Rapport final

INF4173 - Projet synthèse
Université du Québec en Outaouais

Cédric Bastien
BASC08048202

sous la supervision de
M. Jurek Czyzowicz

20 décembre 2004

Table des matières

Description du projet

Le projet réalisé avait pour objectif d'améliorer un logiciel *Java* servant à présenter certaines notions théoriques du cours de *Langages formels*. Ce logiciel possède une interface graphique permettant à l'utilisateur d'interagir visuellement et intuitivement, afin de créer des automates, des automates à pile et des machines de Turing. Il permet aussi de visualiser leur fonctionnement ainsi que le déroulement de divers algorithmes qui se rattachent à l'une ou l'autre de ces machines.

Le travail effectué dans le cadre de ce projet visait à améliorer cette application et se situait à deux niveaux. D'une part, quelques notions du cours de *Langages formels* n'ont pas été implémentées dans la version actuelle du logiciel, notamment le processus de minimisation et de déterminisation d'un automate fini. Ces deux algorithmes ne sont pas en soit très complexes à implémenter, la difficulté principale réside plutôt au niveau de leur animation. En effet, ces deux algorithmes ont comme point commun de produire un nouvel automate comme valeur de sortie. Or, ces automates doivent être affichés à l'écran d'une manière automatisée, c'est-à-dire sans intervention de l'utilisateur pour spécifier l'emplacement en x et y de chaque état. La première partie du travail se rapportait donc à la théorie de l'affichage des graphes (*Graph Drawing*), qu'il faut ici adapter au contexte des automates.

D'autre part, l'interface utilisateur devait être amélioré afin non seulement de la rendre plus attrayante d'un point de vue visuel et fonctionnel, mais aussi afin d'en améliorer son caractère pédagogique. Ce deuxième critère pouvait être réalisé par exemple en offrant à l'utilisateur différents modes de visualisation, en ajoutant davantage d'information pertinante, en utilisant différentes couleurs et formes géométriques, en élaborant davantage l'animation, etc.

Déroulement du projet

1. Affichage des automates

La majeur partie du travail effectué dans le cadre de ce projet a consisté à développer un algorithme devant servir à l'affichage automatique des automates. Le travail à été effectué en plusieurs étapes allant de l'étude théorique sur le domaine jusqu'à l'implémentation de l'algorithme dans le logiciel et son utilisation dans l'animation des algorithmes de déterminisation et de minimisation. Voici en détail ces différentes étapes :

a) Étude sur la théorie de l'affichage des graphes (*Graph Drawing)*

b) Choix des principaux heuristiques à utiliser dans le développement d'un algorithme d'affichage des automates et établissement de leur ordre d'importance.

c) Prise de décision entre le développement d'un nouvel algorithme et l'utilisation d'un logiciel externe pour exécuter le travail.

d) Développement d'un schéma général pour l'algorithme à concevoir.

e) Recherche d'information sur les algorithmes classiques nécessaires au schéma développé.

f) Élaboration concrète de l'algorithme final sous forme de pseudo-code.

g) Implémentation de l'algorithme.

h) Utilisation de l'algorithme dans l'animation des algorithmes de déterminisation et de minimisation d'un automate fini.

Cette section présente en détail le travail effectué à chacune de ces étapes ainsi que les résultats obtenus.

A) Étude sur la théorie de l'affichage des graphes

L'objectif de cette étape préliminaire était de me familiariser avec les concepts généraux de la théorie de l'affichage des graphes. Les principales références utilisées à cet effet furent le livre de *G. Di Battista* [1], ainsi que le livre de *Kaufman* [2] et certaines autres ressources [3,4,5,6,7]. Voici brièvement l'information retenue lors de cette partie du travail :

> L'affichage des graphes est un domaine vaste et inexact, en ce sens qu'un algorithme peut être très bon pour une application et beaucoup moins utile dans un différent contexte. Ceci est dû notamment à l'information contenue dans le graphe et qui doit être communiquée par un affichage adéquat ainsi qu'à la topologie même des graphes. Par exemple, on voudra généralement afficher un graphe planaire de manière à ce les arêtes ne se coupent pas, alors qu'un graphe dense sera sans doute affiché d'une toute autre manière. Ainsi, la grande majorité des algorithmes sont basés sur des heuristiques spécifiant les principaux critères à prendre en compte lors de l'affichage. Ces critères peuvent être regroupés en trois différentes classes : ceux qui doivent à tout prix être respectés (par exemple, dessiner le graphe en utilisant uniquement des segments de droites pour les arêtes), les critères à optimiser (réduire l'aire totale du graphe ou la longueur maximale d'une arête) et les contraintes spéciales (placer un noeud particulier sur la surface externe du graphe). Évidemment, il est impossible de satisfaire tous ces critères à la fois et c'est pourquoi il importe de faire une sélection basée sur les besoins concrets de l'application dans lequel l'algorithme sera utilisé. Aussi, compte tenu des critères retenus, il faut également établir un ordre de priorité dans leur application, de telle sorte que le critère principal soit appliqué en premier, puis le second critère est appliqué en tenant compte des restrictions imposées par le premier, etc. Ceci résulte généralement en des algorithmes s'exécutant par étape.

Évidemment, un survol des principaux algorithmes existant ainsi que des principales approches algorithmiques connues a été effectué lors de cette phase du projet. Les conclusions se rapportant à ce survol seront explicitées dans la section C.

B) Choix des principaux heuristiques

Avant de faire le choix des heuristiques (ou critères) à appliquer, il fallait tout d'abord déterminer les caractéristiques des graphes qui seront affichés par l'application. En particulier, dans quelles circonstances est-ce que l'algorithme d'affichage sera utilisé? Deux réponses sont possibles : ou bien l'algorithme doit chercher à effectuer correctement l'affichage de tous types de graphes (par exemple, à partir d'une description formelle quelconque), ou bien l'algorithme doit être développé avec pour objectif d'en faire une utilisation dans des circonstances précises (par exemple, pour afficher un automate minimisé ou déterminisé). La principale différence entre ces deux alternatives est qu'un automate tel que ceux produits par les algorithmes de minimisation et de déterminisation dans le contexte du cours sont des automates presque planaires et dont le nombre de noeuds dépasse rarement 15 ou 20. Aussi, ces automates ne correspondent que très rarement à des langages importants, la nature sémantique propre de l'automate peut donc être ignorée. Ce ne serait par contre pas le cas d'un algorithme développé pour afficher tous types d'automates. Celui-ci devrait être suffisamment flexible pour afficher par exemple des automates denses (certains automates peuvent former un graphe complet), ou dont le langage possède une propriété particulière qu'il faudrait exprimer par l'affichage. Ce problème est beaucoup plus complexe et nécessiterait un algorithme élaboré devant détecter ces différents éléments. Compte tenu de ces difficultés et du fait que l'intérêt principal de l'algorithme d'affichage est de permettre l'implémentation de l'animation des algorithmes de déterminisation et de minimisation des automates, l'approche la plus simple a été privilégiée.

Ainsi, les critères choisis et le schéma général de l'algorithme (voir section C) ont été décidés en considérant que :

- Les automates à afficher sont peu denses (mais pas nécessairement planaire).
- Les automates contiennent en général un maximum de 20 noeuds (l'algorithme n'empêche pas qu'il y en ait davantage mais n'aura pas à garantir pas une performance accrue. Plus de détails dans les sections D et E).
- Les langages acceptés par les automates à afficher ne possédent pas des propriétés particulières à exposer par l'affichage.

Aussi, certaines limites du logiciel imposent comme restriction d'effectuer l'affichage en utilisant uniquement un segment de droite pour dessiner les arêtes. Notez que l'algorithme a été conçu d'une manière telle que seule la dernière étape devrait être modifiée pour prendre en compte d'autres types d'arêtes (courbes ou

lignes brisées).

Dans ces conditions, les heuristiques suivants ont été privilégiés, dans cet ordre :

1. Affichage de l'état initial sur la surface externe du dessin de l'automate.
2. Affichage des noeuds sur une grille (*grid drawing*).
3. Minimisation du nombre d'intersection des arêtes.

Ces critères ont été établis suite à l'observation de nombreux automates dessinés intuitivement, à la main. Le critère spécifiant que l'état initial doit être situé sur la surface externe du dessin de l'automate à été établi suite à la constatation que la presque totalité des automates sont dessinés en ayant l'état initial situé à l'extrémité supérieure gauche. Algorithmiquement, pour effectuer un tel affichage, il faut tout d'abord s'assurer de placer l'état initial sur l'enveloppe externe du dessin de l'automate, puis appliquer une rotation permettant de placer cet état à la position désirée. Ensuite, la plupart des automates sont dessinés de manière à mettre l'emphase sur leurs éléments symétriques lorsque cela est possible. Bien que l'algorithme ne tient pas compte directement de ce critère, l'affichage des noeuds sur une grille permet un minimum de structure. Finalement, vient le critère visant à minimiser le nombre d'intersections, ce qui est très souhaitable dans le cas de graphes peu denses. Ce critère est lui aussi fréquemment utilisé lors du dessin manuel d'un automate.

C) Décision de développer un nouvel algorithme

À ce point de l'élaboration du projet, deux choix étaient possibles : faire appel à un autre logiciel pour exécuter un algorithme déjà existant afin que celui-ci s'occupe de l'affichage des automates (i.e. choisir des coordonnées x et y pour chaque noeud), ou bien développer et implémenter moi-même un nouvel algorithme. La première solution est évidemment idéale, dans la mesure où un algorithme respectant les critères définis en B existe et qu'il soit disponible et accessible à partir de mon logiciel, car ceci réduirait le temps de développement de beaucoup tout en assurant d'avoir une implémentation robuste et efficace. Cependant, les recherches effectuées ont été infructueuses. Il existe de nombreux logiciels permettant d'afficher des graphes et qui implémentent des algorithmes classiques. Par contre, je n'ai trouvé aucun logiciel servant à l'affichage des automates, ou de graphes avec les restrictions mentionnées précedemment. Le choix a donc été fait d'implémenter moi-même un nouvel algorithme.

D) Développement du schéma général de l'algorithme

À partir des critères définis en B, le schéma général suivant a été élaboré comme plan directeur d'un algorithme d'affichage des automates :

- Créer une représentation planaire (*planar embedding*) en s'assurant de conserver l'état initial sur la surface externe de l'automate et en minimisant le nombre d'intersection des arêtes. Pour chaque intersection, un *noeud artificiel* est ajouté de manière à rendre le graphe artificiellement planaire.
- Tout en respectant la représentation planaire créée précédemment, positionner les noeuds de l'automate de manière à ce que le dessin de l'automate soit planaire et que les noeuds (à l'exception des *noeuds artificiels*) soient placés sur un grillage, avec une distance minimale entre eux.

Ce schéma est relativement simple, mais sépare l'exécution de l'algorithme en deux parties distinctes qui permettent de travailler de manière totalement indépendante sur différents heuristiques. Ainsi, la première partie sert à minimiser le nombre d'intersection tout en conservant l'état initial sur la surface externe, alors que la seconde partie s'occupe des autres éléments, notamment d'accomplir un affichage structuré. Ceci permet d'expérimenter avec différents algorithmes pour l'une ou l'autre des deux tâches, sans que les modifications apportées à l'un des algorithmes affecte l'autre.

E) Recherche d'information sur les algorithmes classiques

La première partie du schéma de l'algorithme doit produire une représentation planaire de l'automate. Plusieurs algorithmes existent pour résoudre ce problème et une recherche approfondie des solutions possibles a dû être effectuée. Le critère principal utilisé lors de cette recherche était de trouver un algorithme relativement simple (considérant que le problème est à la base assez complexe), au risque d'avoir une complexité algorithmique non optimal. Il faut se rappeller que l'algorithme doit fonctionner pour des automates d'au plus 20 états, donc un algorithme fonctionnant en un temps polynomial moyennement élevé (par exemple $O(n^4)$) demeure un algorithme dont le temps d'exécution est raisonnable, pour le problème à résoudre. Aussi, il sera toujours possible par la suite d'améliorer l'efficacité globale de l'algorithme d'affichage en modifiant l'algorithme utilisé pour la création de la représentation planaire car l'implémentation exacte de cet algorithme est indépendante du reste.

Ainsi, l'algorithme retenu provient du livre de *G. Di Battista* [1] et a été adapté pour y intégrer la contrainte qui exige que l'état initial soit situé sur la surface externe du dessin de l'automate. Cette contrainte peut faire en sorte que l*a* représentation planaire qui est produite contient davantage de noeuds artificiels que nécessaire pour l'obtention d'un graphe planaire, mais l'implémentation cherche tout de même à en réduire le nombre, malgré cette contrainte. Cet algorithme utilise comme l'une de ses sous-routines principales un autre algorithme servant à tester si un graphe est planaire ou non.[1] Cet algorithme provient également du livre de *G. Di Battista* [1].

1 En fait, l'algorithme pour tester si un graphe est planaire n'a pas été utilisé lors de l'implémentation de l'algorithme servant à créer l*a* représentation planaire. Des explications à cet effet sont données plus loin.

Pour ce qui est du positionnement des noeuds, le livre de *G. Di Battista* [1] présente divers algorithmes pour afficher des graphes planaires de type *st* (*st-graph*), c'est-à-dire des graphes ayant un seul noeud d'origine (*source*) et un seul noeud d'arrivée (*sink*) tel que pour chaque noeud du graphe, il existe un chemin simple allant de la source vers la destination et passant par ce noeud. Un de ces algorithmes, appelé *Dominance-Straight-Line*, semble tout indiqué pour s'occuper en partie du positionnement des noeuds. Il travaille sur une grille, fait un affichage symétrique, orienté dans une direction particulière (*upward drawing*) tout en minimisant l'aire totale du graphe. Bien sûr, un automate n'est pas nécessairement un *st-graph*, mais nous verrons plus loin de quelle manière cet algorithme peut être utilisé pour positionner les noeuds de certaines sections de l'automate. Notez que cet algorithme n'a pas été implémenté lors du projet, et qu'il ne s'agit pour l'instant que d'une idée qui reste encore à préciser. Tout de même, il me semble bien réaliste de croire que cet algorithme pourra être adapté afin d'effectuer une bonne partie du positionnement final des noeuds.

Le reste des algorithmes utilisés comme sous-routine pour l'un ou l'autre de ces deux algorithmes principaux sont relativement standards. Par exemple, il faut utiliser le parcours en profondeur, le parcours en largeur, effectuer la recherche d'un cycle, décomposer le graphe en ses composantes biconnexes, etc.

F) Élaboration des algorithmes

Compte tenu de la complexité des algorithmes à implémenter, une description sous forme de pseudo-code de haut niveau a été effectuée afin de mieux structurer le fonctionnement des algorithmes. Le premier algorithme qui a été ainsi décrit est celui permettant de créer une représentation planaire. Il est initialement inspiré d'un algorithme issu du livre de *G. Di Battista* [1], mais il a subit plusieurs modifications afin d'assurer que l'état initial soit sur la surface externe du dessin de l'automate tout en minimisant le nombre d'intersections des arêtes. Voici le pseudo-code :

```
Algorithme     : create_embedding

Entrées        : GrapheOriginal, EtatInitial

Sorties        : EmbeddedGraph

Particularités : Assure que l'état initial est sur la surface externe du graphe.
Tente de minimiser le nombre d'intersections des arêtes.

Déroulement :

1. Transformer le GrapheOriginal en graphe manipulable par l'algorithme, donc:
     • Un graphe non-dirigé
     • Un graphe sans arêtes bouclant sur un même noeud
     • Un graphe dont les cycles de longueur 2 sont réduits à une arête.

Ce processus de transformation doit être réversible afin de pouvoir retrouver la
topologie du GrapheOriginal à la fin de l'exécution de l'algorithme.
```

2. Enlever un à un les noeuds du graphe dans l'ordre d'un BFS et les ajouter dans une pile. Débuter la recherche en largeur à partir de l'état initial.
3. Conserver une structure de données de l'enveloppe externe de la représentation planaire du graphe, initialement vide.
4. Dépiler un à un les noeuds et les ajouter au graphe ainsi:
    - Ajouter le noeud au graphe, sans ses arêtes adjacentes, sur la surface externe du graphe.
    - Pour chacune de ses arêtes adjacentes :
        - Si le noeud à la deuxième extrémité n'a pas encore été ajouté au graphe, ne rien faire.
        - Si le noeud à la deuxième extrémité a déjà été ajouté au graphe mais ne fait pas partie de l'enveloppe du graphe, ajouter l'arête à la file des *arêtes non-planaires.*
        - Si le noeud à la deuxième extrémité a déjà été ajouté au graphe et fait partie de l'enveloppe externe du graphe, ajouter l'arête au graphe.
    - Choisir une nouvelle enveloppe externe en fonction du noeud et des arêtes ajoutées en respectant la condition suivante :
        - Soit $v_1,v_2,...,v_n$, les noeuds d'une enveloppe externe particulière. Soit $d_1,d_2,...,d_n$, les degrés respectifs de ces noeuds et $c_1,c_2,...,c_n$ le nombre d'arêtes adjacentes à chacun de ces noeuds dont le noeud à la deuxième extrémité n'est pas encore présent au graphe. Choisir un embedding de tel sorte que l'enveloppe externe résultante maximise l'équation suivante : $(d_1 - c_1) + (d_2 - c_2) + ... + (d_n - c_n)$. En cas d'égalité, choisir l'enveloppe qui contient un maximum de noeuds.
        - Mettre à jour la structure de données de l'enveloppe externe en fonction du choix effectué.
5. Créer le graphe dual du graph obtenu jusqu'à présent.
6. Pour chaque arête qui ont été ajoutées dans la file des arêtes *non-planaires*:
    - Trouver le plus court chemin dans le graphe dual entre une face adjacente au noeud de départ et une face adjacente au noeud d'arrivée.
    - Ajouter l'arête en respectant la représentation planaire représentée par ce chemin.
    - Ajouter un *noeud artificiel* sur chaque intersection créée, afin de conserver un graphe planaire.
    - Mettre à jour le graphe dual.
7. Reprendre la topologie originale du graphe d'entrée et retourner le graphe obtenu.

Voici une explication des différentes étapes de l'algorithme.

- L'étape 1 n'est qu'un prétraitement simplifiant le graphe pour n'en conserver que les éléments essentiels.
- L'étape 2 établi l'ordre dans lequel les noeuds seront ajoutés au graphe. L'ordre choisi (celui d'un BFS) est important pour deux raisons : la première est que l'état initial étant visité en premier, c'est lui qui se retrouve au fond de la pile, ce qui implique qu'il sera ajouté en dernier et se retrouvera

nécessairement sur l'enveloppe externe de la représentation planaire du graphe. La deuxième raison est qu'un tel ordre constitue une première étape visant à minimiser le nombre d'intersection des arêtes. En effet, si l'on numérote les sommets selon leur niveau dans l'arbre BFS (distance par rapport à la racine), on peut prouver qu'un noeud de niveau i ne peut être adjacent qu'à des noeuds de niveau i – 1, i ou i + 1. En ajoutant les noeuds selon leur niveau dans l'arbre BFS en commençant par le niveau inférieur, on assure que les noeuds de niveaux rapprochés (donc ceux qui sont suceptibles d'être reliés par des arêtes) seront ajoutés presqu'au même moment, donc de manière rapprochées dans la représentation planaire du graphe, diminuant ainsi les risques d'intersection des arêtes.

- L'étape 3 ne fait que définir une structure de données nécessaire à l'exécution de l'étape 4.

- L'étape 4 permet d'ajouter itérativement tous les noeuds du graphe. Pour chaque noeud, on ajoute également toutes ses arêtes adjacentes qui vont vers des noeuds faisant déjà partie de l'enveloppe externe du graphe. Il faut maintenant choisir quelle sera la nouvelle enveloppe externe. Ce choix est fait de manière à minimiser les risques éventuels d'intersections des arêtes. Chaque noeud de l'enveloppe possède un certain nombre d'arêtes adjacentes qui n'ont pas encore été ajoutés au graphe. Le choix de l'enveloppe externe est fait de manière à ce que la somme de toutes les arêtes qu'il reste à ajouter à des noeuds de l'enveloppe externe soit maximal. Ce choix est justifié par le fait qu'ajouter une arête à partir d'un noeud sur la surface externe vers un noeud de l'enveloppe externe ne cause aucune intersection, alors qu'ajouter une arête à partir d'un noeud sur la surface externe vers un noeud interne du graphe cause au minimum une intersection.

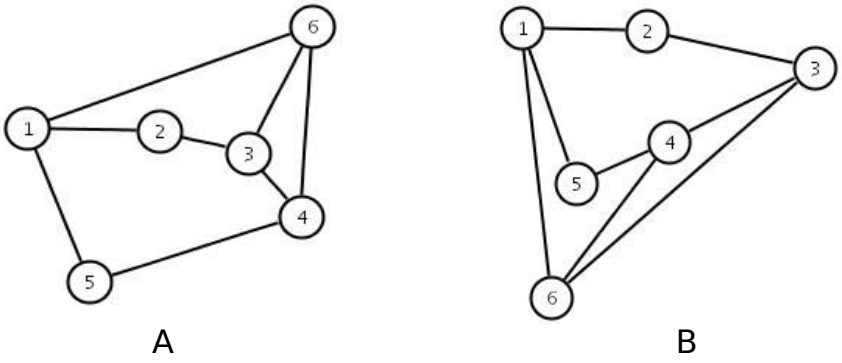

Image 1 – Exemple du choix de l'enveloppe externe

Dans ces deux illustrations, le noeud 6 vient d'être ajouté. L'enveloppe externe initiale est : [1, 2, 3, 4, 5]. Les illustrations A et B représentent deux possibilités dans le choix de l'enveloppe externe, pour l'ajout du noeud 6 et de 3 arêtes. Le choix A donne l'enveloppe externe : [1, 6, 4, 5], tandis que le choix B donne l'enveloppe externe : [1, 2, 3, 6].

- L'étape 5 sert à construire le graphe dual qui servira à l'ajout des arêtes non planaire. Il permettra de minimiser le nombre d'intersections pour chaque arête.

- L'étape 6 ajoute les arêtes non-planaires tout en minimisant le nombre d'intersection. Chaque intersection est remplacée par un *noeud artificiel* pour faire en sorte que le graphe demeure artificiellement planaire.

- L'étape 7 effectue le procédé inverse de l'étape 1.

La version originale de l'algorithme *create_embedding* provient du livre de *G. Di Battista* [1]. Cette version de l'algorithme ne cherche qu'à minimiser légèrement le nombre d'intersection des arêtes, et a besoin d'un algorithme pour tester la planarité d'un graphe. C'est pourquoi, dans ma version originale de l'algorithme *create_embedding*, j'avais prévu moi aussi d'utiliser un algorithme pour tester la planarité d'un graphe. Ce deuxième algorithme provient lui aussi de [1], et a été entièrement implémenté dans le logiciel. Ce n'est que par la suite que j'ai constaté qu'il n'était pas nécessaire de l'utiliser compte tenu des restrictions supplémentaires que j'impose pour créer la représentation planaire. J'ai tout de même conservé le code de l'algorithme, car il pourra possiblement être utile à d'autres fins, dans ce logiciel ou pour un autre projet. Je ne le présenterai pas en détail dans ce rapport car il n'est pas utilisé dans la version finale de l'application.

Une fois la représentation planaire générée, l'étape suivante consiste à positionner chaque noeuds à l'aide de coordonnées *x* et *y*. Une version préliminaire de cet algorithme a été conçue afin d'obtenir un premier aperçu du résultat final, et de permettre d'implémenter et de tester les algorithmes de déterminisation et de minimisation. L'algorithme qui a été utilisé est une version simplifiée et incomplète d'un éventuel algorithme qui accomplira correctement cette tâche. Ce meilleur algorithme n'a pas été développé en entier, et seuls les grandes lignes sont présentement définies. Tout de même, il me semble réaliste d'affirmer qu'une fois le développement complété, les résultats obtenus devraient être satisfaisants.

Cet algorithme fonctionne à deux niveaux. Le premier niveau consiste à positionner chaque *composante biconnexe externe*, c'est-à-dire les composantes biconnexes que l'on obtient en ne considérant que les noeuds de l'enveloppe externe. On peut constater que toutes ces composantes forment ensemble un arbre de composantes. Le deuxième niveau consiste à positionner les noeuds internes à chacune de ces composantes, s'il y a lieu. Ainsi, on peut imaginer que l'algorithme consisterait à positionner les noeuds de chaque composante comme s'il s'agissait d'un graphe indépendant, puis à positionner chaque composante les unes par rapport aux autres en parcourant l'arbre des composantes.

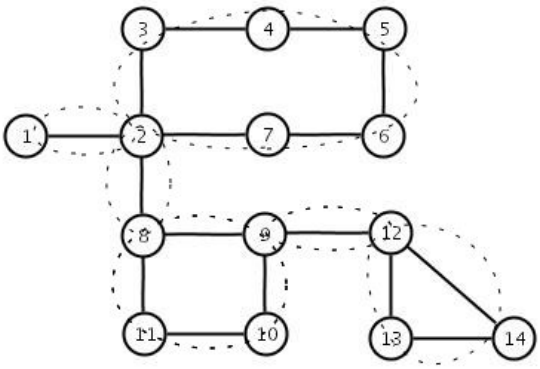

Image 2 – Arbre des composantes externes

Cette illustration présente l'enveloppe externe de la représentation planaire d'un automate. Chaque section entourée de lignes pointillées correspond à une *composante biconnexe externe*. L'algorithme proposé consiste à dessiner de manière indépendante chaque composante à l'aide de l'algorithme *Dominance-Straight-Line*, puis à positionner ces composantes les unes à côtés des autres, d'une manière similaire au dessin d'un arbre dont les noeuds (chaque composante) seraient de dimensions variées.

Pour afficher chaque composante avec ses noeuds interne, l'algorithme *Dominance-Straight-Line* décrit dans le livre de *G. Di Battista* [1] pourrait être utilisé, en l'adaptant bien sûr aux particularités des automates dans le logiciel. Il faudrait notamment transformer ces composantes pour qu'ils deviennent des *st-graph*.

Une fois que l'affichage de chaque composante est complété, et que l'on connait leur dimension, il reste à les placer correctement les uns aux côtés des autres, ce qui revient en quelque sorte à afficher un arbre dont les noeuds sont de dimensions variées. Il faut également s'assurer de positionner l'état initial dans la partie supérieur gauche du graphe.

G) Implémentation des algorithmes

L'implémentation des algorithmes s'est avérée être un des éléments les plus difficiles du projet. Bien que les algorithmes soient bien définis, il en va autrement de la manière de les implémenter. La difficulté réside surtout au niveau de la manipulation des données nécessaires aux différentes étapes de chaque algorithme : quand recueillir les données, comment les représenter en mémoire et à quel moment y accéder afin de garantir que l'algorithme s'exécute en temps optimal sans utiliser une trop grande quantité d'espace mémoire. Arriver à ce résultat a demandé énormément de travail d'analyse et de préparation, une quantité de travail bien supérieure à celle requise pour réellement coder l'algorithme, tester et déboguer.

Le premier algorithme qui a été implémenté a été celui servant à tester la planarité d'un graphe. L'implémentation s'est déroulée correctement et le résultat est très bon, cependant cet algorithme n'a pas été utilisé dans la version finale du logiciel.

Le deuxième algorithme, celui permettant d'obtenir une représentation planaire d'un graphe a été lui aussi entièrement implémenté, à l'exception de l'étape 6 car je n'ai pas encore décidé à quel moment je préfère ajouter les arêtes *non-planaires*. Tout de même, le graphe dual existe et est mis à jour à chaque ajout d'une nouvelle arête, il serait donc facile d'ajouter l'étape 6, si je juge que c'est bien à ce moment que je désire ajouter les arêtes *non-planaires*.

Bien que l'étape d'implémentation a représenté un certain défi, la véritable difficulté pour ce deuxième algorithme était l'étape de conception. Cette étape était cruciale afin d'assurer que le résultat obtenu soit aussi optimal que possible. Le critère que j'ai utilisé pour mesurer l'optimalité est le nombre d'arêtes qui sont mises de côté par l'algorithme parce-que celui-ci les considère comme étant *non-planaires*. Il est intéressant de noter que les premières versions de l'algorithme utilisaient des méthodes beaucoup plus rudimentaires pour choisir la nouvelle enveloppe externe, lors de l'étape 4. Un résultat intéressant montre que pour un des graphes sur lesquels j'ai effectué des tests, une version précédente de l'algorithme éliminait plus d'une vingtaines d'arêtes en les déclarant comme étant *non-planaires*, alors que le graphe était en fait presque planaire. Pour sa part, la dernière version de l'algorithme n'élimine qu'une seule arête pour ce même graphe, ce qui est une nette amélioration. De manière générale, le résultat obtenu par la version finale de l'algorithme se rapproche énormément du résultat que j'obtiens lorsque je dessine un automate manuellement, ce qui correspond tout à fait au résultat désiré. Plus important encore, le nombre d'arêtes éliminées est presque toujours minimal et aucun test n'a donné un nombre d'arêtes anormalement élevé. Ainsi, non seulement le résultat est généralement optimal, il n'est également jamais catastrophique.

L'implémentation du troisième algorithme, celui servant à positionner les noeuds, n'est présentement que partiellement complétée. La version actuelle ne sert qu'à afficher l'enveloppe externe de la représentation planaire de l'automate. Aucun traitement n'est fait sur les noeuds internes. Tout de même, le résultat pour plusieurs automates est très bon. Ceci est principalement dû au fait que la représentation planaire reçue en entrée a été bien construite par l'algorithme précédent et aussi au fait que la taille des automates est relativement petite. L'implémentation actuelle est donc suffisante pour effectuer certains tests ou même utiliser le logiciel pour de petits automates. Cependant, il sera important d'implémenter entièrement l'algorithme décrit à la section F, si l'on désire avoir un logiciel qui fonctionne bien.

H) Animation des algorithmes de déterminisation et de minimisation

Une fois l'algorithme d'affichage des graphes complété, il restait encore une dernière étape importante : celle d'implémenter les algorithmes de déterminisation et de minimisation d'un automate fini, et surtout leur animation. Cette partie du travail était sans doute la plus simple, car le niveau de complexité n'est pas du tout le même que pour la création de l'algorithme d'affichage des graphes. Le logiciel possède déjà de nombreuses fonctionnalités d'affichage et d'animation, la seule difficulté est donc de choisir quoi afficher à l'écran, plutôt que de trouver comment le faire.

Ces 2 algorithmes ont donc été implémentées assez facilement et les options pour les utiliser ont été ajoutées au logiciel. L'algorithme d'affichage des graphes décrit plus tôt est utilisé par ces 2 algorithmes pour afficher l'automate résultant de la déterminisation et de la minimisation.

2. Amélioration de l'interface utilisateur

Peu de travail a été effectué au niveau de l'interface utilisateur, la grande partie de l'effort ayant été consacré à l'élaboration de l'algorithme d'affichage des automates. Tout de même, certaines améliorations y ont été apportée, à divers niveau. Voici un résumé de ces modifications :

- Les algorithmes de déterminisation et de minimisation sont plus complexes que les algorithmes qui étaient déjà présents dans le logiciel, et pour présenter adéquatement leur fonctionnement aux utilisateurs, il fallait développer de nouvelles animations plus élaborées que celles déjà existantes. Parmis ces animations, notons celle permettant de déplacer, centrer et redimensionner un automate dans une section de l'espace d'affichage, celle permettant de faire converger un ensemble d'états vers un même point ainsi que la construction automatique du tableau servant à minimiser un automate.

- L'utilisateur a maintenant la possibilité de spécifier lui-même le nom des états, au lieu de laisser le logiciel générer une numérotation automatique.

- La classe définissant une arête et son affichage offre désormais la possibilité de dessiner une arête sous forme d'une série de segments de droites, plutôt qu'en un seul segment. Cette fonctionnalité n'est par contre pas encore disponible à l'utilisateur car l'interface ne permet pas d'interagir avec cette option.

- Il est possible pour l'utilisateur de modifier les dimensions du graphe et de rétablir les dimensions par défaut, à l'aide des options du menu d'affichage.

- Le menu d'affichage contient aussi une nouvelle option permettant d'effacer le contenu de la section d'affichage des traces des algorithmes.

Toutes ces modifications sont relativement simples et servent à rendre l'utilisation du logiciel un peu plus flexible pour l'utilisateur. Il pourrait tout de même être souhaitable d'aller plus loin dans l'amélioration de l'interface afin d'offrir un environnement de travail encore plus intuitif, ce qui serait très bénéfique considérant le caractère éducatif de l'application. Il pourrait être intéressant aussi d'ajouter un mode *tutoriel* qui expliquerait en mots et en animation le fonctionnement de base de chaque algorithme. De tels ajouts nécessiteraient par contre de retoucher certaines parties de la conception du logiciel, afin d'offrir plus de services de base au niveau de l'affichage. L'animation est déjà automatisée en bonne partie, mais il y aurait place à amélioration, surtout si l'on désire obtenir une qualité supérieure d'affichage et une interface plus élaborée.

Conclusion

L'affichage des graphes est un domaine relativement vaste et complexe et concevoir un nouvel algorithme d'affichage devant répondre à des besoins bien particuliers n'était pas une tâche triviale. Il était donc primordial d'utiliser une démarche bien structurée afin d'obtenir un résultat satisfaisant, d'autant plus que ce domaine m'était inconnu au début du projet.

Dans ces conditions, les résultats que j'ai obtenus, principalement au niveau de l'algorithme permettant de créer une représentation planaire d'un automate, me semble être très bons. J'obtiens une représentation qui respecte les heuristiques choisis et qui ressemble grandement à ce qu'un être humain obtiendrait comme résultat de manière intuitive.

Pour ce qui est de l'algorithme servant à positionner les noeuds sur l'espace d'affichage, l'implémentation n'est pas complétée, mais je suis confiant d'en arriver à un résultat tout aussi concluant que pour le premier algorithme, lorsque le travail sera terminé.

Au niveau de l'interface utilisateur, les éléments ajoutés offrent essentiellement plus de flexibilité à l'utilisateur. Les animations développées pour les algorithmes de déterminisation et de minimisation sont plus élaborées que celles qui existaient déjà pour les algorithmes qui avaient été implémentés avant le début du projet, ce qui est un pas dans la bonne direction. Il reste cependant beaucoup de travail à faire si l'on souhaite avoir un logiciel hautement intuitif et qui exploite au maximum le potentiel éducatif que les technologies actuelles permettent.

Bibliographie

[1] DI BATTISTA, Giuseppe, et autres. *Graph Drawing: Algorithms For the Visualization of graphs*, New Jersey, Prentice-Hall, 1999, 397 p.

[2] KAUFFMANN, Michael, et Dorothea WAGNER. *Drawing Graphs: Methods and models*, Berlin, Springer, 2001, 312 p.

[3] F. COHEN, Robert, et autres. «A framework for dynamic graph drawing»*, Proc. 8th Symp. Computational Geometry*, ACM, juin 1992, p. 261–270.

[4] HOPCROFT, John et Robert TARJAN. «*Efficient planitary testing*», *Journal of the ACM*, Volume 21, n° 4, octobre 1974, p. 549-568.

[5] JUNGER, Michael et Petra MUTZEL. «Maximum Planar Subgraphs and Nice Embeddings: Practical Layout Tools», *Algorithmica*, Volume 162, p. 33-59, 1996.

[6] CIMIKOWSKI, Robert, «An analysis of some heuristics for the maximum planar subgraph problem», *Proc. 6th annual ACM-SIAM symp. on Discrete algorithms*, ACM, 1995, p. 322-331.

[7] F. CRUZ, Isabel et Roberto TAMASSIA. *Graph Drawing Tutorial*, [En ligne], 1998, [http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf], (11 septembre 2004).

La version finale du logiciel est disponible à l'adresse suivante :
http://w3.uqo.ca/basc01/labo1/index.php?labo1_s=logiciel