meilleur que celui obtenu par les exécutions longues.

Dans le cadre du graphe CC 50, ce résultat décevant s'explique en partie par la valeur très haute du score à la fin de la phase d'initialisation. En effet, ce score de 0.59 est supérieur au seuil à partir duquel des génomes peuvent être enregistrés dans la population « novelty ». De ce fait, la recherche guidée par la nouveauté ne fonctionne pas pour ce graphe avec la structure telle qu'elle est définie pour l'instant.

Cependant, cela n'explique pas entièrement les résultats obtenus. En effet, un problème similaire se pose le graphe Cascade 54, pour lequel l'évolution apportée par la structure utilisant la base de données est très faible. Or pour ce deuxième graphe, il n'y a aucun problème lié à la valeur de ce seuil. Par contre, lors de l'étude précise des meilleurs génomes obtenus pour ces graphes, on observe qu'ils n'ont que très peu de gènes (3 pour CC 50 et 4 pour Cascade 54). Or, ces graphes présentent de nombreux sommets dessinés de manière très différente dans le cadre du layout objectif. Il est donc nécessaire d'avoir de nombreuses règles pour bien dessiner ce graphe en respectant l'ensemble des contraintes. Or plusieurs éléments de la structure favorisent les génomes ayant moins de gènes à scores équivalents. Cette pression sur le nombre de gènes a été mise en place pour éviter d'avoir une multiplication de gènes inutiles, en particulier pour les petits graphes qui n'ont justement besoin que de peu de règles différentes, mais elle empêche aussi dans le même temps l'apparition de génomes bien adaptés à de plus grands graphes.

Pour pousser plus loin cette analyse sur le nombre de gènes, les résultats des graphes fortement

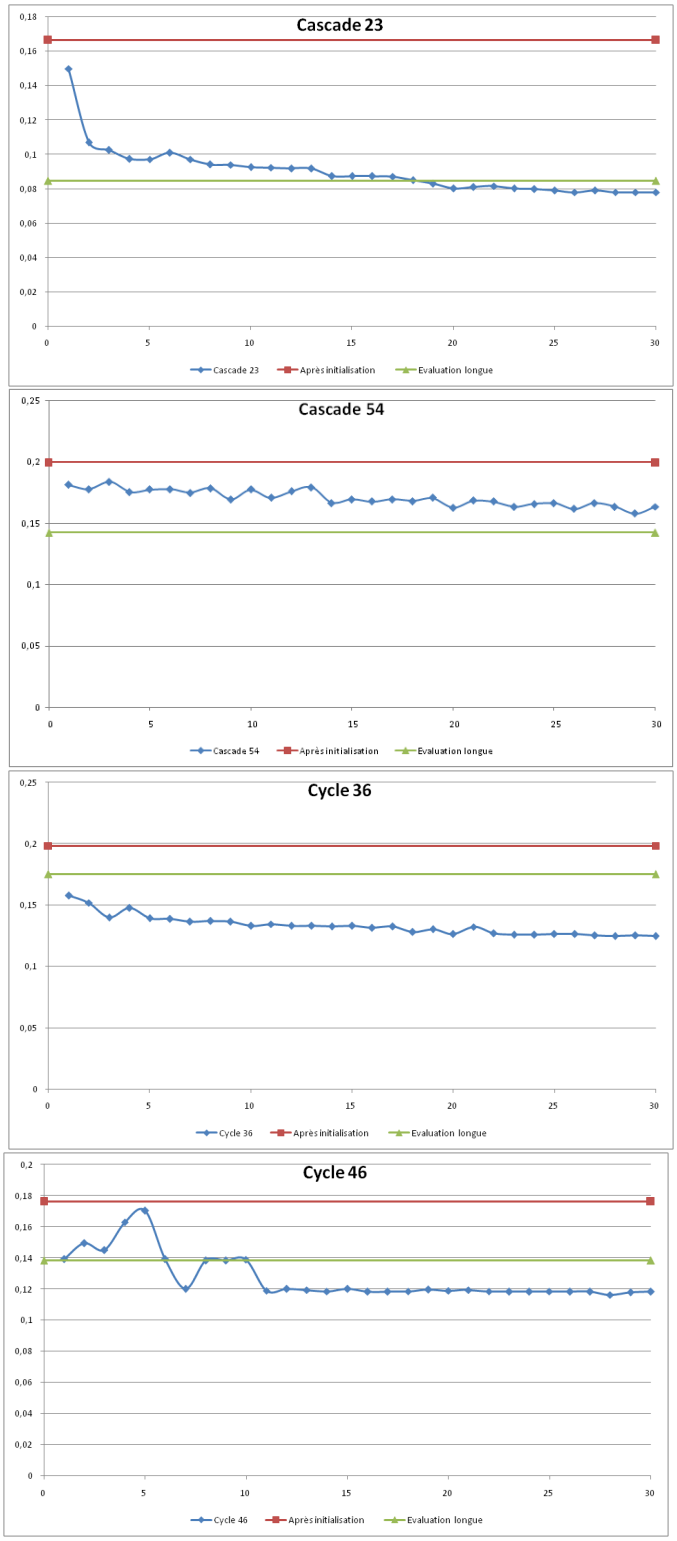


FIGURE 4.9 – Les résultats obtenus pour les graphes cascades et cycles

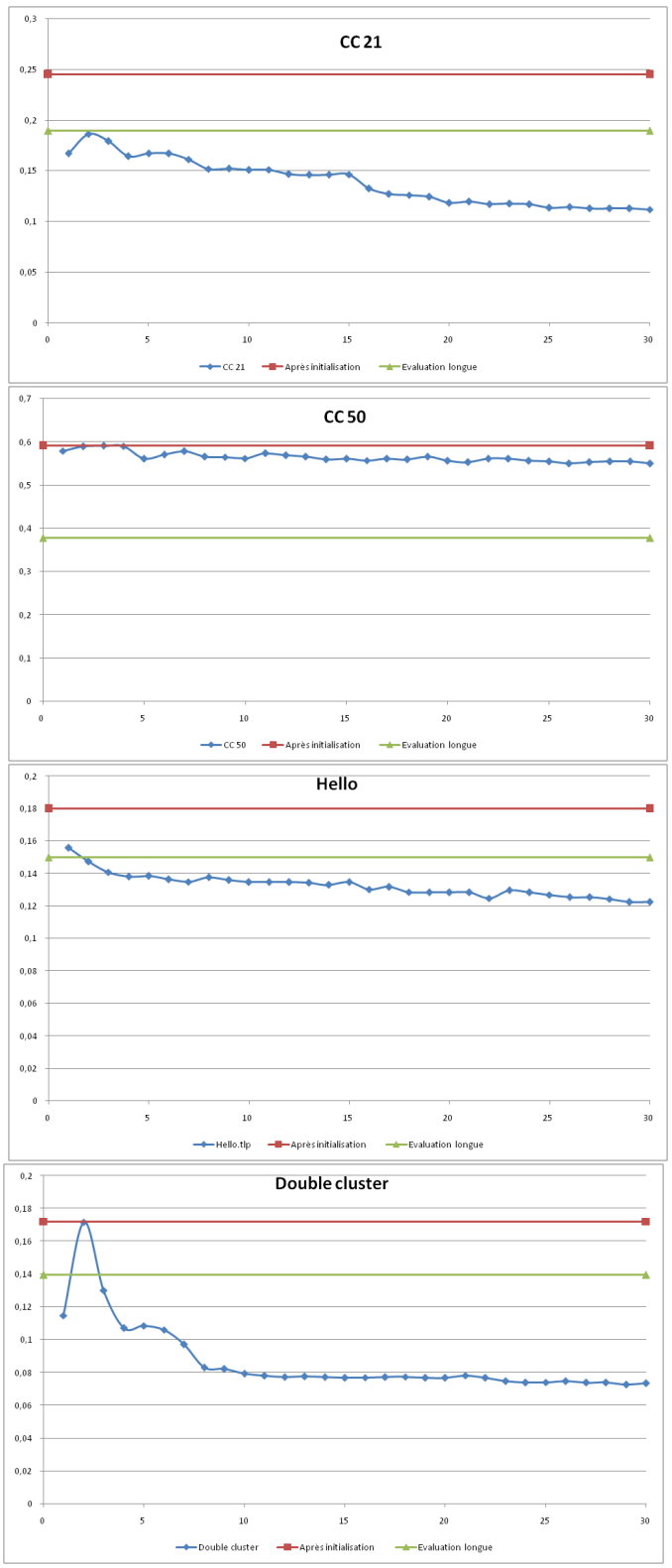


FIGURE 4.10 – Les résultats obtenus pour les graphes CC, Hello et double cluster

améliorés ont été observés eux aussi, et il est intéressant de voir que les génomes de ces graphes présentent généralement plus de gènes que ceux des grands (par exemple, le génome obtenant le meilleur score pour le graphe Hello a 9 gènes).

Les figures 4.11 et 4.12 présentent respectivement les objectifs des 8 graphes utilisés, et les meilleurs résultats pour chacun de ces graphes.

4.4 Exemple de structures interactives

La problématique considérée lors de la mise au point de structures interactives est complètement différente de celle des deux précédents exemples. En effet, l'objectif des tâches interactives est de trouver de nouvelles représentations intéressantes pour un graphe donné. Or, comme nous l'avons vu lors des chapitres précédents, un tel objectif ne peut pas s'exprimer sous la forme d'une fonction unique à optimiser pour l'ensemble des graphes.

Une représentation intéressante peut prendre des formes très différentes en fonction du graphe et de l'utilisateur, et dépend donc très fortement du contexte de l'évaluation. Cependant, comme nous l'avons vu dans la partie 2.4.2, de nombreuses métriques esthétiques sont disponibles pour orienter cette recherche. Ces métriques vont donc être un des outils de base des structures interactives.

Cependant, utiliser une seule métrique pour guider l'algorithme aboutit généralement à des résultats très peu intéressants. Par exemple, tous les essais réalisés en utilisant uniquement le nombre de croisements d'arêtes aboutissaient à des graphes complètement écrasés et très longs, mais qui ne contenaient que très peu de croisement d'arêtes. La métrique était donc bel et bien optimisée, mais au prix d'une très forte dégradation du layout. Une solution envisageable serait possiblement d'utiliser un paramétrage très fin d'un score utilisant l'ensemble des métriques esthétiques disponibles. Cependant, un tel paramétrage dépendrait lui aussi beaucoup du graphe traité et serait bien trop complexe à mettre au point.

La solution que nous avons choisie à consister à demander à l'utilisateur de fournir un dessin de base pour son graphe (ou par défaut le dessin obtenu par GEM avec les paramètres fixés aux valeurs par défaut). Ce layout fournit une base correcte pour ce graphe. L'idée consistera ensuite à tenter d'améliorer ce layout par pas successifs, chaque pas ayant la possibilité de s'éloigner un peu du précédent pour tenter d'optimiser une métrique esthétique. Pour cela, différents critères esthétiques sont mis en compétition avec une métrique de similarité.

Une séance d'évaluation se déroule généralement en plusieurs fois. Lors de la mise en route, l'utilisateur fournit un graphe (et son layout de base) et choisit une première structure interactive à utiliser. Cette première structure doit être une structure d'initialisation, ce qui signifie qu'elle ne requiert pas d'évaluation utilisateur préalable pour fonctionner. Le serveur lance l'exécution dès qu'une machine est disponible. À la fin de cette dernière, le serveur informe l'utilisateur et lui soumet la liste de candidats générés par l'AG. L'utilisateur évalue alors chacun des candidats et sélectionne son préféré. Il choisit ensuite la structure utilisée pour l'étape suivante et ainsi de suite jusqu'à l'obtention d'un graphe qui lui convient.

Un autre élément important des structures interactives est la sélection des candidats. Ces candidats correspondent aux différents layouts proposés à l'utilisateur. Ce dernier doit alors sélectionner celui qu'il préfère et il peut aussi évaluer l'ensemble des candidats affichés de diverses manières. Ces différentes évaluations seront présentées dans les paragraphes suivants. Toutefois, il est important de ne pas proposer plusieurs fois à l'utilisateur le même layout. Pour cela, il

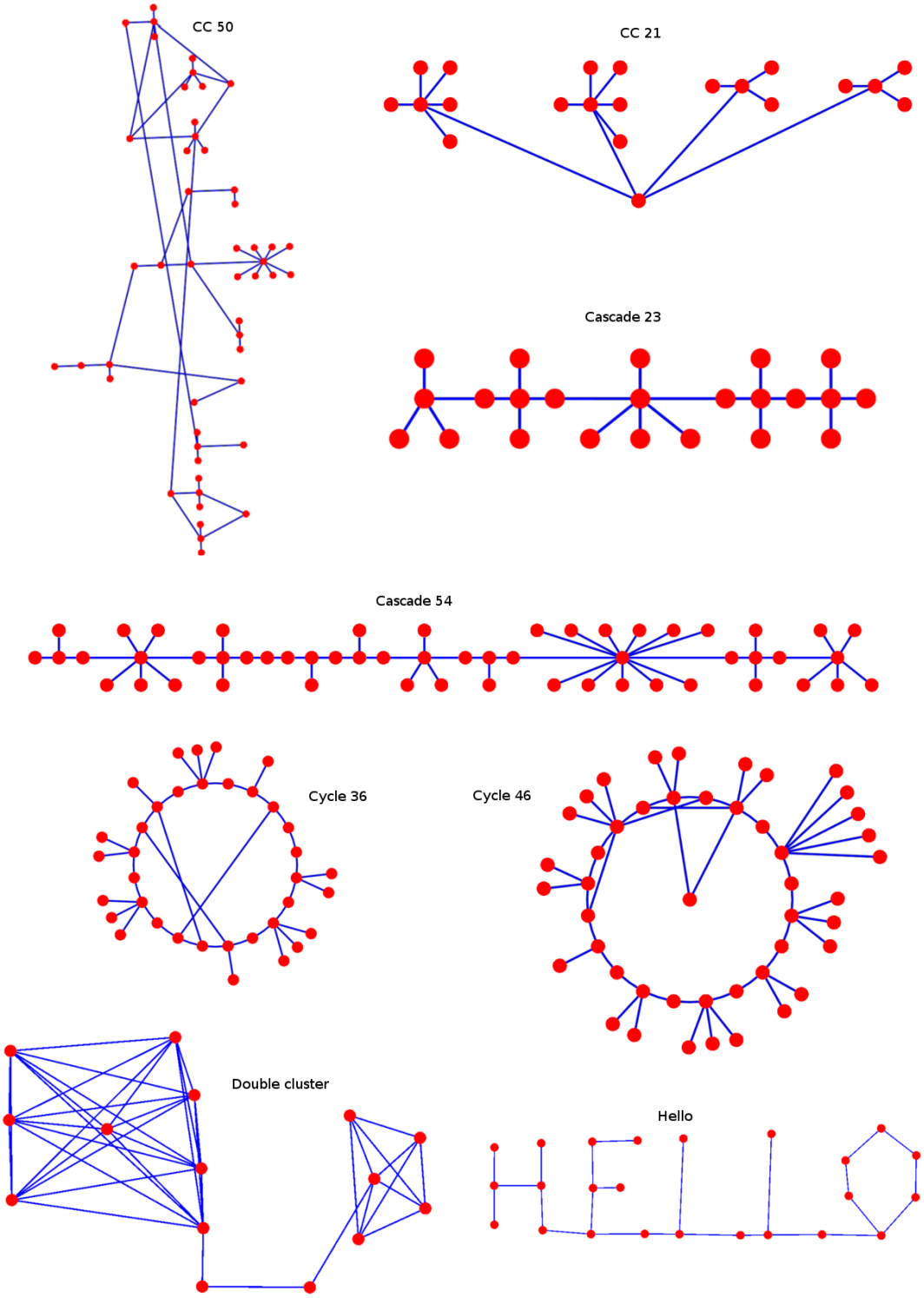


FIGURE 4.11 – Les layouts correspondant aux objectifs pour les 8 graphes utilisés pour cet exemple

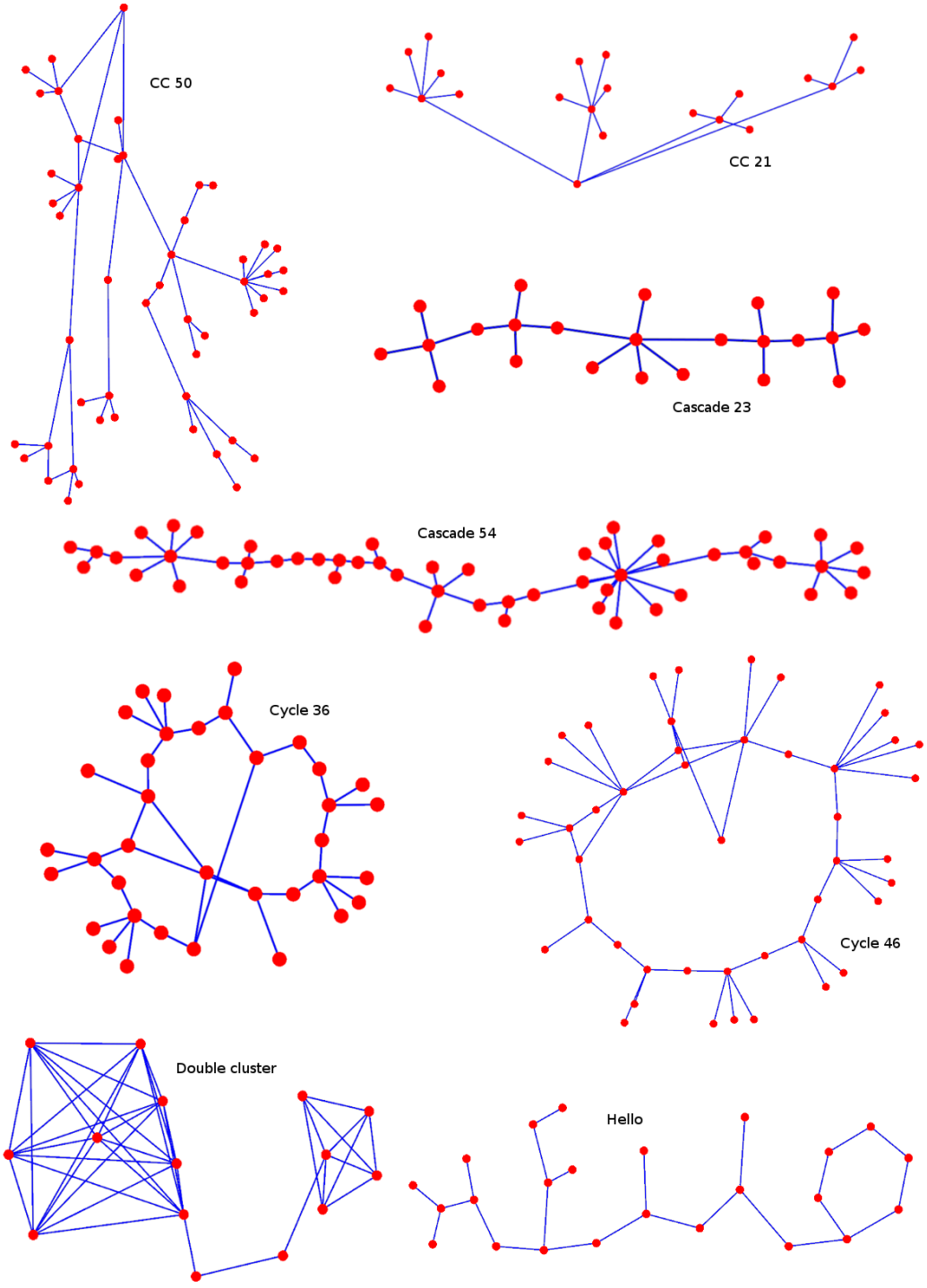


FIGURE 4.12 – Les layouts résultants des meilleurs génomes obtenus avec la nouvelle structure

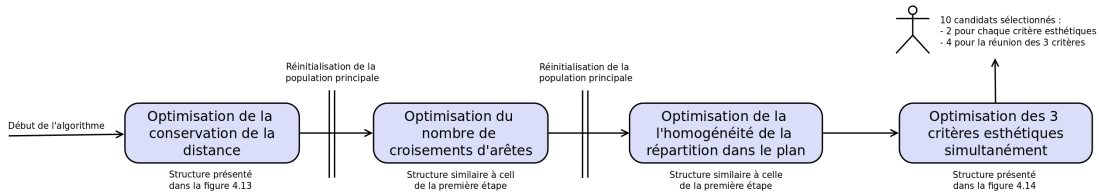


FIGURE 4.13 – Les 4 étapes qui composent cette première structure interactive

suffit de vérifier lors de la sélection d'un candidat que ce dernier n'est pas trop ressemblant aux candidats précédents. De nouveau, c'est la métrique de similarité qui est utilisée pour cela.

La première structure présentée est une structure d'initialisation. Cela signifie qu'elle n'a pas besoin d'évaluations utilisateurs afin de s'exécuter. De ce fait, elle ne prend que très peu en compte les préférences de l'utilisateur. Son objectif est donc de fournir différents graphes pouvant être intéressants dans le but d'amasser plusieurs évaluations. La deuxième structure présentée se base justement sur ces évaluations pour essayer de produire des graphes correspondants aux besoins de l'utilisateur.

4.4.1 Première structure interactive : Optimisation de métriques esthétiques

Comme nous venons de le voir, cette première structure est une structure d'initialisation. Il est donc probable qu'elle soit la première d'une séance d'évaluation. Cette structure se base en grande partie sur un layout modèle, dont elle ne doit pas trop s'éloigner. Lors de la première exécution, ce layout correspond au dessin de base fourni par l'utilisateur. Lors des exécutions suivantes, il s'agit du candidat sélectionné par l'utilisateur au sein de la liste des candidats générés par l'exécution précédente. Ce processus itératif permet de s'éloigner peu à peu du dessin de base tout en conservant les propriétés graphiques importantes du dessin au travers des choix de candidats de l'utilisateur.

Cette métrique tente d'optimiser les 3 critères esthétiques suivants :

- La conservation des distances, soit la minimisation de l'écart-type du ratio $\frac{\text{distance topologique}}{\text{distance euclidienne}}$ entre chaque paire de sommets du graphe.
- Le nombre de croisements d'arêtes
- L'homogénéité de la répartition des sommets sur le dessin

La figure 4.13 présente une vue schématique des différentes étapes qui composent cette structure.

Une étape est dédiée à chacun de ces critères. Une quatrième étape utilise ensuite les scores minimums et maximums obtenus durant ces 3 premières étapes pour essayer d'obtenir un graphe optimisant ces 3 métriques en même temps. De plus, des étapes intermédiaires très courtes sont ajoutées avant les étapes 2 et 3 afin de réinitialiser la population principale à un état aléatoire. Cela permet d'éviter que l'optimisation d'une première métrique biaise le calcul d'un minimum et d'un maximum pour la suivante.

La première étape est basée sur la structure présentée par la figure 4.14.

Cette étape reste relativement simple et reprend les éléments vus dans les exemples précédents. Seule la définition du score change, puisqu'il s'agit maintenant d'une somme de deux

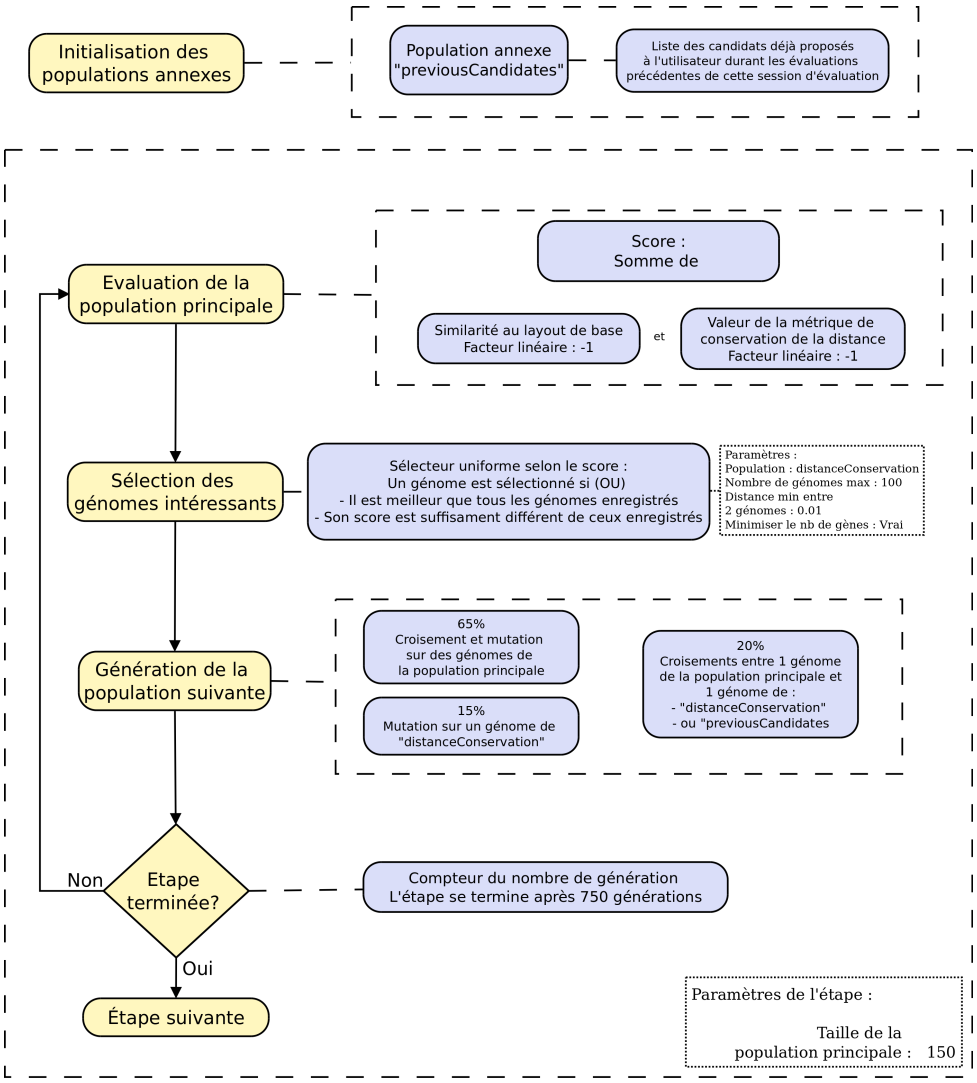


FIGURE 4.14 – La première étape de la structure interactive d'initialisation

métriques. Ces deux métriques tendent idéalement vers 0, un coefficient de -1 leur est donc appliqué à toutes les deux afin de permettre à l'AG d'avoir à maximiser le score.

Le fait d'utiliser la métrique de similarité conjointement à un critère esthétique a un effet très intéressant. En effet, le fait d'améliorer le critère esthétique impliquera en même temps une moins bonne similarité. Les modifications du layout ont donc un aspect positif sur le score que si elles améliorent plus le critère esthétique qu'elles diminuent la similarité. Cela permet d'éviter que des modifications qui modifieraient complètement le layout ne soient acceptées.

Les deux autres critères esthétiques sont traités exactement de la même manière. C'est la moyenne du nombre de croisements d'arêtes par sommet qui est utilisée dans la deuxième étape. La troisième étape tente quant à elle d'homogénéiser la répartition des sommets dans le dessin. Pour cela, elle utilise la métrique de densité graphique, dont elle essaye de minimiser l'écart-type. Pour rappel, cette métrique est calculée pour chaque sommet et correspond à la moyenne de l'inverse des carrés des distances à chaque autre sommet du graphe (voir paragraphe 2.4.2). Une population annexe est associée à chacune de ces étapes. Elle permet de stocker les meilleurs génomes pour le critère de l'étape. Ces populations sont surtout très utiles pour la dernière étape

de la structure.

En effet, cette dernière étape a pour objectif l'obtention de génomes optimisant les 3 critères esthétiques en même temps. Pour cela, elle considérera comme score la somme des scores des étapes précédentes. Cependant, n'utiliser que cette somme risquerait de donner plus de force à une des métriques par rapport aux autres. En effet, différentes métriques peuvent tout à fait prendre des valeurs sur des intervalles très différents. Si cela se produit, un critère utilisant un intervalle beaucoup plus large aura un impact sur le score final beaucoup plus important.

Il est donc nécessaire de normaliser les valeurs de ces différentes métriques. La première solution serait d'utiliser les valeurs moyennes obtenues sur la base de données. Cependant, cela ne permet pas de prendre en compte le graphe et le layout de base fourni pour l'exécution. Il est par contre possible d'utiliser les populations annexes des 3 premières étapes pour cela. L'intervalle considéré pour chaque métrique est donc défini par le score minimum et maximum des génomes contenus dans la population annexe correspondait à la métrique. C'est donc la somme de ces 3 scores ainsi normalisés qui sert de score à la dernière étape.

La figure 4.15 présente la structure complète de cette quatrième et dernière étape. La structure globale reste similaire à celles des étapes précédentes. Cette étape est toutefois relativement plus longue, et la taille de la population principale est plus importante (400 génomes au lieu de 150 dans les étapes précédentes). Cela permet une recherche plus poussée lors de l'optimisation des 3 métriques simultanément. Les sélecteurs des 3 populations annexes des étapes précédentes sont encore présents pour sélectionner de meilleurs génomes pour leur critères si certains se présentent.

Une quatrième population annexe est aussi mise en place durant cette étape. Son objectif est de stocker les génomes optimisant les 3 critères esthétiques. Cette population est nommée "meltingPot", et c'est à cette dernière qu'est dédiée le 4^{e} sélecteur ajouté à l'étape.

Un point important de cette structure qui n'a pas encore été abordé concerne la sélection des candidats. En effet, tout ce qui concerne cette sélection se déroule en fin d'algorithme génétique, à l'aide d'un filtre défini sur chaque population annexe et agissant justement à la fin de l'AG.

Ces filtres sont des **extractorToAnotherPopulation**. Ils ne servent pas à décider si un génome doit ou non être conservé dans la population (la valeur renvoyée par ces filtres est toujours **vrai**, pour indiquer que le génome doit être conservé), leur rôle étant de copier des génomes dans une autre population.

Pour cela, ils ont besoin du nom de la population de destination et d'un sélectionneur de génomes qui indique si oui ou non le génome doit être copié dans l'autre population. Ces filtres vont donc nous permettre de transférer les génomes sélectionnés dans une population de candidats. Le sélectionneur utilisé est un **CriterionGenomeFilter**, qui permet de sélectionner les k meilleurs génomes d'une population, en triant les génomes selon un score précisé par l'utilisateur. Ici, k est fixé à 2 pour les 3 populations annexes des critères esthétiques et 4 pour celle de la dernière étape.

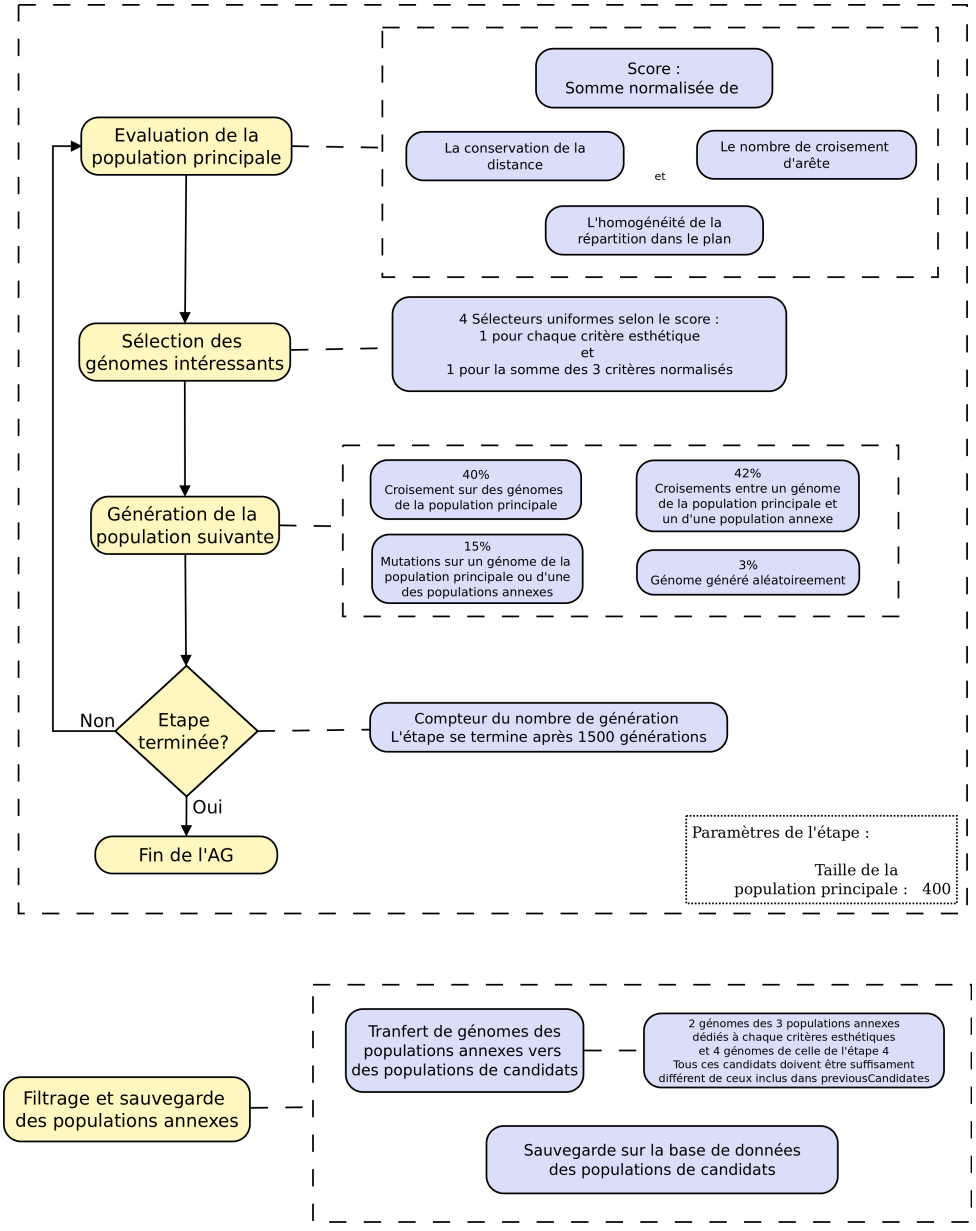


FIGURE 4.15 – La 4e et dernière étape de la structure interactive d'optimisation des critères esthétiques

Cependant, si le score considéré est directement celui d'une étape, il y a un risque de sélectionner un layout identique à celui d'un candidat déjà présenté à l'utilisateur (lors d'une évaluation précédente). Pour éviter cela, un critère de nouveauté est utilisé, qui se base sur la population "previousCandidate". Un génome ne peut être sélectionné que si sa similarité avec l'ancien candidat le plus proche de lui est supérieure à 0.1. Cela permet d'éviter toute proposition trop proche d'un candidat précédent. Cependant, si trop de candidats ont déjà été proposés, cela peut conduire à une diminution de la qualité des candidats sélectionnés, car tous les bons génomes ont déjà été soumis à l'utilisateur. Ce filtrage en fonction de la valeur de nouveauté par rapport aux candidats précédents est réalisé avec un **CriterionFilter**, déjà utilisé dans la structure du deuxième exemple.

Résultats et évaluation

Cette structure ne peut être utilisée que dans le cadre d'une session interactive. Les exécutions peuvent quand même être suivies grâce à l'interface de monitoring, mais les courbes ne sont pas des plus intéressantes. Cette possibilité de suivi a toutefois été fortement utilisée lors de la mise au point de la structure afin de paramétrer correctement les différentes étapes. De plus, dans sa version actuelle, la structure utilise des scores qui sont fortement dépendants du layout sur lequel se base l'exécution (à cause de l'utilisation massive de la métrique de similarité). De ce fait, les scores sont très variables et ne sont que peu indicatif du résultat finalement obtenu.

L'évaluation d'un graphe peut pour l'instant prendre deux formes. La première, la plus simple, consiste à sélectionner un unique candidat, qui correspond au candidat « préféré » de l'utilisateur. Cette sélection est obligatoire, et le layout sélectionné est utilisé comme layout de base durant l'exécution suivante. Le deuxième type d'évaluation permet d'attribuer un score (entre -10 et 10) à chaque candidat. De plus, il est possible de faire porter cette évaluation sur une sélection de sommets (au lieu de l'ensemble du layout). Cela permet par exemple d'indiquer que tel ou tel motif est intéressant ou non.

La structure présentée dans ce premier exemple interactif n'utilise quant à elle que la sélection du candidat préféré. Cela est en grande partie lié au fait que c'est une tâche d'initialisation, qui doit donc pouvoir être exécutée sans avoir la moindre évaluation utilisateur. Mais cela induit aussi un effet immédiat sur la capacité de cette tâche à fournir des layouts qui répondent précisément aux besoins de l'utilisateur. Cela correspond justement à l'objectif de la structure de l'exemple suivant.

Les résultats présentés correspondent aux layouts de certains des candidats proposés par l'AG durant une session interactive. Cette session se base sur le graphe présenté par la figure 4.16. Ce graphe contient un graphe de Petersen en son centre, et 4 branches autour de ce dernier. Chaque branche contenant un type de sous graphe différent, un cycle, un petit arbre, un losange avec une diagonale, et une étoile. Le layout de ce graphe a été initialement créé manuellement. Une des branches contient 2 liens à des sommets non contigus du graphe de Petersen, comme nous le verrons cela pose de grosses difficultés à GaGEM pour reproduire ce motif.

Deux exécutions successives de la tâche d'initialisation sont tout d'abord exécutées sur ce graphe. La figure 4.17 présente 4 layouts issus de la première exécution. Le premier de ces layouts sert de modèle pour la deuxième. La figure 4.18 présente quant à elle 4 layouts issus de cette dernière.

Chaque image présente un layout issu de chaque population annexe. Dans l'ordre de gauche à

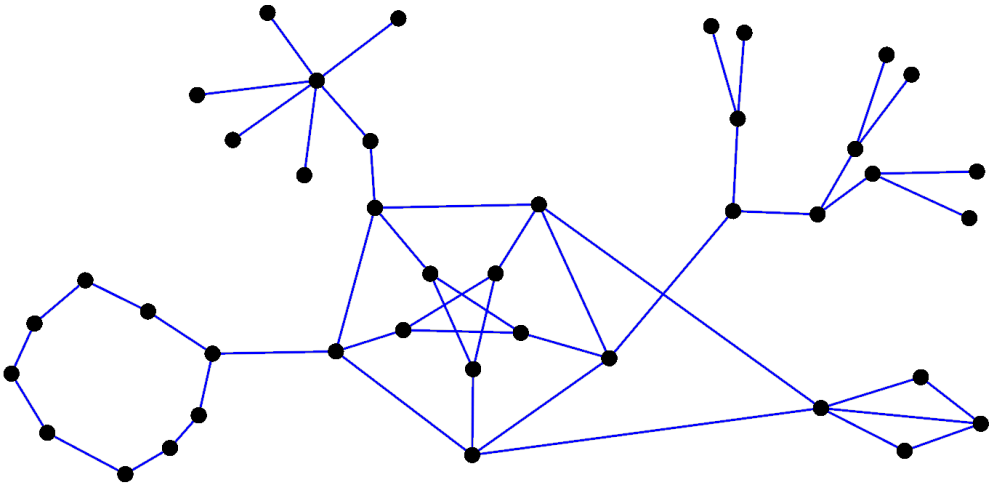


FIGURE 4.16 – Le graphe utilisé lors de la session interactive

droite puis de bas en haut, ils correspondent aux trois critères esthétiques (croisements d'arêtes, densité graphique puis conservation de la distance). Le 4e layout est issu de la quatrième population, qui tente d'optimiser les 3 critères en même temps. Certains de ces graphes ont été évalués en vue de l'exécution de la dernière structure. Ce sont ceux pour lesquels un score est affiché à côté du graphe. De plus, pour certains d'entre eux, des sommets sont colorés en rouge, cela indique que l'évaluation concerne uniquement cette sélection de sommets.

4.4.2 Deuxième structure interactive : Prise en compte des évaluations utilisateurs

L'objectif de cette deuxième structure est donc de prendre compte les évaluations réalisées par l'utilisateur. Ces évaluations portent sur les candidats précédemment proposés à l'utilisateur dans le cadre d'une session interactive. Cette tâche n'est pas une tâche d'initialisation, et ne peut donc pas être utilisée comme première tâche lors d'une session interactive. Dans le cadre des résultats présentés, cette tâche sera utilisée lors de la 3e exécution de notre session interactive.

Cette structure se base principale sur le module **GraphNoveltyCriterion** déjà utilisé dans le cadre de la recherche guidée par la curiosité dans le 2e exemple. Cependant, elle fait un usage très différent de ce module. Pour rappel, il permet de comparer un layout à tous ceux d'une population pour trouver ses plus proches voisins. Pour la recherche guidée par la curiosité, seul nous intéressait le plus proche voisin et l'objectif était de maximiser la distance à ce dernier.

C'est une autre fonctionnalité de ce module qui est utilisé ici. Tout d'abord, la recherche des plus layouts le plus similaires se fait maintenant au sein d'une population contenant les candidats précédents évalués. Aucun génome n'est donc ajouté à cette population au cours de l'exécution. De plus, le voisinage considéré correspond maintenant aux 10 layouts les plus proches. Enfin, ce n'est pas seulement la similarité à ces layouts qui est considérée, mais aussi le score qui leur a été attribué par l'utilisateur. Dans les cas où seule une partie du graphe est évaluée, seuls

les sommets sélectionnés sont considérés lors du calcul de la similarité (la position des autres sommets n'a donc aucun impact sur le résultat).

L'agrégation de ces différents scores correspond à la formule suivante :

$$\sum_{l \in L_{10}} -sim(l) * score(l)$$

Avec L_{10} l'ensemble des 10 layouts les plus similaire au layout testé.

Cette formulation du score va obliger l'AG à tenter de diminuer au maximum toutes ces similarités en donnant plus d'importance à certains layouts en fonction du score qui leur a été attribué. Cependant, cette définition ne fonctionne correctement que si toutes les évaluations utilisateurs sont positives. En effet, dans le cas d'une évaluation négative, l'AG tente de trouver un résultat le plus lointain possible des layouts concernés, car c'est cela qui lui permet de maximiser le plus fortement le score, la similarité aux autres layouts devenant rapidement négligeable. Nous verrons à la fin de cette partie quelles solutions peuvent être envisagées à ce problème.

Le reste de cette structure est relativement classique. Elle se base sur une seule étape, avec un sélectionneur uniforme utilisé pour remplir une population annexe. Seule la sélection des candidats est quelque peu différente de la méthode employée dans la structure précédente. En effet, cette dernière utilisait un filtre afin de s'assurer que les candidats sélectionnés étaient suffisamment différents des candidats déjà soumis à l'utilisateur. Cependant, elle ne vérifiait pas que les candidats sélectionnés étaient différents les uns des autres.

Pour répondre à ce problème, un sélectionneur selon la nouveauté est utilisé comme condition pour le module **ExtractorToAnotherPopulation**. Ce sélectionneur, utilisé pour les populations "novelty" et "noveltyRaw" du 2^{e} exemple, sélectionne des génomes suffisamment différents les uns des autres tout en optimisant un score fourni par l'utilisateur. Ce score se base sur un filtre (**CriterionFilter**) similaire à celui de l'exemple précédent pour éviter les candidats trop proches des précédents et utilise comme score de base celui de l'étape. La structure complète de cette étape est présentée par la figure 4.19.

Deux des layouts obtenus avec cette structure sont présentés par la figure 4.20. Tous n'arrivent pas à prendre l'ensemble des motifs choisis lors des évaluations précédentes. Cela est lié au fait que cette structure converge trop rapidement. De ce fait, il arrive régulièrement qu'elle optimise la similarité à un des layouts au détriment des autres sans arriver à s'éloigner par la suite de cet optimum local. Certains layouts présentent toutefois des caractéristiques très intéressantes. Les deux layouts sélectionnés sont ceux qui présentaient le plus de différence avec les candidats des exécutions précédentes.

Une exécution de la première structure a ensuite été réalisée avec comme modèle de base le premier layout de la figure 4.20. Cela permet de tenter d'optimiser les différents critères esthétiques à partir de ce layout. Ce dernier étant très différent de la plupart des candidats obtenus auparavant, cela permet d'explorer une nouvelle « branche » des dessins de ce graphe. L'objectif étant de construire petit à petit un ensemble d'évaluation permettant d'aboutir au final à un layout répondant correctement à nos attentes. Un des résultats de cette dernière exécution est présenté dans la figure 4.21.

4.4.3 Améliorations possibles de ces structures

Les différentes structures interactives qui viennent d'être présentées sont très loin d'être optimales. En effet, elles présentent une approche simple de la facette interactive de notre système d'AG. Cependant, elles n'exploitent que très peu les fonctionnalités disponible, en terme d'utilisation de modules avancés et d'exploitation de la base de données. De très nombreuses améliorations sont donc envisageables pour aller plus loin dans l'exploration des dessins qu'il est possible de produire pour un graphe.

Le problème énoncé plus tôt concernant les scores négatifs serait par exemple très intéressant à traiter. En effet, évaluer négativement tout ou une partie d'un layout signifie que tout graphe se rapprochant trop de ce layout n'est pas intéressant. L'AG va donc chercher à produire quelque chose de différent. Cependant, avec la définition actuelle du score, cela amène l'AG à être un petit peu « trop » créatif. En effet, s'il arrive à rendre le layout suffisamment différent, cela peut facilement écraser les autres composantes de la somme. Il faudrait donc trouver une définition du score qui permet d'éviter cet effet. Il faudrait aussi équilibrer cet éloignement des layouts mauvais avec le respect de critères esthétiques afin d'éviter la production de candidats trop cassés.

Mais il est aussi possible d'aller beaucoup plus loin, en mettant par exemple en place une recherche guidée par la nouveauté. L'objectif de cette dernière serait de trouver des graphes intéressants pour les critères esthétiques, et étant les plus éloignés possible des candidats proposés jusque-là. Cela permettrait par exemple d'obtenir une très bonne structure d'initialisation. Ce type de recherche pourrait aussi être utilisé dans une structure similaire à la deuxième afin d'éviter toute convergence de la population autour d'un unique candidat. L'objectif serait donc de trouver de nouveaux candidats qui sont compatibles avec les évaluations de l'utilisateur.

L'utilisation de la base de données pourrait aussi offrir de nombreuses possibilités. Par exemple, il serait possible de créer une structure d'initialisation s'appuyant massivement sur les génomes enregistrés lors du dessin d'un graphe déjà connu par le système. En utilisant une comparaison des métriques topologiques des graphes (via le contexte), il serait aussi possible de s'inspirer des génomes utilisés sur des graphes similaires. Cela permettrait par exemple de faire des structures extrêmement courtes (une centaine de générations), qui permettrait de fournir à l'utilisateur des idées de dessins pour son graphe très rapidement.

Il pourrait aussi être intéressant de faire intervenir d'autres critères esthétiques, afin d'augmenter le nombre de voies à explorer lors de l'affinage d'un layout. De telles structures n'ont pas été présentées ici, car nous avons souhaité limiter « au maximum » la complexité des structures des exemples. Il a d'ailleurs été relativement difficile de se fixer sur certaines structures alors même qu'elles présentaient des lacunes évidentes. Cependant, l'objectif de ce chapitre était plus de fournir une idée des possibilités offertes par notre système plutôt que des structures optimales. Ces dernières auraient été extrêmement complexes, et de plus, comme nous le verrons dans la conclusion, nous n'avons pas eu suffisamment de temps à consacrer à la mise au point de telles structures durant la durée de cette thèse.

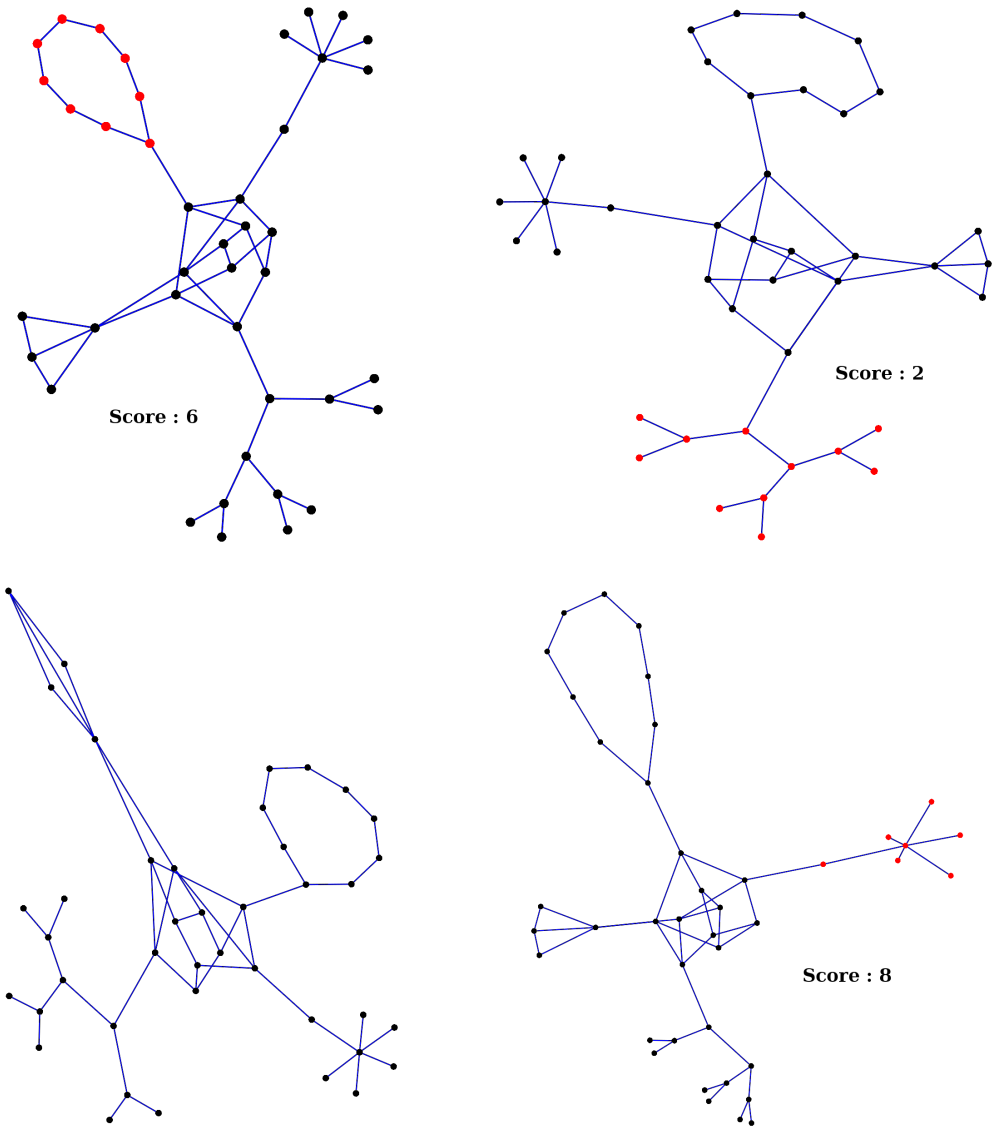


FIGURE 4.17 – Les layouts produits par la première exécution de la structure interactive d'initialisation

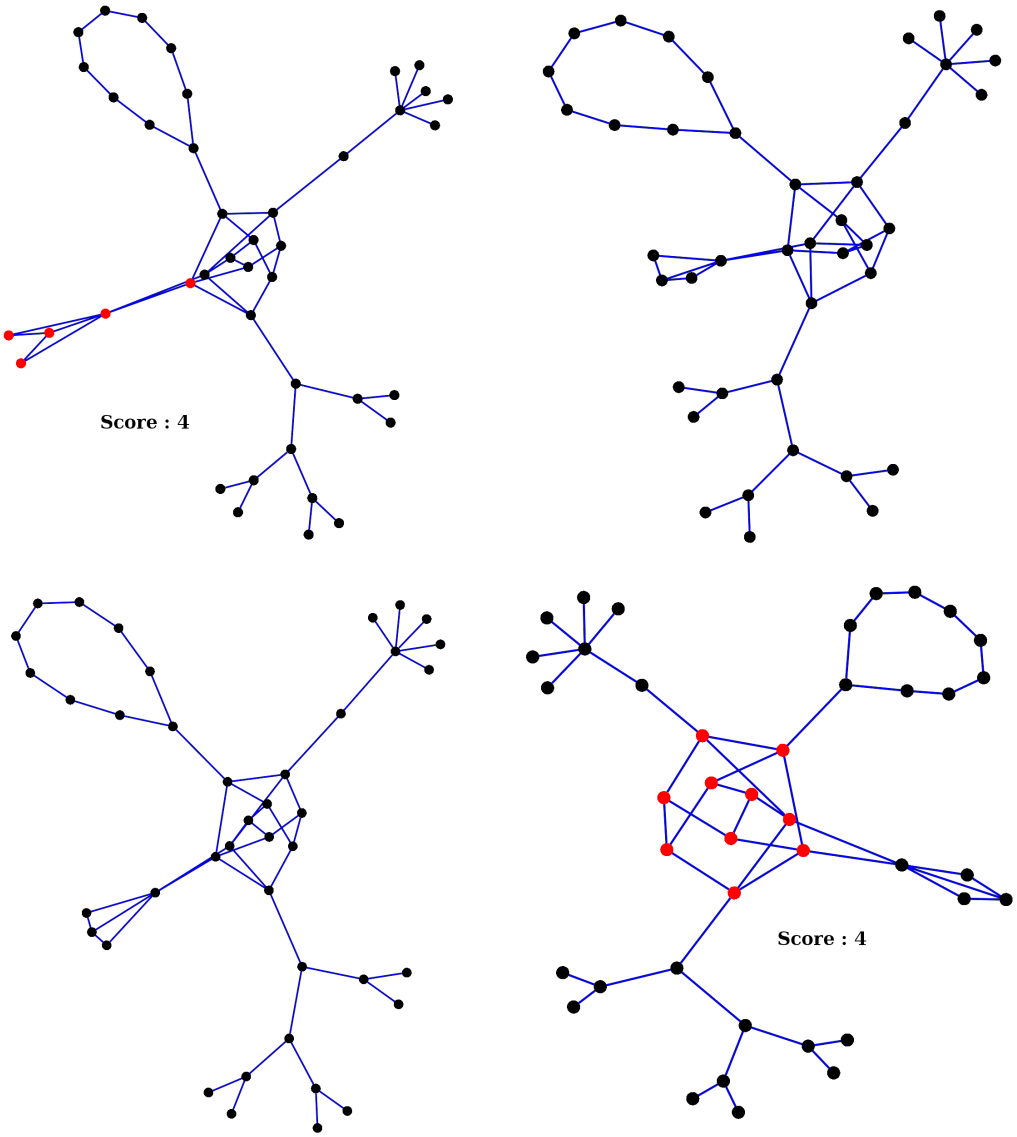


FIGURE 4.18 – Les layouts produits par la deuxième exécution de la structure interactive d'initialisation

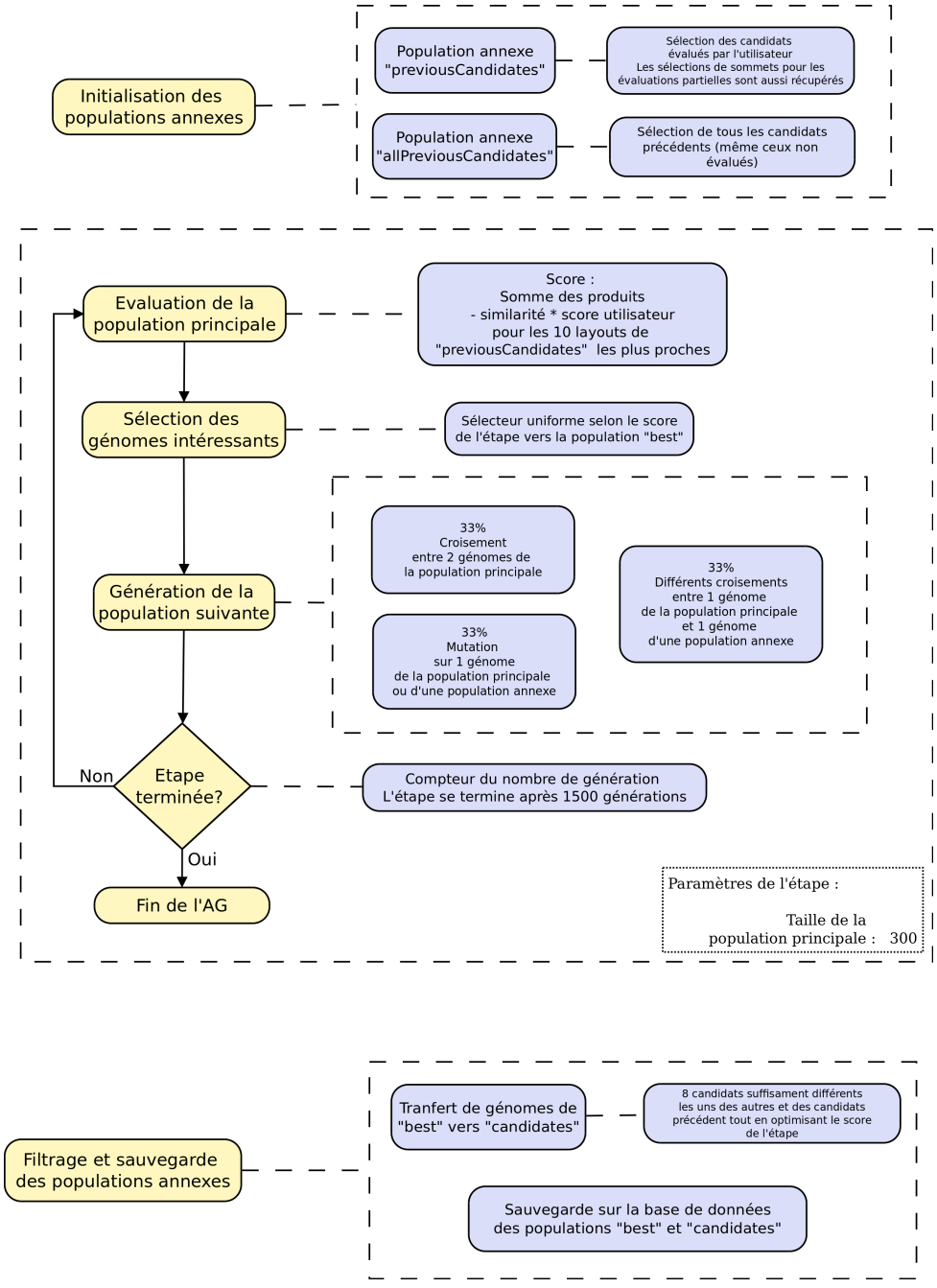


FIGURE 4.19 – La structure interactive permettant de prendre en compte les évaluations de l'utilisateur

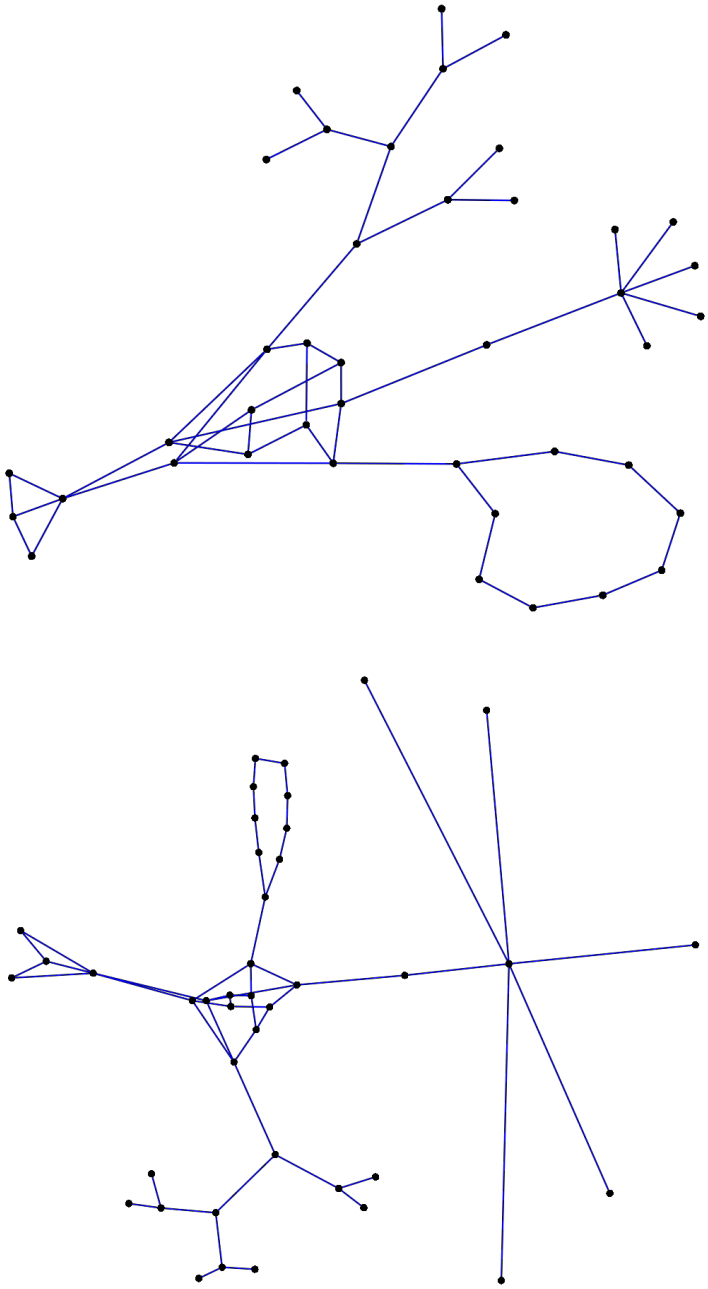


FIGURE 4.20 – Les layouts obtenus suite à l'exécution de la structure prenant en compte les évaluations utilisateurs

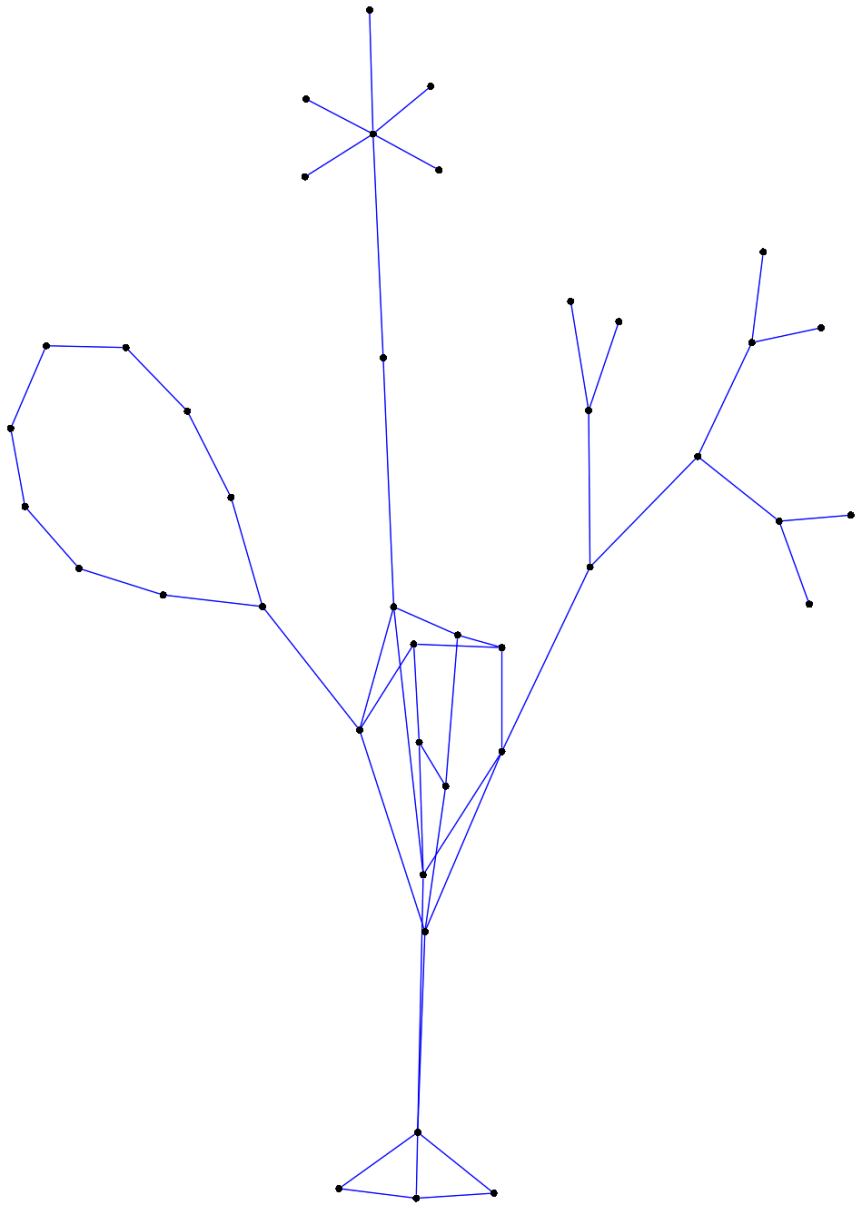


FIGURE 4.21 – Les layouts obtenus après la dernière exécution de la structure d'initialisation

Chapitre 5

Besoins identifiés et solution logicielle

Ce chapitre va être dédié à différentes besoins, techniques ou en termes de fonctionnalités, rencontrés au cours de cette thèse. Ces besoins ont fortement influencés certaines décisions au cours de la mise en place du système. Les réponses apportées prennent la forme de modules spécifiques ou sont plus diffuses au sein de l'ensemble du système. Les parties suivantes aborderont les 3 besoins principaux. La première concerne la distribution des calculs, la deuxième le paramétrage des algorithmes génétiques, et la dernière l'interaction entre les utilisateurs novices et le système.

5.1 Distribution des calculs

La problématique première de cette thèse consiste à assister un utilisateur novice dans le cadre du dessin de graphe. Cette problématique induit un corollaire immédiat sur la puissance disponible pour réaliser cette tâche sur la machine de l'utilisateur, qui doit être considérée comme faible. Ce corollaire est de plus fortement amplifié par l'évolution actuelle des périphériques informatiques orientés vers la mobilité et qui de ce fait sont relativement peu puissants.

Cette limite prend encore plus d'importance dans le cadre des problématiques liées à l'intelligence artificielle. Les méta-heuristiques actuelles sont généralement gourmandes en terme de puissance de calcul, et cela va croissant avec la complexité de l'espace à explorer. Le cas du dessin de graphe pouvant être considéré comme complexe (au moins à la vue de la taille de la communauté scientifique se penchant sur ce point), il n'est pas envisageable d'espérer réaliser tous les calculs sur la machine de l'utilisateur final.

La réponse apportée a été de concevoir un système dans lequel tous les calculs sont exécutés sur des machines distantes dédiées uniquement à cette tâche. Notre architecture se décompose en 4 éléments distincts. La figure 5.1 présente le rôle de chacun et différentes interactions entre ces entités.

L'idée principale reposant derrière cette structure est qu'il est possible d'avoir une importante puissance de calcul sur des machines dédiées à l'exécution continuelle de l'AG. Le système dans son état actuel permet l'exécution d'une centaine de structures simultanément. Nous n'avons pas eu la possibilité de réaliser des tests à plus grandes échelles, mais il est fort possible qu'une gestion plus fine de la mémoire au sein du serveur soit nécessaire pour cela.

Le programme **Genetips**, qui repose en grande partie sur la bibliothèque **genLib**, permet l'exécution de n'importe quelle structure de l'AG, dans le cadre du dessin de graphe. Ce pro-

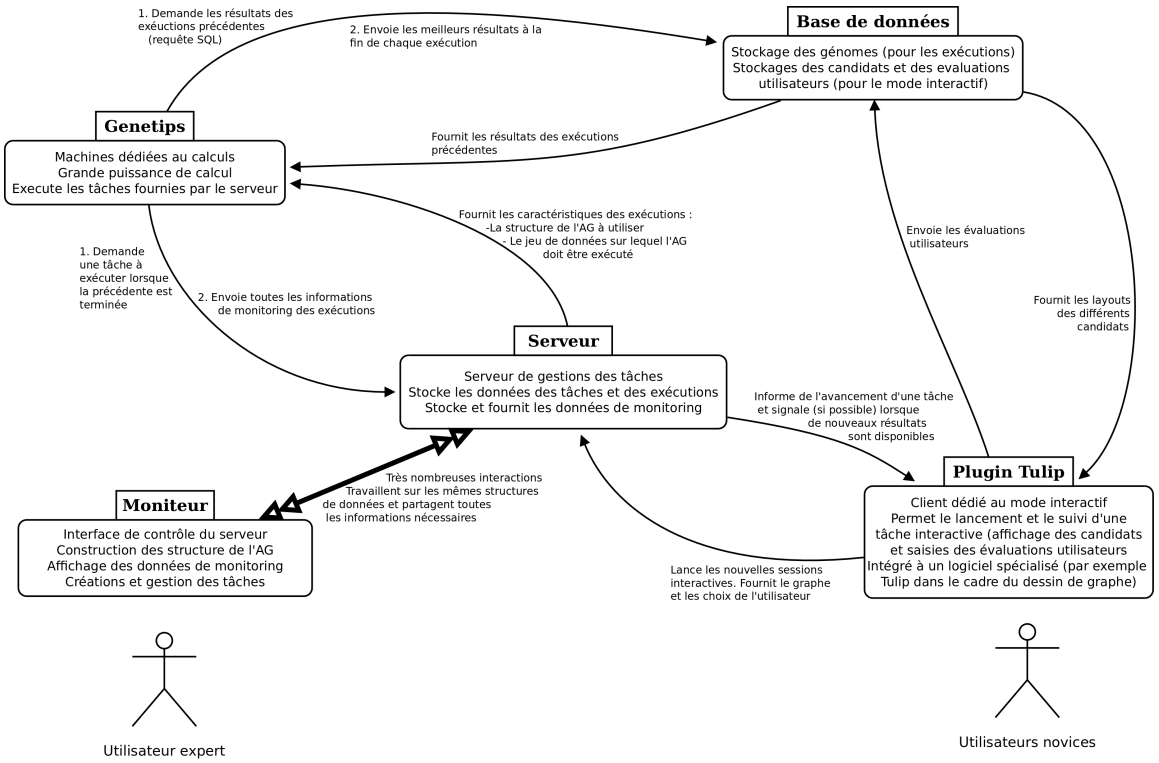


FIGURE 5.1 – Les interactions entre les différents éléments du système mis en place

gramme est intégralement écrit en C++ afin de permettre une exécution la plus efficace possible. Une parallélisation des calculs au sein du client n'a pas encore été mise en place mais elle serait tout à fait possible puisque l'évaluation de chaque génome est réalisée indépendamment des autres. De manière générale, pour chaque cas d'utilisation, il est toujours possible de paralléliser les calculs réalisés par l'évaluateur.

Le serveur et le moniteur sont tous les deux écrits en java. Ce choix a été dicté par des contraintes de temps et le relâchement de la contrainte sur l'efficacité de calcul, puisque ces deux programmes ne sont finalement pas très gourmands en terme puissance de calcul. En effet, la seule action continue du serveur consiste à enregistrer les données de monitoring, tout le reste ne consiste qu'en actions ponctuelles simples en début ou en fin d'exécution. En ce qui concerne le moniteur, ce dernier ne constitue finalement qu'une grande interface graphique permettant d'accéder aux données du serveur. Les différentes interactions entre le serveur et le moniteur seront plus amplement abordées dans la suite de ce chapitre.

Le module de la base de données est sûrement un des plus simples de cette architecture, dans la mesure où il consiste simplement en une base de données PostGreSQL dans laquelle une base a été créée et est accessible pour un utilisateur distant. Lors du premier lancement d'un programme **Genetips** sur cette base, toutes les tables nécessaires seront automatiquement créées.

Le dernier module est le plug-in Tulip. L'objectif de ce dernier est de permettre à un utilisateur final d'interagir simplement avec notre système. Le principe du plug-in consiste donc à pouvoir communiquer avec le serveur, lui fournir un graphe et demander le lancement d'une exécution interactive de l'AG. Ce plug-in n'a pas encore été réellement mis en oeuvre, pour des questions de temps. Toutes les fonctionnalités sont pour l'instant mises en oeuvres dans une section dédiée du moniteur. Cependant, le système de récupération des résultats directement sur

la base de données est déjà utilisé afin de se rapprocher du comportement qu'aurait un plug-in directement intégré dans Tulip.

Comme le montre l'image, il est prévu que les utilisateurs finaux n'interagissent qu'avec le plug-in Tulip. L'utilisateur qui utilise le moniteur, et qui configure le serveur est quant à lui un utilisateur expert. Il est en effet nécessaire qu'il connaisse très finement notre système ainsi que le domaine du dessin de graphes. En effet, comme nous le verrons dans la dernière partie de ce chapitre, il doit créer les structures que l'AG utilisera pour le mode interactif.

Cette architecture répond de manière satisfaisante au problème de l'absence de puissance de calcul sur la machine exécutant le client final. De plus, elle permet aussi l'utilisation de manière intensive de l'AG à des fins de recherche « pure » en permettant l'exécution parallèle d'un grand nombre d'AG et la comparaison des résultats obtenus au cours et à la fin de ces exécutions. Cette possibilité est intimement liée à la possibilité de paramétrer finement une structure d'AG, ce qui correspond exactement au besoin traité dans le paragraphe suivant.

5.2 Paramétrage et suivi des exécutions

Notre volonté de pouvoir définir très finement la structure de chaque algorithme génétique exécuté, et de stocker cette structure au sein d'un fichier modifiable manuellement (un fichier XML dans notre cas) a un conséquence immédiate. Il faut pouvoir créer ces fichiers de description. La première possibilité consiste à le faire manuellement, mais il est très difficile de mémoriser toutes les caractéristiques de chaque balise. En effet, il existe actuellement plus de 80 modules différents, acceptant un ou plusieurs types d'enfants, avec des relations d'héritages entre eux. Nous avons donc rapidement estimé qu'il était nécessaire de faciliter la création de ces structures.

La solution proposée consiste en une interface graphique dans laquelle chaque module apparaît avec l'ensemble de ses paramètres et de ses enfants. De plus, un menu dédié à chaque module permet quand cela est possible d'ajouter un enfant. Ce menu présente uniquement les modules qui sont compatibles avec les espaces disponibles.

Une contrainte liée à cette interface concerne la possibilité d'ajouter facilement de nouveaux modules au système. Pour répondre à cela, cette interface utilise plusieurs fichiers XML décrivant l'ensemble des modules disponibles. Actuellement deux fichiers sont utilisés, le premier décrivant l'ensemble des blocs génériques (définis au sein de la bibliothèque **genLib**), et le deuxième contenant tous ceux dédiés uniquement au cas d'utilisation des dessins de graphe. Lors de l'ajout d'un nouveau module, il suffit d'ajouter dans un de ces deux fichiers sa description pour qu'il soit automatiquement pris en compte par le programme.

L'ensemble des structures XML est stocké au sein du serveur. Lors d'une action d'édition, la structure est d'abord rapatriée sur le moniteur. Les modifications sont ensuite réalisées localement puis envoyées au serveur. Ce dernier garde un suivi des modifications réalisées sur chaque structure afin de pouvoir par la suite mesurer l'impact d'une modification sur les résultats obtenus.

L'interface du moniteur permet aussi un mécanisme de copier-coller d'un module et de ses enfants, ou uniquement des paramètres d'un module. Cela peut être fait au sein d'une même structure, mais aussi entre deux structures. Ce mécanisme prend toute son utilité dans le cadre d'édition de structure complexe. À titre d'exemple, la structure du deuxième exemple du chapitre

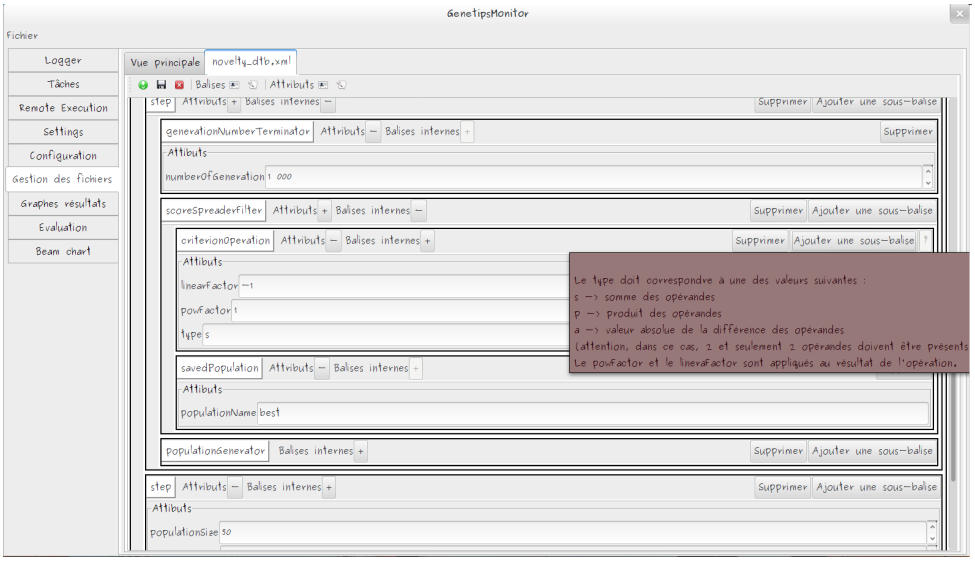


FIGURE 5.2 – L'interface d'édition des fichiers XML
La pop-up affichée fournit la documentation du bloc **CriterionOperation**

4 contient 146 blocs, dont certains très similaires les uns aux autres.

La figure 5.2 présente une capture d'écran de l'interface d'édition des structures. Au moment de la capture, la souris survolait le bouton permettant d'afficher la documentation d'un bloc, d'où la présence de la pop-up.

Une telle capacité de paramétrage est cependant inutile s'il n'est pas possible d'avoir une idée précise de l'impact des modifications sur les résultats. Même sans avoir besoin de comparer deux structures différentes, il est important d'avoir un retour précis sur le déroulement d'un AG pour bien comprendre l'effet des différents modules disponibles. Pour cela, un système de suivi (monitoring) des exécutions a été mis en place.

Cette fonctionnalité a deux aspects. Tout d'abord, au niveau de l'algorithme génétique, il est possible d'indiquer depuis n'importe quel endroit du code source qu'une valeur doit être envoyée au serveur. Chaque valeur est caractérisée par son nom, l'heure à laquelle elle a été enregistrée, le niveau d'importance de cette valeur (afin de pouvoir limiter la quantité de données envoyées) et la valeur à enregistrer. L'enregistrement d'une valeur est très facile (un simple appel à une fonction) et ce mécanisme a été régulièrement utilisé lors de la mise au point de nouveaux modules.

Actuellement, peu de valeurs sont enregistrées par défaut. Des modules de suivis des populations annexes sont disponibles, mais aucune statistique avancée n'a pour l'instant été mise en place. Il serait très intéressant de s'inspirer des travaux réalisés par la communauté pour mettre en place des métriques de suivi du déroulement de l'AG réellement performante. De plus, il serait ensuite possible de mettre en place des conditions de fin d'étape dépendant de ces mesures. De nombreux travaux ont été réalisés sur ce point sur les mesures de diversité de la population.

Du côté du moniteur, il est possible de suivre des exécutions particulières, et pour chacune d'elle, l'utilisateur choisit quelles valeurs il souhaite afficher.

Le serveur se place entre les exécutions et le moniteur. Son rôle est d'enregistrer toutes les

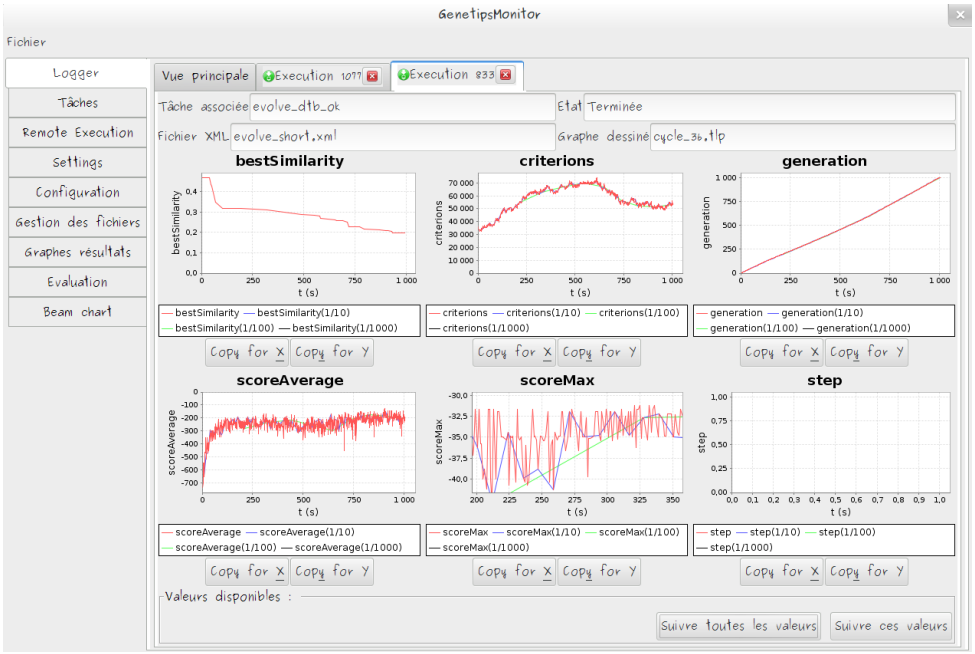


FIGURE 5.3 – L'interface de suivi des valeurs d'une exécutions

valeurs afin de les fournir au moniteur lorsque ce dernier le souhaitera. Cela permet en particulier de consulter les valeurs enregistrées même après la fin de l'exécution. Afin de limiter le nombre de valeurs enregistrées, le serveur ne garde que les 1000 dernières valeurs, puis 900 valeurs de 10 en 10, puis 900 de 100 en 100, puis 900 de 1000 en 1000. Cet échantillon de valeurs permet généralement d'avoir une idée de l'évolution de valeurs au cours de longues exécutions. Cependant, ce système d'échantillonnage est amené à être amélioré, en fournissant par exemple les écarts-types des blocs de valeurs anciens, et avec choix des valeurs à conservées pour maximiser l'information.

La figure 5.3 présente une capture d'écran de l'interface de suivi d'une exécution. Cette interface permet de facilement zoomer sur des parties spécifiques des courbes. De plus, il est aussi possible de créer des graphiques présentant deux valeurs l'une par rapport à l'autre.

Un dernier élément dont la mise en œuvre n'est toutefois pas complètement terminée concerne la génération du résumé d'une exécution. Son objectif est de permettre la comparaison rapide de nombreuses exécutions utilisant des structures différentes. Le résumé d'une exécution correspond à des valeurs caractéristiques (minimum, maximum, moyenne, écart-type) pour chaque valeur enregistrée. L'objectif de l'interface associée aux résumés sera de pouvoir facilement représenter ces valeurs en les regroupant par graphe et/ou par structure. Cela sera indispensable pour pouvoir paramétrer très finement des structures complexes. En effet, l'impact de la modification d'un paramètre peut être masqué localement à cause du caractère aléatoire des AG. Mais cet effet de masquage disparaît, ou est en tout cas fortement diminuer, si la comparaison porte sur un échantillon représentatif d'exécutions.

5.3 Gestion des exécutions par tâches et mode interactif

La dernier besoin identifié a été le fait de choisir, lorsqu'un calculateur est disponible, quelle tâche doit être effectuée. Initialement, nous avions prévu de réaliser cette tâche manuellement.

Cette possibilité reste toujours disponible, puisqu'il est possible de lancer **Genetips** indépendamment de tout serveur en ne lui fournissant qu'une structure XML, un graphe sur lequel travailler et les informations de connexion à la base de données. Cependant, réaliser cette tâche manuellement a rapidement représenté une perte de temps importante, en particulier lors de la mise à jour d'un fichier XML.

Nous avons donc profité de la mise en place du serveur de suivis des exécutions pour lui ajouter la fonction de fournir les informations de lancement d'une exécution. Pour cela, nous avons mis en place les tâches. Une tâche correspond à un ensemble de graphes et de structure XML, le serveur s'occupant d'énumérer les couples possibles petit à petit. Il est possible de limiter le nombre de cycles à réaliser pour une tâche, et de définir une priorité afin de favoriser une tâche lorsque plusieurs sont actives simultanément.

De plus, une tâche peut être prévue pour le mode interactif du système. Dans ce cas, cette tâche sera proposée à l'utilisateur final lorsqu'il demandera au système de dessiner un graphe via le plug-in dédié. Les tâches du mode interactif sont identiques aux tâches classiques, sauf qu'elles ne peuvent être activées en dehors d'une requête d'une utilisateur final. De plus, leur liste de graphes n'est pas utilisée puisque c'est toujours le graphe de l'utilisateur qui est fourni à **Genetips**.

Une autre différence entre les tâches interactives et les tâches classiques concerne l'utilisation d'une population de candidats. Comme nous l'avons vu dans les deux derniers exemples du chapitre 4, les tâches interactives doivent utiliser au moins une structure utilisant une population de candidats, **CandidatesPopulation**, pour indiquer quels dessins doivent être présentés à l'utilisateur à la fin de la tâche.

Il est possible d'ajouter une description à chaque tâche. Cette description est fournie à l'utilisateur novice lorsqu'il doit choisir quelle tâche interactive il souhaite utiliser pour son graphe. Il est envisageable de dédier certaines structures de GA à des types de graphes spécifiques, comme les graphes de voies métaboliques, qui présentent des spécificités importantes en termes de représentations. Dans ce cas, une description est des plus utiles pour aider l'utilisateur novice dans son choix. Il est toutefois préférable de limiter le nombre de choix possibles afin de ne pas retourner au problème du choix d'une méthode de dessin dans une liste immense.

Chapitre 6

Conclusion

6.1 Apprentissage et dessin de graphe

L'ensemble du travail de cette thèse prend place dans deux domaines distincts. Le premier est la visualisation de données, et même plus précisément la construction de layout pour un graphe, et le deuxième est l'apprentissage. Cette dichotomie apparaît très clairement dans la structure de ce manuscrit, à travers les chapitre 2 et 3 qui sont chacun dédiés à un de ces domaines, et le chapitre 4 qui présente l'association de ces deux domaines à travers la solution proposée.

Cette séparation est très courante dans les travaux de recherche liés à l'apprentissage. Il est presque toujours possible d'identifier le système d'apprentissage, qui correspond à la mise en œuvre de la méta-heuristique choisie, et le cas d'application dans lequel ce système est utilisé. Cette séparation a toujours été très sensible au cours de ces trois années, et un équilibre a dû être trouvé entre ces deux domaines. L'objectif final étant d'aboutir à une solution la plus complète et fonctionnelle possible.

6.1.1 La genèse d'un projet

Le sujet initial de cette thèse était « Dessin de graphes dynamiques à l'aide d'une méthode d'apprentissage ». L'objectif était alors de mettre au point une méthode qui pourrait apprendre à dessiner dynamiquement des graphes en adaptant le dessin au fur et à mesure des ajouts et des suppressions d'éléments dans le graphe.

L'idée derrière cet objectif était qu'une méthode qui aurait appris à dessiner des graphes pourrait facilement s'adapter à l'aspect dynamique d'un graphe.

À partir de là, plusieurs contraintes ou besoins sont apparus pour pouvoir construire une solution. La première est assez intuitive : pour pouvoir s'adapter à l'évolution d'un graphe, il faut pouvoir dessiner ce graphe avant qu'il évolue. Il était donc nécessaire d'avoir une méthode de dessin de graphes statiques. L'interrogation suivante immédiate a donc été de savoir comment dessiner correctement un graphe.

Comme nous l'avons vu dans le chapitre 2, centré sur le dessin de graphes, de très nombreuses méthodes de dessins existent. Mais elles sont souvent difficiles d'accès et ne produisent qu'un seul layout par graphe. L'usage d'une seule de ces méthodes ne permet donc généralement pas une adaptation fine aux besoins de l'utilisateur, ce qui devrait être pourtant nécessaire pour la production d'un « bon » layout.

C'est à partir de ce point que le sujet de cette thèse a évolué vers sa forme finale : « Assister l'utilisateur novice dans le cadre du dessin de graphe à l'aide d'une méthode d'apprentissage ». En effet, l'obtention d'une méthode de dessin automatique pouvant s'adapter aux besoins de l'utilisateur répondait très bien à la problématique de l'assistance à un utilisateur novice. L'objectif devenait de pouvoir fournir différents dessins pour un même graphe et de laisser l'utilisateur choisir ceux qui correspondaient le mieux à ses besoins.

Nous verrons dans la partie dédiée aux perspectives futures que l'adaptation de notre système au dessin de graphes dynamiques serait relativement facile.

La mise en place d'une méthode d'apprentissage au sein d'un système interactif soulève aussi une autre contrainte. En effet, l'interactivité nécessite un temps de réponse du système relativement court. Or, les méthodes d'apprentissages classiques nécessitent généralement une grande quantité de calculs. Il nous a donc semblé nécessaire de proposer une méta-heuristique qui permettrait des situations d'apprentissages, mais aussi des situations de restitution rapide des « connaissances ». De plus, dans l'objectif de conserver une interactivité et une exploration dépendantes des besoins des utilisateurs, la séparation entre apprentissage et restitution doit être la plus fine possible. L'idée était alors de proposer un système réalisant continuellement des tâches d'apprentissage pur tout en ayant la possibilité de mettre en œuvre des tâches de restitution plus ou moins longues en parallèle.

6.1.2 Une proposition de solution

Ces deux contraintes nous ont guidés vers la solution proposée dans le cadre de cette thèse.

GaGEM a été mis au point afin de nous permettre de générer des layouts différents. Nous avons pu montrer qu'il était possible de générer des layouts intéressants très différents de ceux obtenus avec la version originale de GEM. Il suffisait ensuite de mettre au point une méthode d'exploration de l'espace des paramètres.

C'est dans cet objectif qu'a été mis au point notre système d'algorithme génétique (AG). Cependant, comme nous l'avons vu tout au long de ce manuscrit, la définition d'un bon layout (et donc de bons paramètres) dépend énormément du graphe et des besoins de l'utilisateur. Par extension, la manière de réaliser une exploration de l'espace des paramètres dépendra beaucoup de l'objectif souhaité. C'est pour permettre cette adaptabilité que nous avons mis au point un système aussi paramétrable. De nombreuses fonctionnalités découlent directement de cette volonté de pouvoir contrôler finement les structures de nos algorithmes génétiques.

Cependant, le monde des algorithmes génétiques comme celui du dessin de graphes sont en perpétuelle évolution. Il nous a donc semblé indispensable de permettre à cette évolution de prendre place dans notre système. De ce fait, de nouveaux modules peuvent facilement être ajoutés à notre système et tous ceux existants peuvent être remplacés facilement. Comme nous le verrons dans la partie dédiée aux perspectives pour la suite de cette thèse, cette évolutivité permet d'envisager de nombreuses applications à notre système.

La dernière contrainte est de pouvoir fournir des résultats rapidement. Or, cela ne peut être associé au fait d'obtenir des résultats de bonne qualité dans le cadre de la définition classique des AG. En effet, cette dernière implique une génération aléatoire de la population initiale et un

affinement progressif des résultats au cours de l'algorithme. C'est pour répondre à cette problématique de temps que la réutilisation des résultats a été mise en avant dans notre système. Pour cela, il était nécessaire de pouvoir facilement stocker des génomes dans la base de données et y accéder par la suite. Cela nous a ensuite conduits à mettre en place les contextes et les comportements afin de pouvoir caractériser chaque évaluation d'un génome.

6.1.3 Un système aux nombreuses possibilités

Le système mis en place est aujourd'hui fonctionnel. Et il semble apporter une réponse correcte à notre problématique centrale. Il permet à un utilisateur novice de fournir un graphe, puis de sélectionner une tâche de haut niveau pour lancer une session interactive. Il peut ensuite guider cette session à travers la sélection de ses candidats préférés et les évaluations qu'il réalise. Plusieurs tâches de haut niveau peuvent être facilement ajoutées au système afin de proposer aux utilisateurs novices des outils efficaces et simples pour dessiner leurs graphes.

Cependant, comme nous le verrons dans la partie suivante, de très nombreuses perspectives d'amélioration du système s'offrent à nous. Chacune d'elles permettrait l'obtention de layouts bien plus intéressants que ceux obtenus pour l'instant. En particulier, les tâches proposées dans le chapitre 4 ne constituent qu'un exemple extrêmement simplifié d'une utilisation du système. Et il reste encore de nombreux éléments à améliorer dans l'optique d'une mise à disposition de ce système à de « vrais » utilisateurs novices. Cependant, l'ensemble du code écrit durant ces trois ans l'a été dans une optique de réutilisation future en « production », et non pas d'un prototype destiné uniquement à valider les idées proposées dans cette thèse.

Ce système pourra aussi trouver de nombreux usages en dehors de l'assistance à un utilisateur novice. En effet, il est tout à fait possible de se passer de l'aspect interactif pour n'utiliser que la capacité de l'AG à générer des layouts en essayant d'optimiser une métrique particulière. Il suffirait pour cela d'ajouter un module correspondant au calcul de cette métrique et ensuite de définir une tâche utilisant ce module dans sa définition du score. Et, de manière encore plus générale, il serait possible de changer l'algorithme de dessin, voire même de changer complètement de cas d'utilisation. Cette modularité, couplée à une automatisation des exécutions et un paramétrage simple des AG, représente selon moi la production exploitable la plus importante de cette thèse. Il serait possible d'aller beaucoup plus loin dans ce projet. Mais, le temps de réflexion dédié à cette recherche étant limité à trois ans, toutes ces améliorations restent pour l'instant à l'état de perspectives.

6.2 Perspectives

6.2.1 Amélioration de certains modules

La première perspective d'amélioration de notre système concerne la reprise de certains modules. En effet, quelques uns, occupant parfois une place centrale dans l'AG, ont une responsabilité importante dans la limitation des résultats que nous avons obtenus.

Le premier de ces modules est bien évidemment GaGEM. Il consomme aujourd'hui toujours plus de 90 % du temps de calcul de **Genetips**, ce qui est tout à fait normal pour un évaluateur dans le cadre des AG. Globalement, la complexité de l'AG entier correspond à la complexité de l'évaluateur multiplié par un facteur linéaire (la taille de la population multipliée par le nombre de générations dans notre cas). GEM a pour sa part une complexité en $O(n^2)$, associé à une constante très importante (en utilisant une seule thread sur un processeur cadencé à 1,86GHz,

le calcul d'un layout pour un graphe de 200 sommets et 965 arêtes prend environ 1,5 secondes). Cela rend tout grand graphe impossible à dessiner à l'aide de l'AG, car il est pour cela nécessaire de réaliser un grand nombre d'essais pour obtenir un résultat intéressant. Il a été un temps envisagé de mettre au point une version multi-niveaux de GaGEM afin d'éviter ce problème, mais ce projet a dû être abandonné par manque de temps. En effet, cette modification aurait été autrement plus lourde que celle déjà réalisée lors du passage de GEM à GaGEM.

De plus, GEM, et donc GaGEM, n'a à sa disposition qu'un nombre limité de forces différentes, et en particulier n'a que très peu d'outils pour agir sur les arêtes et sur le positionnement absolu d'un sommet par rapport à un autre. Or il me semble que le positionnement de deux sommets (pas uniquement en terme de distance entre les sommets) a une importance primordiale sur la perception qu'un utilisateur a d'un dessin de graphe. Cela induit une limitation sur la variété des layouts que GaGEM peut atteindre « facilement », c'est-à-dire en un temps raisonnable et sans avoir besoin de nombreux objectifs intermédiaires pour arriver à ce résultat.

Un dernier point problématique de GEM est sa sensibilité aux variations même très faible des paramètres, en particulier dans le cadre d'une recherche guidée par la métrique de similarité. En effet, la part aléatoire de GEM est assez importante et la moindre variation d'un paramètre peut provoquer de petites transformations sur l'ensemble du graphe. Cela nous a posé problème lors de la restauration de génomes de la base de données. Un arrondi était alors réalisé sur les valeurs des paramètres (à 10^{-3}). L'impact sur le résultat était particulièrement fort puisque des génomes qui obtenaient une similarité inférieure à 0.1 obtenaient des scores s'approchant de 0.3 une fois leurs paramètres arrondis. Cette forte dégradation du score a disparu une fois les génomes stockés à l'identique sur la base. Mais cela indique une discontinuité de GEM, et donc de GaGEM, au niveau de la similarité de layouts produits par des paramètres modifiés même très faiblement. Cette discontinuité entre l'espace des paramètres et les layouts finalement obtenus ne peut pas être ignorée trop longtemps, car le bon fonctionnement d'un AG nécessite un minimum de continuité afin de pouvoir exploiter correctement le potentiel de la mutation, qui est bien évidemment l'opérateur génétique le plus concerné par ce problème.

Cependant, cette amélioration nécessiterait un temps important. En effet, une réponse correcte à ces différents problèmes correspondrait à la mise au point d'un nouvel algorithme par modèle de force, fonctionnel et rapide et permettant un passage à l'échelle. Il serait sûrement possible de reprendre un algorithme plus récent en le modifiant afin de le rendre aussi paramétrable que GaGEM. Il faudrait cependant trouver un algorithme dans lequel de nombreux paramètres sont accessibles, et qui permettent d'influer sur le processus de dessin sans pour autant casser complètement le layout.

D'autres modules gagneraient aussi à être repris et améliorés, la situation étant toutefois beaucoup moins critique que celle de GaGEM. Il s'agit de la définition du comportement et de l'expressivité du langage utilisé pour décrire les conditions associée à chaque jeu de paramètres dans les gènes.

Le comportement pose pour l'instant problème, car il est inutile au sein d'une exécution. En effet, puisque le comportement est uniquement fonction du graphe dessiné, ce dernier ne change jamais au cours d'une exécution. Il ne peut donc pour l'instant être utilisé que pour réaliser une requête sur la base de données. La définition du comportement vient de notre définition précédente des génomes, où chaque génome contenait un jeu de paramètres pour un unique sommet, et donc chaque sommet se voyait attribuer un génome. Dans cette situation, le contexte permettait de différencier les différents sommets d'un graphe. Les opérateurs génétiques pouvaient donc

sélectionner des génomes en fonction des contextes topologiques locaux dans lesquels seraient évalués les génomes produits. Toutefois, la modification du génome rend obsolète cet usage du comportement. Il serait donc intéressant d'en trouver une nouvelle définition afin de rendre plus utile ces informations, cependant, nous n'avons pour l'instant pas eu le temps de nous pencher plus sur cette question.

Le dernier module dont l'amélioration serait intéressante concerne les jeux de règles de chaque gène. Pour l'instant, ces conditions s'expriment sous la forme d'une combinaison non-linéaire des valeurs des différentes métriques topologiques pour chaque sommet. Or ce système ne permet que très difficilement de faire apparaître des impacts de voisinage. Par impact de voisinage, j'entends qu'il est très difficile que les paramètres associés à un sommet soient déterminés par la présence d'un sommet caractéristique dans le voisinage de ce dernier. Or ce type de mécanisme est très courant dans le dessin de graphe, lorsque l'on souhaite par exemple que tel groupe de sommets soit dessiné d'une certaine manière. Il suffirait pour permettre cela de fournir des opérations à ajouter dans la définition des règles afin de prendre aussi en compte le voisinage. Ces opérations seraient à priori similaires aux fonctions d'agrégations des bases de données (minimum, maximum, moyenne, décompte, accumulation). Toutefois, de telles fonctions posent le problème du coût de calcul. Dans la mesure où ces règles sont générées aléatoirement, il serait tout à fait possible de faire apparaître une chaîne d'opérations sur le voisinage qui deviendrait alors très coûteuse à évaluer. Nous n'avons malheureusement pas pu dédier suffisamment de temps à cette réflexion pour trouver une solution viable. D'autres améliorations seraient envisageables pour ce système de règles, par exemple des modules se basant sur le contenu de l'étiquette associée au sommet. Cela serait par exemple extrêmement utile pour les graphes de voies métaboliques dans lesquels la manière de placer un sommet dépend énormément de la molécule qu'il représente.

6.2.2 Validation et valorisation

Une autre perspective très intéressante serait de réaliser une validation des différents éléments mis en place dans cette thèse. Certains modules se prêteraient plus facilement que d'autres à cet exercice. Par exemple, un travail intéressant sera facilement réalisable autour de la métrique de similarité. Il existe en effet de nombreuses méthodes permettant de réaliser des comparaisons similaires. Une comparaison exhaustive de ces méthodes avec la nouvelle métrique permettrait sûrement d'identifier de nouveaux domaines d'application.

De plus, cette métrique pourrait facilement être modifiée pour permettre un passage à l'échelle. L'idée consistant à ne considérer que certains couples de points lors de la comparaison, en utilisant une sorte de « spanner » du graphe complet ou une approximation de grille issue de l'ensemble de sommets à comparer. Une telle approche permettrait aussi d'utiliser cette mesure de similarité pour comparer des nuages de points plutôt que des layouts (la topologie du graphe n'est pas utilisée, sauf lors de l'utilisation du coefficient pour calculer une similarité locale). Enfin, cette métrique pourrait aussi être beaucoup plus précise si elle considérait les angles entre les triplets de points en plus des distances (cette modification aura par contre un impact fort sur la complexité).

Il serait ensuite intéressant de valoriser ces possibilités à travers des publications scientifiques. Cette mesure occupe d'ailleurs une place importante dans la publication la plus importante réalisée durant cette thèse, *One Graph, Multiple Drawings* [71]. Cette publication, présentée dans le cadre de la conférence IV 2013 à Londres, porte principalement sur la modification apportée à GEM pour obtenir GaGEM et sur cette mesure. Il reste toutefois encore de nombreux points à explorer qui pourraient selon moi présenter un intérêt certain pour la communauté.

La question de l'indexation des layouts est elle aussi très intéressante. Lors des différents essais réalisés durant cette thèse, les méthodes envisagées n'ont fourni que de très pauvres résultats. Mais de nombreuses pistes restent à explorer. Par exemple, il serait peut-être intéressant de toujours considérer à la place d'une distance le rapport de cette distance divisé par la distance moyenne entre deux points pour l'ensemble du layout. Cela rendrait ces grandeurs comparables d'un layout à l'autre, ce qui offrirait à priori de nouvelles possibilités pour permettre une indexation.

Une autre perspective intéressante serait la validation du système mis en place pour l'algorithme génétique. La plus grande nouveauté mise en œuvre dans ce domaine concerne les populations annexes. Il faudrait pour cela pouvoir mettre en place une comparaison entre notre système et d'autres AG utilisés dans un même objectif afin de déterminer précisément le gain de performance. La validation de notre système de réutilisation des résultats serait par contre bien plus délicate. En effet, il n'existe pas d'autres systèmes d'AG permettant à notre connaissance de réutiliser ainsi des résultats issus de toutes les exécutions précédentes. Il faudrait donc trouver une méthode de comparaison valide qui permet de confirmer l'apport de cette réutilisation. Il serait peut-être intéressant pour cela d'étudier les protocoles de validation employés dans le cadre de l'optimisation multi-objectif.

Cependant, toutes ces validations nécessiteraient tout d'abord de mettre au point des structures efficaces pour l'algorithme génétique. La mise au point de telles structures correspond justement à la dernière famille de perspectives envisagées, abordée plus en détail dans le paragraphe suivant.

6.2.3 Mise au point de nouvelles structures

Cette dernière perspective est sûrement la plus accessible, mais aussi la plus longue de toutes. Il s'agit de la mise au point de nouvelles structures pour l'AG. En effet, celles présentées dans le chapitre 4 restent très simples et ne permettent pas une exploitation complète des possibilités de notre système.

Une première voie intéressante pour exploiter le système serait de trouver une métrique à optimiser. Il faudrait pour cela mettre au point des structures efficaces similaires à celles des deux premiers exemples du chapitre 4. Ces structures ne seraient donc pas interactives, mais permettraient d'optimiser efficacement la métrique choisie. La réalisation de telles structures nécessite par contre un long temps de paramétrage afin de bien étudier le comportement des différents modules et leurs interactions.

La deuxième possibilité d'amélioration concerne la conception de structures interactives. Celles proposées dans le chapitre 4 sont basées sur des définitions de score extrêmement simples. Il me semble qu'à l'aide de scores plus complets, et faisant moins appel à la contrainte de similarité, il serait possible d'avoir une génération de dessins beaucoup plus différents tout en restant intéressants. La constitution d'une grande collection d'évaluations d'utilisateurs pourrait sûrement être exploitée dans ce but. C'est d'ailleurs dans cette optique que les index de populations ont initialement été mis en place. Cependant, comme nous venons de le voir, les résultats que nous avons obtenus dans cette direction n'ont pour l'instant pas été satisfaisants. Cependant, s'il devenait possible d'exploiter toutes les anciennes évaluations utilisateurs pour dessiner un nouveau graphe, cela serait des plus intéressants.

Ce dernier point rejoint un dernier projet qui a un temps été envisagé durant cette thèse. Il s'agit de la spécialisation de l'AG dans un domaine précis. C'est dans le cadre des graphes de voies métaboliques que cette idée a vu le jour. L'objectif était de dessiner un grand nombre de graphes de ce type afin de stocker dans la base de données toutes les informations permettant de dessiner cette famille. Le concept peut même être poussé plus loin en évaluant un même génome sur plusieurs graphes simultanément (cela revient à utiliser les mêmes gènes lors de l'attribution des paramètres à chaque sommet). Pour obtenir un bon score pour chaque graphe, un génome est obligé d'avoir des gènes répondant à tous les types de sommets présents dans l'ensemble des graphes utilisés. Si cet ensemble est assez « couvrant », il est possible d'espérer que ce génome puisse dessiner correctement d'autres graphes similaires à ceux de l'ensemble. Ce principe pourrait être particulièrement intéressant pour réaliser un apprentissage sur un ensemble de petits graphes pour ensuite en dessiner des biens plus grands. C'est dans cette dernière optique qu'il serait possible d'adapter le système aux graphes dynamiques.

6.3 Un système aux vastes possibilités

Il semble que le plus grand apport de cette thèse concerne l'approche qui a été faite des AG ainsi que le système mis en place pour permettre cette approche. Les AG, comme la plupart des méta-heuristiques d'intelligence artificielle, sont extrêmement gourmands en terme de puissance de calcul. Permettre leur distribution et leur automatisation est donc un pré-requis indispensable pour pouvoir envisager leur utilisation dans le cadre d'un système plus vaste et plus ambitieux qui approcherait une intelligence plus « humaine ».

6.3.1 Une nouvelle approche du paramétrage des AG

Comme nous l'avons vu au cours de ce manuscrit, les AG correspondent à une méthode d'exploration naturelle d'un espace vaste adaptée au monde de l'informatique. Cependant, d'énormes différences subsistent entre le modèle naturel et le modèle informatique pour ces algorithmes. La différence majeure concerne le fait que dans la nature, l'ensemble de la structure de l'algorithme est contenu au sein du génome, ce qui n'est pas du tout le cas en informatique. Une autre variation importante concerne la méthode de sélection pour réaliser un croisement. Un processus de sélection majoritairement aléatoire est utilisé en informatique, alors que dans la nature, l'accouplement et la phase de séduction correspondent souvent au comportement le plus complexe d'une espèce. Cela pourrait rejoindre le principe présenté par Goodall [41] concernant le croisement, pour lequel il dit que le but de ce dernier est de faire apparaître des génomes qui soient nouveaux **et** intéressants.

Michalevicz dans son livre sur le sujet [69] met en avant les grandes possibilités des AG lorsqu'ils sont utilisés conjointement à des structures de données complexes. Cependant, il est a priori possible de supposer que plus ces données seront complexes, plus il sera possible de faire des choses différentes avec. Dans le cadre des AG, cela peut se principalement se traduire à travers la manière de mener la mutation, le croisement et sur la sélection des différents génomes utilisés par ces opérateurs.

C'est dans cette optique que notre système a été mis en place, pour permettre de construire, d'essayer, et d'évaluer rapidement une nouvelle configuration. L'étape suivante étant de comparer cette configuration aux précédentes afin de tenter de comprendre au plus vite en quoi elle

améliore la recherche et quels sont les points qui devraient encore pouvoir être optimisés. Et il me semble, une fois arrivé au terme de ces 3 années, que nous fournissons une réponse correcte à cette problématique.

J'espère que l'exploitation de ces capacités de configuration permettra à terme de mettre au point des AG complexes remplissant des tâches nouvelles et le faisant efficacement. Même si nos capacités en termes de calculs ont explosé ces dernières années, il reste encore un très long chemin à parcourir pour rattraper l'évolution naturelle et sa capacité à produire des systèmes complexes et intelligents.

6.3.2 La modularité, un besoin primordial

Cependant, de tels systèmes n'apparaîtront pas en un jour, et il faudra de nombreux essais pour réussir à aboutir dans cette quête. Un système intelligent est en effet bien trop complexe pour pouvoir marcher « du premier coup ». C'est aussi pour prendre en compte cela que notre système a été prévu pour permettre un maximum d'évolutions, et cela sans nécessiter une refonte complète du code à chaque modification.

Nous avons déjà exploité à plusieurs reprises cette capacité au cours de cette thèse. La modification la plus importante étant sûrement le changement du type de génome utilisé (passage d'un génome par sommet avec un jeu de paramètres par sommet à un génome par graphe avec des ensembles de règles pour attribuer les paramètres). Cette modification n'a finalement nécessité que la création d'un gène permettant de représenter nos nouvelles données, d'un bloc d'initialisation de population, et la modification de la fonction de l'évaluateur permettant d'appeler l'algorithme de dessins à partir des génomes. En effet, le reste du système fait complètement abstraction du contenu des génomes lors des manipulations.

De manière plus générale, la modularité est au cœur de notre système, et cela permet de laisser toute liberté à l'utilisateur quant à sa manière de diriger son AG selon ses besoins et ses envies. Sur ce point, j'estime que la réponse que nous avons apportée à notre besoin est là aussi adaptée et satisfaisante. Grâce à cela il devrait rapidement être possible d'appliquer cet AG à un autre cas d'utilisation que le dessin de graphe.

6.3.3 Apprendre à apprendre, un Graal ?

Dans la quête d'un système intelligent, d'une manière au moins comparable à un être humain, il reste encore un chemin immense à parcourir. L'apprentissage est un domaine sans limites, et qui représente pour moi un des grands défis de l'informatique de demain. Un système réellement intelligent devra pouvoir apprendre quelque chose pour lequel il n'a pas été conçu, ce qui rendra justement sa conception des plus délicate. Et c'est un défi que je trouve fascinant.

Trois ans m'ont été « offerts » pour que je puisse explorer une idée, pour que je puisse faire un pas dans une direction qui me semble intéressante. Il reste encore une très grande route avant d'obtenir un système réellement capable d'apprendre à apprendre et il est encore impossible de dire si ce système représente réellement une avancée pour cet objectif encore très utopique. Les intuitions sont souvent fausses, mais elles fournissent au moins une direction à suivre, et permettent souvent de découvrir de nombreuses choses tout au long du chemin. De plus, la thèse comme elle est proposée dans le cadre de la recherche en France permet dans une certaine mesure de

suivre de telles idées, de voir vers où elles peuvent mener sans pour autant sanctionner l'explorateur s'il n'aboutit pas à ce qui était attendu. C'est une chance qui m'a été proposée et dont j'ai profité avec le plus grand des plaisirs. Je ne sais pas encore si le point auquel j'ai abouti me permettra d'aller plus loin dans cette quête d'un programme capable d'apprendre, mais j'ai en tout cas beaucoup appris lors de ce parcours, et je garderai un très bon souvenir des moments qui l'ont constitué.

Bibliographie

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning : Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1) :39–59, 1994.

[2] a.M.S. Barreto and H.J.C. Barbosa. Graph layout using a genetic algorithm. *Proceedings. Vol.1. Sixth Brazilian Symposium on Neural Networks*, pages 179–184, 2000.

[3] J Andre, P Siarry, and T Dognon. An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. *Advances in engineering software*, 32, 2001.

[4] Daniel Archambault, David Auber, and Tamara Munzner. TopoLayout : Multi-Level Graph Layout by Topological Features. *IEEE transactions on visualization and computer graphics*, 2007.

[5] DA Ashlock, KM Bryden, and S Corns. Small population effects and hybridization. *... Computation, 2008. CEC ...*, 2008.

[6] GD Battista, P Eades, R Tamassia, and IG Tollis. *Graph drawing : algorithms for the visualization of graphs*. 1998.

[7] M Y Becker and I Rojas. A graph layout algorithm for drawing metabolic pathways. *Bioinformatics (Oxford, England)*, 17(5) :461–7, May 2001.

[8] JL Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 1975.

[9] Jacques Bertin and Marc Barbut. *S{é}miologie graphique : les diagrammes, les r{é}seaux, les cartes*. Mouton Paris, 1967.

[10] Jacques Bertin and Serge Bonin. *La graphique et le traitement graphique de l'information*. Flammarion Paris, 1977.

[11] T Biedl, J Marks, and K Ryall. Graph multidrawing : Finding nice drawings without defining nice. *Graph Drawing*, 1998.

[12] F L Bookstein. *Morphometric tools for landmark data : geometry and biology*, volume 10. Cambridge University Press, 1991.

[13] FL Bookstein. Principal warps : Thin-plate splines and the decomposition of deformations. *Pattern Analysis and Machine Intelligence,*, 1989.

[14] R. Bourqui, D. Auber, V. Lacroix, and F. Jourdan. Metabolic network visualization using constraint planar graph drawing algorithm. *Tenth International Conference on Information Visualisation (IV'06)*, pages 489–496.

[15] U Brandes. Keynote address : Why everyone seems to be using spring embedders for network visualization, and should not. *Visualization Symposium (PacificVis), 2011 IEEE*, 2011.

[16] Ulrik Brandes. A faster algorithm for betweenness centrality*. *Journal of Mathematical Sociology*, 25(1994) :163–177, 2001.

[17] Jurgen Branke and Frank Bucher. Using genetic algorithms for drawing undirected graphs. *Workshop on Genetic Algorithms and*, pages 1–13, 1996.

[18] Stina Bridgeman and Roberto Tamassia. A user study in similarity measures for graph drawing. *Journal of graph algorithms and applications*, 6(3) :225–254, 2002.

[19] Sung-bae Cho and Joo-young Lee. A human-oriented image retrieval system using interactive genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics - Part A : Systems and Humans*, 32(3) :452–458, May 2002.

[20] KC Cox, SG Eick, GJ Wills, and RJ Brachman. Brief application description ; visual data mining : Recognizing telephone calling fraud. *Data Mining and Knowledge ...*, 1997.

[21] C Dangalchev. Residual closeness in networks. *Physica A : Statistical Mechanics and its Applications*, 365 :556–564, 2006.

[22] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[23] René Descartes. Lettre à mesland du 9 février 1645. 1645.

[24] Tim Dwyer and Yehuda Koren. Dig-CoLa : directed graph layout through constrained energy minimization. *IEEE Symposium on Information Visualization (InfoVis 2005)*, 2005.

[25] Tim Dwyer, Yehuda Koren, and Kim Marriott. IPSEP-COLA : An Incremental Procedure for Separation Constraint Layout of Graphs. *Computer*, 12(5) :821–828, 2006.

[26] P Eades. *Drawing Free Trees.* IIAS-RR-. International Institute for Advanced Study of Social Information Science, Fujitsu Limited, 1991.

[27] P Eades and ML Huang. Navigating clustered graphs using force-directed methods. *J. Graph Algorithms Appl.*, 4(3) :157–181, 2000.

[28] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42 :149–160, 1984.

[29] Timo Eloranta and Erkki Mäkinen. TimGA : A Genetic Algorithm for Drawing Undirected Graphs. *Computer*, 9(2) :155–171, 2001.

[30] Martin J Eppler and Jeanne Mengis. The Concept of Information Overload : A Review of Literature from Organization Science, Accounting, Marketing, MIS, and Related Disciplines. *The Information Society*, 20(5) :325–344, 2004.

[31] Leonard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8 :128–140, 1741.

[32] Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). *Graph Drawing*, 1995.

[33] JH Friedman, JL Bentley, and RA Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on ...*, 1549(July), 1977.

[34] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software : Practice and Experience*, 21(11) :1129–1164, November 1991.

[35] Pawel Gajer and S Kobourov. Grip : Graph drawing with intelligent placement. *Graph Drawing*, 6(3) :203–224, 2001.

[36] Severino F. Galan and Ole J. Mengshoel. Generalized crowding for genetic algorithms. *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, page 775, 2010.

[37] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. On the readability of graphs using node-link and matrix-based representations : a controlled experiment and statistical analysis. *Information Visualization*, 4(2) :114–135, May 2005.

[38] M Girvan and M E J Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12) :7821–6, June 2002.

[39] DE Goldberg. Computer-aided gas pipeline operation using genetic algorithms and rule learning. page 242, 1983.

[40] DE Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison we edition, 1989.

[41] C Goodall. Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society. Series B (*, 53(2) :285–339, 1991.

[42] Lars Grammel. Supporting end users in analyzing multiple data sources. In *Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on*, pages 246–247. IEEE, 2009.

[43] Lars Grammel. *User interfaces supporting information visualization novices in visualization construction*. PhD thesis, 2012.

[44] Lars Grammel, Melanie Tory, and Margaret-Anne Storey. How information visualization novices construct visualizations. *IEEE transactions on visualization and computer graphics*, 16(6) :943–52, 2010.

[45] JJ Grefenstette. Optimization of control parameters for genetic algorithms. *..., Man and Cybernetics, IEEE Transactions on*, 00(February) :122–128, 1986.

[46] LJ Groves, Z Michalewicz, P V Elia, and C Z Janikow. Genetic algorithms for drawing directed graphs. *for Intelligent Systems,*, 1990.

[47] S Hachul and M Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. *Graph Drawing*, 2005.

[48] Minhi Hahn, Robert Lawson, and Young Gyu Lee. The effects of time pressure and information load on decision quality. *Psychology and Marketing*, 9(5) :365–378, 1992.

[49] J H Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[50] W. Huang. Using eye tracking to investigate graph layout effects. *2007 6th International Asia-Pacific Symposium on Visualization*, pages 97–100, February 2007.

[51] M. Hutter and S. Legg. Fitness uniform optimization. *Arxiv preprint cs/0610126*, 2006.

[52] T Kamada and S Kawai. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(April) :7–15, 1989.

[53] M Kaufmann and D Wagner. *Drawing graphs : methods and models*. 2001.

[54] Hee-Su Kim and Sung-Bae Cho. Application of interactive genetic algorithm to fashion design. *Engineering Applications of Artificial Intelligence*, 13(6) :635–644, December 2000.

[55] Corey Kosak, Joe Marks, and Stuart Shieber. Automating the Layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(3) :440–454, 1994.

[56] Bongshin Lee, Catherine Plaisant, Cynthia Sims Parr, Jean-Daniel Fekete, and Nathalie Henry. Task taxonomy for graph visualization. *Proceedings of the 2006 AVI workshop on BEyond time and errors novel evaluation methods for information visualization - BELIV '06*, page 1, 2006.

[57] J Lehman. *Evolution Through the Search for Novelty*. PhD thesis, 2007.

[58] Joel Lehman and Kenneth O. Stanley. Efficiently evolving programs through the search for novelty. *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, (Gecco) :837, 2010.

[59] Joel Lehman and Kenneth O Stanley. Abandoning objectives : evolution through the search for novelty alone. *Evolutionary computation*, 19(2) :189–223, January 2011.

[60] Joel Lehman and K.O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. *Artificial Life*, 11(Alife Xi) :329, 2008.

[61] SJ Louis and Judy Johnson. Solving similar problems using genetic algorithms and case-based memory. *... International Conference on Genetic Algorithms*, 1997.

[62] SJ Louis and J McDonnell. Learning with case-injected genetic algorithms. *Evolutionary Computation, IEEE ...*, 13(0704), 2004.

[63] A Markus. Experiments with Genetic Algorithms for Displaying Graphs. pages 62–67, 1991.

[64] D Masson, A Demeure, and G Calvary. Magellan, an evolutionary system to foster user interface design creativity. *Proceedings of the 2nd ACM SIGCHI ...*, 2010.

[65] T. Masui. Graphic object layout with interactive genetic algorithms. *Visual Languages, 1992. Proceedings., 1992 IEEE ...*, pages 74–80, 1992.

[66] Toshiyuki Masui. Evolutionary learning of graph layout constraints from examples. *Proceedings of the 7th annual ACM symposium on User interface software and technology - UIST '94*, (November) :103–108, 1994.

[67] ML Mauldin. Maintaining Diversity in Genetic Search. *AAAI*, 1984.

[68] Mitchell Melanie. *An introduction to genetic algorithms*. MIT Press, 1996.

[69] Z Michalewicz. *Genetic algorithms+ data structures= evolution programs*. Springer, 3rd edition, 1996.

[70] A Moraglio. Abstract convex evolutionary search. *Proceedings of the 11th workshop proceedings on ...*, 2011.

[71] Maurin Nadal and Guy Melançon. One Graph, Multiple Drawings. In *IEEE Proceedings of the 16th International Conference Information Visualization IV13*.

[72] HAD Nascimento and P Eades. User hints for directed graph drawing. *Graph Drawing*, 2002.

[73] M E J Newman and M Girvan. Finding and evaluating community structure in networks. *Physical review E*, pages 1–16, 2004.

[74] T Takao Nishizeki and Md Saidur Rahman. *Planar graph drawing*, volume 12. World Scientific, 2004.

[75] Mathias Pohl, Markus Schmitt, and Stephan Diehl. Comparing the readability of graph layouts using eyetracking and task-oriented analysis. *... of the Fifth Eurographics conference on ...*, 2009.

[76] HC Purchase, RF Cohen, and M James. Validating Graph Drawing Aesthetics. *Graph Drawing*, 1995.

[77] Helen Purchase. Which aesthetic has the greatest effect on human understanding ? *Graph Drawing*, 1997.

[78] Helen Purchase. Effective information visualisation : a study of graph drawing aesthetics and algorithms. *Interacting with Computers*, 13(2) :147–162, December 2000.

[79] Helen Purchase and M McGill. Graph drawing aesthetics and the comprehension of UML class diagrams : an empirical study. *Proceedings of the 2001 ...*, 2001.

[80] CL Ramsey and JJ Grefenstette. Case-based initialization of genetic algorithms. *... Conference on Genetic Algorithms*, 1993.

[81] EM Reingold and JS Tilford. Tidier drawings of trees. *Software Engineering, IEEE ...*, (2) :223–228, 1981.

[82] Neil Robertson and Daniel Sanders. The four-colour theorem. *Journal of Combinatorial ...*, 70(1) :2–44, May 1997.

[83] F James Rohlf and Dennis Slice. Extensions of the Procrustes Method for the Optimal Superimposition of Landmarks. *Systematic Zoology*, 39(1) :40–59, 1990.

[84] Jimmy Secretan, Nicholas Beato, David B D'Ambrosio, Adelein Rodriguez, Adam Campbell, Jeremiah T Folsom-Kovarik, and Kenneth O Stanley. Picbreeder : a case study in collaborative evolutionary exploration of design space. *Evolutionary computation*, 19(3) :373–403, January 2011.

[85] Paolo Simonetto, Daniel Archambault, David Auber, and Romain Bourqui. ImPrEd : An Improved Force-Directed Algorithm that Prevents Nodes from Crossing Edges. *Computer Graphics Forum*, 30(3) :1071–1080, June 2011.

[86] Cheri Speier, Joseph S. Valacich, and Iris Vessey. The Influence of Task Interruption on Individual Decision Making : An Information Overload Perspective. *Decision Sciences*, 30(2) :337–360, March 1999.

[87] R Spence. *Information Visualization : Design for Interaction.* Pearon/Prentice Hall, 2007.

[88] H. Takagi. Interactive evolutionary computation : fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE*, 89(9) :1275–1296, 2001.

[89] WT Tutte. How to draw a graph. *Proc. London Math. Soc*, 8(May 1962) :743–767, 1963.

[90] J Utech, J Branke, H Schmeck, and Peter Eades. An evolutionary algorithm for drawing directed graphs. *Proc. of the Int. Conf. on*, 1998.

[91] D C Van Essen, C H Anderson, and D J Felleman. Information processing in the primate visual system : an integrated systems perspective. *Science*, 255(5043) :419–423, January 1992.

[92] JQ Walker. A node-positioning algorithm for general trees. *Software : Practice and Experience*, 1990.

[93] Colin Ware. *Information Visualization : Perception for Design.* Interactive Technologies. Elsevier Science, 2004.

[94] Colin Ware, Helen Purchase, Linda Colpoys, and Matthew McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2) :103–110, June 2002.

[95] Tatsumi Watanabe and H Takagi. Recovering system of the distorted speech using interactive genetic algorithms. *... Cybernetics, 1995. Intelligent Systems ...*, 1995.

[96] Darrell Whitley, Soraya Rana, and RB Heckendorn. Island model genetic algorithms and linearly separable problems. *Evolutionary computing*, 1997.

[97] LD Whitley. The GENITOR Algorithm and Selection Pressure : Why Rank-Based Allocation of Reproductive Trials is Best. *ICGA*, 1989.

[98] BG Woolley and KO Stanley. Exploring promising stepping stones by combining novelty search with interactive evolution. *arXiv preprint arXiv :1207.6682*, pages 1–15, 2012.

[99] Qing-Guo Zhang, Hua-yong LIU, Wei Zhang, Ya-jun GUO, and ZHANG WEI. Drawing undirected graphs with genetic algorithms. *Lecture notes in computer science*, 2005.