MASTER EN SCIENCES INFORMATIQUES

Développement d'un SIAD sous Windows basé sur la méthode du Recuit Simulé pour l'optimisation d'un problème de transport

Christodoulidis, Christos; Delhaye, Marc

Award date:
1994

Awarding institution:
Universite de Namur

acultés
niversitaires
.D. de la Paix
amur

stitut d'Informatique

Développement d'un SIAD sous Windows basé sur la méthode du Recuit Simulé pour l'optimisation d'un problème de transport

Christodoulidis Christos
Delhaye Marc

Promoteur : J. Fichefet
Copromoteur : J.P. Leclercq

Mémoire présenté en vue de l'obtention du titre de Licencié et Maître en informatique

Année académique 1993-1994

Nous tenons à remercier

notre promoteur, Monsieur J. Fichefet pour la liberté et l'aide lors de la rédaction de ce mémoire;

Monsieur J.P. Leclercq pour ses précieux conseils;

Monsieur J. Christodoulidis pour nous avoir fourni l'idée de départ du sujet de ce mémoire;

Mademoiselle Lia pour sa patience lors de notre stage en Crète;

Mademoiselle Marie-Jeanne Collard pour l'inspiration.

Abstract

Ce mémoire présente la conception et l'implémentation d'un logiciel d'aide à la décision (*Mythos*), ayant comme but l'optimisation d'un problème de ramassage scolaire où chaque étudiant est soumis à des contraintes horaires. La conception de ce logiciel repose sur une réduction du problème initial en une généralisation du problème de coloration des noeuds d'un graphe non-orienté à l'aide d'un algorithme de calcul du plus court chemin dans un graphe orienté. Le problème de coloration des noeuds d'un graphe étant de nature NP-Complet, l'utilisation d'une heuristique pour sa résolution nous a permis d'obtenir des résultats assez étonnants pour des problèmes de taille. L'heuristique utilisée repose sur un hybride du recuit simulé. Cette heuristique produit des résultats meilleurs que ceux qui ont été précédemment présentés dans la littérature concernant les problèmes de collecte, en général. Le logiciel a été implémenté en utilisant le *langage basé sur objets "Visual Basic 3.0 pour MS-Windows"*.

Mots clés : recuit simulé, heuristique, optimisation combinatoire, problèmes de transport, NP-Complet, coloration d'un graphe, plus court chemin dans un graphe.

Abstract

This master thesis presents the design and implementation of a software (*Mythos*) intended to optimize a school bus transport problem with time-windows. The design of this software is based on a reduction of the initial problem to a generalisation of a non-oriented graph nodes coloration problem, using of a shortest path algorithm. It can be shown that the graph node coloration problem is NP-Hard, hence the need for heuristic solution methods. Owing to the computational complexity of the problem, we have used a simulated annealing hybrid heuristic which produces very good solutions reasonably quickly. The software has been implemented using the *object based language "Visual Basic 3.0 for MS-Windows"*.

Keywords : simulated annealing, heuristics, time-windows, combinatorial optimization, transport problems, NP-Hard, graph nodes coloration problem, shortest path problem.

Table des matières

Chapitre Un : Modélisation du problème

Chapitre Deux : Le Recuit Simulé

Chapitre Trois : Le plus court chemin dans un graphe

Chapitre Quatre : Implémentation des différents modules

Avant-propos

Le problème de tournée de véhicules consiste à déterminer pour une flotte de véhicules les trajets à effectuer, c'est-à-dire quels clients visiter et dans quel ordre. De nombreuses applications existent: réparation d'appareils ménagers à domicile, livraison de fuel, collecte d'argent dans les agences bancaires par des véhicules blindés, ramassage d'étudiants dans les écoles, ...

L'objet de ce mémoire consiste à concevoir et à implémenter un logiciel d'aide à la décision à la planification d'un ramassage scolaire. La méthode utilisée est inspirée d'un exposé donné par Monsieur Marc J.A. Lescrenier dans le cadre du cours de "*Programmation d'applications scientifiques*" de Monsieur J.P. Leclercq. La conception du logiciel a été effectuée lors d'un stage Comett en Crète (Grèce) supervisé par Monsieur J. Fichefet, tandis que l'implémentation de celui-ci a été réalisée aux Facultés Universitaires Notre Dame de la Paix de Namur (Belgique).

Etant donné la volonté de développer un logiciel de qualité malgré les contraintes temporelles qui nous étaient imparties, l'utilisation d'un langage de prototypage tel que Visual Basic pour Microsoft Windows s'est avérée nécessaire.

Tout au long de la réalisation de ce travail de fin d'études, dans le but de rester fidèles à une démarche universitaire, nous nous sommes efforcés de mettre en pratique la diversité des connaissances acquises lors de notre formation. Les bases mathématiques de notre enseignement nous ont permis de développer un modèle basé sur un formalisme rigoureux, cette démarche de formalisation faisant d'ailleurs partie intégrante du travail d'informaticien. D'autre part, c'est grâce à cette formation d'informaticien qu'il nous a été loisible de réaliser une solution abstraite correcte, efficace, et indépendante des outils de réalisation ainsi que de traduire celle-ci pour une machine et un langage de programmation spécifique.

Chapitre Un
Modélisation du problème

1. Présentation du problème

1.1 Introduction

Dans ce chapitre, nous allons présenter le problème traité tel qu'il a été posé par la direction d'une école primaire. En nous basant sur cette présentation, nous allons réduire le problème posé à un problème **d'optimisation combinatoire** comprenant une **fonction-objectif** et certaines **contraintes**. Nous déduirons ensuite la **modélisation mathématique** du problème combinatoire et nous présenterons enfin la découpe en modules du problème modélisé.

1.2 Le problème traité

La direction d'une école primaire est confrontée au problème suivant. Chaque matin un certain nombre de bus scolaires, de capacités différentes, se charge du ramassage des étudiants à partir de leur domicile pour les conduire à l'école.

Chaque bus a un nombre maximal de places disponibles (la capacité du bus) et une fois arrivé à l'école, il ne peut plus repartir. De plus, tous les bus doivent être à l'école au plus tard à une heure fixée par le directeur d'école (l'heure à laquelle les cours commencent).

Chaque étudiant impose une contrainte horaire du type : *"l'étudiant X doit être pris par un bus (quelconque) entre l' heure Y et l' heure Y+h"* où h est un intervalle (en minutes) propre à chaque étudiant.

Le directeur de l'école souhaite savoir s'il existe une solution possible à son problème, en tenant compte des contraintes horaires des étudiants. Dans le cas où il en existerait une, il souhaiterait minimiser le nombre de bus utilisés pour l'opération de ramassage, vu qu'un bus représente un très grand budget pour l'école. Finalement, il souhaiterait une répartition uniforme des étudiants dans les bus utilisés, de telle manière qu'aucun bus ne ni soit trop vide ni trop plein.

Le logiciel développé doit fonctionner sur un micro-ordinateur du type PC-AT et l'environnement utilisé doit être convivial afin de permettre à des personnes qui ne sont pas des informaticiens de l'utiliser.

1.3 Caractéristiques du problème

La résolution du problème consiste à trouver un chemin pour chaque bus sur un graphe orienté représentant les routes de la ville, respectant les contraintes du paragraphe précédent. On peut remarquer qu'il existe des similarités entre notre problème et le problème du facteur chinois (CPP).

Le CPP consiste à trouver un chemin de **coût minimal** qui traverse chaque arc du graphe au moins une fois, et qui à la fin, aboutit à l'origine. Mais dans notre cas, il existe des contraintes supplémentaires :

- certains arcs du graphe ne sont pas utilisés (non servis);
- la capacité de chaque bus limite les services offerts par celui-ci (s'il est rempli), et impose le besoin d'utiliser plusieurs véhicules.

Un second problème similaire au nôtre est le CARP (*Capacitated Arc Routing Problem*). Ce problème a été trivialement introduit par Golden and Wong de la manière suivante.

Soit un graphe non orienté, **G=(V,A,C,D)**

où **V** est l'ensemble des noeuds du graphe;

A est l'ensemble des arcs du graphe;

C est la matrice des coûts d'utilisation des arêtes;

D est la matrice des demandes de service des arêtes.

En supposant que tous les coûts et toutes les demandes soient non négatifs, il est demandé de trouver un circuit , de coût minimal, qui passe par un noeud de dépôt (e.g. école) et satisfaisant toutes les demandes sans dépasser la capacité limite de chaque véhicule.

Golden et Wong ont démontré que ce problème est NP-Hard (NP-Complet). Ils ont également démontré que même le problème trouvant une solution 1,5 fois moins bonne que la solution optimale est dans la classe NP-Hard.

Par conséquent, nous pouvons dire que, à première vue, notre problème risque d'être dans la classe NP.

2. Formalisation mathématique du problème

2.1 Définition des ensembles

Il n'est pas impossible que des notations et appellations explicitées ci-après heurtent quelque peu certaines personnes. Disons simplement ici qu'elles nous ont paru commodes pour des questions d'implémentation et de convivialité du logiciel associé à notre mémoire. Nous utiliseront les notations suivantes :

- **N** est l'ensemble des noeuds du graphe dont chaque élément représente un domicile possible;
- $\mathbf{A} \subset \mathbf{N} \times \mathbf{N}$ est l'ensemble des arcs du graphe dont chaque élément représente un segment d'une route qui relie deux domiciles possibles;
- **C**=(**A**,Coût) est appelé l' "*ensemble des coûts d'utilisation*" des arcs du graphe dont chaque élément $((n_i,n_j),c)$ représente le coût $c \in$ Coût, en secondes, du passage à partir d'un élément $n_i \in \mathbf{N}$ vers un autre élément $n_j \in \mathbf{N}$ (telle que $(n_i,n_j) \in \mathbf{A}$);
- **E**=(**N**,H,H) est un triplet qui sera appelé l' "*ensemble des étudiants*" où
  - **N** est l'ensemble défini ci-dessus,
  - H est l'ensemble des heures possibles (00:00 → 23:59);
- **B**=(Pref,Cap, $\wp$(**E**)) est un triplet qui sera appelé l' "*ensemble des bus*" où

  Pref représente l'appréciation du directeur de l'école pour un bus par rapport aux autres bus, appréciation mesurée sur une échelle d'ordre strict total < : {Weak, Medium, Strong, Absolute}

  avec la convention :

  Weak < Medium

  Medium < Strong

  Strong < Absolute

- Cap représente la capacité du bus, i.e. le nombre maximal de places dans le bus,
- $\wp(\mathbf{E})$ représente une partition de l'ensemble **E** des étudiants.

- **P=(N,C,E,B)** est l'ensemble des problèmes de transport tels qu'ils ont été définis au début de ce chapitre.

2.2 Sémantique des éléments des ensembles

2.2.1 Sémantique des éléments de l'ensemble E

Un élément $e_i=(n,h_{tôt},h_{tard}) \in \mathbf{E}$ représente un étudiant qui habite au noeud n, dont l'*heure au plus tôt* du ramassage est $h_{tôt}$ et l'*heure au plus tard* est h_{tard}.

2.2.2 Sémantique des éléments de l'ensemble B

Un élément $b_j=(p,c, \wp_j(\mathbf{E})) \in \mathbf{B}$ représente un bus dont

- la préférence (donnée par le directeur d'école) par rapport aux autres bus, est p,
- le nombre maximal de places est c,
- le sous-ensemble des étudiants ramassés par ce bus est $\wp_j(\mathbf{E})$.

2.3 Contraintes

- **Contraintes concernant l'ensemble C**

1. *Le coût pour passer d'un noeud vers un autre noeud, s'il existe un arc entre eux deux, est strictement positif;*

$\forall k : 1 \leq k \leq \|\mathbf{C}\|, \; c_k \in \mathbf{C} : c_k=(n_i,n_j,c) \Rightarrow (c>0)$

où

$n_i \in \mathbf{N}$

$n_j \in \mathbf{N}$

$n_i \neq n_j$

2. *Le coût pour passer d'un noeud vers le même noeud, en une transition, est nul, c'est-à-dire :*

$\forall k : 1 \leq k \leq \|\mathbf{C}\|, c_k \in \mathbf{C} : c_k=(n,n,c) \Rightarrow (c=0)$

où $n \in \mathbf{N}$

3. *Il existe au plus un arc entre deux noeuds;*

$\forall i : 1 \leq i \leq \|\mathbf{N}\|,$

$\forall j : 1 \leq j \leq \|\mathbf{N}\| \Rightarrow ! \; c_k \in \mathbf{C}, 1 \leq k \leq \|\mathbf{C}\| : c_k=(n_i,n_j,c)$ où $n_i,n_j \in \mathbf{N}$

- **Contraintes concernant l'ensemble E**

4. *L'heure au plus tôt est strictement plus petite que l'heure au plus tard;*

$\forall i : 1 \leq i \leq \|\mathbf{E}\| \Rightarrow [\; e_i=(n,h_{i\,tôt},h_{i\,tard}) \in \mathbf{E} \Rightarrow h_{i\,tôt}<h_{i\,tard}) \;]$

où $n \in N$, $h_{i\,tôt} \in H$, $h_{i\,tard} \in H$

- **Contraintes concernant l'ensemble B**

5. *La capacité de chaque bus est strictement positive;*

$\forall i : 1 \leq i \leq \|\mathbf{B}\| \Rightarrow [\; b_i=(p,c,\wp_i(\mathbf{E})) \in \mathbf{B} \; \Rightarrow c>0 \;]$

où $p \in$ Pref, $c \in$ Coût, $\wp_i(\mathbf{E}) \in \wp(\mathbf{E})$

6. *La capacité d'un bus est plus grande ou égale au nombre d'étudiants dans ce bus;*

   $\forall i : 1 \leq i \leq \|\mathbf{B}\| \Rightarrow [\, b_i=(p,c,\wp_i(\mathbf{E})) \in \mathbf{B} \Rightarrow c \geq \|\wp_i(\mathbf{E})\| \,]$

   où $p \in$ Pref, $c \in$ Coût, $\wp_i(\mathbf{E}) \in \wp(\mathbf{E})$

7. *Chaque étudiant est ramassé par un et un seul bus, i.e. $\wp(\mathbf{E})$ est bien une partition de $\mathbf{E}$;*

   $\forall e \in \mathbf{E} \Rightarrow [\, \exists! i : (1 \leq i \leq \|\mathbf{B}\|)$

   $\wedge\ b_i=(p,c,\wp_i(\mathbf{E})) \in \mathbf{B}$

   $\wedge\ p \in$ Pref

   $\wedge\ c \in$ Coût

   $\wedge\ \wp_i(\mathbf{E}) \in \wp(\mathbf{E})$

   $\wedge\ e \in \wp_i(\mathbf{E})\ ]$

8. *La somme des capacités sur tous les bus est plus grande ou égale au nombre total d'étudiants;*

$$\sum_{\substack{b_i=(Pref_i,Cap_i,\wp_i) \in \mathbf{B} \\ i=1}}^{\|\mathbf{B}\|} Cap_i \leq \|\mathbf{E}\|$$

2.4 Contrainte principale

Deux étudiants ne peuvent se trouver dans un même bus que s'il existe un chemin entre eux, de coût c, tel qu'il est possible d'en ramasser un des deux dans les limites de ses contraintes horaires, de traverser le chemin de coût c, et de ramasser l'autre dans les limites de ses contraintes;

$$\forall e,e' \in E, \begin{cases} e = (n, h_{tôt}, h_{tard}) \\ e' = (n', h'_{tôt}, h'_{tard}) \\ n, n' \in N \\ h_{tôt}, h_{tard}, h'_{tôt}, h'_{tard} \in H \end{cases} :$$

$$\begin{cases} \left[\begin{array}{l} \exists i : (1 \le i \le \|B\|) \wedge (b_i = (p, c, \wp_i(E)) \in B) \\ \wedge (p \in Pref) \wedge (c \in Coût) \wedge (\wp_i(E) \in \wp(E)) \\ \wedge (e \in \wp_i(E)) \wedge (e' \in \wp_i(E)) \end{array}\right] \\ \Rightarrow \left[\begin{array}{l} (\exists h \in [h_{tôt}, h_{tard}]) \wedge (\exists h' \in [h'_{tôt}, h'_{tard}]) \\ \wedge (\exists \text{ une suite } c_1, c_2, \ldots, c_k \in C) : \\ \begin{cases} \langle c_1 = ((n, n_1), coût_1), c_2 = ((n_1, n_2), coût_2), \ldots, c_k = ((n_{k-1}, n'), coût_k) \rangle \\ \text{telle que } h + \sum_{l=1}^{k} coût_l \le h' \end{cases} \\ \vee \\ \begin{cases} \langle c_1 = ((n', n_1), coût_1), c_2 = ((n_1, n_2), coût_2), \ldots, c_k = ((n_{k-1}, n), coût_k) \rangle \\ \text{telle que } h' + \sum_{l=1}^{k} coût_l \le h \end{cases} \end{array}\right] \end{cases}$$

2.5 La fonction de coût

La fonction de coût doit satisfaire aux propriétés suivantes :

1. Elle doit exprimer l'objectif du problème (d'où l'appellation *fonction-objectif*);
2. Elle doit être rapidement calculable, car elle sera évaluée des milliers des fois.

Vu que l'on veut minimiser le nombre de bus nécessaires au ramassage des étudiants, la première fonction de coût qui vient à l'esprit serait ce nombre de bus. L'objectif serait dès lors atteint en minimisant cette fonction. Observons le comportement de cette fonction sur l'exemple suivant:

Le seul moyen pour faire varier la valeur de la fonction f est la répartition des étudiants dans les bus. Supposons deux répartitions (deux solutions possibles) dans les différents bus pour effectuer le ramassage scolaire de 40 étudiants:

Solution A:	Bus 1 (capacité=15) : 10 étudiants
	Bus 2 (capacité=15) : 10 étudiants
	Bus 3 (capacité=15) : 10 étudiants
	Bus 4 (capacité=15) : 10 étudiants

Solution B:	Bus 1 (capacité=15) : 14 étudiants
	Bus 2 (capacité=15) : 12 étudiants
	Bus 3 (capacité=15) : 13 étudiants
	Bus 4 (capacité=15) : 1 étudiant

L'évaluation de la fonction de coût pour ces deux solutions donne chaque fois la même valeur : 4 bus ! Or, intuitivement, la solution B est meilleure que la solution A car elle est plus proche d'une solution à trois bus. Cette fonction n'est sensible qu'aux transitions qui font passer le dernier étudiant d'un bus vers un autre. Elle ne reflète pas les *taux de remplissages* des différents bus. Cette fonction n'est donc pas adaptée à notre problème.

C'est pourquoi, étant donné un problème $p \in \mathbf{P}$, nous avons choisi de **maximiser** la fonction f définie par :

$$f: \mathbf{P} \mapsto \mathbb{R}$$
$$p \rightarrow f(p) = \sum_{b_i=(p_i,c_i,\wp_i(\mathbf{E}))\in \mathbf{B}} p_i \cdot \|\wp_i(\mathbf{E})\|$$

La multiplication externe entre un élément $p_i \in$ Pref et un entier n'est pas encore définie. Pour ce faire et vu que le nombre d'éléments distincts de l'ensemble Pref est très réduit, il suffit d'attribuer à chaque élément de Pref une valeur entière de telle manière que les conditions d'antisymétrie et de transitivité soient vérifiées.

Nous poserons donc :

Weak	=	0
Medium	=	10
Strong	=	20
Absolute	=	30

Montrons intuitivement que la maximisation de la fonction f, comme décrite ci-dessus, exprime bien l'objectif du problème :

Si tous les bus ne possèdent pas les mêmes préférences, alors, la valeur de la fonction de coût prendra une valeur plus ou moins grande suivant la répartition des étudiants dans les bus de préférences différentes. Cette fonction de coût est donc sensible aux transitions qui transfèrent un étudiant entre deux bus de préférences différentes. La maximisation de la fonction f se ramène dès lors au remplissage des bus possédant une *forte* préférence, et la fonction de coût atteindra sa valeur maximale pour une répartition des étudiants telle que l'on ait rempli d'abord tous les bus de préférence *Absolute*, ensuite (si il y a encore des étudiants qui ne sont pas ramassés par des bus de préférence *Absolute*) ceux de préférence *Strong*, etc.

Par conséquent, plus la préférence d'un bus est élevée (par rapport aux autres bus), plus ce bus sera rempli.

3. Découpe du problème en sous-problèmes

3.1 Introduction

Dans ce paragraphe nous allons réduire un modèle simplifié de notre problème en un problème de coloration des noeuds d'un graphe, et ensuite nous généraliserons la réduction pour trouver une découpe en modules du problème initial.

La seule différence entre le modèle simplifié considéré ici et le modèle présenté aux paragraphes 1.2.1 et 1.2.2 concerne la contrainte horaire de l'étudiant. Dans le modèle simplifié, *l'heure de ramassage au plus tôt* et *l'heure de ramassage au plus tard* ***sont égales***.

3.2 Le problème de coloration des noeuds d'un graphe

Etant donné un graphe non orienté G=(S,Ar), où S représente l'ensemble des sommets du graphe et Ar l'ensemble des arêtes, **la coloration des noeuds du graphe G** consiste à attribuer à chaque noeud une couleur (p.e. un nombre entier) de telle manière que deux noeuds adjacents (reliés par une arrête) ont toujours une couleur différente. Autrement dit, la coloration du graphe G consiste à trouver un *mapping* f de S dans $\{1,2,\ldots,k\}$ telle que

$$\forall\, u,v \in S : (f(u)=f(v)) \Rightarrow ((u,v) \notin Ar).$$

Le **problème de coloration d'un graphe** consiste à *minimiser* le nombre k (le nombre des couleurs utilisées pour colorer le graphe).

Définition : La **matrice d'adjacence Adj** d'un graphe G=(S,Ar) est une matrice $\|S\| \times \|S\|$ dont chaque élément Adj_{ij} vérifie les conditions suivantes :

$$\forall i,j \in S : \{\ (i,j) \in Ar \Leftrightarrow Adj_{ij}=1\ \}$$

Remarque : La matrice d'adjacence est une matrice carrée **symétrique** car le graphe est non orienté.

3.3 Réduction du modèle simplifié à un problème de coloration d'un graphe

Définissons un graphe non orienté G=(E,Ar) où

- E est l'ensemble des noeuds du graphe correspondant à l'ensemble **E**=(N,H,H)[1] défini dans le paragraphe 1.2.1. Chaque noeud du graphe G représente un étudiant *à une heure précise*.
- Ar est l'ensemble des arêtes du graphe, et est défini comme suit :

  $\forall i,j \in E$:

  $\{(i,j) \notin Ar \Leftrightarrow$ "l'étudiant i peut se trouver dans le même bus que l'étudiant j"$\}$

Autrement dit, il existe une arête entre deux étudiants si et seulement si il est impossible de ramasser l'un des deux dans les limites de ses contraintes horaires, de traverser le chemin **de coût minimal** joignant les deux étudiants, et de ramasser l'autre dans les limites de ses contraintes horaires.

Remarque : La contrainte présentée ci-dessus implique la contrainte principale du paragraphe 1.2.3.

Nous pouvons constater que, en faisant correspondre un bus à chaque couleur, la solution au problème de coloration des noeuds de ce graphe est une solution à notre problème de transport. En effet, cette solution est telle que :

- le nombre de couleurs (donc le nombre de bus) est minimisé,
- deux sommets (donc deux étudiants) ne possèdent la même couleur (ne sont ramassés par le même bus) que s'il n'y a pas d'arête entre eux (que s'ils peuvent être ramassés par le même bus).

Notre problème de transport se "*résume*" donc à deux sous-problèmes:

1. la construction du graphe représentant les données du problème;
2. la résolution du problème de coloration des noeuds de ce graphe.

---

[1] Dans le modèle *simplifié* chaque élément $e_i=(n_i,h_{i\,tôt},h_{i\,tard})$ de **E** est de la forme $h_{i\,tôt}=h_{i\,tard}$.

3.4 Construction du graphe

Il est évident que si nous connaissons le plus court chemin (le chemin de coût minimal) entre chaque paire d'étudiants, alors nous pouvons *déduire* la matrice d'adjacence du graphe G.

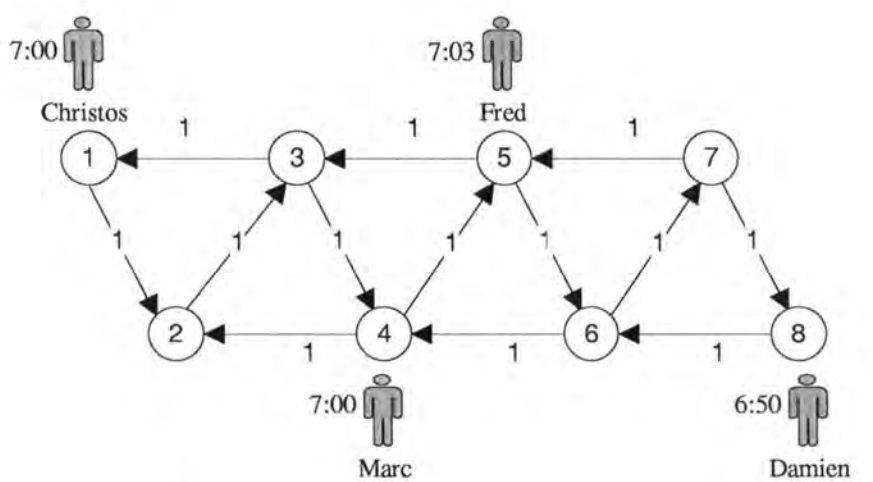
Exemple :

Soit un problème défini par les ensembles suivants:

N = {1,2,3,4,5,6,7,8 },

C = {((1,2),1), ((2,3),1), ((3,4),1), ((4,5),1), ((5,6),1), ((6,7),1), ((7,8),1), ((3,1),1), ((5,3),1), ((7,5),1), ((8,6),1), ((6,4),1), ((4,2),1)},

E = { Christos = (1, 7:00, 7:00), Fred = (5, 7:03, 7:03), Marc = (4, 7:00, 7:00), Damien = (8, 6:50, 6:50)}.



Le calcul du plus court chemin (en minutes) entre chaque paire d'étudiants nous donne :

Christos → Fred = 4,	Marc → Damien = 4
Christos → Marc = 3,	Marc → Christos = 3
Christos → Damien = 7	Marc → Fred = 1
Fred → Christos = 2	Damien → Christos = 5
Fred → Marc = 2	Damien → Marc = 2
Fred → Damien = 3	Damien → Fred = 3

Pour créer le graphe G=(E,Ar) nous devons savoir, pour chaque paire d'étudiants, s'ils peuvent se trouver dans un même bus. A cette fin, nous utilisons la formule suivante :

> Deux étudiants $e_i=(i,h_{i\ tôt},h_{i\ tard})$[2] et $e_j=(j,h_{j\ tôt},h_{j\ tard})$ peuvent se trouver dans un même bus
>
> si et seulement si
>
> $(h_{i\ tôt}+plus\_court\_chemin_{ij} \leq h_{j\ tard}) \vee (h_{j\ tôt}+plus\_court\_chemin_{ji} \leq h_{i\ tard})$
>
> où $plus\_court\_chemin_{ij}$ est une fonction de $\mathbf{C} \rightarrow \mathbf{IN}_+$ qui donne comme résultat la valeur (en minutes) du plus court chemin pour aller de l'étudiant i vers l'étudiant j.

Vérifions cette condition pour Christos et Fred:

Nous avons (c.f. tableau précédent) $plus\_court\_chemin_{15}=4$, $plus\_court\_chemin_{51}=2$, $h_{1\ tôt}=7:00$ et $h_{5\ tôt}=7:03$. Par la formule précédente on obtient pour la partie gauche de la condition :

$$(7:00+4 \leq 7:03) \vee (7:03+2 \leq 7:00) \Rightarrow$$

$$(7:04 \leq 7:03) \vee (7:05 \leq 7:00) \Rightarrow$$

$$\underline{Faux} \vee \underline{Faux} \Rightarrow$$

$$\underline{Faux} \Rightarrow$$

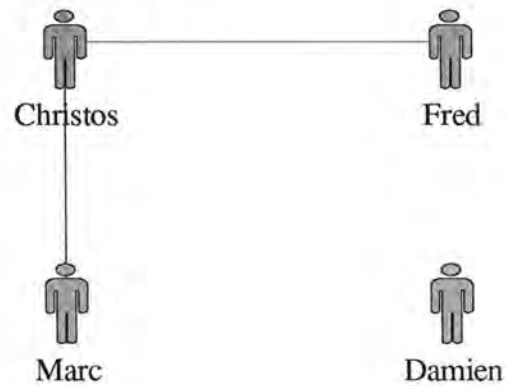
Christos et Fred ne peuvent pas se trouver dans le même bus $\Rightarrow$

il existe une arête entre Christos et Fred dans le graphe G $\Rightarrow$

$Adj_{Christos\ Fred}=1$ *et* $Adj_{Fred\ Christos}=1$.

[2] Avec $h_{i\ tôt}=h_{i\ tard}$ et $h_{j\ tôt}=h_{j\ tard}$

En faisant ce raisonnement pour chaque paire d'étudiants, on obtient le graphe G :



et la matrice d'adjacence associée au graphe G:

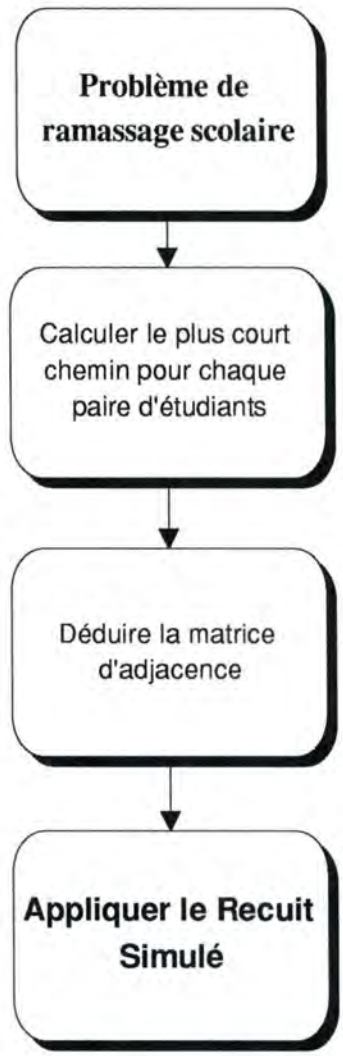
$$\text{Adj} = \begin{array}{c} \\ \text{Christos} \\ \text{Fred} \\ \text{Marc} \\ \text{Damien} \end{array} \begin{pmatrix} & & & \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Nous venons de construire un procédé pour *déduire* la matrice d'adjacence à partir les données initiales du problème, nécessitant le calcul des plus courts chemins entre chaque paire d'étudiants. Ce calcul s'effectuera à l'aide d'un algorithme de calcul des plus courts chemins qui sera développé dans le chapitre 3.

3.5 Résolution du problème de coloration des noeuds du graphe

La résolution du problème de coloration des noeuds du graphe se fera grâce à l'heuristique du *recuit simulé*. Le développement de cette heuristique est l'objet du chapitre 2.

Une première découpe du problème simplifié en modules est représentée par la figure suivante.


Problème de ramassage scolaire
Calculer le plus court chemin pour chaque paire d'étudiants
Déduire la matrice d'adjacence
Appliquer le Recuit Simulé

3.6 Le principe des supervertex

Dans les paragraphes précédents nous avons réduit notre problème simplifié à un problème de coloration de graphe. Dans les deux paragraphes qui suivent nous allons généraliser cette réduction pour qu'elle soit conforme au problème initial.

Rappelons que dans le problème de transport initial, l'heure au plus tôt de ramassage d'un étudiant était strictement plus petite que l'heure au plus tard de ce même étudiant, et cela pour tout étudiant.

Dans le modèle simplifié, un noeud du graphe représentait le ramassage d'un étudiant à un instant précis.

Dans le modèle général, nous devons considérer un intervalle de temps (*time-window*) et plus un instant précis. Dès lors, nous allons faire correspondre un ensemble des noeuds à chaque étudiant, dont chaque noeud représente un intervalle de temps très court (de l'ordre de la minute). Autrement dit, chaque noeud de cet ensemble correspond à une heure de ramassage possible pour un étudiant donné. Dorénavant nous appellerons chaque sous ensemble de noeuds, ayant la propriété de représenter un même étudiant à des intervalles distincts, un ***supervertex***. Chaque étudiant est par conséquent représenté par un et un seul supervertex, et chaque élément de ce supervertex correspond à un intervalle de temps différent. Le nombre de noeuds dont nous avons besoin pour représenter la tranche horaire d'un étudiant, est fonction de :

- l'heure au plus tôt et au plus tard du ramassage (time-window) concernant cet étudiant;
- le précision que nous imposons sur l'heure exacte du ramassage.

Le deuxième point mérite une étude plus approfondie.

Supposons que pour l'étudiant *Damien* :

l'heure au plus tôt est égale à 7:00;

l'heure au plus tard est égale à 7:15.

Si nous voulons une précision à une minute près, nous avons besoin de ((7:15)-(7:00))/1 min = (15 min) / (1 min) = 15 noeuds pour représenter le supervertex de cet étudiant!

Dans ce supervertex, le premier noeud représente la tranche horaire [7:00→7:01], le deuxième la tranche horaire [7:01→7:02], etc.

Remarque : Les instants 7:01, 7:02, ... , 7:14, sont représentes deux fois chaque, car ils sont repris dans les deux intervalles successifs. Nous avons fait le choix d'inclure les instants limites dans les deux intervalles successifs car la précision est (dans ce cas) de l'ordre de la minute, et par conséquent le résultat est précis à une minute près!

Pour généraliser ce raisonnement pour un étudiant quelconque, notons par

- *inter* la précision du résultat (en minutes), i.e. la longueur du plus court intervalle considéré;
- $h_{tôt}$ l'heure au plus tôt de ramassage;
- h_{tard} l'heure au plus tard de ramassage.

Nous aurons donc besoin de $(h_{tard} - h_{tôt})/inter$ noeuds pour représenter le supervertex de l'étudiant.

Conventions

- La précision du résultat (variable *inter*) est commune pour tous les étudiants et la précision maximale est de l'ordre de la minute.
- Deux supervertex distincts ne contenant pas nécessairement le même nombre de noeuds, la longueur du *time - window* de chaque étudiant peut être quelconque, la seule restriction étant : $h_{tard} > h_{tôt}$.

3.7 Le problème étendu de la coloration d'un graphe

Après avoir étendu la notion de graphe en lui ajoutant la structure de supervertex, nous allons définir une extension au problème de coloration d'un graphe. Cette extension doit exprimer le fait que nous ne voulons plus colorer tous les noeuds du graphe, mais seulement un noeud de chaque supervertex.

Une définition étendue du problème de coloration d'un graphe qui tient compte des supervertex est la suivante :

> Trouver une coloration minimale i.e. un *mapping* *'color'* de $\mathbf{E}^*$ dans {0, 1, 2, ... , nb_bus} telle que
>
> - nb_bus soit minimal;
> - tous les noeuds d'un supervertex sont colorés avec la couleur 0, sauf un qui est coloré avec une couleur entre 1 et nb_bus;

- $\forall$ et_1, $et_2 \in \mathbf{E}^*$ tel que $Adj_{et1,et2}=1$, nous avons $color(et_1) \neq color(et_2)$ ou $color(et_1) = color(et_2) = 0$

où $\mathbf{E}^*$ est l'ensemble des étudiants (c.f. 1.2.1) muni de la structure de supervertex.

Remarque : $\|\mathbf{E}^*\| = \sum_{i \in E} (h_{i\,tard} - h_{i\,tôt}) / inter.$

Dans la paragraphe 1.3.4, nous avons créé un procédé pour déduire la matrice d'adjacence à partir du graphe initial, en utilisant un algorithme de plus court chemin. Si nous généralisons ce procédé pour qu'il soit compatible au principe des supervertex nous obtenons le graphe non orienté $\mathbf{G}=(\mathbf{E}^*,Ar)$ où

- $\mathbf{E}^*$ est l'ensemble des noeuds du graphe, correspondant à l'ensemble **G**. Chaque noeud du graphe **G** représente un étudiant à une heure précise suivant le principe des supervertex présenté dans le paragraphe précèdent.
- Ar est l'ensemble des arrêtes du graphe qui est défini par :

$$\forall\ i,j \in \mathbf{E}^* :$$

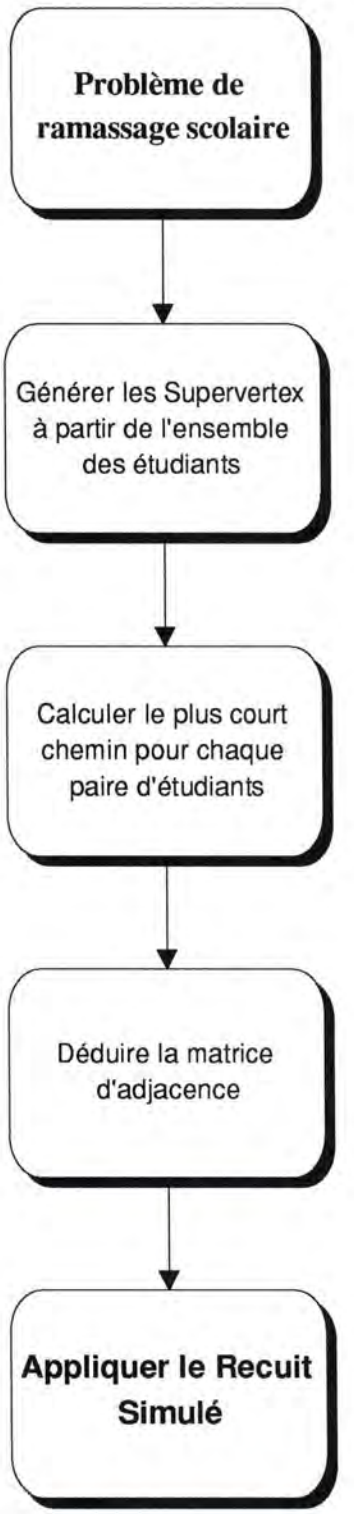
  - Si i n'appartient pas au même supervertex que j alors

    $\{(i,j) \notin Ar \Leftrightarrow$ "l'étudiant *e*, ramassé à l'heure *h* correspondant au noeud *i* de $\mathbf{E}^*$, peut se trouver dans le même bus que l'étudiant *e'*, ramassé à l'heure *h'* correspondant au noeud *j* de $\mathbf{E}^*$"$\}$;

  - Si i appartient dans le même supervertex que j, il n'a pas de sens de parler du fait qu'un étudiant peut ou pas se retrouver dans le même bus que lui même !

Autrement dit, il existe une arrête entre deux noeuds (étudiants distincts), appartenant chacun dans un supervertex distinct, *si et seulement si*, il est impossible de ramasser un des deux étudiants dans les limites de sa contrainte horaire (déterminé par le noeud du supervertex), parcourir le chemin de coût minimal entre ceux là, et ramasser l'autre dans les limites de sa contrainte horaire (déterminé par le noeud du supervertex).

En ajoutant le principe des Supervertex dans la découpe en modules du problème simplifié, nous obtenons la découpe en modules suivante pour le problème initial.


Problème de ramassage scolaire
Générer les Supervertex à partir de l'ensemble des étudiants
Calculer le plus court chemin pour chaque paire d'étudiants
Déduire la matrice d'adjacence
Appliquer le Recuit Simulé

Chapitre Deux
Le Recuit Simulé

1. Introduction

En 1953, Metropolis et al. ont proposé un algorithme qui permet de simuler, de façon efficace, l'évolution d'un solide vers un état d'équilibre thermique. Cependant, il aura fallu plus de 30 ans avant que Kirkpatrick et al. et Cerny réalisent qu'il existe une profonde analogie entre la minimisation de la fonction économique d'un problème d'optimisation combinatoire et le refroidissement lent d'un solide jusqu'au moment où il atteint son énergie minimale. Ce processus d'optimisation peut être réalisé en appliquant successivement le critère de Metropolis. En substituant le coût à l'énergie et en exécutant l'algorithme de Metropolis avec une séquence de valeurs de température décroissantes, Kirkpatrick et ses coéquipiers ont obtenu un algorithme d'optimisation combinatoire qu'ils ont appelé "**simulated annealing**" (**recuit simulé**).

Encore actuellement, la solution de beaucoup de problèmes combinatoires de grandes tailles peut seulement être approximée, surtout à cause du fait que beaucoup de ces problèmes ont été démontrés *NP-Hard*[1] . Pour ce problème, nous sommes forcés d'utiliser des algorithmes qui approximent la solution, autrement dit, des **heuristiques**. L'utilisation de tels algorithmes ne garantit pas que la solution trouvée soit optimale, mais assure par contre, que le temps d'exécution de ceux-ci puisse être borné par un polynôme dépendant de la taille du problème[2] .

[1] de complexité non polynomiale

[2] cfr Annexe 1

2. Formalisation d'un problème combinatoire

Un **problème combinatoire** peut être formalisé comme un couple ($\mathfrak{R}$,C) où

- $\mathfrak{R}$ est un ensemble fini ou dénombrable appelé **espace de configurations**,
- et C une **fonction de coût**, autrement appelée fonction économique.

La fonction C est de la forme: C: $\mathfrak{R} \rightarrow \mathbb{R}$, ce qui veut dire qu'à chaque configuration de l'espace, elle fait correspondre un nombre réel. Sans perte de généralité supposons que la meilleure configuration pour la fonction C est celle qui correspond à la plus petite valeur de C, i.e. celle qui correspond au minimum de la fonction.

Le problème sera donc de trouver cette configuration pour laquelle C prend sa valeur minimale, i.e. $C_{opt} = C(i_o) = \min_{i \in \mathfrak{R}} C(i)$ où C_{opt} représente le coût minimal.

Comme nous l'avons déjà dit, il existe deux approches possibles pour essayer de résoudre un problème d'optimisation combinatoire.

La première approche est d'utiliser un **algorithme d'optimisation** qui garantit une solution optimale mais le prix à payer sera le temps d'exécution de celui-ci.

La deuxième approche sera d'utiliser une **heuristique** qui approxime la solution optimale dans un temps d'exécution acceptable.

Ces algorithmes d'approximation peuvent à leur tour être classés en deux catégories:

- les algorithmes construits pour résoudre un problème **spécifique** (*Tailored Algorithms*) et
- les algorithmes généraux applicables à une **grande variété** de problèmes d'optimisation combinatoire.

Le recuit simulé peut être vu comme un algorithme appartenant à la deuxième catégorie d'algorithmes approximatifs. C'est une technique d'optimisation générale applicable à des problèmes d'optimisation combinatoire et capable de fournir une solution très proche de la solution optimale.

3. Introduction au recuit simulé

Cet algorithme est basé sur une **technique d'optimisation aléatoire** mais en plus, il intègre des aspects relatifs aux **algorithmes d'amélioration itérative**. Dans ce qui suit, ces deux aspects seront expliqués avant d'entrer dans les détails de l'étude de l'algorithme.

3.1 L'algorithme de Metropolis (d'amélioration itérative)

L'application d'un algorithme dit d'amélioration itérative (A.I.) présuppose la définition:

- de l'**espace de configurations**,
- de la fonction économique ou **fonction de coût**,
- d'un **mécanisme capable de gérer des transitions d'un état de l'espace de configurations vers un autre état** (voisin du premier) appartenant au voisinage du précédant.

Utilisons la formalisation des **réseaux de Pétri** pour définir le terme **voisinage** d'un état de l'espace de configurations. Supposons que les noeuds du graphe du réseau de Pétri représentent les différentes configurations de l'espace et que les arcs représentent les transitions possibles d'un état vers un autre.

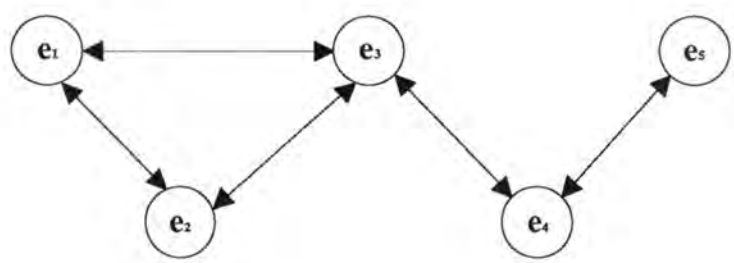


Figure 1

Le **voisinage** de e_i est défini comme l'ensemble des noeuds adjacents au noeud e_i. Autrement dit, le voisinage (*neighbourhood*) R_i pour une configuration (état) i est l'ensemble des différentes configurations qui peuvent être atteintes à partir de i en

faisant une seule transition. Pour cette raison-ci, la technique d'AI est aussi connue sous le nom de **recherche locale** ou **recherche au voisinage**. Pour utiliser une telle technique il est évident qu'on a besoin d'une configuration initiale appelée **solution initiale**.

L'organigramme de l'algorithme d'AI est le suivant:

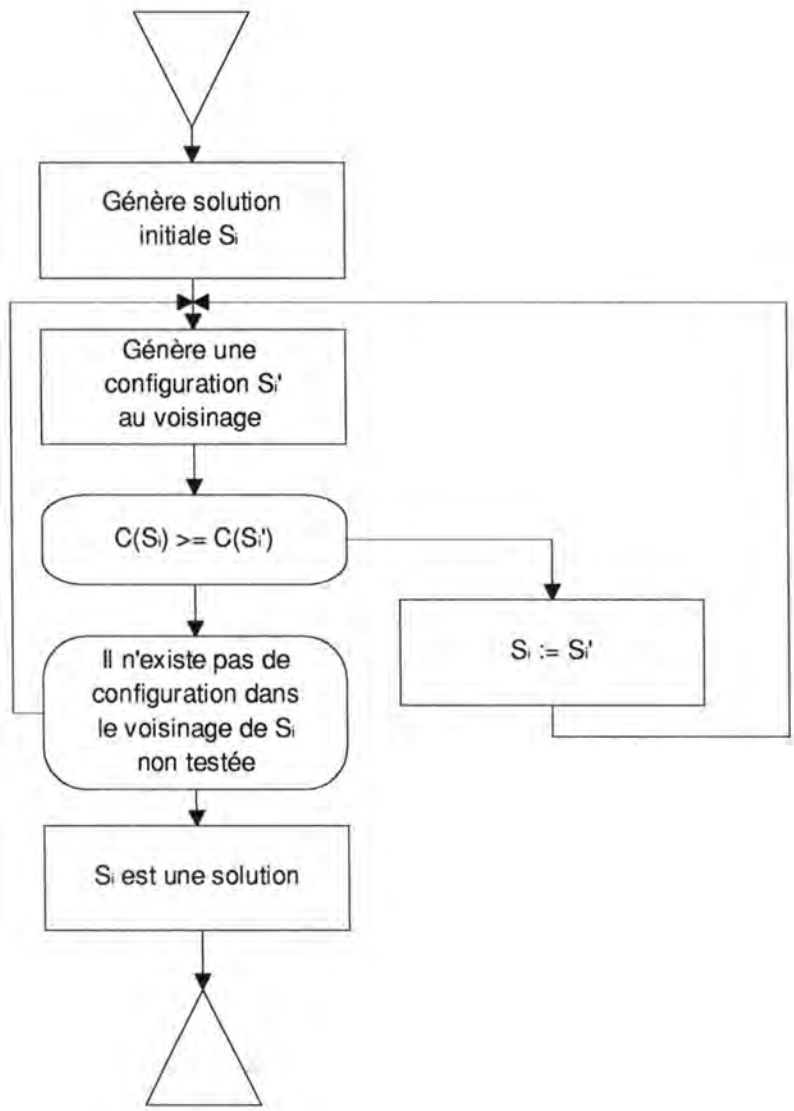


Figure 2: Algorithme d'amélioration itérative

3.2 Critiques de la méthode

Par définition, cette technique trouve un optimum (minimum) local, et on n'a aucune information sur l'écart entre cet optimum local et l'optimum global. Donnons un exemple d'une fonction C sur un espace $\Re$ où cette technique peut très mal fonctionner :

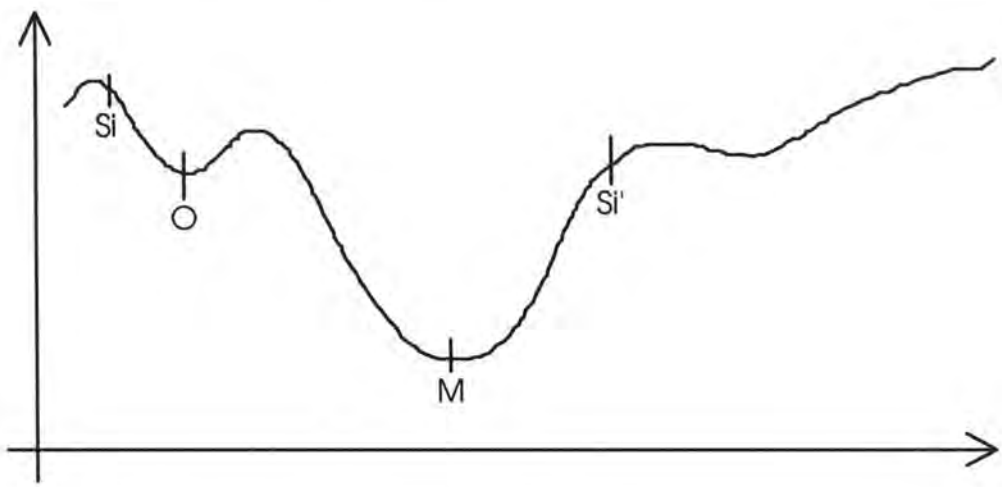


Figure 3

où Si: solution initiale

O: optimum trouvé par l'algorithme d'AI

M: optimum (minimum) global

Si': autre solution initiale

On peut remarquer sur la figure 3 que l'optimum local dépend fortement de la solution (configuration) initiale. Si, à la place de la configuration **Si** on avait choisi **Si'**, comme configuration initiale, la solution finale serait le **minimum global**.

On ne peut pas connaître d'avance le nombre d'itérations dont l'algorithme a besoin pour trouver un optimum local. Le temps d'exécution de cet algorithme ne peut donc pas être borné par un polynôme dépendant de la taille du problème[3].

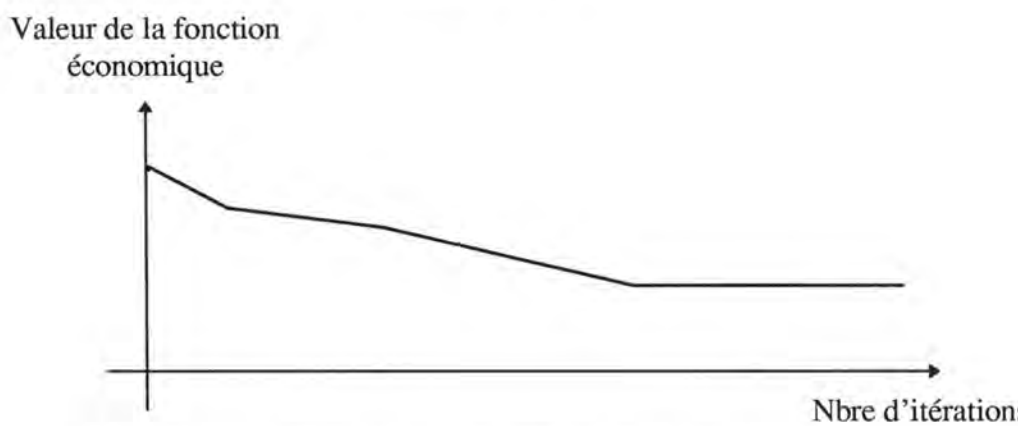
[3] cfr Annexe 1

3.3 La technique d'optimisation aléatoire

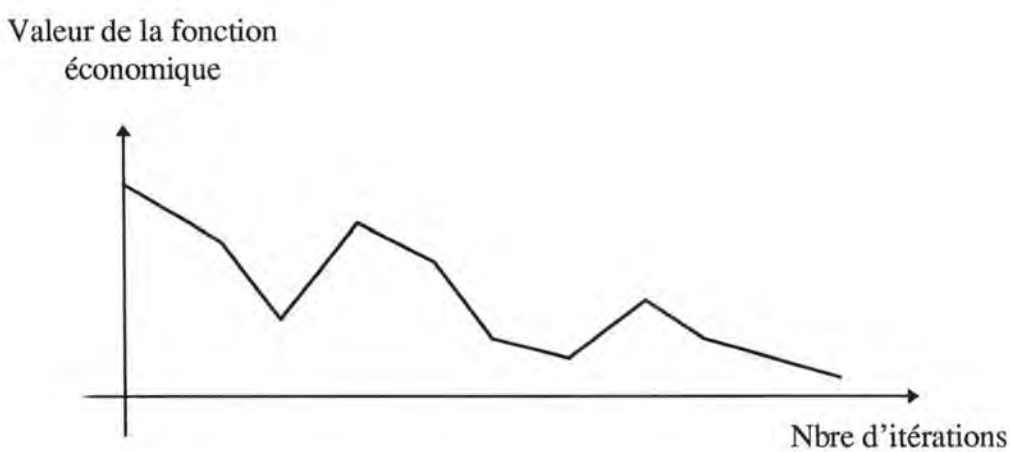
Malgré tout, l'algorithme d'amélioration itérative reste une méthod*e facilement implementable et générale*. Mais, pour éviter certains des désavantages cités ci-dessus, on pourrait imaginer l'alternative suivante:

L'algorithme pourrait **parfois** accepter des transitions qui correspondent à une augmentation de la fonction économique en espérant qu*'il pourra ainsi échapper aux minima locaux.*

En effet, les itérations successives de l'exécution de l'algorithme d'AI (pour un problème de minimisation) vont toujours dans le sens d'une évolution décroissante de la fonction économique, comme dans la forme :



Or il serait intéressant d'obtenir un algorithme dont les itérations succesives aillent dans le sens d'une évolution de la fonction économique plus *"chaotique"*, comme dans la forme:



Kirkpatrick et al. ont développé l'algorithme du recuit simulé, qui utilise cette alternative. Celui-ci est également connu sous les appellations "*simulated annealing*", "*Monte Carlo annealing*", "*statistical cooling*", "*probabilistic hill climbing*", "*stochastic relaxation*", *ou "probabilistic exchange algorithm*".

4. Le Recuit simulé

4.1 Introduction à l'algorithme

En physique de la matière, "**annealing**" représente un processus physique dans lequel un solide subit un **bain de chaleur** (*"heat bath"*): on augmente d'abord la température du bain jusqu'à une valeur maximale, telle que toutes les particules du corps se positionnent **aléatoirement** pendant la *phase liquide*, et on diminue ensuite la température du bain jusqu'à ce que le solide se refroidisse. En fixant au départ la température à une valeur maximale assez élevée, l'énergie interne du solide augmente (car l'énergie cinétique des particules du solide augmente), ce qui permet aux particules du solide de circuler à des vitesses élevées d'une manière aléatoire. En diminuant **progressivement** la température du bain, l'énergie interne du solide diminue et, par conséquent, l'énergie cinétique des particules de celui-ci. Il existe une basse température pour laquelle **aucun mouvement de particules n'est permis**. Si au départ la température est assez élevée, pour avoir un mélange aléatoire de distribution uniforme, et si la température diminue assez lentement et progressivement, alors, durant le passage à l'état de température minimale, les particules ont l'occasion de s'arranger de telle manière que l'entropie du système (désordre) soit minimale. L'aspect physique sera étudié en détail dans le paragraphe 2.6.

4.2 Description du processus de recuit

A chaque valeur de température T, il est permis au solide d'atteindre un **équilibre thermique** (“*thermal equilibrium*”) caractérisé par une probabilité de se trouver en un état d'énergie E décrit par la distribution de Boltzmann:

$$\Pr\{IE = E\} = \frac{1}{Z(T)} \cdot \exp\left(-\frac{E}{k_B \cdot T}\right) \qquad \textbf{(1)}$$

où

- IE représente l'état d'énergie du solide;
- Z(T) est un facteur de normalisation appelé **fonction de partition**. Ce facteur dépend de la température T;
- k_B est la **constance de Boltzmann**.

Le facteur $\exp\left(-\frac{E}{k_B \cdot T}\right)$ est connu sous le nom "**facteur de Boltzmann**".

Comme la température tendra vers zéro, seuls les états d'énergie minimales auront une probabilité non nulle d'apparaître.

Remarque: Si le refroidissement est trop rapide, c.à.d. si le solide n'a pas le temps d'atteindre son **équilibre thermique** pour chaque étage de température, il peut geler (“*froze*”) sans atteindre sa structure cristalline correspondant à son énergie minimale mais en contenant des structures amorphes (Figure 6). C'est pour cette raison que le refroidissement doit être assez lent.

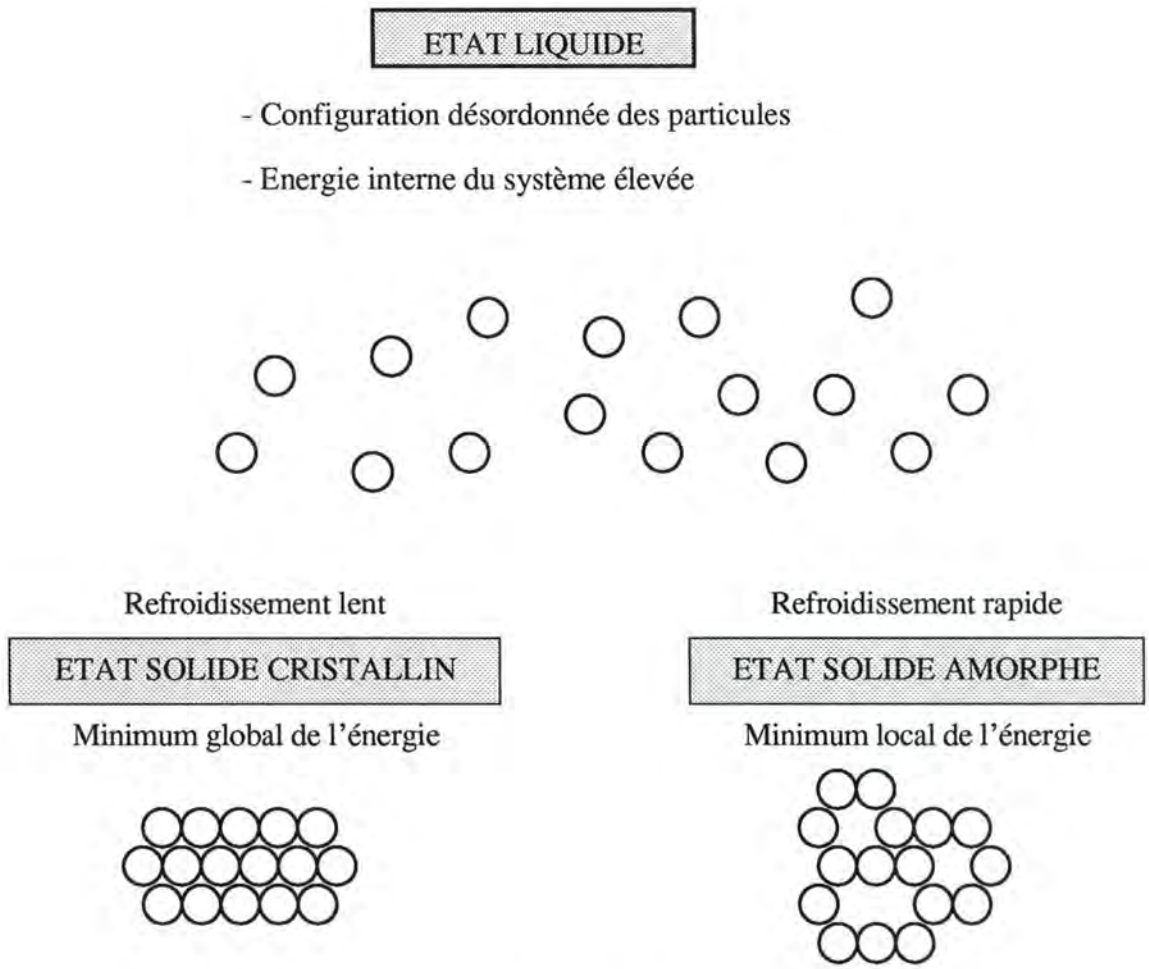

ETAT LIQUIDE
- Configuration désordonnée des particules
- Energie interne du système élevée
Refroidissement lent
Refroidissement rapide
ETAT SOLIDE CRISTALLIN
ETAT SOLIDE AMORPHE
Minimum global de l'énergie
Minimum local de l'énergie

Figure 6

4.3 Le critère de Metropolis

Pour simuler l'évolution des états de particules d'un solide vers l'équilibre thermique, pour une valeur de température T fixée, Metropolis et al. ont proposé une méthode dite "*de Monte Carlo*", qui génère des séquences de configurations de particules d'un solide de la manière qui suit.

Etant donné l'état actuel du solide caractérisé par la position de ses particules, une petite perturbation est générée aléatoirement, par exemple par un petit déplacement d'une particule choisie au hasard.

Si la différence d'énergie ΔE entre l'état après la perturbation et l'état avant celle-ci est **négative**, i.e. si la perturbation provoque une *diminution de l'énergie interne du solide*, alors l'état après la perturbation est accepté et devient donc le nouvel état actuel (courant) et le processus s'exécutera de nouveau à partir de cet état.

Si la différence d'énergie ΔE est **plus grande ou égale à 0**, alors *la probabilité d'acceptation de l'état après la perturbation* en tant qu'état actuel est donnée par:

$$\exp\left(\frac{-\Delta E}{k_B \cdot T}\right).$$

Cette règle d'acceptation est appelée "**critère de Metropolis**". Après un grand nombre de perturbations en utilisant le critère de Metropolis, *la fonction de distribution de la probabilité des états approxime la distribution de Boltzmann* et donc le système évolue vers un état d'équilibre thermique.

Cette technique est connue sous le nom d'"**algorithme de Metropolis**". Cet algorithme peut être utilisé pour générer des séquences de configurations d'un problème d'optimisation combinatoire. Dans ce cas, on obtient l'analogie suivante:

Solide	**Problème combinatoire**
Etat	Configuration
E (énergie)	C (fonction de coût)
$k_B T$ (température)	c (paramètre de contrôle)

4.4 Recuit simulé

Le recuit simulé peut être vu comme *une suite d'algorithmes de Metropolis* évalués pour une suite décroissante de valeurs du paramètre de contrôle. Il peut être décrit comme suit:

Notons ΔC_{ij} = C(j) - C(i), alors **la probabilité que la configuration j sera la configuration suivante** est donnée par:

$$\Pr\{\text{config} = j\} = \begin{cases} 1 & \text{si } \Delta C_{ij} \le 0 \\ \exp\left(-\dfrac{\Delta C_{ij}}{c}\right) & \text{si } \Delta C_{ij} > 0 \end{cases} \quad \{\textbf{critère de Metropolis}\} \quad (2)$$

On voit donc qu'*il existe une probabilité non nulle de choisir une configuration d'un coût plus élevé que celui de la configuration actuelle.* Le processus est continué tant que l'équilibre n'est pas atteint, i.e. jusqu'à ce que la distribution de la probabilité de configuration approche suffisamment la distribution de Boltzmann qui est donnée par:

$$\Pr\{\text{config} = i\} \overset{\text{def}}{=} q_i(c) = \frac{1}{Q(c)} \cdot \exp\left(-\frac{C(i)}{c}\right) \qquad (3)$$

où Q(c) est une constante de normalisation dépendante du paramètre de contrôle c, étant équivalente à la fonction de partition.

Une fois l'équilibre atteint, la valeur du paramètre de contrôle c est diminuée (le système refroidit) et le processus se répète comme décrit plus haut, par la génération d'une nouvelle configuration. L'algorithme se termine pour une petite valeur de c pour laquelle, *virtuellement*, aucune perturbation pouvant diminuer la fonction de coût C n'est permise. La dernière configuration obtenue est la solution "approchée" fournie par l'algorithme pour le problème d'optimisation combinatoire donné.

Une fois que les *configurations*, la *fonction de coût*, la *structure du voisinage* et le *mécanisme de génération des configurations* sont définis, nous obtenons l'**heuristique générale** de la figure 7, où le paramètre c a été appelé "*température*" par analogie avec la technique physique du recuit mais avec un certain abus de langage.

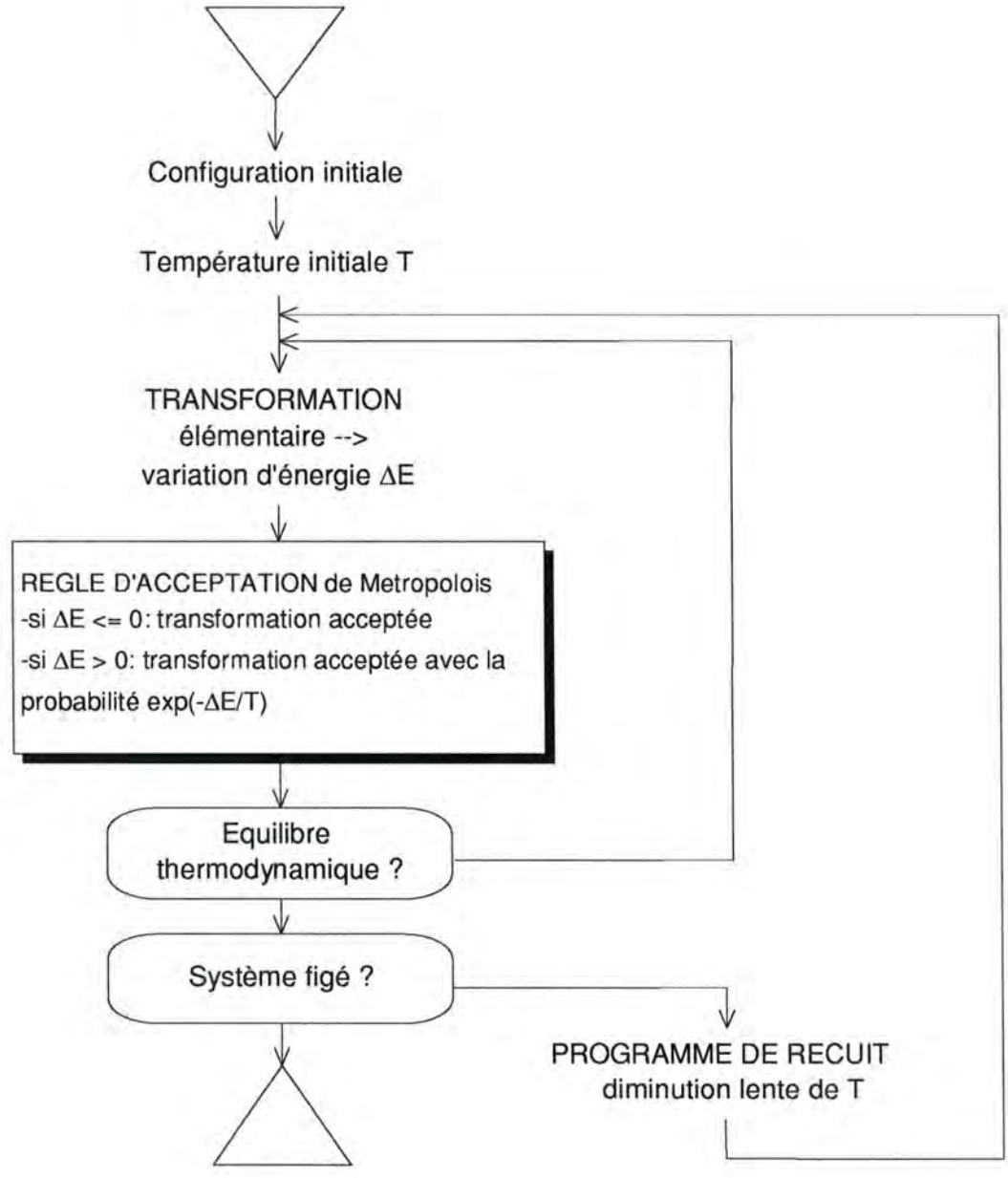


Figure 7: Algorithme général de recuit

5. Modèle mathématique de l'algorithme

Etant donné une structure de voisinage le recuit simulé peut être vu comme un algorithme qui essaye continuellement de transformer la configuration courante en une autre appartenant à l'ensemble formé par les configurations *voisines* de la première.

Ce mécanisme peut être décrit dans un formalisme mathématique par le moyen des **chaînes de Markov**. Dans ce formalisme, étant donné une configuration (la configuration courante), la suivante ne dépend que de la configuration courante.

Une chaîne Markov est décrite à l'aide des probabilité conditionnelles $P_{ij}(k-1,k)$ pour chaque paire de configurations (i,j) où $P_{ij}(k-1,k)$ est la probabilité que la configuration j *sortira* au $k^{ème}$ *tir*, sachant que juste avant le $k^{ème}$ *tir* on avait la configuration i.

Notons $a_i(k)$ *la probabilité que la configuration i sortira au* $k^{ème}$ *tir*

Calculons cette probabilité :

$a_i(1) = a_i(0)$ i.e. que $a_i(1)$ est la configuration initiale

$$a_i(2) = \sum_j a_j(1) \cdot P_{ji}(1,2) \text{ , } j \in \|\Re\|$$

(...)

$$a_i(k) = \sum_j a_j(k-1) \cdot P_{ji}(k-1,k) \text{ , } j \in \|\Re\| \text{ , } k = 1,2 \text{ , } ...$$

Remarque : La somme sur $j \in |\Re|$ signifie la somme sur toutes les configurations possibles.

Notons par X(k) la configuration sortie au $k^{ème}$ tir, on a donc:

$P_{ij}(k-1,k) = Pr\{X(k) = j \mid X(k-1) = i\}$

et $a_i(k) = Pr\{X(k) = i\}$

Définition : Si la probabilité conditionnelle ne dépend pas de k, la **chaîne de Markov** correspondante est appelée **homogène**.

Définition : Si la probabilité conditionnelle dépend de k, **la chaîne de Markov** correspondante est appelée **inhomogène.**

5.1 La matrice des transitions

Dans le cas du recuit simulé, la probabilité conditionnelle P_{ij} (k-1,k) représente la probabilité que la $k^{ème}$ *transition* soit une transition de la configuration i à la configuration j.

Définition : La probabilité $P_{ij}(k-1,k)$ est appelée **probabilité de transition.**

Définition : La matrice $P(k-1,k) = (P_{ij}\ (k-1,k))_{(i,j)\ \in\ \Re\times\Re}$ est appelée **matrice des transitions.**

Comme on l'a déjà remarqué, les probabilités de transitions dépendent de la valeur du paramètre de contrôle c (où c représente la température dans le processus du recuit physique).

Si c est une constante, la chaîne de Markov correspondante est homogène et la matrice des transitions P=P(c) peut être définie comme suit:

$$P_{ij}(c) = \begin{cases} G_{ij}(c) \cdot A_{ij}(c) & \forall j \neq i \\ 1 - \sum_{\substack{l=1 \\ l \neq i}}^{|\Re|} G_{il}(c) \cdot A_{il}(c) & \forall j = i \end{cases} \qquad (4)$$

où $G_{ij}(c)$ est la probabilité de générer la configuration j à partir de la configuration i et s'appelle la **probabilité de génération.**

$A_{ij}(c)$ est la probabilité d'acceptation (dans le cas du recuit cette acceptation se fait suivant le **critère de Métropolis**) de la configuration j générée à partir de la configuration i et est appelée le **probabilité d'acceptation.**

Et quand i = j :

$$\begin{aligned} P_{ij}(c) &= P_{ii}(c) \\ &= \Pr(\text{la configuration i est générée à partir de la configuration i}) \\ &= 1 - \sum_{\substack{\text{toutes les} \\ \text{config} \\ \text{sauf i}}} \Pr(\text{La configuration i est générée à partir d'une config. différente de i}) \end{aligned}$$

Définition : La matrice $G(c) = (G_{ij}(c))_{(i,j)\ \in\ \Re\times\Re}$ est appelée **matrice de génération.**

Définition : La matrice $A(c) = (A_{ij}(c))_{(i,j)\ \in\ \Re\times\Re}$ est appelée **matrice d'acceptation.**

Remarque : La matrice P(c) est une matrice **stochastique**

i.e. $$\sum_{j} P_{ij}(c) = 1$$

car $$\sum_{j} P_{ij}(c) = \sum_{\substack{j=1 \\ j\neq i}}^{|\Re|} G_{ij}(c)\, A_{ij}(c) + 1 - \sum_{\substack{l=1 \\ l\neq i}}^{|\Re|} G_{il}(c)\, A_{il}(c) = 1.$$

5.2 Formulations du recuit simulé à l'aide des chaînes de Markov

Dans les paragraphes précédents on a considéré le paramètre de contrôle c comme étant constant. Mais la valeur de c diminue pendant l'exécution de l'algorithme. En se basant sur cette diminution de la valeur de c, nous pouvons distinguer deux formulation possibles de l'algorithme à partir de la formulation des chaînes de Markov:

1. L'algorithme peut être considéré comme un algorithme **homogène** si on considère qu'il est décrit par une suite de chaînes de Markov **homogènes**. Chaque suite de Markov de la suite est générée pour une valeur fixée de cette valeur de c diminue entre deux éléments de la suite.
2. L'algorithme peut être considéré comme un algorithme **inhomogène** si on considère qu'il est décrit par une seule de chaîne de Markov **inhomogène**. Dans ce cas, la valeur de c diminue entre deux transitions successives.

On peut remarquer que dans le premier cas, on tient compte de toutes les configurations considérées pendant l'exécution complète de l'algorithme, tandis que dans le second cas, on ne tient compte que des configurations correspondant à l'équilibre pour une valeur de température c.

5.3 Convergence asymptotique de l'algorithme

Le recuit simulé donne comme résultat un minimum global après k transitions si la condition suivante est vérifiée:

$$\Pr (X(k) \in \Re_{opt}) = 1 \qquad \textbf{(5)}$$

où $\Re_{opt}$ est l'ensemble des configurations globalement minimales.

5.3.1 L'algorithme homogène

Nous allons montrer que pour l'algorithme homogène on a:

$$\lim_{c \xrightarrow{c>0} 0} \left(\lim_{k \to \infty} \Pr (X(k) \in \Re_{opt}) \right) = 1$$

si
1. toute chaîne de Markov est de longueur infinie
2. certaines conditions sur les matrices $A(c_l)$ et $G(c_l)$ sont satisfaites

où c_l représente la valeur du paramètre de contrôle pour le $l^{ème}$ élément de la suite des chaînes de Markov.

5.3.1.1 Plan de la démonstration de la convergence asymptotique de l'algorithme homogène

Définition : La **distribution stationnaire** d'une chaîne de Markov est définie comme un vecteur q dont la $i^{ème}$ composante est donnée par:

$$q_i = \lim_{k \to \infty} \Pr (X(k) = i \text{ étant donné que } X(0) = j) \text{ pour un j arbitraire} \qquad \textbf{(6)}$$

Un point essentiel dans la démonstration de l'algorithme homogène est le fait que sous certaines conditions la **distribution stationnaire** d'une chaîne de Markov homogène existe.

Nous avons fait l'hypothèse qu'une configuration ne dépend que de la précédente (hypothèse de Markov). Nous avons donc:

$$q_i = \lim_{k \to \infty} \Pr (X(k) = i) \qquad \textbf{(7)}$$

Si nous notons par a(0) la distribution des probabilités initiales, nous obtenons:

$[a(0)]_i = a_i(0)$, $\forall i \in \Re$	
$a_i(0) \geq 0$, $\forall i \in \Re$ $\sum_{i \in \Re} a_i(0) = 1$	Conditions pour que $a_i(0)$ soit une fonction de distribution. **(8)**

Nous obtenons alors

$$q = \lim_{k \to \infty} a(0)^T P^k \qquad \textbf{(9)}$$

Nous avons considéré ci-dessus que cette distribution stationnaire existe pour déduire qu'elle est égale à $\lim_{k \to \infty} a(0)^T P^k$, ce qui veut dire que la distribution stationnaire est la distribution de la probabilité des configurations après un nombre infini de transitions.

Dans un paragraphe précédent, nous avons expliqué que la matrice des transitions P dépendait de la valeur du paramètre de contrôle c (i.e. P=P(c)). On a alors $q_i=q_i(c)$ et par conséquent q=q(c).

Plan de la démonstration

1. Nous allons trouver des conditions sur les matrices A(c) et G(c) qui définissent la chaîne homogène de Markov de telle sorte qu'une distribution stationnaire q(c) existe (c.f. 2.5.3.1.2).
2. Ensuite, nous raffinerons ces conditions en tenant compte du fait que la valeur du paramètre de contrôle c décroît et nous montrerons que q(c) converge vers une distribution uniforme sur l'ensemble des configurations qui sont des minimums globaux (c.f. 2.5.3.1.3).

i.e. $$\lim_{c \xrightarrow{c>0} 0} q(c) = \pi \qquad \textbf{(10)}$$

où π est R-vecteur défini par

$$\pi_i = \begin{cases} \dfrac{1}{\|\Re_{opt}\|} & \text{si } i \in \Re_{opt} \\ 0 & \text{Partout ailleurs} \end{cases} \qquad \textbf{(11)}$$

où $\Re_{opt}$ est l'ensemble des configurations correspondant aux minima globaux

Par (9) et (10) nous obtenons pour la $i^{ème}$ composante:

$$\lim_{c \xrightarrow{c>0} 0} q_i(c) = \pi_i$$

$$\Rightarrow \lim_{c \xrightarrow{c>0} 0} (\lim_{k \to \infty} \Pr (X(k) = i)) = \pi_i \qquad \textbf{(12)}$$

La preuve de la convergence de l'algorithme vers une solution de $\Re_{opt}$ se déduit alors directement du fait que l'on a considéré la distribution uniforme (c.f. éq(11)):

$$\lim_{c \xrightarrow{c>0} 0} (\lim_{k \to \infty} \Pr (X(k) \in \Re_{opt}))$$

$$= \lim_{c \xrightarrow{c>0} 0} (\lim_{k \to \infty} \sum_{l \in \Re_{opt}} \Pr (X(k) = l))$$

$$= \lim_{c \xrightarrow{c>0} 0} (\lim_{k \to \infty} (\frac{1}{\|\Re_{opt}\|} + \frac{1}{\|\Re_{opt}\|} + ... + \frac{1}{\|\Re_{opt}\|} + 0 + ... + 0))$$

$$= \lim_{c \xrightarrow{c>0} 0} (\lim_{k \to \infty} \|\Re_{opt}\| \frac{1}{\|\Re_{opt}\|}))$$

$$= 1$$

et donc :

$$\lim_{c \xrightarrow{c>0} 0} (\lim_{k \to \infty} \Pr (X(k) \in \Re_{opt})) = 1 \qquad \textbf{(13)}$$

5.3.1.2 Existence de la distribution stationnaire

Définition : Une chaîne de Markov est **irréductible**

SSI pour tous les couples (i,j) il existe une probabilité non nulle d'atteindre la configuration j à partir de la configuration i en un nombre **fini** de transitions

i.e. $\forall\, i,j\ \exists\, n : 1 \leq n < \infty$ et $(P^n)_{ij} > 0$

Définition : Une chaîne de Markov est **apériodique**

SSI pour toutes les configurations $i \in \Re$ le P.G.C.D. de tous les entiers $n \geq 1$ t.q. $(P^n)_{ii} > 0$ est égal à 1

Grâce au théorème suivant, nous allons prouver l'existence de la distribution stationnaire d'une chaîne de Markov générée par l'algorithme de Recuit Simulé.

Théorème 1 (Feller)

La **distribution stationnaire** q d'une chaîne de Markov **homogène** et **finie** existe **SI** la chaîne de Markov est **irréductible** et **apériodique.**

De plus, on sait que, par définition, la distribution stationnaire (le vecteur q) est définie par les équations suivantes:

$$\forall\, i : q_i > 0\ ,\ \Sigma_i\, q_i = 1 \qquad \textbf{(14)}$$

$$\forall\, i : \Sigma_j\, q_j\, P_{ij} = q_i \qquad \textbf{(15)}$$

Remarque : Par (14) et (15), on voit aussi que le vecteur q est le vecteur propre de la matrice P de valeur propre égale à l'unité.

Cherchons les conditions pour que les chaînes de Markov générées par l'algorithme de Recuit Simulé soient irréductibles et apériodiques.

- **Irréductibilité**

Dans le cas du recuit simulé, la matrice P est définie par l'équation (4) :

Comme $\forall$ i, j, c > 0 on a $A_{ij}(c) > 0$ [4], il suffit que G(c) soit irréductible.

Une <u>première condition</u> à imposer est donc la suivante :

$\forall i, j \in \Re, \exists\ 1 \le p \le \infty$ et $\exists$ une suite $l_0, l_1, l_2, \ldots, l_p \in \Re$ t.q. $l_0=1$ et $l_p=j$

t.q. $G_{l_k l_{k+1}}(c) > 0$, k = 0, 1, 2, ... , p-1 **(16)**

- **Apériodicité**

Pour établir l'apériodicité, on utilise le fait qu'une chaîne de Markov irréductible est apériodique si la condition suivante est satisfaite :

$$\forall\ c > 0, \exists\ i_c \in \Re : P_{i_c i_c}(c) > 0 \qquad \textbf{(17)}$$

Pour que cette condition soit satisfaite, on impose une <u>seconde condition</u> sur la matrice A d'acceptation :

$$\forall\ c > 0, \exists\ i_c, j_c \in \Re : A_{i_c j_c} < 1 \qquad \textbf{(18)}$$

En effet, nous savons que $\forall$ i, j : $A_{ij} \le 1$ **(**)**

donc : $\sum_{\substack{l=1 \\ l \neq i_c}}^{|\Re|} A_{i_c l}(c) \cdot G_{i_c l}(c) =$

$$\begin{aligned}
&= \sum_{\substack{l=1 \\ l \neq i_c \\ l \neq j_c}}^{|\Re|} A_{i_c l}(c) \cdot G_{i_c l}(c) + A_{i_c j_c}(c) \cdot G_{i_c j_c}(c) \\
&\overset{(18)}{<} \sum_{\substack{l=1 \\ l \neq i_c \\ l \neq j_c}}^{|\Re|} A_{i_c l}(c) \cdot G_{i_c l}(c) + G_{i_c j_c}(c) \\
&\overset{(**)}{\le} \sum_{\substack{l=1 \\ l \neq i_c \\ l \neq j_c}}^{|\Re|} G_{i_c l}(c) + G_{i_c j_c}(c) \\
&= \sum_{\substack{l=1 \\ l \neq i_c}}^{|\Re|} G_{i_c l}(c) \\
&\overset{(*)}{\le} \sum_{l=1}^{|\Re|} G_{i_c l}(c) \\
&= 1
\end{aligned}$$

[4] critère de Métropolis

(*) car $G_{il}(c)$ est une fonction de probabilité et elle est donc toujours ≥ 0.

Nous avons donc montré que :

$$\sum_{\substack{l=1 \\ l\neq i_c}}^{|\Re|} A_{il}(c) \cdot G_{il}(c) < 1 \qquad \textbf{(19)}$$

On obtient alors :

$$P_{i_c i_c} = 1 - \sum_{\substack{l=1 \\ l\neq i_c}}^{|\Re|} A_{il}(c) \cdot G_{il}(c) > 0 \quad \textbf{(20)}$$

ce qui est bien le même chose que l'équation (17).

Nous avons prouvé que, si les matrices A(c) et G(c) satisfont aux conditions (16) et (18), la distribution stationnaire d'une chaîne homogène de Markov de probabilité conditionnelle donnée par l'équation (4) existe.

5.3.1.3 Convergence de la distribution stationnaire

Nous imposerons des conditions supplémentaires sur les matrices A(c) et G(c) pour assurer la convergence de q(c) vers la distribution π donnée par l'équation (11). Plusieurs auteurs ont proposé de telles conditions, mais les moins restrictives ont été donné par Romeo et Sangiovanni - Vincentelli. La dérivation des conditions trouvées est basée sur le fait que pour une configuration arbitraire $i \in \Re$, la composante correspondante du vecteur de la distribution stationnaire peut être écrite sous la forme :

$$q_i(c) = \frac{\psi(C(i),c)}{\sum_{j\in\Re}\psi(C(j),c)} \qquad \textbf{(21)}$$

où $\psi(\gamma,c)$ est une fonction à deux arguments qui satisfait aux conditions suivantes :

1. $\forall\, i \in \Re, \forall\, c > 0 : \psi(C(i),c)>0$ **(22)**
2. **(équations de balance)**

$$\forall\, j \in \Re : \sum_{\substack{i=1 \\ i\neq j}}^{\Re} \psi(C(i),c) \cdot G_{ij}(c) \cdot A_{ij}(c) = \psi(C(i),c) \cdot \sum_{\substack{i=1 \\ i\neq j}}^{\Re} G_{ij}(c) \cdot A_{ij}(c) \qquad \textbf{(23)}$$

Montrons que pour toute fonction $\psi(.,.)$ vérifiant les conditions (22) et (23), les $q_i(c)$ définis par l'équation (21) vérifient bien les équations (14) et (15) et sont par conséquent les composantes de la distribution stationnaire.

Preuve :

a) Montrons d'abord que $\forall i \in \Re : q_i(c)>0 \sum_i q_i(c) = 1$.

$$\sum_i q_i(c) = \sum_i \frac{\psi(C(i),c)}{\sum_j \psi(C(j),c)} = \frac{\sum_i \psi(C(i),c)}{\sum_j \psi(C(j),c)} = 1$$

b) Montrons ensuite que $\forall i \in \Re : \; q_i(c) = \sum_j q_j(c) P_{ji}(c)$.

$$\sum_j q_j(c) P_{ji}(c)$$

$$\overset{(17)}{=} \sum_j \left(\frac{\psi(C(j),c)}{\sum_l \psi(C(l),c)} P_{ji}(c) \right)$$

$$\overset{(4)}{=} \frac{1}{\sum_l \psi(C(l),c)} \left\{ \left[\sum_{j \atop j \neq i} \psi(C(j),c) G_{ji}(c) A_{ji}(c) \right] + \left[\psi(C(i),c) \left(1 - \sum_{k \atop k \neq i} G_{ik}(c) A_{ik}(c) \right) \right] \right\}$$

$$\overset{(23)}{=} \frac{1}{\sum_l \psi(C(l),c)} \left\{ \left[\psi(C(i),c) \sum_{j \atop j \neq i} G_{ij}(c) A_{ij}(c) \right] + \psi(C(i),c) - \left[\psi(C(i),c) \sum_{k \atop k \neq i} G_{ik}(c) A_{ik}(c) \right] \right\}$$

$$= \frac{\psi(C(i),c)}{\sum_l \psi(C(l),c)} \overset{(21)}{=} q_i(c)$$

Pour assurer que $\lim_{c \xrightarrow{c>0} 0} q(c) = \pi$ (convergence de la distribution stationnaire), les trois conditions suivantes sur la fonction $\psi(\gamma,c)$ sont suffisantes :

1. $\lim_{c \xrightarrow{c>0} 0} \psi(\gamma,c) = \begin{cases} 0 \text{ si } \gamma > 0 \\ \infty \text{ si } \gamma < 0 \end{cases}$ **(24)**

2. $\frac{\psi(\gamma_1,c)}{\psi(\gamma_2,c)} = \psi(\gamma_1 - \gamma_2,c)$ **(25)**

3. $\forall c > 0 : \psi(0,c) = 1$ **(26)**

Montrons que si on a les conditions (24), (25) et (26) la distribution stationnaire q(c) tend vers π quand c tend en décroissant vers 0 i.e. $\lim_{c \xrightarrow{c>0} 0} q(c) = \pi$.

Preuve :

$$\lim_{c \xrightarrow{c>0} 0} q_i(c)$$

$$\overset{(21)}{=} \lim_{c \xrightarrow{c>0} 0} \frac{\psi(C(i),c)}{\sum_j \psi(C(j),c)}$$

$$= \lim_{c \xrightarrow{c>0} 0} \frac{1}{\sum_j \frac{\psi(C(j),c)}{\psi(C(i),c)}}$$

$$\overset{(25)}{=} \lim_{c \xrightarrow{c>0} 0} \frac{1}{\sum_j \psi(C(j)-C(i),c)}$$

- **Si** $i \in \Re_{opt}$

  $\Rightarrow$ C(i) est un minimum global de la fonction C sur $\Re$ et par conséquent $\begin{cases} C(j)-C(i) > 0, \ \forall i \in \Re / \Re_{opt} \\ C(j)-C(i) = 0, \ \forall i \in \Re_{opt} \end{cases}$ (**).

  Par (24) et (**) on a :

$$\lim_{c \xrightarrow{c>0} 0} q_i(c) = \lim_{c \xrightarrow{c>0} 0} \frac{1}{\sum_{\substack{j \\ j \neq i}} \psi(C(j)-C(i),c)} \overset{(**) \text{ et } (24)}{=} \frac{1}{(\|\Re_{opt}\|)+0} = \frac{1}{\|\Re_{opt}\|}$$

- **Si** $i \notin \Re_{opt}$

  $\Rightarrow$ C(i) n'est pas un minimum global de la fonction C sur $\Re$

  $\Rightarrow \exists\, k_1, k_2, \ldots, k_n \neq i$ t.q. $C(k_1)-C(i) < 0,\ C(k_2)-C(i) < 0,\ \ldots,\ C(k_n)-C(i) < 0$ (***)

  $\Rightarrow$

$$\lim_{c \xrightarrow{c>0} 0} q_i(c)$$

$$= \lim_{c \xrightarrow{c>0} 0} \frac{1}{\psi(0,c) + \sum_{\substack{j \\ j \neq i}} \psi(C(j)-C(i),c)}$$

$$\overset{(26)}{=} \lim_{c \xrightarrow{c>0} 0} \frac{1}{1 + \sum_{\substack{j \\ j \neq i}} \psi(C(j)-C(i),c)}$$

$$= \frac{1}{1+\psi(C(k_1)-C(i),c)+\ldots+\psi(C(k_n)-C(i),c) + \sum_{\substack{j \\ j\neq i \\ j\neq k_1 \\ \ldots \\ j\neq k_n}} \psi(C(j)-C(i),c)}$$

$$\overset{\substack{(***) \\ \text{et} \\ (24)}}{=} \frac{1}{1+\infty+\ldots+\infty+0+\ldots+0} = \frac{1}{\infty} = 0$$

Nous avons montré que $\lim_{c \xrightarrow{c>0} 0} q(c) = \pi$, où $\pi_i = \begin{cases} \dfrac{1}{\|\Re_{opt}\|} & \text{si } i \in \Re_{opt} \\ 0 & \text{Partout ailleurs} \end{cases}$

Remarquons que le théorème montre les conditions suffisantes pour la convergence mais pas les conditions nécessaires, ce qui veut dire que nous pouvons trouver une fonction $\psi(.,.)$ qui ne vérifie pas les conditions (22), (23), (24), (25), et (26), mais telle que q(c) converge tout de même vers la distribution π.

5.3.2 Remarque

Dans beaucoup de problèmes traités par le recuit simulé, la matrice G(c) ne dépend pas de c, i.e. la probabilité de générer une configuration j à partir d'une configuration i est indépendante de la température. Dans ce cas on peut avoir une forme plus explicite de la fonction à deux variables $\psi(\gamma,c)$. C'est l'objet traité par le théorème suivant :

Théorème 2 (Folklore)

Si $A_{i_0 i}(c)$ (où $i_0 \in \Re_{opt}$ est une configuration arbitraire) est prise comme fonction $\psi(C(i),c)$ et si G(c) ne dépend pas de c alors la distribution stationnaire q(c) est donnée par :

$$\forall i \in \Re : q_i(c) = \frac{A_{i_0 i}(c)}{\sum_{j \in \Re} A_{i_0 j}(c)} \quad \textbf{(27)}$$

si les matrices A(c) et G satisfont aux conditions suivantes :

(a1) $\forall i, j \in \Re : G_{ij} = G_{ji}$ (matrice symétrique)

(a2) $\forall i, j, k \in \Re : C(i) \le C(j) \le C(k) \Rightarrow A_{ik}(c) = A_{ij}(c)A_{jk}(c)$

(a3) $\forall i, j \in \Re, c > 0 : C(i) \ge C(j) \Rightarrow A_{ij}(c) = 1$

(a4) $\forall i, j \in \Re, c > 0 : C(i) < C(j) \Rightarrow 0 < A_{ij}(c) < 1$

(a5) $\forall i, j \in \Re : C(i) < C(j) \Rightarrow \lim_{c \xrightarrow{c>0} 0} A_{ij}(c) = 0$

Preuve :

Montrons d'abord que q(c) est une fonction de distribution stationnaire. Autrement dit, il faut montrer que q(c) est une fonction de distribution :

$$\forall i \in \Re, q_i(c) > 0 : \sum_{i \in \Re} q_i(c) = 1$$

et ensuite que cette fonction de distribution définit une distribution stationnaire :

$$\forall i \in \Re : \sum_{j \in \Re} q_j(c) P_{ji}(c) = q_i(c)$$

Montrons que q(c) est une fonction de distribution :

$$\forall i \in \Re, q_i(c) > 0 : \sum_{i \in \Re} q_i(c) = 1 \quad \textbf{(28)}$$

Preuve :

$\forall i \in \Re, q_i(c) > 0$:

$$\sum_{i\in\Re} q_i(c) \overset{(27)}{=} \sum_{i\in\Re} \frac{A_{ioi}(c)}{\sum_{j\in\Re} A_{ioj}(c)}$$

$$= \frac{\sum_{i\in\Re} A_{ioi}(c)}{\sum_{j\in\Re} A_{ioj}(c)}$$

$$= 1$$

Montrons que cette fonction de distribution définit une distribution stationnaire:

$$\forall i \in \Re: \sum_{j\in\Re} q_j(c) P_{ji}(c) = q_i(c)$$

Preuve :

$$\sum_{j\in\Re} q_j(c) P_{ji}(c) = q_i(c) P_{ii}(c) + \sum_{\substack{j\in\Re \\ j\neq i}} q_j(c) P_{ji}(c)$$

$$\overset{(4)}{=} q_i(c) P_{ii}(c) + \sum_{\substack{j\in\Re \\ j\neq i}} q_j(c) G_{ji} A_{ji}(c)$$

$$\overset{(27)}{=} q_i(c) P_{ii}(c) + \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} \frac{1}{N} A_{ioj}(c) G_{ji} A_{ji}(c) + \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)> C(i)}} \frac{1}{N} A_{ioj}(c) G_{ji} A_{ji}(c)$$

$$\overset{\substack{(a1) \text{ et } (a2) \\ \text{et} \\ (a3)}}{=} q_i(c) P_{ii}(c) + \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} \underbrace{\frac{1}{N} A_{ioi}(c)}_{(27)\ q_i(c)} G_{ji} + \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)> C(i)}} \underbrace{\frac{1}{N} A_{ioj}(c)}_{(27)\ q_j(c)} G_{ij}$$

$$= q_i(c) P_{ii}(c) + q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij} + \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)> C(i)}} q_j(c) G_{ij}$$

où $N = \sum_{j\in\Re} A_{ioj}(c)$ (le dénominateur de l'équation 27)

Nous avons prouvé que :

$$\sum_{j\in\Re} q_j(c) P_{ji}(c) = q_i(c) P_{ii}(c) + q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij} + \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)> C(i)}} q_j(c) G_{ij} \quad \textbf{(29)}$$

Calculons $q_i(c)P_{ii}(c)$

$$q_i(c)P_{ii}(c) \overset{(4)}{=} q_i(c)\left(1 - \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij}\underbrace{A_{ij}(c)}_{=1\ (a3)} - \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)>C(i)}} G_{ij}A_{ij}(c)\right)$$

$$\overset{(a3)}{=} q_i(c) - q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij} - q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)>C(i)}} G_{ij}A_{ij}(c)$$

$$\overset{(27)}{=} q_i(c) - q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij} - \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)>C(i)}} \frac{1}{N} A_{i0i}(c)G_{ij}A_{ij}(c)$$

$$\overset{(a2)}{=} q_i(c) - q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij} - \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)>C(i)}} q_j(c)G_{ij}$$

où $N = \sum_{j\in\Re} A_{i0j}(c)$ (le dénominateur de l'équation 27)

Nous avons prouvé que :

$$q_i(c)P_{ii}(c) = q_i(c) - q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij} - \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)>C(i)}} q_j(c)G_{ij} \qquad \textbf{(30)}$$

En utilisant ce résultat montrons que :

$$\forall i \in \Re: \sum_{j\in\Re} q_j(c)P_{ji}(c) = q_i(c) \qquad \textbf{(31)}$$

$$\sum_{j\in\Re} q_j(c)P_{ji}(c) \overset{(29)}{=} q_i(c)P_{ii}(c) + q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij} + \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)>C(i)}} q_j(c)G_{ij}$$

$$\overset{(30)}{=} q_i(c) - q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij} - \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)>C(i)}} q_j(c)G_{ij} + q_i(c) \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)\leq C(i)}} G_{ij} + \sum_{\substack{j\in\Re \\ j\neq i \\ C(j)>C(i)}} q_j(c)G_{ij}$$

$$= q_i(c)$$

Par les équations (28), (31) et le théorème 1 nous pouvons déduire que la fonction q(c) comme définie par l'équation (27) est bien une fonction de distribution stationnaire.

Pour finir avec la démonstration du théorème 2, il reste à montrer que :

$$\lim_{c \xrightarrow{c>0} 0} q_i(c) = \pi_i ,$$

$$\text{où } \pi = \begin{cases} 1/\|\Re_{opt}\| & \text{si } i \in \Re_{opt} \\ 0 & \text{sinon} \end{cases}$$

Preuve :

Par (a5) on a que : $\forall i, j \in \Re : C(i) < C(j) \Rightarrow \lim_{c \xrightarrow{c>0} 0} A_{ij}(c) = 0$.

Distinguons deux cas :

Si $i \in \Re_{opt}$

soit $k \in \Re_{opt}$

$$\lim_{c \xrightarrow{c>0} 0} q_i(c) =$$

$$= \lim_{c \xrightarrow{c>0} 0} \left[\frac{A_{i0i}(c)}{\sum_{j \in \Re} A_{i0j}(c)} \cdot \frac{A_{ki0}(c)}{A_{ki0}(c)} \right]$$

$$= \lim_{c \xrightarrow{c>0} 0} \frac{A_{ki}(c)}{\sum_{j \in \Re} A_{kj}(c)} \qquad \text{par(a2) car} \begin{cases} C(k) \le C(i0) \le C(i) \\ C(k) \le C(i0) \le C(j) \\ \text{vu que } k \in \Re opt \text{ et } i \in \Re opt \end{cases}$$

$$= \frac{\lim_{c \xrightarrow{c>0} 0} A_{ki}(c)}{\lim_{c \xrightarrow{c>0} 0} \sum_{j \in \Re} A_{kj}(c)}$$

$$= \frac{1}{\lim_{c \xrightarrow{c>0} 0} \sum_{j \in \Re} A_{kj}(c)} \qquad \text{par (a5) car } i \notin \Re opt \Rightarrow C(k) > C(i)$$

$$= \frac{1}{\lim_{c \xrightarrow{c>0} 0} \left(A_{kk}(c) + \sum_{\substack{j \in \Re \\ j \neq k}} A_{kj}(c) \right)}$$

$$= \frac{1}{1 + \lim_{c \xrightarrow{c>0} 0} \sum_{\substack{j \in \Re \\ j \neq k}} A_{kj}(c)} \qquad \text{par (a3) car } C(k) \ge C(k)$$

$$= \frac{1}{1 + (\|\Re_{opt}\| - 1)}$$

Si $i \notin \mathfrak{R}_{opt}$

$$\lim_{c \xrightarrow{c>0} 0} q_i(c) = \lim_{c \xrightarrow{c>0} 0} \left(\frac{A_{i0i}(c)}{\sum_{j \in \mathfrak{R}} A_{i0j}(c)} \right)$$

$$= \lim_{c \xrightarrow{c>0} 0} \left[\frac{A_{i0i}(c)}{\sum_{j \in \mathfrak{R}} A_{i0j}(c)} \frac{A_{ki0}(c)}{A_{ki0}(c)} \right]$$

$$= \lim_{c \xrightarrow{c>0} 0} \left(\frac{A_{ki}(c)}{\sum_{j \in \mathfrak{R}} A_{kj}(c)} \right) \quad \text{car} \begin{cases} C(k) \le C(i0) < C(i) \\ C(k) \le C(i0) \le C(j) \\ \text{vu que } k \in \mathfrak{R}opt \text{ et } i \notin \mathfrak{R}opt \end{cases}$$

$$= \frac{\lim_{c \xrightarrow{c>0} 0} A_{ki}(c)}{\lim_{c \xrightarrow{c>0} 0} \left(\sum_{j \in \mathfrak{R}} A_{kj}(c) \right)}$$

$$= \frac{0}{\lim_{c \xrightarrow{c>0} 0} \left(\sum_{j \in \mathfrak{R}} A_{kj}(c) \right)} \quad \text{car (a5) et } \left\{ i \notin \mathfrak{R}opt \Rightarrow \left(C(k) \le C(i) \right) \right\}$$

$$= \frac{0}{1 + \lim_{c \xrightarrow{c>0} 0} \left(\sum_{\substack{j \in \mathfrak{R} \\ j \neq k}} A_{kj}(c) \right)} \quad \text{car (a3) pour } C(k) \le C(k)$$

$$= \frac{0}{1 + \left(\|\mathfrak{R}_{opt}\| - 1 \right)}$$

$$= 0$$

Nous avons prouvé que :

$$\lim_{c \xrightarrow{c>0} 0} q(c) = \pi ,$$

$$\text{où } \pi = \begin{cases} 1 / \|\mathfrak{R}_{opt}\| & \text{si } i \in \mathfrak{R}_{opt} \\ 0 & \text{sinon} \end{cases}$$

5.3.2.1 Résumé

Dans ce chapitre nous avons montré que sous certaines conditions sur les matrices A(c) et G(c), le recuit simulé converge vers un minimum global avec une probabilité 1 pour chaque valeur c_l du paramètre de contrôle (l=0,1,2,...). La chaîne de Markov correspondante est de longueur infinie. Dans le cas où $\lim_{l \longrightarrow \infty} c_l = 0$, nous avons montré que :

$$\lim_{c \xrightarrow{c>0} 0} \left(\lim_{k \longrightarrow \infty} \Pr\{X(k) = i\} \right) = \lim_{c \xrightarrow{c>0} 0} q_i(c) = \begin{cases} 1/\|\Re_{opt}\| & \text{si } i \in \Re_{opt} \\ 0 & \text{si } i \notin \Re_{opt} \end{cases} \qquad \textbf{(32)}$$

5.3.3 L'algorithme inhomogène

Les résultats que nous avons présentés jusqu'ici s'appuient sur l'hypothèse d'une décroissance de la température par **paliers** (qui assure une convergence rapide de l'algorithme du recuit simulé, comme nous l'avons déjà décrit plus haut). Certains auteurs[5] se sont intéressés à la convergence de l'algorithme du recuit simulé en se plaçant dans le cadre plus général de la théorie des chaînes de Markov **inhomogènes**. Dans ce cas le comportement asymptotique est plus délicat à étudier. Nous nous contenterons ici d'évoquer le principal résultat de ces travaux d'intérêt essentiellement théorique : "*l'algorithme du recuit converge vers un optimum global, avec une probabilité égale à l'unité si, lorsque le temps t tend vers l'infini, la température T(t) ne décroît pas plus vite que l'expression C/ln(t), en désignant par C une constante qui est liée à la profondeur des 'puits d'énergie' du problème*".

[5] P.Siarry, G. Dreyfus, J. Physique Lett **45** (1984) L-39

J.G. Gay, R. Richter, B.J. Berne, Integration, the VLSI journal **3** (1985) 271

P. Siarry, La méthode du recuit simulé : application à la conception de circuits électroniques. Thèse, Université Pierre et Marie Curie (1986)

6. Analogie entre le recuit simulé et la physique statistique

En physique statistique, on étudie les systèmes formés d'un très grand nombre de particules en interaction. Comme le nombre de ces particules se compte en milliards, la physique statistique ne considère que les propriétés de petits échantillons de quelques centaines de particules. Elle étudie l'évolution de ces échantillons par des simulations. De plus, comme on ne peut étudier en continu le mouvement de chaque particule, on détermine un éventail représentatif de toutes les configurations microscopiques du système à partir duquel on calcule les propriétés macroscopiques.

La relation entre la physique statistique et l'optimisation de problèmes combinatoires peut être exprimée comme suit.

Etant donné un système physique en équilibre thermique dont les états d'énergie interne sont distribués suivant l'expression:

$$\Pr[E = E] = \frac{1}{Z(T)} \cdot \exp\left(-\frac{E_i}{k_B \cdot T}\right) \qquad \textbf{(33)}$$

où T est la température du système,

k_B est la constante de Boltzmann

et la fonction de partition Z(T) est définie par $Z(T) = \sum_i \exp\left(-\frac{E_i}{k_B \cdot T}\right)$,

La somme sur i représente la somme sur tous les états macroscopiques possibles.

et un problème d'optimisation combinatoire dont les configurations sont données par l'équation (3) (qui est d'ailleurs identique à l'équation (33)), on peut définir un ensemble de quantités macroscopiques pour les problèmes d'optimisation par analogie aux quantités définies pour le système physique.

Dans ce document, on va se contenter de donner la définition de l'entropie. Pour une étude plus approfondie, nous renvoyons à [AERTS, E., and VAN LAARHOVEN, P., Simulated Annealing, Theorie and applications]

L'**entropie** à l'équilibre d'un système physique est donnée par l'équation:

$$S(c) = -\sum_{i \in R} q_i(c) \cdot \ln(q_i(c))$$

En physique statistique, l'entropie peut être interprétée comme une mesure naturelle de l'ordre dans ce système. Une grande valeur de l'entropie correspond au chaos (désordre complet) et une petite valeur à l'ordre. La **troisième loi de thermodynamique** qui est exprimée par l'équation :

$$\lim_{c \xrightarrow{c>0} 0} S(c) = 0$$

exprime le fait que **l'entropie d'un système physique dont la température décroît vers 0, tend également vers 0**, i.e. le système a tendance à s'ordonner quand la température diminue (suffisamment lentement).

Dans le cas du recuit simulé (et plus généralement de l'optimisation combinatoire), l'entropie peut être interprétée comme une mesure quantitative du degré d'optimalité. Au départ, dans un problème d'optimisation, la configuration est chaotique et correspond par conséquent à une valeur d'entropie très élevée. Pendant le processus d'optimisation, pour les configurations observées, la valeur de la fonction de coût tend de plus en plus vers le minimum. Les configurations deviennent donc moins chaotiques et l'entropie du système décroît. A l'optimum, l'entropie atteint également sa valeur minimale.

Tout comme en physique statistique, le recuit simulé n'explore donc que certaines configurations représentatives du problème. Or, quand on baisse la température à laquelle on simule le système, on explore à l'aide de l'algorithme de Metropolis que les configurations microscopiques voisines de celles de l'énergie minimale, conformément aux lois de la thermodynamique.

Observons ce qui se passerait lors de l'exécution de l'algorithme de recuit simulé pour trouver la solution au problème du plus court chemin passant une et une seule fois par tous les sommets du graphe ci-dessous (problème du chemin hamiltonien minimu).

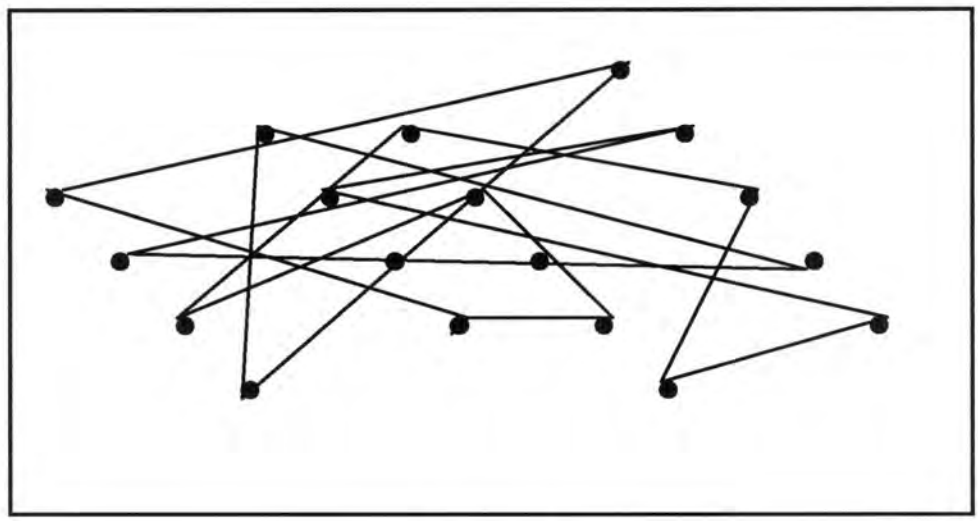


Schéma 1

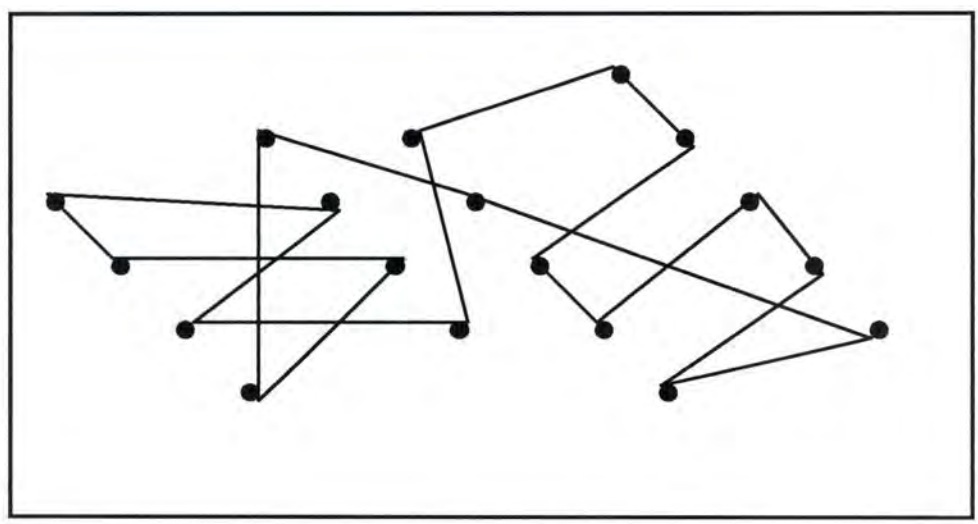


Schéma 2

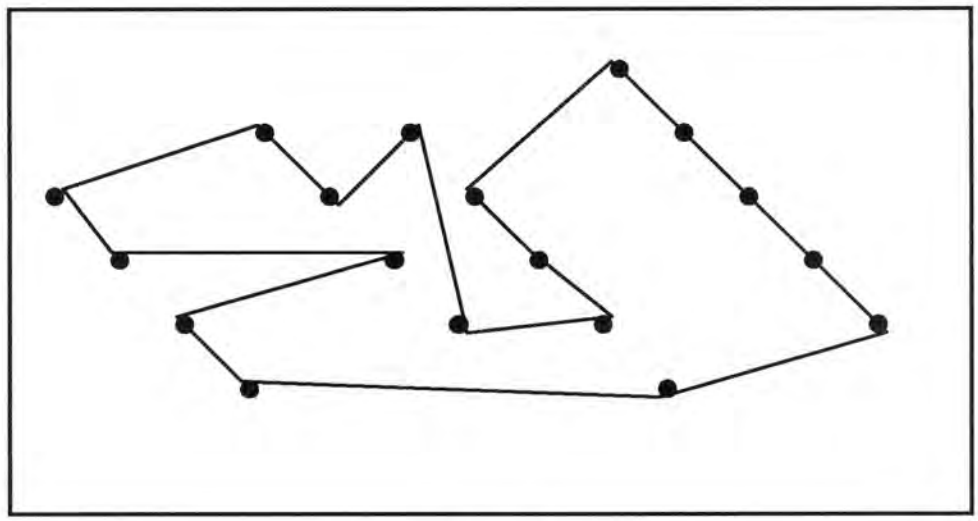


Schéma 3

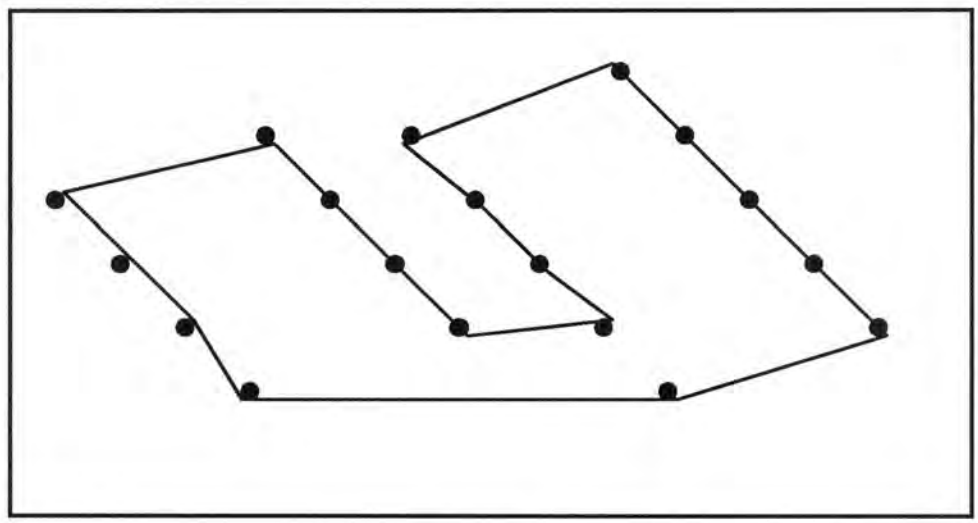


Schéma 4

On peut remarquer sur ces quatre schémas que l'on passe progressivement d'une configuration initiale (schéma 1) complètement désordonnée d'entropie maximale, à une configuration finale (schéma 4) totalement ordonnée, d'entropie minimale.

7. The Cooling Schedule

Nous avons vu que la convergence de l'*algorithme* du recuit simulé est assurée lorsque la température tend vers zéro et lorsqu'une chaîne de Markov de longueur *infinie* est générée pour chaque palier de température.

Une fois confrontés au problème d'implémentation du recuit simulé, pour laquelle une suite de chaînes homogènes de Markov de longueurs *finies* est générée pour les valeurs décroissantes du paramètre de contrôle, les paramètres suivants doivent être spécifiés:

- la **valeur initiale** du paramètre de contrôle c_0
- la **valeur finale** du paramètre de contrôle c_f (condition d'arrêt)
- la **longueur des chaînes de Markov** (le nombre d'itérations pendant lesquelles le paramètre de contrôle est gardé constant, autrement dit le nombre d'itérations pour chaque puit d'énergie)
- la **règle** utilisée pour la diminution du paramètre de contrôle c_k à sa valeur suivante c_{k+1}: $c_k > c_{k+1}$

Dans la littérature, la politique suivie pour la spécification de ces paramètres s'appelle "**cooling schedule**" (politique de refroidissement).

Une notion centrale pour la construction d'une politique de refroidissement est la notion de **quasi équilibre**.

Définition : Si L_k est la longueur de la $k^{ème}$ chaîne de Markov, alors l'algorithme du recuit simulé se trouve en **quasi équilibre** en c_k si $a(L_k,c_k)$ est près de $q(c_k)$,

où

- c_k est la $k^{ème}$ valeur du paramètre de contrôle
- $a(L_k,c_k)$ est la distribution de la probabilité en c_k, après L_k transitions
- $q(c_k)$ est la distribution stationnaire de la chaîne homogène de Markov pour la valeur c_k du paramètre de contrôle.

Autrement dit, l'algorithme se trouve en quasi équilibre si $\|a(L_k,c_k) - q(c_k)\| < \varepsilon$, pour un $\varepsilon > 0$ arbitraire.

7.1 Valeur initiale du paramètre de contrôle (c_0)

La valeur initiale de c doit être déterminée de telle manière que presque toutes les transitions soient acceptées,

i.e. c_0: $\exp\left(-\frac{\Delta C_{ij}}{c_0}\right) \cong 1$ pour presque tout $i \in \Re$, $j \in \Re_i$, où $\Re_i$ est le voisinage de i.

7.2 Valeur finale du paramètre de contrôle (c_f)

Pour définir la condition d'arrêt, il suffit de déterminer la valeur de c_f. Une manière de le faire consiste à choisir un nombre suffisamment près de 0 et de le donner comme valeur à c_f. Quand la valeur de c_k devient plus petite que celle de c_f, alors on arrête l'algorithme.

Une autre politique consiste à arrêter l'algorithme si, pendant un certain nombre d'itérations, la valeur de la fonction de coût reste constante. c_f prend la valeur d'un c_k pour lequel la fonction de coût $C(i_k)$ est telle que

$$C(i_{k-l}) = C(i_{k-l+1}) = \ldots = C(i_k), \qquad \text{où l est à définir.}$$

7.3 La longueur des chaînes de Markov (L_k)

La longueur des chaînes de Markov représente le nombre d'itérations exécutées avant que la valeur du paramètre de contrôle soit diminuée.

La manière la plus simple de choisir la valeur de L_k consiste à choisir une valeur qui dépend polynômialement de la taille du problème. L_k est donc indépendant de k et est par conséquent une constante.

Une autre manière de définir L_k est la suivante: pour chaque valeur de c_k, la valeur de L_k est telle qu'un nombre n_{min} de transitions a été accepté (où n_{min} est une constante). Cette politique pose un problème quand c_k est suffisamment proche de c_f, car la probabilité d'acceptation d'une transition est petite; pour une configuration suffisamment proche de l'optimum, on a $L_k \rightarrow \infty$. Pour cette raison L_k doit être bornée par une constante $\overline{L}$ (dépendant de la taille du problème) pour empêcher d'avoir des chaînes de Markov très longues pour les petites valeurs de c_k.

Il ne faut pas oublier que la politique suivie pour le choix de la valeur de L_k doit être telle que le quasi équilibre soit suffisamment approché.

7.4 Diminution du paramètre de contrôle

La diminution du paramètre de contrôle doit être telle que des chaînes de Markov suffisamment courtes suffisent pour établir le quasi équilibre après la diminution de la valeur de c_k.

La règle la plus fréquente est la suivante: $c_{k+1} = \alpha \, . \, c_k$,

où α est une constante strictement plus petite (mais suffisamment proche) que 1.

La valeur de α la plus fréquemment utilisée est $\alpha = 0{,}93$ ou $\alpha = 0.95$. Vu le choix arbitraire de α pour le problème d'optimisation, diverses valeurs de α doivent être testées pour $\alpha \in [0.5,0.99]$.

Chapitre Trois
Le plus court chemin dans un graphe

1. Présentation générale des algorithmes

1.1 Types de problèmes rencontrés

Afin de résoudre les problèmes des plus courts chemins dans un graphe, des dizaines d'algorithmes ont déjà été proposés et ont donné lieu à l'édition de plus d'une centaine d'articles. Certaines de ces méthodes sont meilleures que d'autres, chacune ayant sont adaptabilité propre à des types particuliers de structure. D'autre part, il y a peu de méthodes générales pour résoudre les problèmes de plus courts chemins et certains algorithmes ne sont que des variantes d'algorithmes antécédents.

Par contre, il existe beaucoup de problèmes relatifs aux plus courts chemins. Les plus fréquemment rencontrés sont les six suivants:

1. le plus court chemin entre deux noeuds
2. les plus courts chemins d'un noeud vers tous les autres
3. les plus courts chemins entre tous les noeuds
4. les plus courts chemins entre des noeuds spécifiés et passant par des noeuds fixés
5. le plus court chemin entre deux noeuds et ne comportant pas plus de k arcs, k étant fixé
6. les seconds, troisièmes, etc. plus courts chemins.

Notons que les problèmes du type 2 résolvent les problèmes du type 1; les problèmes du type 2 peuvent être résolus par applications successives d'algorithmes résolvant les problèmes du premier type; et enfin plusieurs applications d'un algorithme du type 2 fournit une solution au problèmes du type 3.

1.2 Notations et remarques

1.2.1 Notations

d_{ij}	désigne la longueur (ou le coût) de l'arc (i,j)
d^{*ab}	indique la longueur du plus court chemin de a vers b
N	sera l'ensemble des noeuds
L	désignera l'ensemble des arcs
n_N	est le nombre de noeuds
n_L	est le nombre d'arcs
G(N,L)	représente le graphe associé au réseau

La longueur des plus courts chemins est définie de manière **récursive**

$$d^{*aj} = \min (d^{*ai} + d_{ij})$$

$$d^{*aa} = 0$$

Cette définition suggère déjà des algorithmes de base pour calculer les plus courts chemins.

1.2.2 Remarques

- Si un arc (i,j) n'existe pas dans le graphe, alors son coût se verra attribuer la valeur ∞.
- Nous supposerons que les coûts des arcs ne satisfont pas aux conditions de triangularité c'est à dire que d_{ij} n'est pas inférieur à $d_{ik}+d_{kj}$ pour tout i, j et k ; sinon, le plus court chemin entre les noeuds i et j sera toujours l'arc (i,j) et le problème devient inexistant.
- Il sera supposé que $d_{ij} \neq d_{ji}$ pour tout i, j: le graphe est non nécessairement symétrique.

1.2.3 Remarques générales sur les algorithmes

Les algorithmes calculant les plus courts chemins entre beaucoup de noeuds se répartissent en deux catégories:

1. les algorithmes par **construction d'arbre** ("*treebuilding method*")
2. les algorithmes **matriciels** ("*matrix method*")

La première catégorie de méthodes est mieux adaptée d'abord aux plus courts chemins d'**un** noeud vers **tous** les autres (ou un sous-ensemble d'entre eux) et ensuite aux plus courts chemins de **k** noeuds vers tous les autres (K arbres devant alors être successivement construits). C'est d'ailleurs ce dernier cas qui nous intéresse.

Les méthodes du second groupe sont favorisées pour obtenir les plus courts chemins entre **tous** les noeuds **simultanément**.

La différence entre ces deux catégories réside au niveau de l'objet de résolution, mais il en est une autre, également fondamentale, concernant le temps d'exécution ainsi que le stockage que requièrent ces différent algorithmes.

2. Les algorithmes par construction d'arbre

Si nous prenons les plus courts chemins d'un noeud s vers chacun des autres noeuds (et ce pour tous ceux accessibles depuis s), alors l'union de ces chemins engendre un arbre dont la racine est s. Chaque chemin de cet arbre constitue, depuis s, le plus court chemin dans le graphe de départ. Un tel arbre est appelé **arbre des plus courts chemins** ou encore **arbre minimum**.

On s'aperçoit vite que l'arbre est entièrement déterminé par la donnée des noeuds avec pour chacun d'eux son prédécesseur dans le plus court chemin menant à lui. Il suffit donc de disposer du précédent de chaque noeud, ce qui fournit une représentation de l'arbre indépendamment du graphe. La littérature identifie ces prédécesseurs par le terme ***"backnodes"***.

2.1 Classification

Le problème de recherche des plus courts chemins (depuis le noeud s) revient donc à trouver l'arbre minimum de racine s.

Les "treebuilding methods" sont en fait des "labeling methods" c'est-à-dire des algorithmes travaillant sur des "labels" (étiquettes) associés à chaque noeud.

Ces méthodes sont réparties en deux classes générales:

- les "**label-setting** methods"
- les "**label-correcting** methods"

Une méthode du premier type part d'un arbre vide et augmente, un par un, le nombre de noeuds et d'arcs à chaque itération. Si x est un noeud entré dans l'arbre, l'unique arc de s à x est le chemin le plus court chemin de s à x. Ainsi, à chaque itération un plus court chemin est trouvé.

Par contre, les méthodes du second type échangent, augmentent et mettent à jour les arcs de l'arbre de manière à replacer ou raccourcir l'unique chemin de la racine à un noeud quelconque de l'arbre. Ces méthodes ne garantissent pas que le nouveau chemin est le plus court (jusqu'à ce que le processus soit terminé).

2.2 Algorithme d'une passe ("*oncetrough*")

La première publication de cet algorithme est due à Moore-Dijkstra (1959) , il s'agit d'un algorithme qui a été très utilisé pour résoudre les problèmes de transport, et qui est de type "*label-correcting*".

Contrairement à d'autre algorithmes, celui-ci n'est valable que pour la résolution de **problèmes à coûts non négatifs**, et c'est le cas du problème que nous cherchons à résoudre.

Donc, $d_{ij} \geq 0 \; \forall \; i,j \in N$.

2.2.1 Description de l'algorithme

La méthode est basée sur une association de chaque noeud avec un label tantôt **temporaire**, tantôt **permanent**; temporaire dans le sens où sa valeur **peut être changée**, et permanent dans le sens où sa valeur est dès lors **définitive**.

Au départ, ces labels ont une valeur temporaire arbitrairement grande. Ils sont au fur et à mesure réduits par une procédure itérative et, à chaque itération, exactement un des labels temporaires devient permanent et indique, à partir de ce moment, la longueur du plus court chemin du noeud source jusqu'au noeud considéré.

Plus formellement, nous avons:

l(i) désigne le label du noeud i

s est le noeud à partir duquel les plus courts chemins sont calculés

p est le noeud rendu permanent à chaque itération

1. Initialisation

$l(s) \leftarrow 0$

$l(i) \leftarrow \infty \quad \forall \; i \neq s$

$p \leftarrow s$

2. Mise à jour des labels

Pour tout noeud i de l'adjacence de p, ayant un label temporaire, faire

$l(i) \leftarrow \min\{l(i), l(p)+d_{pj}\}$

3. Trouver le label à rendre permanent

trouver un noeud x* ayant un label temporaire tel que $l(x^*) = \min\{l(i)\}$

4. Fixer un label comme permanent

$p \leftarrow x^*$

5. Test de terminaison

(i) si seulement le chemin de s jusque a est désiré

si p = a, l(p) est la longueur du plus court chemin recherché

sinon aller en 2

(ii) si les plus courts chemins de s vers tous les autres noeuds sont voulus

si tous les labels sont permanents, alors les coûts des plus courts chemins sont les valeurs des labels

sinon aller en 2

2.2.2 Remarque

S'il y a plusieurs candidats pour le pas 3, n'importe lequel peut être choisi pour devenir permanent.

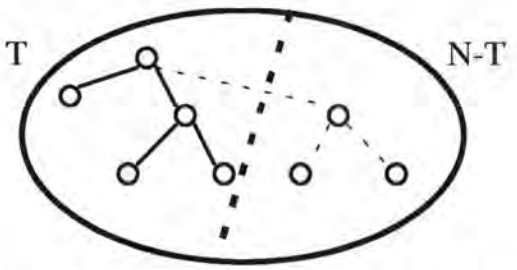
2.2.3 Interprétation

Par cette manière de procéder, le premier noeud à acquérir un label permanent est à une distance nulle de s. Le second (parmi les n_N-1 noeuds restants) est le plus proche de s. Des n_N-2 noeuds demeurant temporaires, celui à devenir permanent est le second noeud le plus proche de s.

En fait, comme nous l'avons précisé au départ, cet algorithme revient à considérer deux ensembles de noeuds: les noeuds faisant partie de l'arbre minimum (les noeuds marqués définitivement) et ceux qui n'en font pas encore partie (les noeuds marqués temporairement), c'est-à-dire, respectivement, T et N-T. Chaque itération consiste à considérer les arcs faisant passer de T à N-T i.e.

$$\{(u,v) \mid u \in T, v \in N\text{-}T, (u,v) \in L\}$$

et de prendre l'arc réalisant le chemin de coût minimum depuis la racine.



Théorème : L'algorithme de Moore-Dijkstra produit bien les plus courts chemins.

Démonstration:

Supposons qu'à une étape quelconque de l'algorithme, les valeurs des labels permanents soient les coûts des plus courts chemins.

Soit T l'ensemble des noeuds marqués définitivement.

Soit U l'ensemble des noeuds ayant un label provisoire.

A la fin du pas 2 de chaque itération, le label temporaire l(i) est le plus court chemin de s à i passant entièrement par des noeuds de l'ensemble T (puisque seulement un noeud s'ajoute à T à chaque itération, la mise à jour de l(i) nécessite seulement la seule comparaison du pas 2).

Supposons que le plus court chemin de s à x* ne passe pas entièrement par T, mais contient au moins un noeud de U.

Soit $j \in U$, le premier noeud de ce genre sur ce chemin et Δ, le coût du morceau de chemin de j à x*. On a que

$$l(j) + \Delta < l(x^*)$$

Les d_{ij} étant non négatifs, Δ doit avoir une valeur non négative tel que

$$l(x^*) - \Delta < l(x^*)$$

En mettant ces deux équations ensembles on obtient

$$l(j) < l(x^*) - \Delta < l(x^*)$$

ou encore

$$l(j) < l(x^*)$$

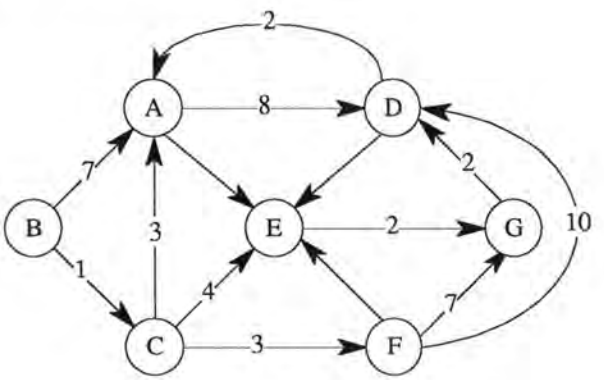
Ceci contredit toutefois l'affirmation que l(x*) est le plus petit label temporaire. Ainsi, le plus court chemin jusque x* n'est possible que par des noeuds de T et l(x*) est alors son coût.

Puisque T est initialement {s} et qu'à chaque itération, un x* s'ajoute à T, la supposition que les longueurs des plus courts chemins, pour tout i de T, est valide à

chaque itération et donc, par induction, la réponse produite en fin d'algorithme est optimale.

2.2.4 Illustration de l'algorithme de Dijkstra

Considérons le réseau suivant:



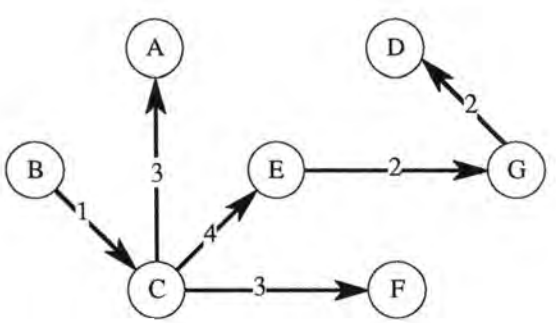
et calculons le plus court chemin à partir du noeud B.

Pour cela, examinons les valeurs successivement prises par les labels. Dans le tableau qui suit, les noeuds à label permanent sont soulignés et les noeuds actifs sont indicés par une étoile.

		A	B	C	D	E	F	G
		∞	0*	∞	∞	∞	∞	∞
Itération 1 :	pas 2	7	0*	1	∞	∞	∞	∞
	pas 3 et 4	7	0	1*	∞	∞	∞	∞
Itération 2 :	pas 2	4	0	1*	∞	5	4	∞
	pas 3 et 4	4	0	1	∞	5	4*	∞
Itération 3 :	pas 2	4	0	1	14	5	4*	11
	pas 3 et 4	4*	0	1	14	5	4	11
Itération 4 :	pas 2	4*	0	1	12	5	4	11
	pas 3 et 4	4	0	1	12	5*	4	11
Itération 5 :	pas 2	4	0	1	12	5*	4	7
	pas 3 et 4	4	0	1	12	5	4	7*
Itération 6 :	pas 2	4	0	1	9	5	4	7*
	pas 3 et 4	4	0	1	9*	5	4	7

et les chemins sont donnés par

Noeuds	A	B	C	D	E	F	G
d^B	4	0	1	9	5	4	7



Tous les pas sont facilement programmables à l'exception du travail consistant à distinguer les noeuds à labels permanents de ceux ayant un label temporaire : c'est une implémentation plus délicate sur laquelle repose l'efficience plus ou moins grande de l'algorithme. Une des premières méthodes venant à l'esprit est d'associer les noeuds aux indices 1, 2, ..., n_N et de conserver un vecteur binaire de longueur n_N; lorsque le $i^{ème}$ noeud devient permanent, le $i^{ème}$ élément de ce vecteur change de 0 à 1.

L'algorithme ainsi décrit ne donne pas encore le plus court chemin du noeud s vers quelqu' autre noeud ; il ne fournit que les plus courtes distances (renseignement quantitatif). Un plus court chemin peut facilement être établi en travaillant à l'envers depuis le noeud terminal de telle sorte qu'on se dirige vers le prédécesseur dont le label diffère exactement de la longueur de l'arc joignant les deux noeuds. Mais si tous les plus courts chemins sont désirés, il suffit de déterminer le prédécesseur de chaque noeud (et non de rechercher n_N-1 chemins par la méthode explicitée ci-dessus).

Remarques :

- Au cours de l'exécution de l'algorithme, plus le nombre de labels permanents augmente, plus le nombre d'additions et de comparaisons nécessaires pour modifier les labels temporaires décroît.

  Ainsi, lors de la première itération, n_N-1 labels sont temporaires ; le pas 2 requiert n_N-1 additions et n_N-1 comparaisons, et le pas 3, n_N-1 comparaisons.

  A la deuxième itération, ces nombres deviennent tous n_N-2, et ils vont diminuer jusqu'à atteindre 1 (l'algorithme se termine en exactement n_N-1 itérations si tous les plus courts chemins sont demandés).

  En faisant la somme on obtient $n_N(n_N-1)/2$ additions et $n_N(n_N-1)$ comparaisons.

  Nous avons écarté le fait qu'il faut distinguer les noeuds temporaires des permanents ; mais le coût de cette opération dépend de la manière dont on l'implémente, plus précisément du moment , dans le processus, où l'on effectuera les comparaisons nécessaires. Une borne supérieure à ce coût est $n_N(n_N-1)/2$ comparaisons.

  Note : Toutes ces évaluations ne sont que des bornes supérieures si l'on ne désire qu'un plus court chemin.

  Ainsi, le coût calcul est de l'ordre de $(n_N)^2$.

- Il faut préciser que pour un nombre n_N (de noeuds) fixés, le temps d'exécution est indépendant du nombre d'arcs que le graphe peut avoir. En effet, il est

implicitement supposé que le graphe est complet (i.e. chaque noeud est relié à **tous les autres**) car associer à chaque arc manquant un coût infini présuppose son existence.

- Si le nombre de connexions n_L est beaucoup plus petit que $n_N(n_N-1)$, on dira que le graphe est **creux** et il est alors possible de réduire le temps de calcul. Il faut pour cela définir un autre test qui n'altère que les labels temporaires des noeuds successeurs de celui le plus récemment rendu permanent. On évite ainsi de parcourir tous les noeuds à chaque itération. Il faut alors bien sûr assurer une différence confortable entre la durée nécessaire pour exécuter ce test, et le temps gagné par rapport à la solution précédente. Cette différence doit augmenter en même temps que le caractère creux du graphe ; de plus, elle dépend fortement de la structure de donnée mise en jeu.
- Nous avons supposé que les poids d_{ij} étaient non négatifs. Si certains coûts ne vérifiaient pas cette condition, l'algorithme ne marcherait pas !
- Pour trouver le plus court chemin entre **deux noeuds seulement** (a et b), certains auteurs proposent de commencer "par les deux bouts" et de construire simultanément les arbres pour ces deux noeuds. La chose la plus importante est alors de décider quand s'arrêter.

3. Evaluation de l'algorithme

3.1 Au niveau du stockage

Nous n'engageons ici encore aucun débat sur une structure de donnée optimale, nous évaluons simplement le nombre d'informations nécessaires ou suffisantes qu'il faut garder en mémoire pour un bon déroulement de l'algorithme. Ensuite, nous essayerons de tirer quelques conclusions par rapport au problème que nous étudions.

- On donne à chaque noeud un backnode; donc n_N éléments ont besoin d'être stockés.
- n_N éléments sont nécessaires pour les longueurs actuelles des plus courts chemins de l'origine vers tous les noeuds.
- Un flag doit en outre indiquer si un noeud doit être inspecté ou non ; donc n_N éléments en plus.
- $n_N + n_L$ éléments (au moins) sont nécessaires pour stocker le réseau lui-même et la longueur des arcs.

Ce qui, en tout, nécessite $4n_N + 2\ n_L$ éléments pour stocker un arbre et le réseau.

Note : Un seul arbre suffit en mémoire, car cet algorithme ne peut travailler que sur un seul arbre à la fois.

3.2 Au niveau temps-calcul

L'efficience des algorithmes construisant un arbre est réputée lorsque les noeuds sont seulement connectés avec quelques autres ou quand les plus courts chemins sont à chercher entre un sous-ensemble de noeuds.

4. Les méthodes heuristiques

Selon certains auteurs, quand on cherche des plus courts chemins entre beaucoup de noeuds, il serait possible de profiter du fait que certains noeuds sont géographiquement très proches. Cette idée serait d'application pour les très grands réseaux. En effet, deux arbres des plus courts chemins d'origines fort proches sont identiques à une "certaine distance" des origines; d'où il devrait être possible d'utiliser l'arbre d'une résolution pour le problème où l'origine serait changée.

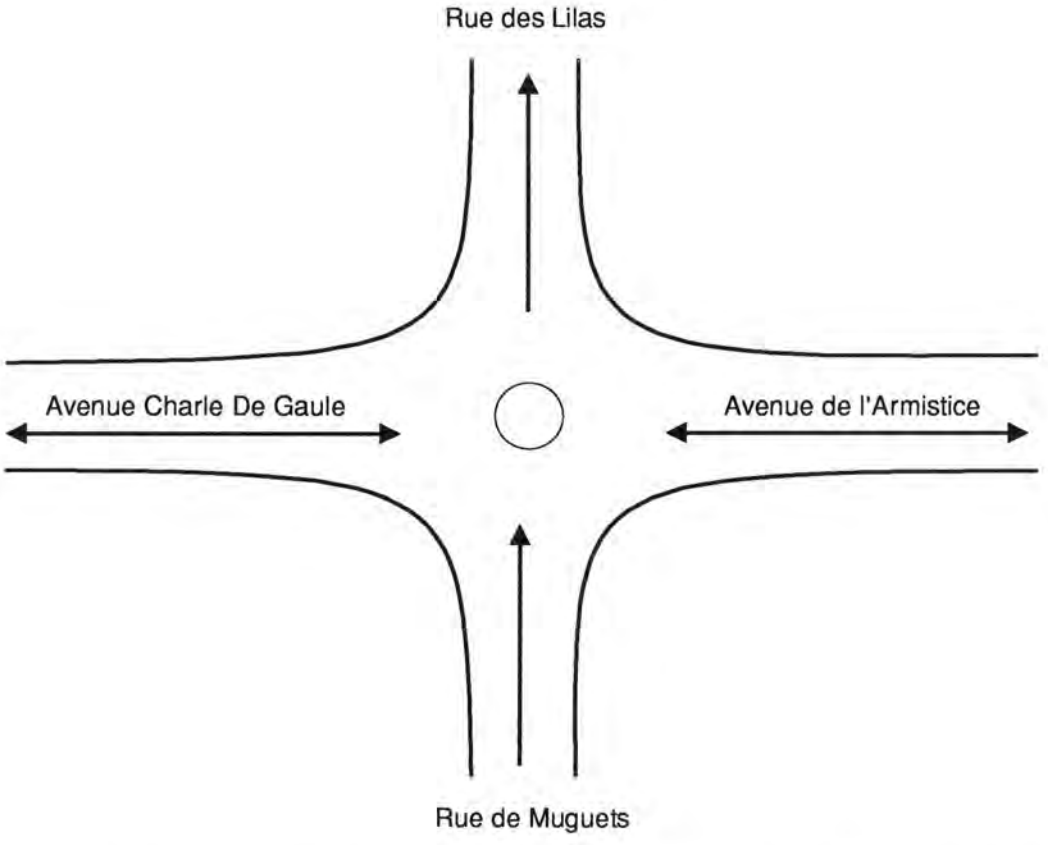
Malheureusement, deux arbres peuvent être identiques pour un certain nombre de noeuds sans l'être nécessairement pour tous. Ce n'est donc pas possible d'utiliser les arbres minimaux de noeuds voisins aussi directement que cela le paraît. Néanmoins, l'on peut considérer des parties d'arbres engendrés par des noeuds suffisamment proches, et épargner un temps-calcul assez long. On pourrait, à cet égard, calculer les arbres non à partir d'**un noeud**, mais d'**un *supernoeud*** englobant un groupe de noeuds voisins. Les chemins à l'intérieur de ce **supernoeud** feraient l'objet d'un autre calcul.

Chapitre Quatre
Implémentation des différents modules

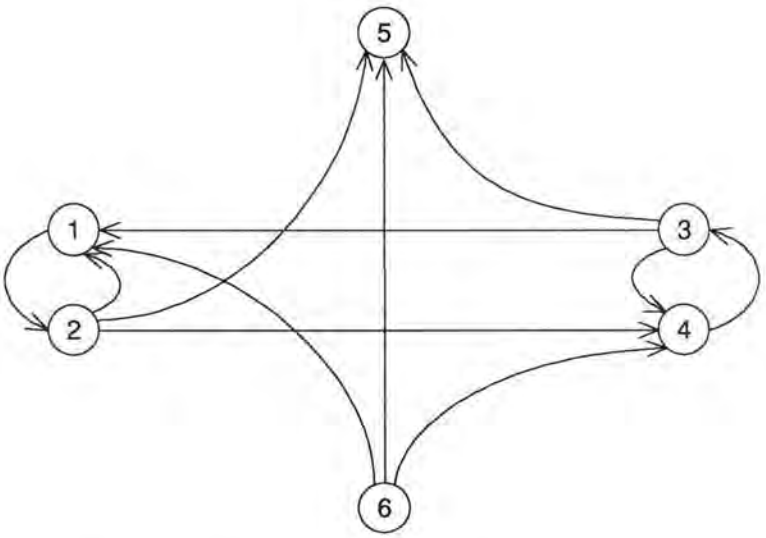
1. Le module de la carte

Le **réseau considéré** est, rappelons-le, associé à un **centre-ville**. Grossièrement, nous associerons un noeud du réseau à une **portion** de route, rue, ou avenue, déterminée par **deux carrefours successifs**, tandis que ces derniers seront à peu de chose près les arcs du réseau.

Exemple:



A partir de la *carte* ci-dessus, nous pouvons déterminer 6 noeuds. Un pour la rue des muguets, un pour la rue des Lilas, deux pour l'avenue Charles De Gaule, et deux également pour l'avenue de l'Armistice. Nous représentons les avenues par deux noeuds car elles sont à double sens et le fait de se trouver d'un côté ou l'autre de la route est d'une grande importance : si on se trouve du côté inférieur de l'avenue Ch. De Gaule, on peut se rendre plus rapidement Rue des Lilas que si on se trouve du côté supérieur (il faudrait d'abord faire demi-tour pour se trouver du côté inférieur). A partir de ces 6 noeuds, on détermine facilement les 11 arcs représentant les différentes transitions possible entre ces 6 noeuds :



L'arc entre les noeuds 1 et 2, ainsi que celui entre 3 et 4 n'existent que si il est possible de faire demi-tour dans cette rue sans passer par un autre noeud.

Quelles sont les caractéristiques d'un graphe associé à un centre-ville ?

- Le nombre de noeuds sera élevé (plusieurs centaines), si l'on désire une bonne représentation.
- Chaque noeud n'est pas connecté avec tous les autres, mais seulement avec certains d'entre eux (trois ou quatre, en général) se trouvant dans le voisinage de premier.
- Ainsi, le nombre d'arcs s'élèvera à trois ou quatre fois le nombre de noeuds.
- Les coûts relatifs aux arcs sont positifs ou nuls.

Le graphe considéré est donc ***de grande taille*** et ***relativement creux***.

Le modèle conceptuel de la carte d'une ville peut être exprimé à l'aide du modèle **entité-association** :

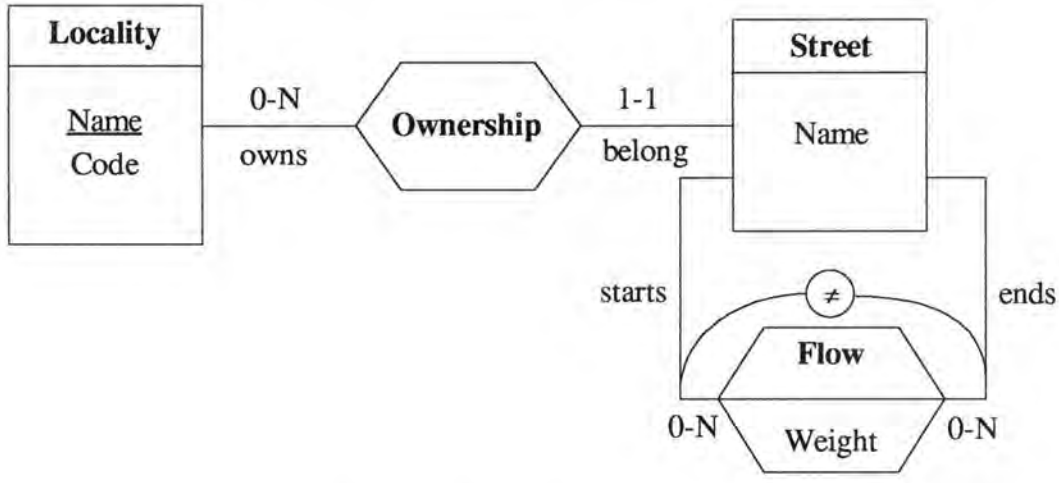


Schéma Entité-Association

Afin de normaliser ce schéma entité-association, nous devons éliminer toute association contenant des attributs. Dans notre cas il n'y en a qu'une seule.

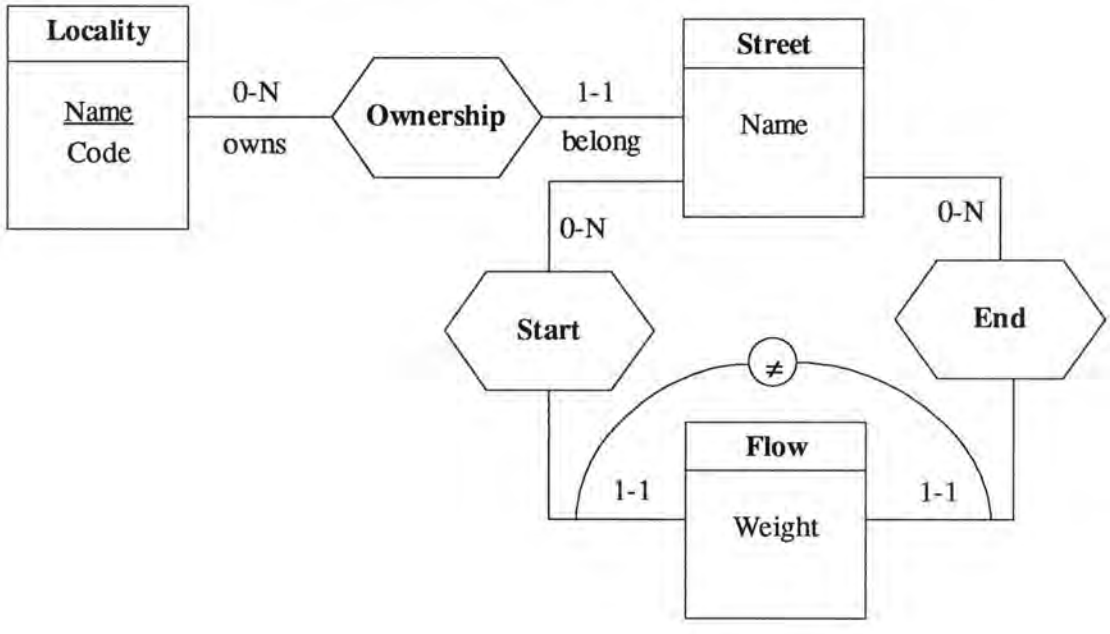
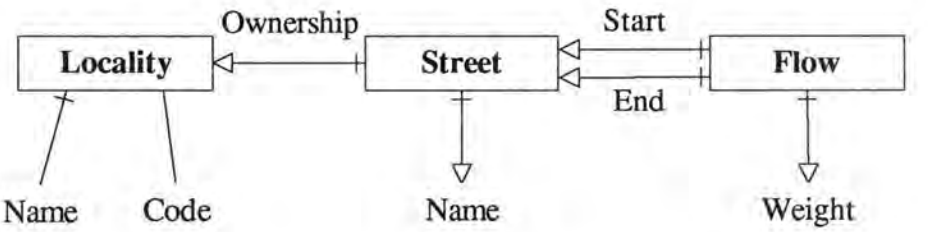


Schéma Entité-Association normalisé

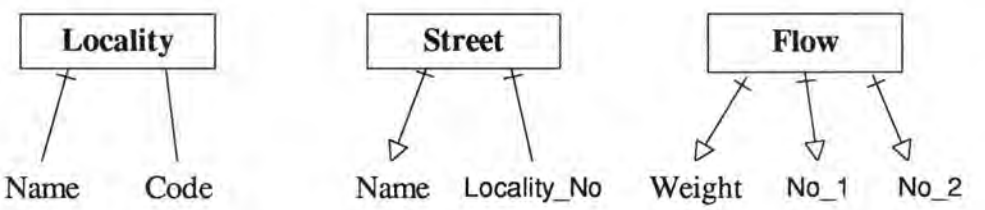
A partir de cette analyse conceptuelle, nous pouvons dériver un modèle d'accès généralisé qui est une solution *technique* indépendante des outils de réalisation.



contrainte d'intégrité : Start.Name(:Street) ≠ End.Name(:Street)

Modèle d'Accès Généralisé (MAG)

Le SGBD utilisé étant du type relationnel, nous devons supprimer les associations du MAG pour obtenir finalement le schéma conforme suivant :



contrainte d'intégrité :

Locality_No(:Street) in Name(:Locality)

No_1(:Flow) in Name(:Locality)

No_2(:Flow) in Name(:Locality)

No_1(:Flow) ≠ No_2(:Flow)

MAG conforme à un SGBD relationnel

1.1 Le logiciel Map Creator

Nous avons développé un logiciel ayant pour objet l'encodage d'une carte. Nous employons un moteur SQL (Microsoft Access 1.1) pour accéder à la base de données décrite au paragraphe précédent. Le développement de ce logiciel n'étant pas le but principal de notre travail, nous citerons simplement les fonctionnalités que ce logiciel offre à l'utilisateur.

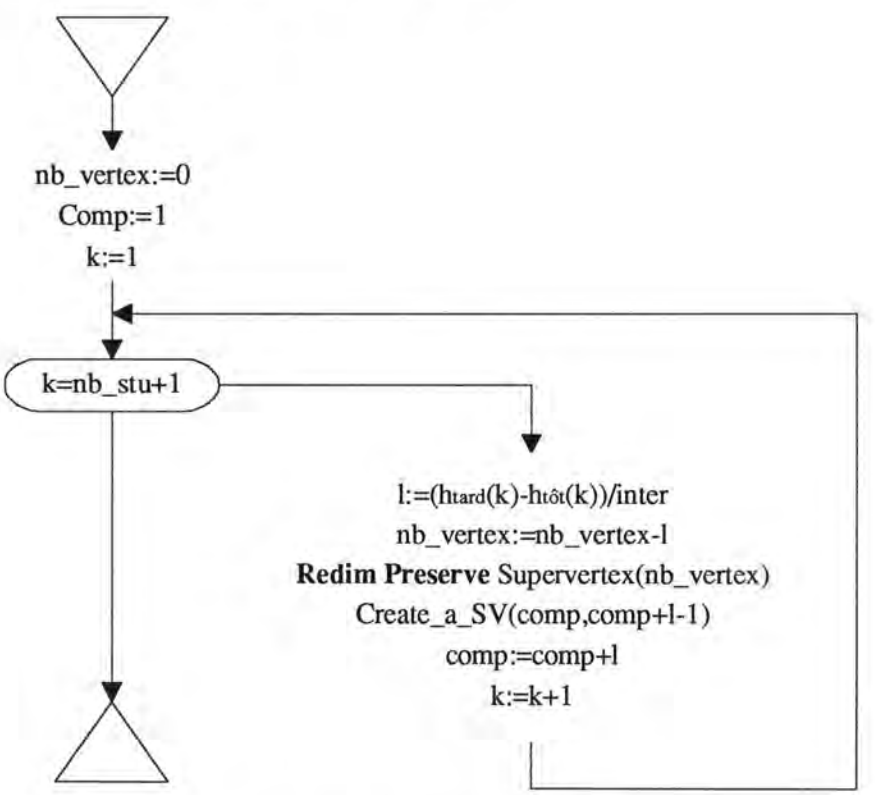
1. Localités :
    - Ajout d'une localité;
    - Mise à jour d'une localité;
    - Suppression d'une localité (cette suppression entraîne la suppression de toutes les routes appartenant à cette localité).
2. Routes :
    - Ajout d'une route dans une localité existante;
    - Mise à jour d'une route existante;
    - Suppression d'une route existante (cette suppression entraîne la suppression de tous les flux ayant cette route comme point de départ ou d'arrivée).
3. Flux[1] :
    - Ajout d'un flux entre deux routes existantes;
    - Mise à jour d'un flux existant;
    - Suppression d'un flux existant.

[1] L'existence d'un *flux* entre deux routes implique que l'on peut se déplacer de la route de départ vers celle d'arrivée en une seule transition. La valeur de ce *flux* indique le temps nécessaire pour effectuer ce déplacement.

2. Construction de la matrice d'adjacence

2.1 Le mécanisme de génération des supervertex

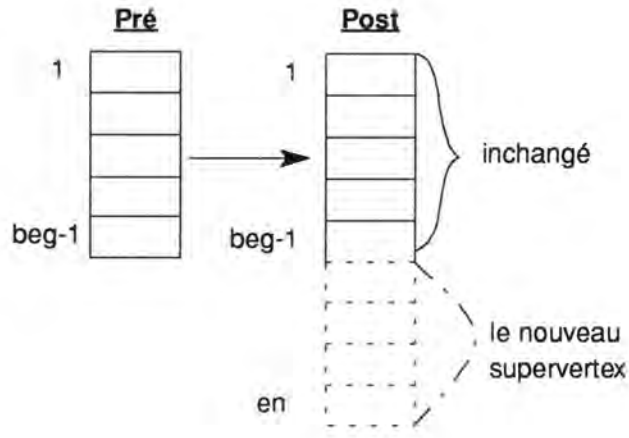
Afin de construire la matrice d'adjacence du graphe $G(E^*,Ar)$, nous devons définir un mécanisme de génération des supervertex. Soit un problème de nb_stu étudiants, l'algorithme suivant crée une suite de supervertex qui représentent chaque étudiant tel que cela a été défini au chapitre 1. A la fin de l'exécution de l'algorithme, la variable nb_vertex représente le nombre $\|E^*\|$.



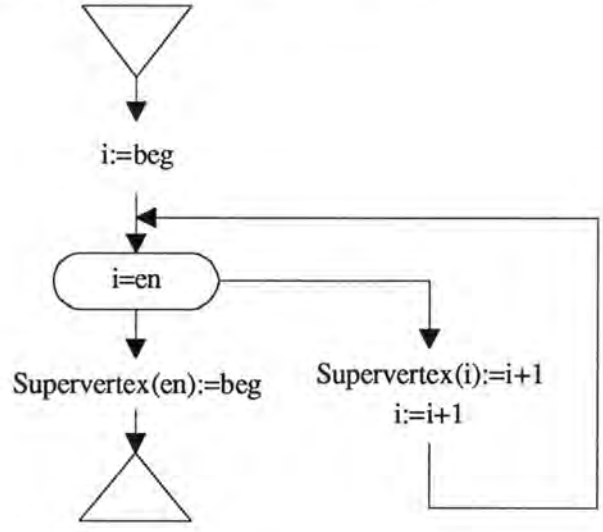
où l compte le nombre des noeuds à créer dans le nouveau supervertex;

comp est un pointeur vers l'élément suivant.

Remarque : **Redim Preserve** Supervertex(nb_vertex) redimensionne le tableau Supervertex() à nb_vertex éléments sans perdre les données existantes. Les nouvelles cases du tableau sont initialisées à zéro.



Create_a_SV(beg As Integer, en As Integer)



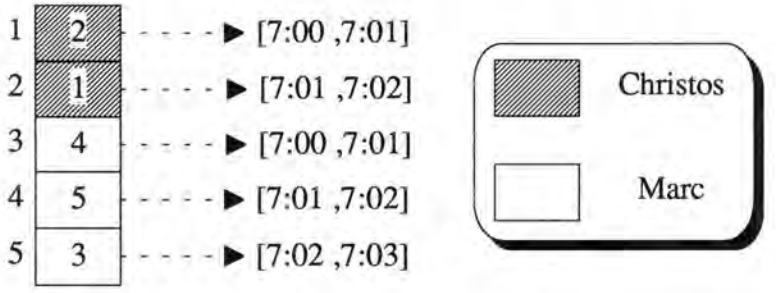
Appliquons cet algorithme sur l'exemple suivant :

Christos(7:00,7:02);

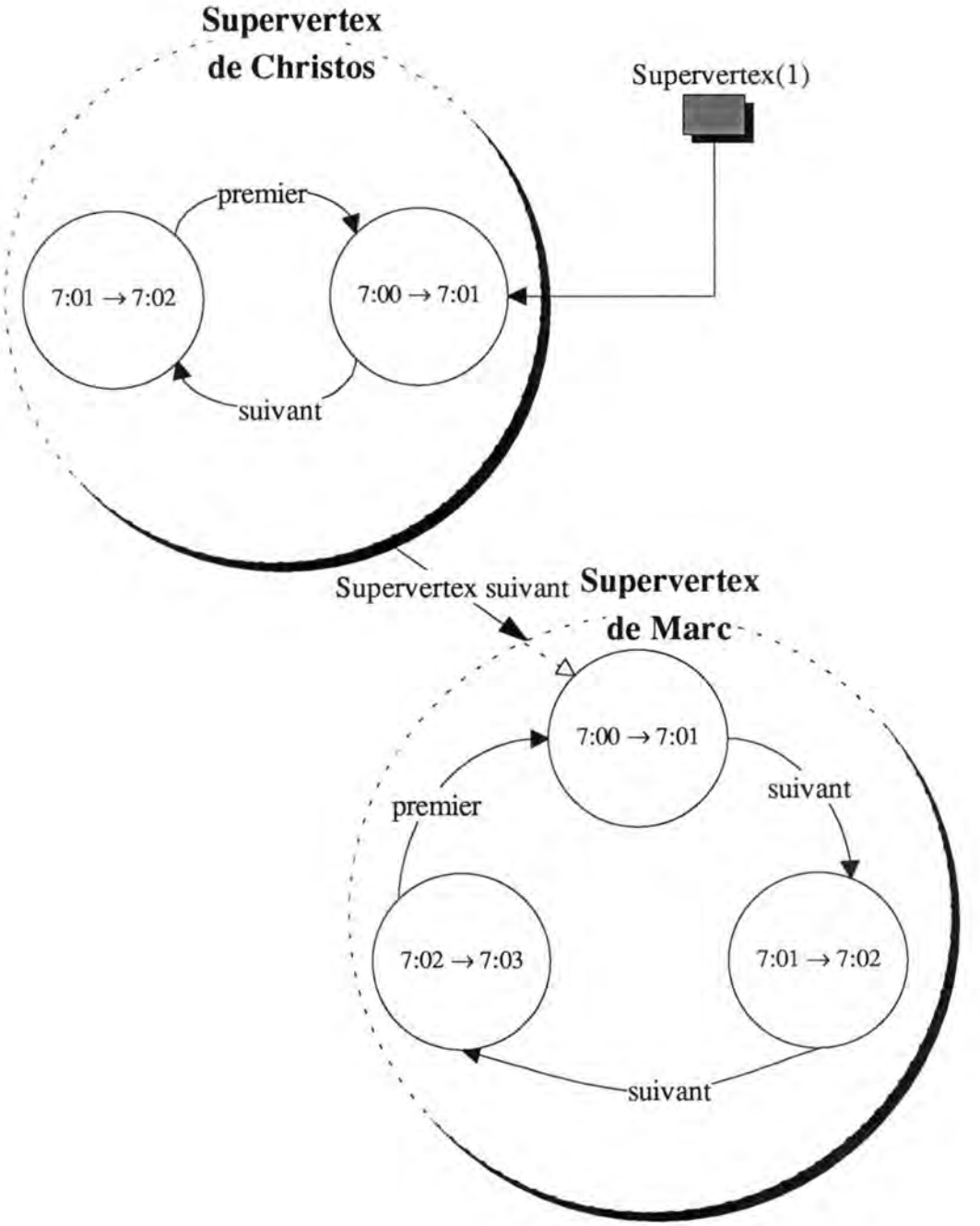
Marc(7:00,7:03);

inter=1.

A la fin de l'exécution de l'algorithme, nous aurons dans le tableau Supervertex:



La figure ci dessus représente une suite des listes circulaires dont chaque élément d'un même supervertex pointe vers son suivant et le dernier vers le premier du supervertex. D'une manière intuitive la figure précédente est équivalente à celle qui suit.



Pour montrer la puissance d'une telle représentation, trouvons ce qui représente le quatrième élément du tableau Supervertex. Nous savons que le premier étudiant est Christos(7:00,7:02), et nous pouvons calculer à l'aide de l'équation $(h_{tard}-h_{tôt})/inter$ le nombre de places qu'il occupe dans le tableau Supervertex. Il occupe par conséquent les (7:02 -7:00) / 1 = 2 premières places du tableau! Nous savons également que le deuxième étudiant est Marc(7:00,7:03). Nous pouvons donc déduire que Supervertex(3) représente Marc(7:00,7:01). Nous accédons à l'élément suivant donné par Supervertex(3)=4, qui se trouve donc à la quatrième place du tableau.

L'élément Supervertex(4) représente de nouveau Marc mais *inter* minutes plus tard que celui représenté par Supervertex(3). Nous pouvons donc déduire que Supervertex(4) représente Marc(7:01,7:02)!

2.2 Implémentation de l'algorithme du plus court chemin

2.2.1 Réécriture de l'algorithme de Dijkstra

2.2.1.1 Rappel de l'algorithme

Nous avions les notations suivantes :

l(i) désigne le label du noeud i

s est le noeud à partir duquel les plus courts chemins sont calculés

P est le noeud rendu permanent à chaque itération

1. Initialisation

$l(s) \leftarrow 0$

$l(i) \leftarrow \infty \quad \forall\, i \neq s$

$p \leftarrow s$

2. Mise à jour des labels

Pour tout noeud i de l'adjacence de p, ayant un label temporaire, faire

$l(i) \leftarrow \min\{l(i), l(p)+d_{pj}\}$

3. Trouver le label à rendre permanent

trouver x* parmi tous les noeuds tel que $l(x^*) = \min\{l(i)\}$

4. Fixer un label comme permanent

$p \leftarrow x^*$

5. Test de terminaison

i) si seulement le chemin de s jusque a est désiré

si p = a, l(p) est la longueur du plus court chemin recherché

sinon aller en 2

ii) si les plus courts chemins de s vers tous les autres noeuds sont voulus

si tous les labels sont permanents, alors les coûts des plus courts chemins sont les valeurs des labels

sinon aller en 2

2.2.1.2 Remarques

Nous désirons un algorithme calculant **tous** les plus courts chemins entre **chaque paire d'étudiants**. L'algorithme décrit ci-dessus sera donc exécuté autant de fois qu'il y a d'étudiants, ou, pour être plus exact, autant de fois qu'il y de noeuds dans le graphe qui sont *habités par au moins un étudiant*[2] . Chacune de ces exécutions se fera avec pour noeud de départ un des noeuds *habités*.

2.2.2 1ère adaptation de l'algorithme de Dijkstra

- Le pas 5 sera inclus dans l'écriture d'une boucle s'effectuant n_N-1 **fois**, puisque le processus pour trouver **tous** les plus courts chemins à partir d'un noeud se termine en n_N-1 itérations de l'algorithme.
- Afin de réaliser le pas 3 d'une itération, nous disposerons d'un masque Temp et d'une valeur Min. Au départ, le masque Temp indiquera que tous les noeuds sauf celui de départ sont temporaires, tandis que la valeur de Min vaudra *l'infini*[3] . Ensuite, nous allons parcourir tous les noeuds du graphe et, chaque fois qu'un noeud temporaire sera rencontré, c'est à dire chaque fois que la valeur du masque Temp indique que ce noeud est temporaire, la valeur Min sera mise à jour: cette valeur ne peut que diminuer et ce, dans le cas où la valeur du plus court chemin courant associé au noeud visité est inférieure à Min (à qui cette dernière valeur sera affectée).

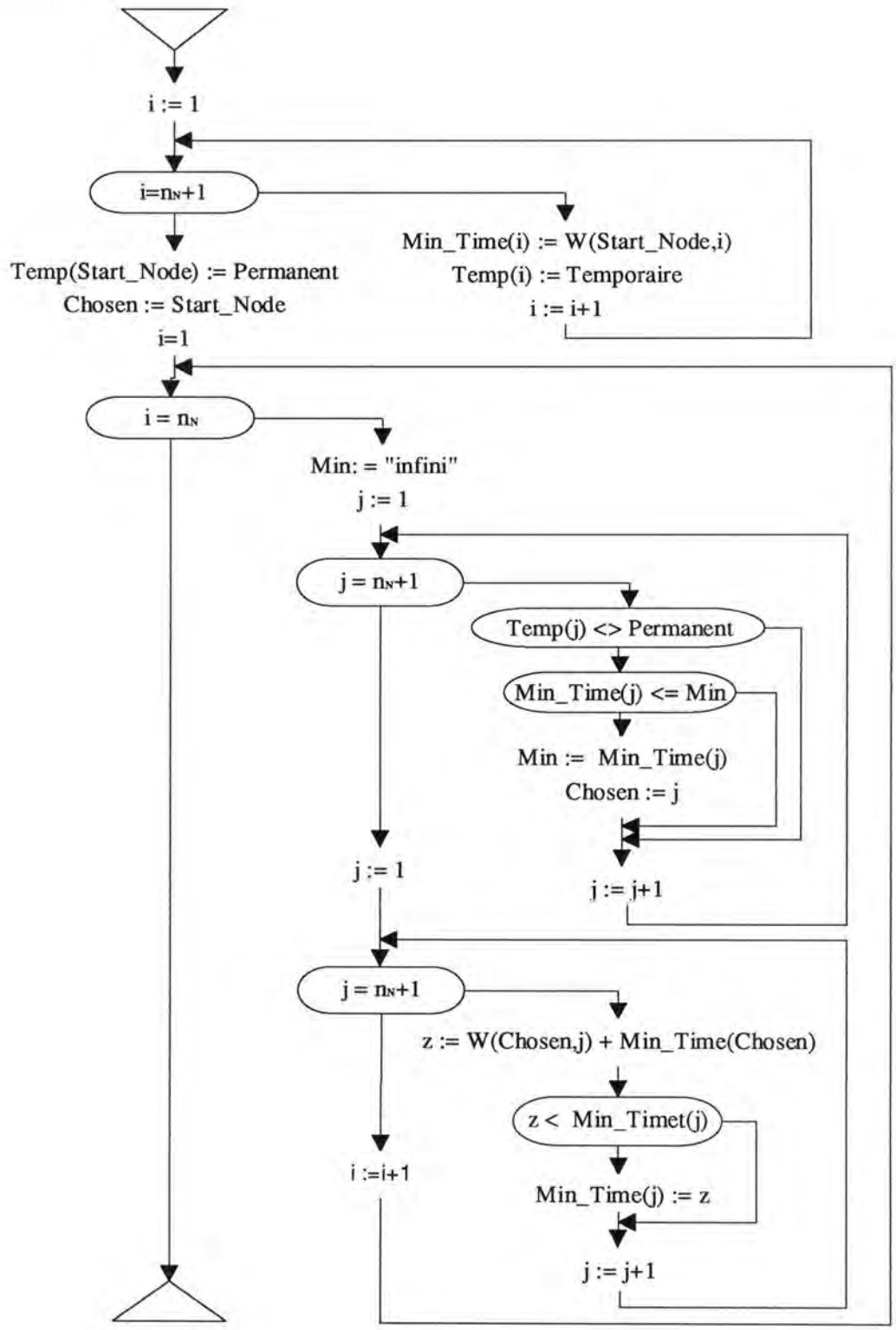
---

[2] Plusieurs étudiants peuvent habiter une même portion de route représentée par un seul noeud.

[3] Pratiquement, la valeur *infinie* n'est pas représentable dans un langage et on utilisera une valeur très grande.

- La méthode choisie pour résoudre le pas 3 empêche de ne parcourir que les adjacents temporaires du noeud p lors d'une itération. Il faut considérer tous les noeuds à label temporaire (pour pouvoir obtenir le label minimum dans Min).
- Il faudra, ensuite, déterminer la manière d'obtenir le d_{Pi} du pas 2. Dans l'algorithme qui suit, nous lui associons *W(i,j)*.

2.2.2.1 Procédure DIJKSTRA-1



La structure du programme ainsi créée nous oblige à concevoir une fonction de coût W(i,j) renvoyant le poids d'un arc (i,j) à partir *d'un dictionnaire d'adjacence.*

2.2.2.2 La fonction de coût

2.2.2.2.1 La structure de donnée du dictionnaire des successeurs

Le dictionnaire des successeurs est un tableau à deux dimensions contenant autant de lignes qu'il y a de noeuds dans le graphe (n_N) et 8 colonnes (4 paires). La $i^{ème}$ ligne contient la liste des noeuds que l'on peut atteindre à partir du $i^{ème}$ noeud (les noeuds adjacents au noeud i) ainsi que le poids des arcs pour atteindre ces noeuds.

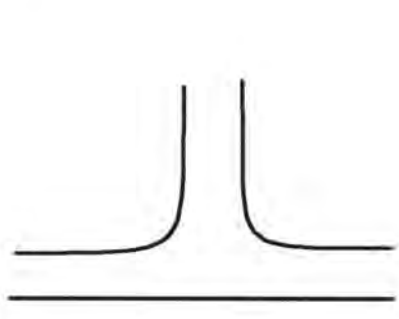
Par exemple, si la $13^{ème}$ ligne est de la forme:

13→	1	5	12	4	46	7	-1	-1

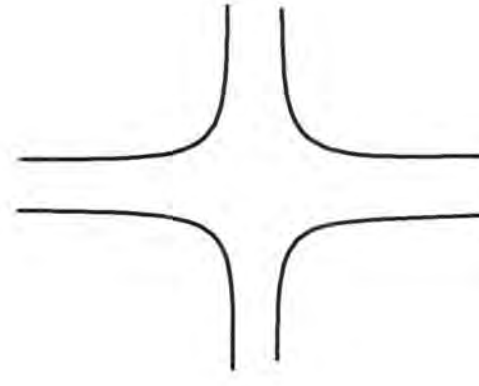
cela veut dire que à partir du noeud 13, on peut atteindre trois noeuds : le noeud 1 en 5 secondes, le noeud 12 en 4 secondes le noeud 46 en 7 secondes.

Nous considérons un maximum de 4 paires par ligne, donc un maximum de 4 noeuds accessibles à partir d'un autre. Dans le cas (rare) où , à partir d'un noeud, on peut accéder à plus que 4 autres, nous *dédoublerons* le noeud et nous placerons un chemin de coût nul entre ces deux noeuds.

Nous avons choisi 4 noeuds adjacents dans la mesure où les carrefours les plus fréquemment rencontrés sont de la forme :



ou bien

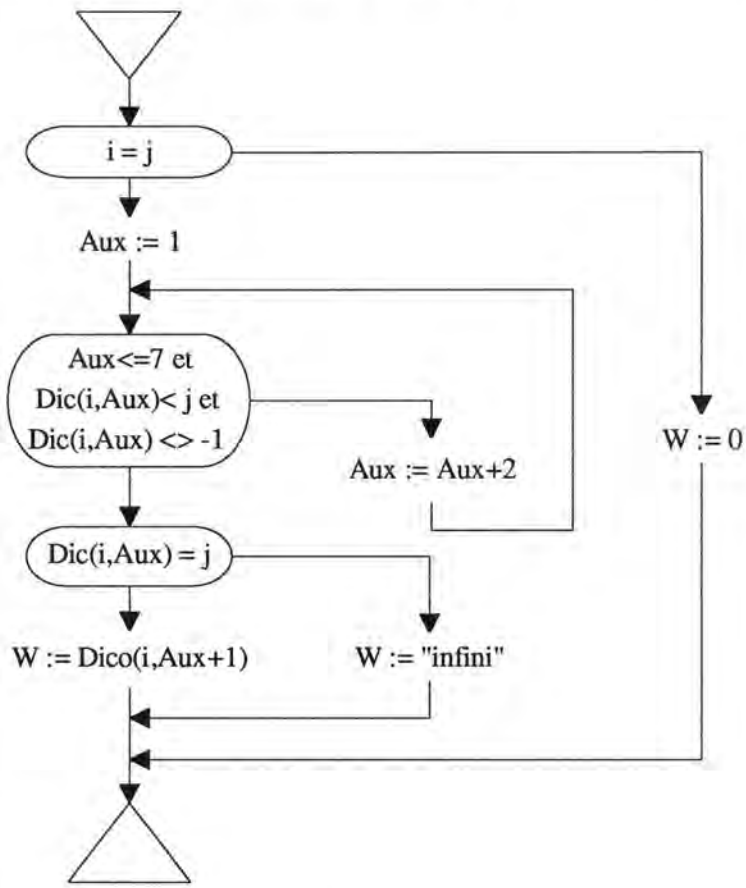


ce qui nous donne un maximum de 4 *suivants* par noeuds si toutes les routes sont à double sens.

Le chargement de ce dictionnaire en mémoire ne s'effectuera qu'une seule fois pour une même carte. En effet, les données nécessaires à sa construction ne seront jamais modifiées entre deux exécutions éventuelles de l'algorithme du calcul des plus courts chemins.

2.2.2.2.2 L'algorithme w(i,j)

Il nous faut parcourir la liste des suivants engendrée à partir du noeud i, si cette dernière est non vide. Vu que nous avons construit le dictionnaire de telle sorte que les noeuds soient ordonnés en ordre croissant, l'arc (i,j) n'existera pas si les noeuds parcourus sont plus grands que j ou si l'on est en fin de liste. Dans ce cas la valeur *infini* sera affectée au coût de l'arc correspondant.



Pour éviter des problèmes lors de l'évaluation de la seconde condition, nous considérerons que lorsqu'une condition de la forme "A et B" doit être évaluée, si la condition "A" est fausse, alors la condition "B" ne sera pas évaluée.

Remarquons que nous considérons un centre-ville, un noeud possède donc toujours un successeur ; sinon, ce serait un point *sans retour* en réalité inconcevable.

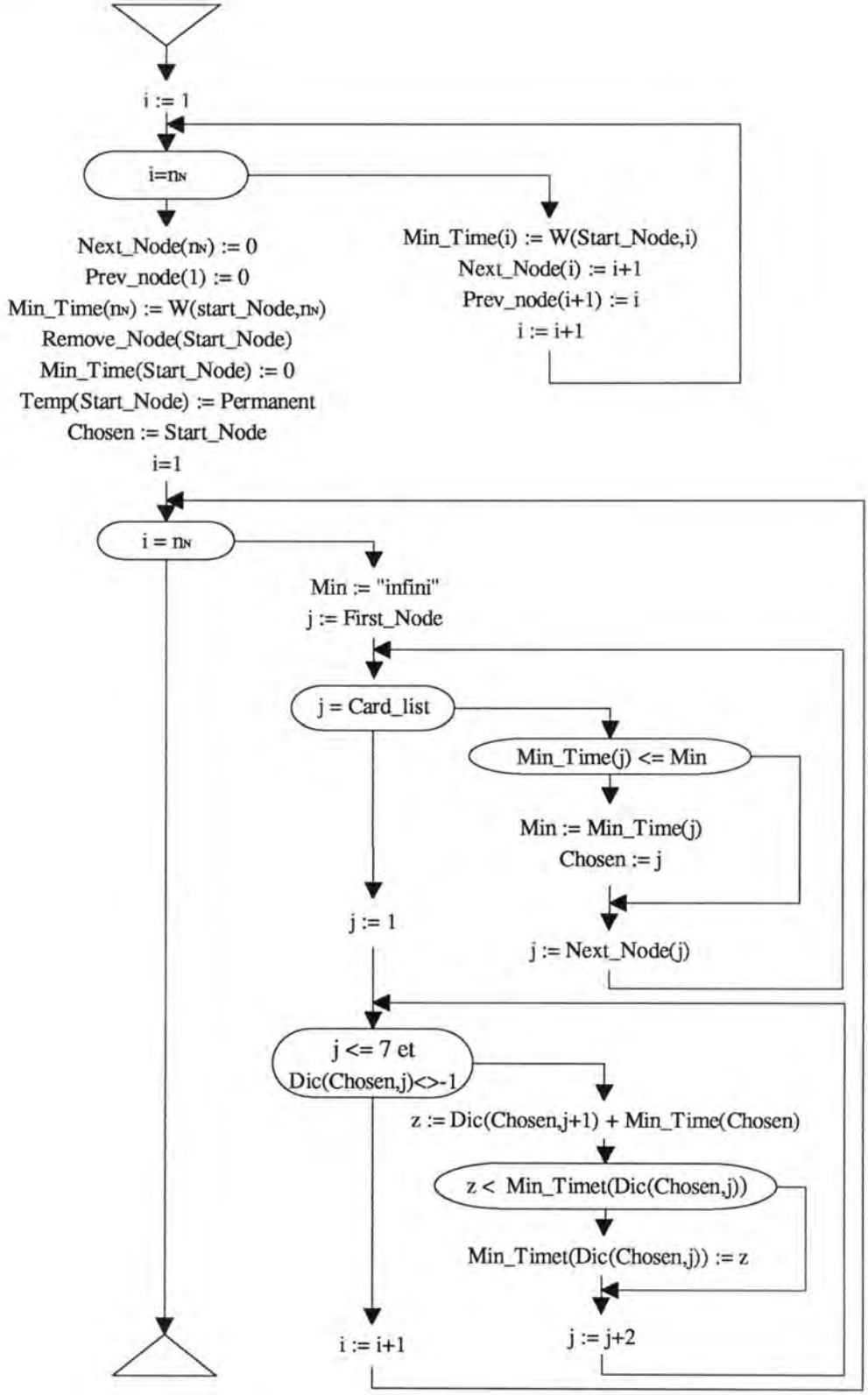
2.2.3 2ème adaptation de l'algorithme de Dijkstra

Dans la version précédente de l'algorithme, les pas 2 et 3 étaient tout deux inclus dans une boucle s'effectuant autant de fois qu'il y a de noeuds dans le graphe.

Etant donné la lourdeur en temps-calcul d'une telle tâche, c'est sur ces deux boucles que porteront les deux premières améliorations de l'algorithme. Ces premières améliorations ne nécessitent pas l'ajout de structure de données supplémentaires, mais demandent la modification de l'une d'entre elles. Nous étudierons ultérieurement les améliorations du second type.

- Afin de réaliser le pas 3 d'une itération, nous disposerons maintenant d'une liste doublement chaînée contenant les noeuds temporaires. Au départ, cette liste contiendra tous les noeuds du graphe sauf le noeud de départ. Nous procéderons de la même manière que dans l'algorithme précédent mais en ne considérant que les noeuds compris dans la liste. Le noeud choisit pour devenir permanent sera évidement retiré de la liste. Par ce procédé, le nombre d'itération de la boucle sera divisé, en moyenne, par deux, mais la maintenance de cette liste pénalisera un peu cet avantage.
- La méthode choisie pour diminuer le temps d'exécution du pas 2 ne nécessite quant à elle aucune modification des structures de données, mais profite mieux des informations que peut nous fournir le dictionnaire des successeurs. On remarque que lors de la mise à jour des labels des noeuds, les seuls labels qui peuvent éventuellement être modifiés sont les suivants du noeud *actif* p. La boucle ne s'effectuera donc, que pour les noeuds successeurs de p, c'est-à-dire la $i^{ème}$ ligne du dictionnaire des successeurs.

2.2.3.1 Procédure DIJKSTRA-2



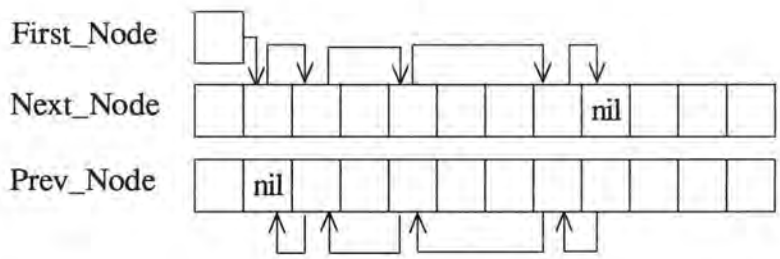
2.2.3.2 Gestion de la liste des noeuds temporaires

2.2.3.2.1 La structure de données de la liste

La structure de donnée de la liste des noeuds temporaires se compose de :

- deux tableaux: Prev_Node(.) et Next_Node(.),
- et d'un *pointeur* vers le premier élément: First_Node.

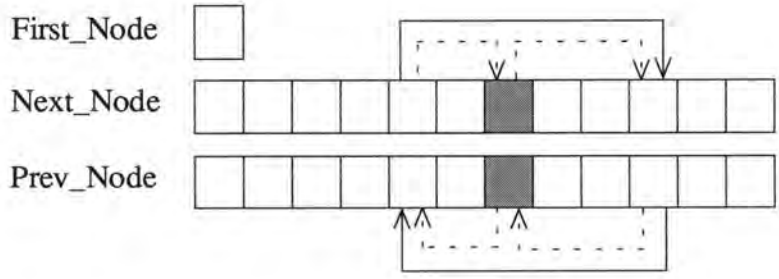
Examinons cette structure sur un exemple:



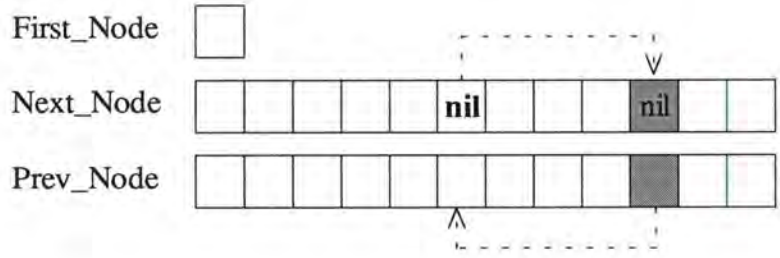
On voit que la liste des noeuds temporaires est 2 (First_Node), 3 (Next_Node(2)), 5 (Next_Node(3)), 8 (Next_Node(5)), 9 (Next_Node(8)), et que 8 est le dernier noeud de la liste vu que Next_Node(8)=nil

La seule tâche délicate lors de la gestion de cette liste de noeuds temporaires, est la suppression d'un noeud. Les quatre cas qui peuvent se présenter sont représentés par les quatre figures qui suivent. Dans ces figures, nous avons représenté en hachuré l'élément à supprimer, en pointillé les pointeurs tels qu'ils étaient avant la suppression, en trait continu les pointeurs tels qu'ils sont après la suppression.

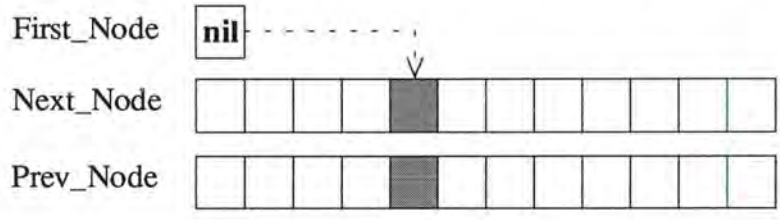
Cas général



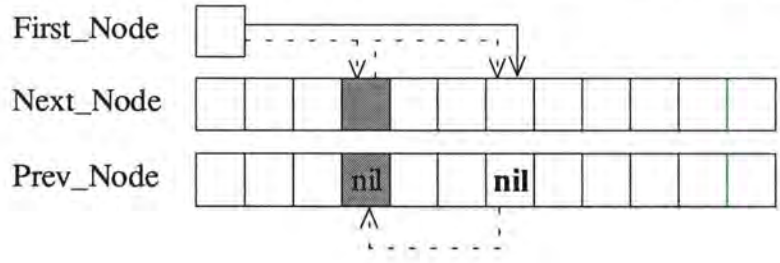
Cas où l'élément à retirer est le dernier élément de la liste mais pas le premier



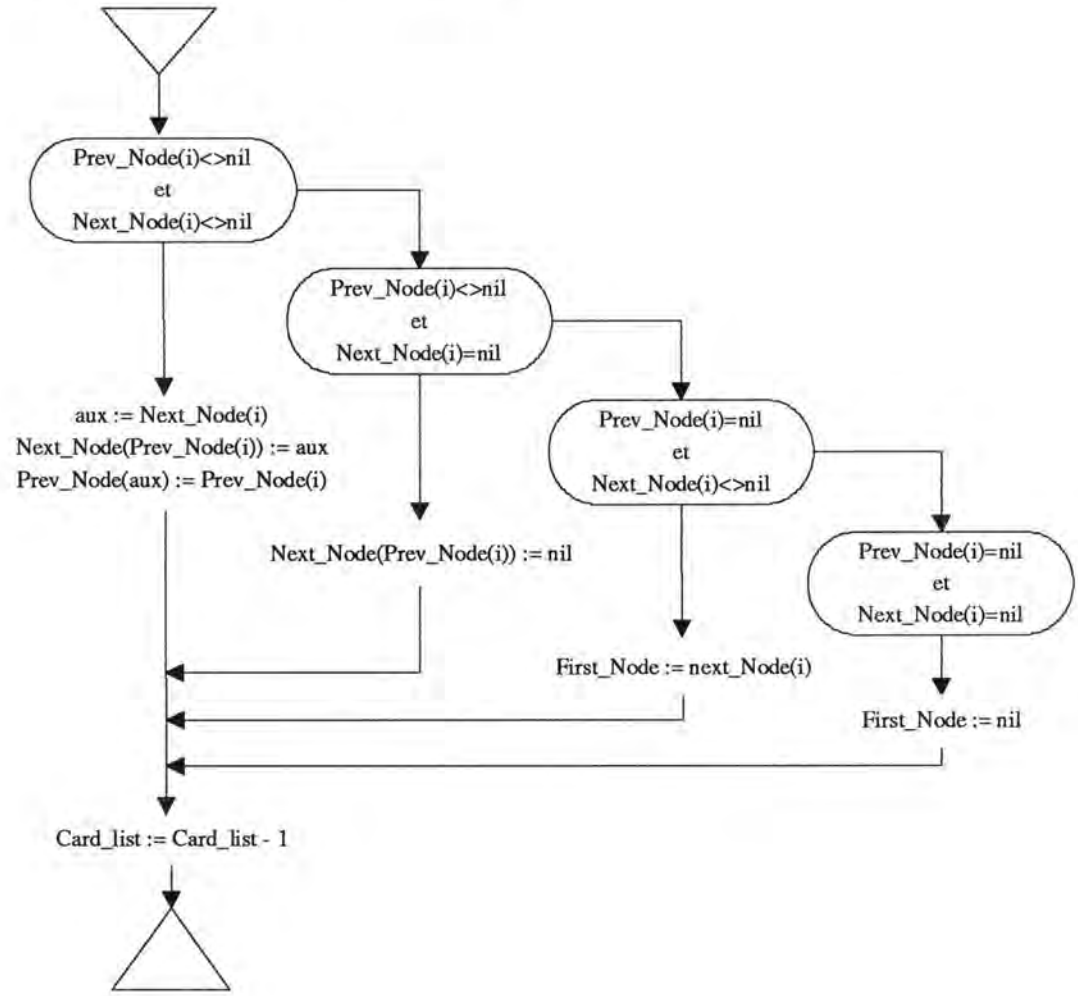
Cas où l'élément à retirer est à la fois le dernier et le premier élément de la liste



Cas où l'élément à retirer est le premier de la liste mauis pas le dernier



2.2.3.2.2 L'algorithme Remove_Node(i)



2.2.4 3ème adaptation de l'algorithme de Dijkstra

Cette amélioration implique l'ajout d'une structure de données supplémentaire EFG qui représente une partition de l'ensemble des étudiants en trois ensembles : E, F, et G. Grâce à cette partition EFG, nous allons réduire le nombre d'itérations de la boucle globale représentant le pas 5 de l'algorithme.

En effet, jusque maintenant, nous effectuons cette boucle n_N-1 fois de telle sorte que les plus courts chemins depuis le noeud habité considéré jusqu'à **tous les autres noeuds du graphe** sont calculés. Cependant, nous ne désirons pas connaître tous ces plus courts chemins mais seulement une partie d'entre eux : les plus courts chemins entre deux paires d'étudiants. En réalité, nous ne sommes donc intéressés que par les chemins qui partent du noeud de départ vers des noeuds habités par des étudiants.

Afin de tenir compte de cette information, avant d'exécuter l'algorithme, nous allons *répertorier* les noeuds qui sont habités en les *marquant* comme faisant partie de l'ensemble G. Ensuite, lors des différentes itérations de l'algorithme, chaque fois qu'un noeud est rendu permanent, on regarde s'il est habité, c'est-à-dire s'il est *marqué* comme appartenant à G, si c'est le cas, on retire ce noeud de l'ensemble G pour le placer dans l'ensemble E. L'ensemble G contient donc : *"les noeuds habités pour lesquels on ne connaît pas encore la valeur des plus courts chemins"* tandis que l'ensemble E contient : *"les noeuds habités pour lesquels on connaît la valeur des plus courts chemins"*.

On pourra évidement arrêter l'exécution de l'algorithme dès que l'ensemble G est vide puisque cela voudra dire que l'on connaît la valeur des plus courts chemins pour tous les noeuds pour lesquels on désirait connaître cette valeur.

La dernière modification de l'algorithme, ainsi que l'emploi du troisième ensemble exploite la particularité même du problème que l'on traite. En effet, ce qui nous intéresse ce n'est pas tellement de connaître la valeur du plus court chemin entre deux étudiants mais plutôt de savoir si l'on peut ou non placer ces deux étudiants dans le même bus. Or nous savons que l'algorithme de Dijkstra trie selon un *ordre de proximité croissant*[4] les noeuds du graphes. Autrement dit, lors d'une itération si le noeud k est choisi pour devenir permanent, on sait que le temps minimum pour atteindre n'importe quel noeud temporaire (entre autre, n'importe quel noeud de G) vaudra au minimum Min_Time(k). Donc, si un noeud est tel que: l'heure minimale de chargement de l'étudiant habitant au noeud de départ[5] + Min_Time(k) > l'heure maximale de chargement de l'étudiant habitant ce noeud, on sait que de toute façon on ne pourra pas charger les deux étudiants dans le même bus (alors qu'on ne connaît pas exactement le temps qui les sépare !). Ces noeuds seront retirés de l'ensemble G pour être placés dans l'ensemble F. L'ensemble F contient donc : *"les noeuds habités pour lesquels on ne connaît pas encore la valeur des plus courts chemins mais pour lesquels on sait que ce temps sera trop long"*.

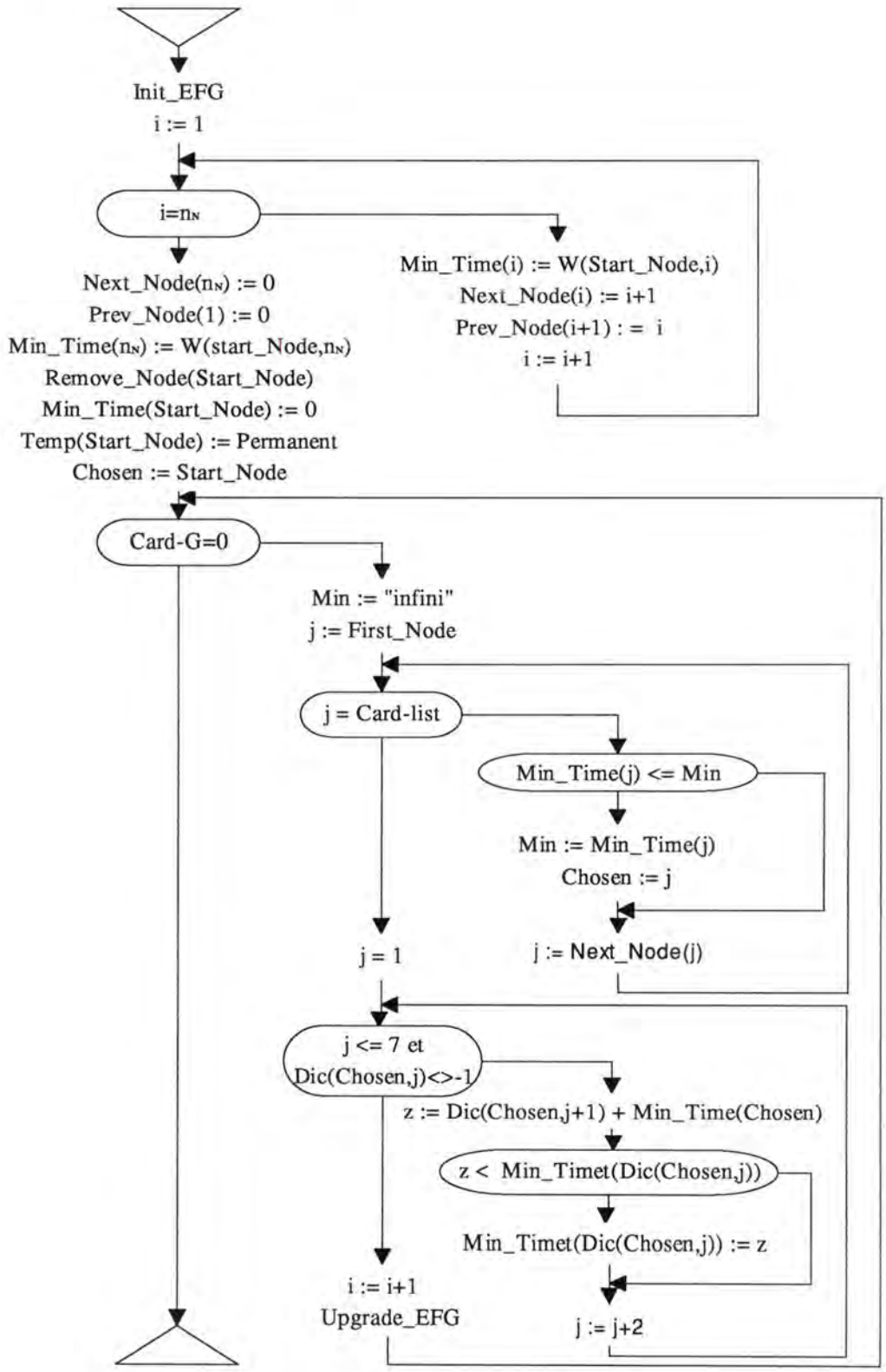
La version finale de l'algorithme de calculs des plus courts chemins est présenté à la page suivante.

---

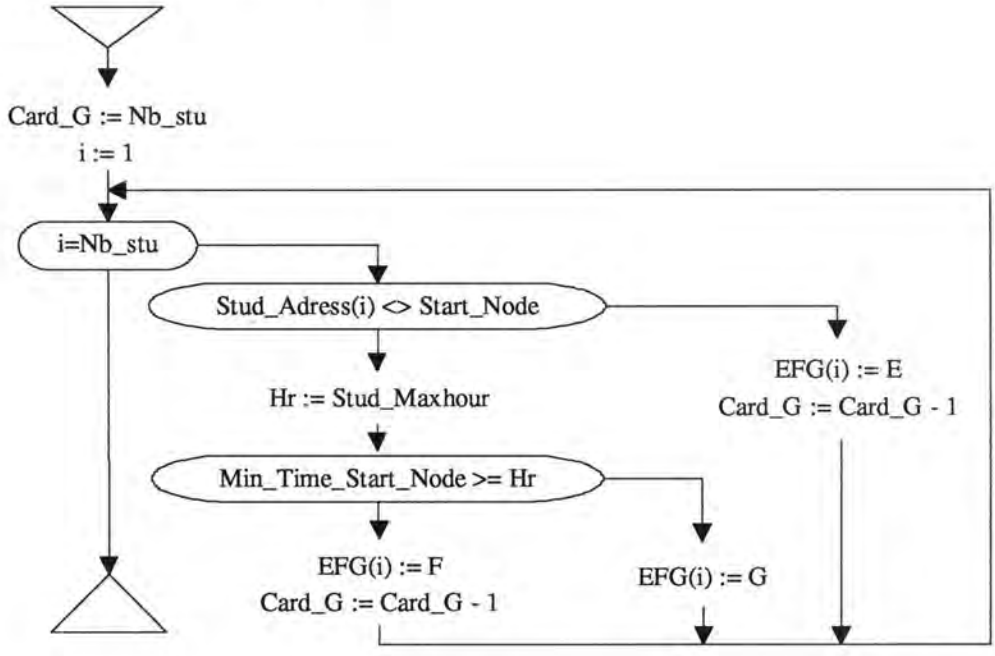
[4] c.f. 3.2.2.3

[5] Start_Node

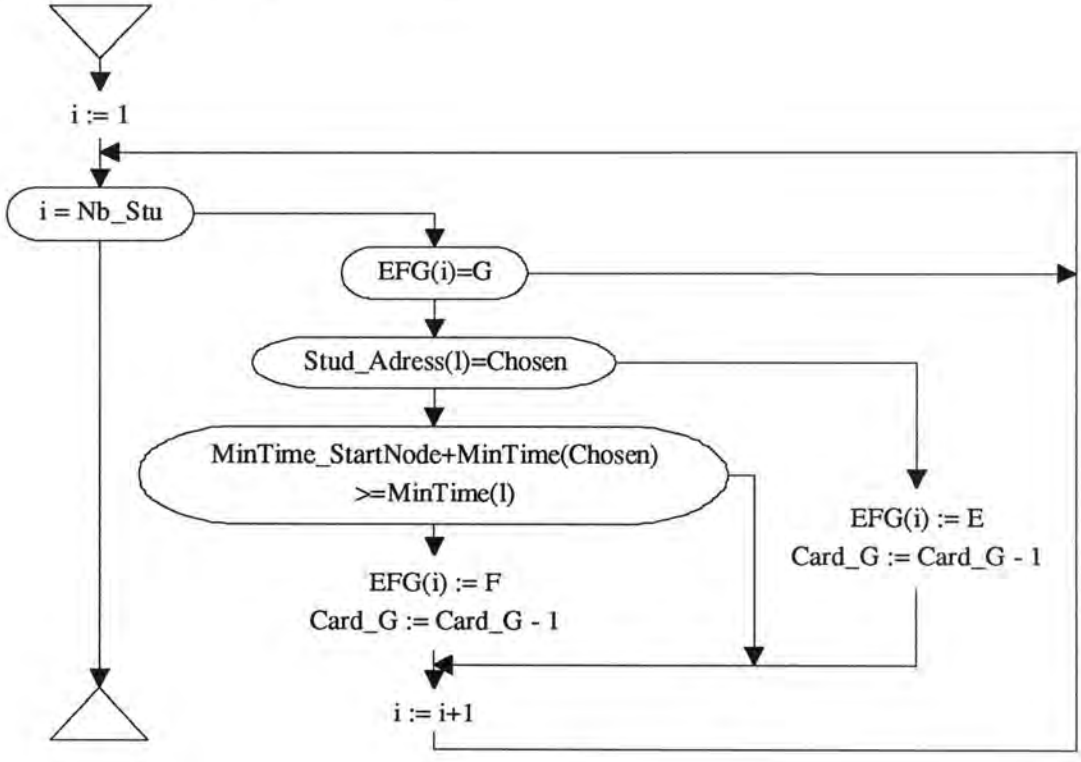
2.2.4.1 Procédure DIJKSTRA-3



2.2.4.1.1 La procédure Init_EFG



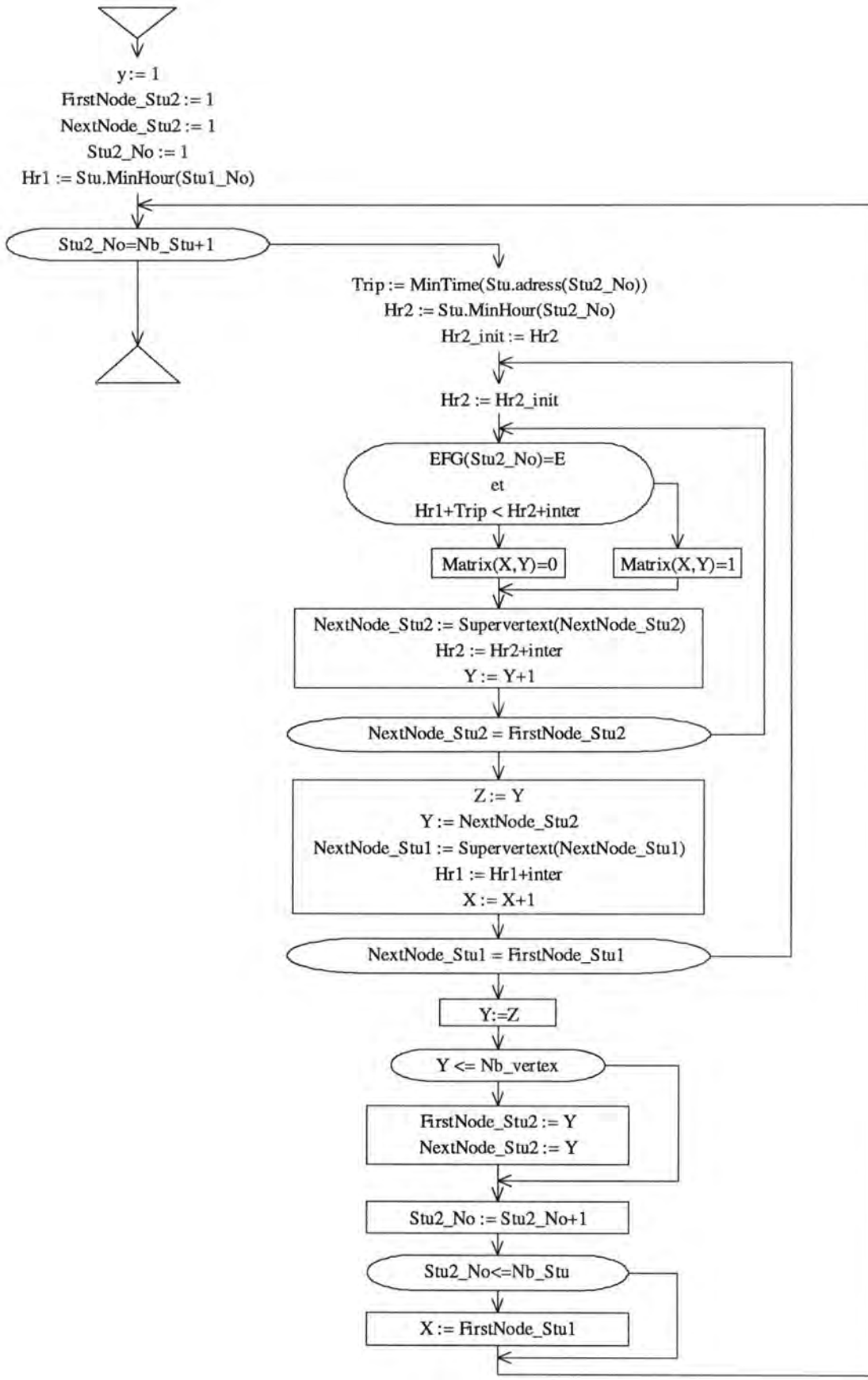
2.2.4.1.2 La procédure Update_EFG



2.3 La matrice d'adjacence

A partir du tableau des Supervertex (2.1), chaque fois que l'algorithme des plus courts chemins (2.2) se termine, nous construisons une *partie* de la matrice d'adjacence.

Comme nous l'avons expliqué au paragraphe 2.2.1.2 de ce chapitre, l'algorithme de calcul des plus courts chemins est lancé autant de fois qu'il y a de noeuds habités et chacune de ces exécutions se fait avec pour noeud de départ un de ces noeuds habités. Dès que cette exécution est terminée, nous allons mettre à jour les valeurs des lignes de la matrice d'adjacence qui correspondent aux différents noeuds du graphe **G**(E*,Ar) représentant le (ou les) étudiant(s) habitant ce noeud de départ.


y := 1
FirstNode_Stu2 := 1
NextNode_Stu2 := 1
Stu2_No := 1
Hr1 := Stu.MinHour(Stu1_No)
Stu2_No=Nb_Stu+1
Trip := MinTime(Stu.adress(Stu2_No))
Hr2 := Stu.MinHour(Stu2_No)
Hr2_init := Hr2
Hr2 := Hr2_init
EFG(Stu2_No)=E
et
Hr1+Trip < Hr2+inter
Matrix(X,Y)=0
Matrix(X,Y)=1
NextNode_Stu2 := Supervertext(NextNode_Stu2)
Hr2 := Hr2+inter
Y := Y+1
NextNode_Stu2 = FirstNode_Stu2
Z := Y
Y := NextNode_Stu2
NextNode_Stu1 := Supervertext(NextNode_Stu1)
Hr1 := Hr1+inter
X := X+1
NextNode_Stu1 = FirstNode_Stu1
Y:=Z
Y <= Nb_vertex
FirstNode_Stu2 := Y
NextNode_Stu2 := Y
Stu2_No := Stu2_No+1
Stu2_No<=Nb_Stu
X := FirstNode_Stu1

3. Une heuristique pour colorer un graphe basée sur le recuit simulé

3.1 Introduction

Dans le paragraphe précédent, nous avons construit un procédé pour déduire la matrice d'adjacence à partir des données initiales du problème. Dans ce chapitre, nous allons utiliser cette matrice pour trouver une solution pour le problème de coloration de graphe à l'aide d'une heuristique basée sur le recuit simulé.

L'algorithme présenté ici est légèrement différent de celui présenté dans le chapitre 2. Nous avons apporté les modifications suivantes :

- Le temps d'exécution de l'algorithme est borné par un polynôme dépendant de la taille du problème traité[6].
- Le nombre maximal de couleurs disponibles pour colorer le graphe décroît pendant l'exécution de l'algorithme. Autrement dit, le nombre de bus disponible décroît !
- Le nombre d'étages (paliers) de température et la longueur des chaînes de Markov pour chaque palier sont fixés à une constante près.
- L'algorithme du recuit simulé est relancé plusieurs fois avant de fournir une première solution à l'utilisateur. Ce procédé est l'objet du paragraphe suivant.

3.2 La structure générale de l'exécution de l'algorithme

Pour qu'une solution trouvée, par un algorithme de recuit simulé, soit proche d'un optimum global, nous avons expliqué (c.f. chapitre 2) que la longueur des chaînes de Markov pour chaque palier de température doit être suffisamment grande. Dans

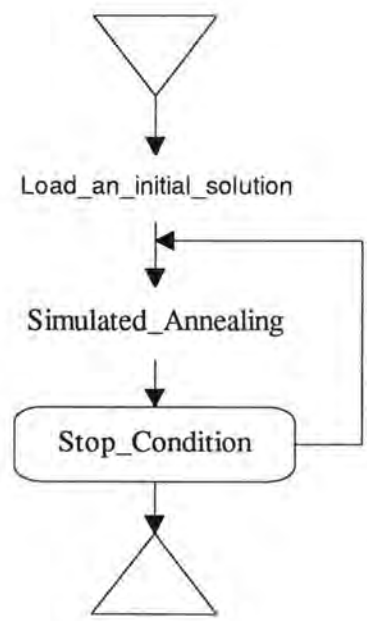
[6] La taille du problème est fonction du nombre d'étudiants et du nombre de bus.

cette optique, l'algorithme du recuit simulé est lancé une et une seule fois et à la fin de son exécution la solution fournie est supposée être proche de la solution optimale !

Plusieurs auteurs se sont posé la question de savoir si la qualité du résultat fourni après l'exécution d'une suite de N algorithmes de recuit simulé de longueur de chaînes de Markov L (identique pour tous les paliers) est meilleure que celle d'un algorithme identique de longueur de chaînes de Markov N.L .

Formellement, il n'existe aucune preuve que le résultat donné après une suite d'exécutions de l'algorithme soit meilleur que celui d'une seule exécution. Dans la pratique au contraire, le gain de temps d'exécution pour une qualité de résultat identique, favorise la suite des algorithmes.

Nous avons adopté l'optique *relancement du recuit simulé* pour résoudre notre problème et le schéma général de cette exécution répétitive est représenté par l'organigramme suivant.



3.3 La solution initiale (le principe des bus virtuels)

Nous pouvons remarquer que le recuit simulé présuppose une répartition initiale des étudiants dans les bus (c.f. 4.3.2) à partir de laquelle, en faisant des petites perturbations au voisinage, il approche la solution optimale. Cette répartition initiale nous l'appellerons dorénavant *solution initiale*. La procédure *'Load_an_initial _solution'* est chargée de générer une solution initiale pour notre problème.

Le principe le plus fréquemment utilisé pour la génération d'une solution initiale consiste à générer une solution initiale non admissible[7], ce qui entraîne l'ajout d'un **facteur de pénalisation**, dépendant de la grandeur de la *non-admissiblité* de la configuration, dans la fonction de coût. Cette technique présente un désavantage : il se peut que l'on ait une solution non-admissible comme solution finale!

La technique utilisée ici permet au recuit simulé de démarrer avec une solution initiale admissible. Le mécanisme de génération des solutions au voisinage d'une solution admissible est tel que les solutions générées sont des solutions admissibles. Par induction, il est donc trivial que toutes les solutions envisagées par notre algorithme seront des solutions admissibles. Le gain apporté en utilisant cette technique se reflète également dans le temps de calcul de la fonction de coût, car le calcul du facteur de pénalisation n'est plus nécessaire.

Supposons que nous travaillons sur un problème de N étudiants et B bus.

- Si N≤B nous considérons la solution initiale suivante :

  Les N premiers bus ramassent un et un seul étudiant et les B-N autres bus sont vides. On peut constater que cette solution est admissible. L'heure de ramassage de chaque étudiant est tirée aléatoirement dans l'ensemble des heures admissibles de cet étudiant;

- Si N>B, ce qui veut dire que nous avons plus d'étudiants que de bus, nous appliquons le même principe pour les B premiers étudiants et, pour les N-B qui restent, nous considérons des bus **virtuels** de capacité égale à deux[8].

---

[7] Une solution est admissible si elle vérifie toutes les contraintes.

[8] Le choix de la valeur de la capacité des bus virtuels sera justifié plus loin.

Nous pouvons alors appliquer à nouveaule même principe sur les bus virtuels[9].

De cette manière, nous avons toujours une solution initiale réalisable et admissible.

La coloration du graphe **G** revient à trouver un *mapping* '*color*' dans $\{1,2,\ldots,nb\_bus\}$ telle que $\forall e_1,e_2 \in \mathbf{G} : \{(color(e_1)=color(e_2)) \Rightarrow Adj_{e1,e2}=0\}$;

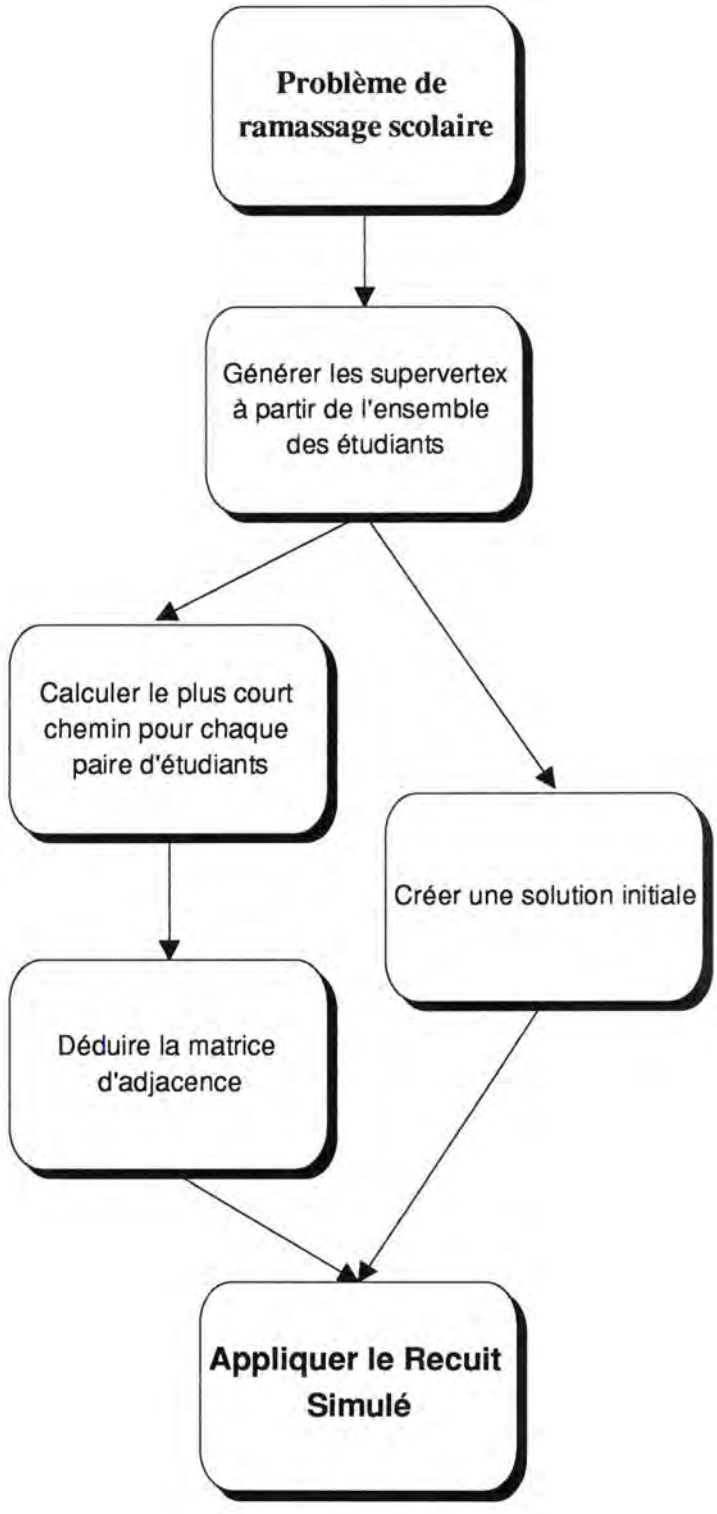
où $nb\_bus = \|\mathbf{B}\| + \|\mathbf{VB}\|$;

VB représente l'ensemble des bus virtuels et la définition de cet ensemble est analogue à celle de **B**;

Adj est la matrice d'adjacence du graphe **G**, déduite comme décrit dans le chapitre 1.

La découpe initiale en modules devient donc :

[9] La valeur de préférence attribuée aux bus virtuels sera justifiée plus loin


Problème de ramassage scolaire
Générer les supervertex à partir de l'ensemble des étudiants
Calculer le plus court chemin pour chaque paire d'étudiants
Créer une solution initiale
Déduire la matrice d'adjacence
Appliquer le Recuit Simulé

3.4 La condition d'arrêt

Nous avons choisi **trois conditions d'arrêt** du relancement de l'algorithme du recuit simulé. L'utilisateur a la possibilité de choisir celle qui lui convient le mieux. Le logiciel *MYTHOS* étant en plus un système *interactif*, chacune de ces conditions correspond à un niveau d'interactivité différent !

L'orientation de la solution peut être réalisée de deux manières différentes. A la fin d'une exécution de l'algorithme du recuit, l'utilisateur peut changer les valeurs des préférences des bus réels et virtuels avant de relancer l'algorithme, ou d'une manière *interactive* il peut essayer de déplacer les étudiants d'un bus vers un autre (*drag & drop principle*).

La première condition correspond à un niveau d'interactivité minimale, ce qui revient à relancer l'algorithme tant que l'optimum n'est pas suffisamment approché. La condition d'arrêt, dans ce cas, se ramène à comparer la valeur finale de la fonction de coût d'une exécution avec la valeur finale de l'exécution précédente, ce qui présuppose au moins deux exécutions de l'algorithme. Le relancement de l'algorithme s'arrêtera quand la valeur finale de la fonction de coût (*energy*) sera égale à la valeur finale de la fonction de coût de l'exécution précédente. Cette condition est très stricte et peut provoquer, pour un problème de grande taille, un nombre important d'exécutions consécutives de l'algorithme. La qualité du résultat, quand cette condition d'arrêt est choisie, est nettement supérieure à celle des autres conditions.

La deuxième condition d'arrêt vise les utilisateurs plus expérimentés, autrement dit des utilisateurs qui désirent agir d'une manière interactive sur un résultat de qualité moyenne pour orienter la solution vers une solution proche de l'optimalité. Dans ce cas la condition d'arrêt revient à comparer le nombre de bus (réels et virtuels) d'une exécution avec celui de l'exécution précédente. De nouveau cette condition présuppose au moins deux exécutions de l'algorithme. Nous pouvons facilement voir que cette condition est moins stricte que la précédente. Pour mieux le comprendre, nous allons étudier un exemple.

Soit la répartition des étudiants suivante :

Solution 1	**Bus Volvo**	**Bus Mercedes**	**Bus DAF**
Capacité	5 places	3 places	3 places
Préférence	Absolute	Medium	Medium
	Christos	Marc	Damien
	Fred		Nick

Supposons que le relancement du recuit à partir de la solution 1, donne comme résultat la solution suivante :

Solution 2	**Bus Volvo**	**Bus Mercedes**	**Bus DAF**
Capacité	5 places	3 places	3 places
Préférence	Absolute	Medium	Medium
	Christos	Marc	Damien
	Fred		
	Nick		

La valeur finale de la fonction de coût de la deuxième exécution est plus élevée que celle de la première car l'étudiant *Nick* se trouvant initialement dans un bus de préférence *Medium*, a été déplacé vers un bus de préférence *Absolute*.

Si nous considérons la première condition d'arrêt, l'algorithme sera relancé une nouvelle fois car la valeur finale de la fonction de coût de la deuxième exécution est plus élevée que celle de la première.

Au contraire, si nous considérons la deuxième condition d'arrêt, l'algorithme ne sera plus relancé car le nombre total de bus est resté constant !

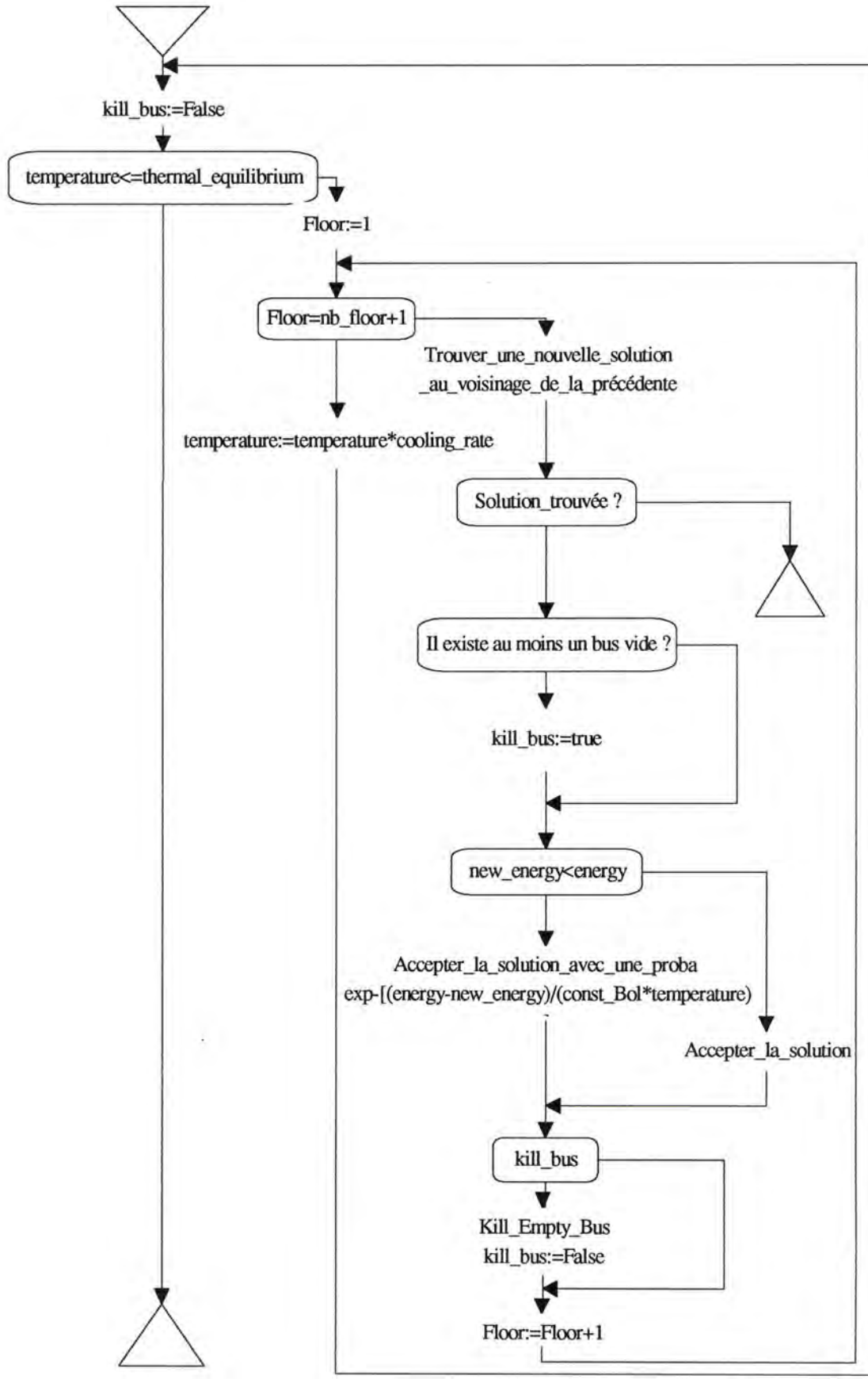
La troisième condition d'arrêt s'adresse à des utilisateurs qui désirent orienter la solution très tôt. La condition d'arrêt dans ce cas revient à comparer à la fin d'une exécution, si le nombre de bus (réels et virtuels) est plus petit ou égal à deux fois le nombre de bus réels! Nous avons considéré qu'une bonne solution de départ, pour l'utilisateur qui veut orienter l'évolution des solutions, est telle qu'elle ne comporte ni trop ni trop peu de bus virtuels.

Remarquons que cette condition ne présuppose pas au moins deux exécutions de l'algorithme.

3.5 Simulated Annealing

Dans ce paragraphe nous allons décrire, d'une manière générale et sans entrer dans des détails techniques, l'algorithme du recuit simulé que nous avons développé. Cet algorithme général que nous allons présenter ici est applicable aussi bien au problème simplifié (c.f. 1.3.1) qu'au problème initial.

Pour mieux comprendre les différents principes de l'algorithme, nous allons d'abord présenter l'organigramme de celui-ci et ensuite expliquer ses particularités!


kill_bus:=False
temperature<=thermal_equilibrium
Floor:=1
Floor=nb_floor+1
Trouver_une_nouvelle_solution
_au_voisinage_de_la_précédente
temperature:=temperature*cooling_rate
Solution_trouvée ?
Il existe au moins un bus vide ?
kill_bus:=true
new_energy<energy
Accepter_la_solution_avec_une_proba
exp-[(energy-new_energy)/(const_Bol*temperature)
Accepter_la_solution
kill_bus
Kill_Empty_Bus
kill_bus:=False
Floor:=Floor+1

3.5.1 Sémantique des variables utilisées

temperature	Le paramètre de contrôle du recuit simulé. Il représente la *température* du système.
thermal_equilibrium	La valeur finale du paramètre de contrôle qui correspond à l'équilibre thermique du système.
nb_floor	La longueur des chaînes de Markov pour un palier de température. Ce nombre est identique pour tous les paliers de température
cooling_rate	Le facteur correspondant au taux de diminution de la température, une fois que la chaîne de Markov pour un palier de température a été générée.
energy	La valeur de la fonction de coût *avant* la perturbation élémentaire.
new_energy	La valeur de la fonction de coût *après* la perturbation élémentaire.
Floor	Compteur de la longueur de la chaîne de Markov générée.
kill_bus	*Flag* signalant s'il existe au moins un bus vide.

3.5.2 Les procédures

Trouver une solution au voisinage de la précédente

Dans cette procédure nous trouvons le mécanisme de génération d'une nouvelle solution au voisinage de la précédente. Nous allons décrire ci-dessous ce mécanisme pour le modèle simplifié. Ensuite, nous généraliserons le mécanisme pour qu'il soit applicable au problème initial.

Le principe de ce mécanisme est relativement facile à imaginer. Nous disposons d'une solution actuelle (*solution courante*) c.à.d. d'une répartition des étudiants dans les différents bus. Nous choisissons un étudiant **"etu"** et un bus **"b"** aléatoirement. Si ce bus n'est pas totalement rempli (il existe au moins une place libre dans ce bus), et si la contrainte principale (c.f. 1.2.4) est satisfaite pour chaque couple (et,et_i) $\forall$ I $\in$ b,

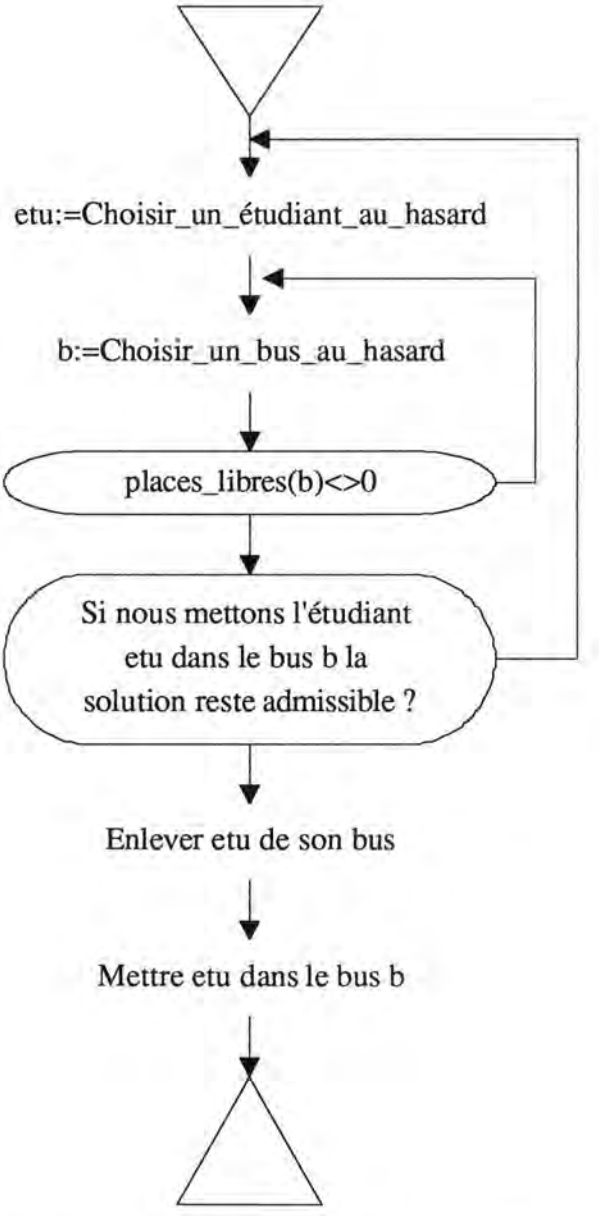
alors nous avons créé une nouvelle solution en appliquant une petite perturbation à la solution courante. Dans le cas où il existe une incompatibilité au niveau de la contrainte principale, ce processus est répété tant qu'une solution admissible n'est pas trouvée.

Remarquons que ce procédé peut *boucler* pour deux raisons!

1. Il n'existe aucune place libre dans aucun bus.
2. Aucune solution admissible ne peut être générée à partir de la solution courante.

A la fin de ce paragraphe nous allons transformer l'algorithme pour faire disparaître ces défauts.

En mettant le procédé présenté ci-dessus sous forme d'organigramme, nous obtenons :



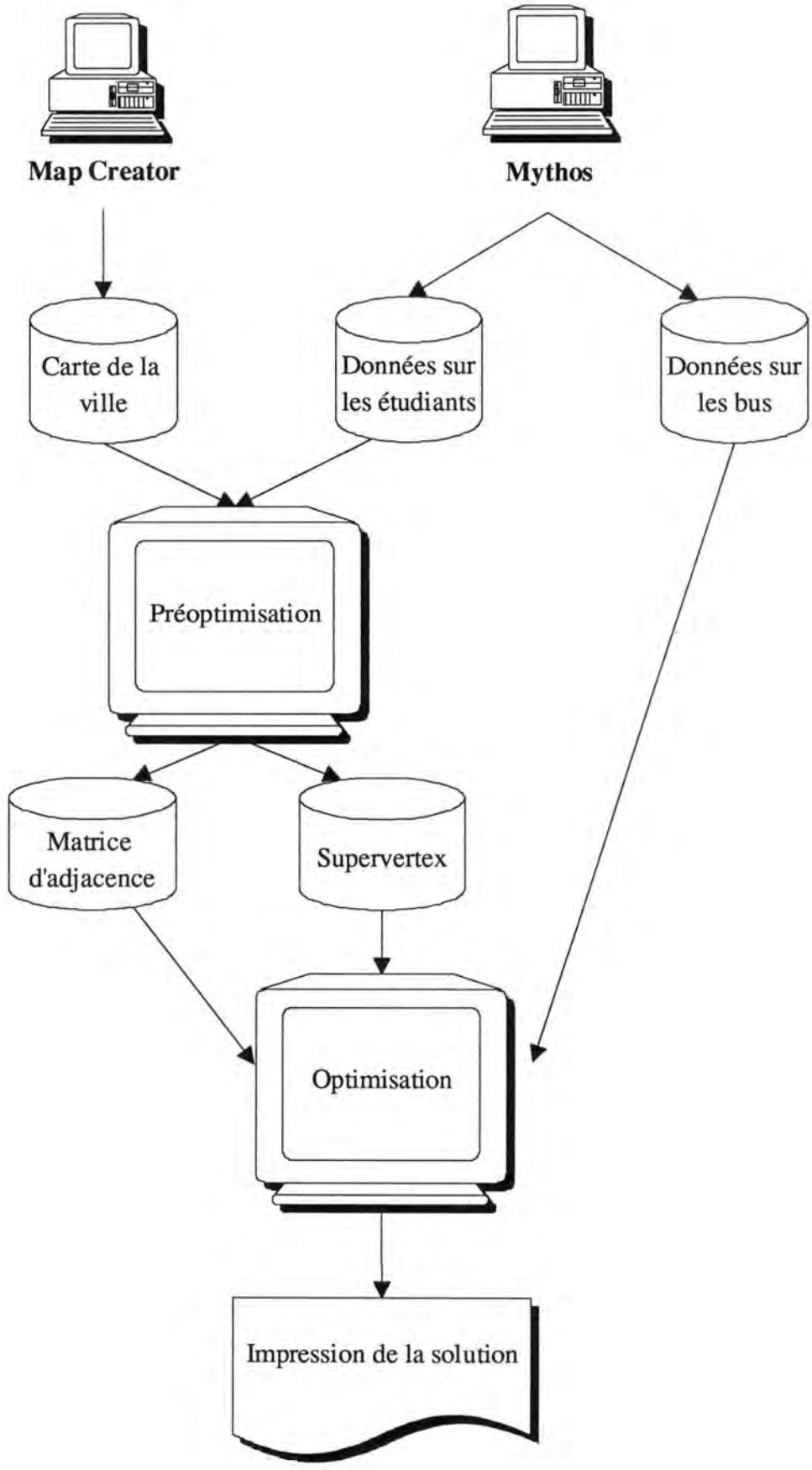
Nous venons de construire un mécanisme pour générer une solution au voisinage d'une précédente pour le problème simplifié. Afin qu'il soit applicable pour le modèle initial, nous le transformerons de la manière suivante:

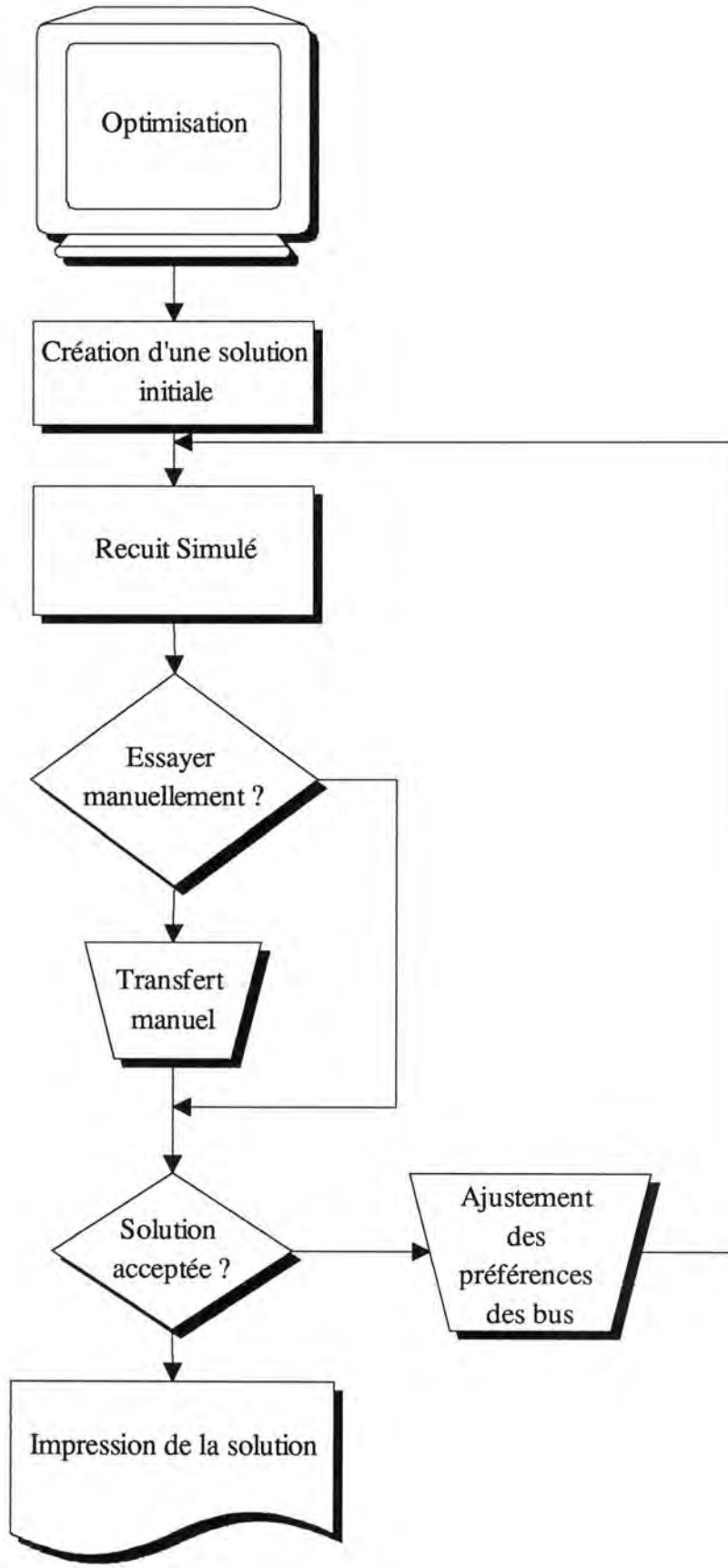
1. Choisir arbitrairement un noeud "n" dans l'ensemble $\mathbf{E}^*$ des noeuds du graphe étendu **G**.
2. Si ce noeud est coloré avec une couleur c>0, choisir arbitrairement une couleur "d" dans [0, nb_bus].

   Si d≠0, colorer ce noeud, si ceci est possible (c.f. contraintes). S'il est impossible d'obtenir une solution admissible en colorant le noeud "n" avec la couleur "d", choisir arbitrairement une nouvelle couleur "d" dans [0, nb_bus]/{d} et réessayer.

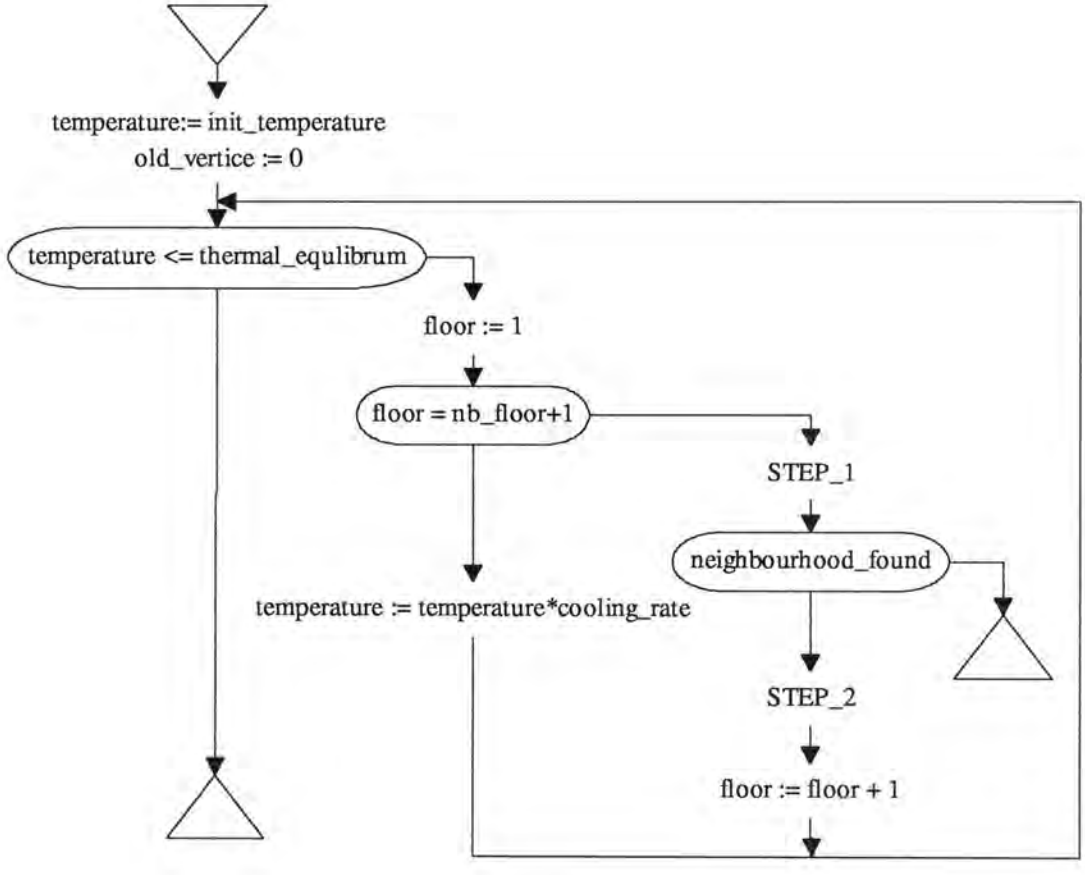
   Si d=0, trouver arbitrairement un autre noeud dans le même supervertex qui peut être coloré avec une couleur non nulle (en appliquant le même principe). Si le procédé réussit, attribuer au noeud "n" la couleur 0.
3. Si le noeud "n" est coloré avec la couleur 0, choisir arbitrairement une couleur dans [1, nb_bus] et essayer de colorer ce noeud avec cette nouvelle couleur. Si le procédé réussit, trouver le noeud de ce supervertex qui est coloré avec une couleur non nulle et le colorer avec la couleur 0.

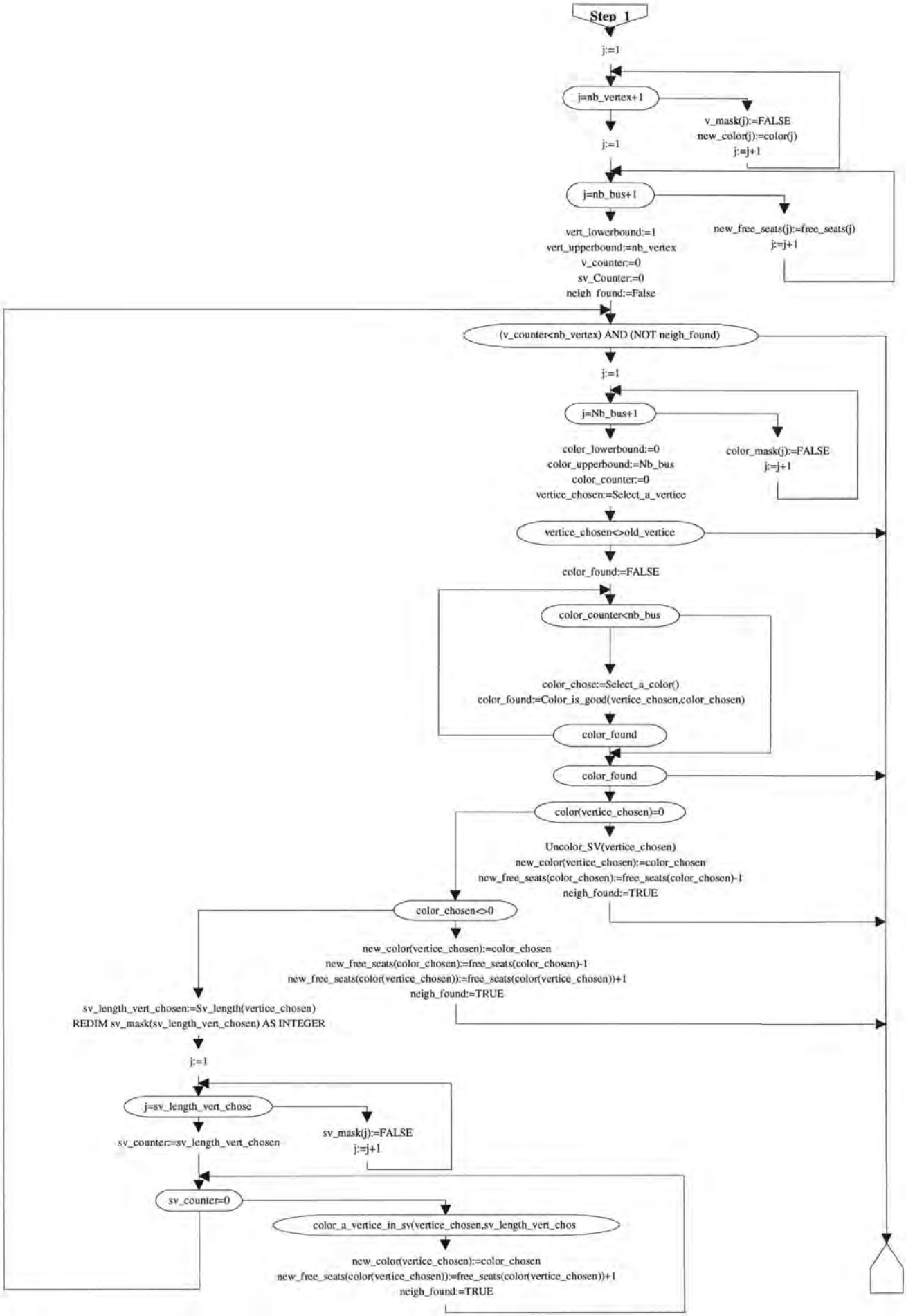
Avant d'étudier en détail notre algorithme final, schématisons les grands principes présentés dans ce chapitre.

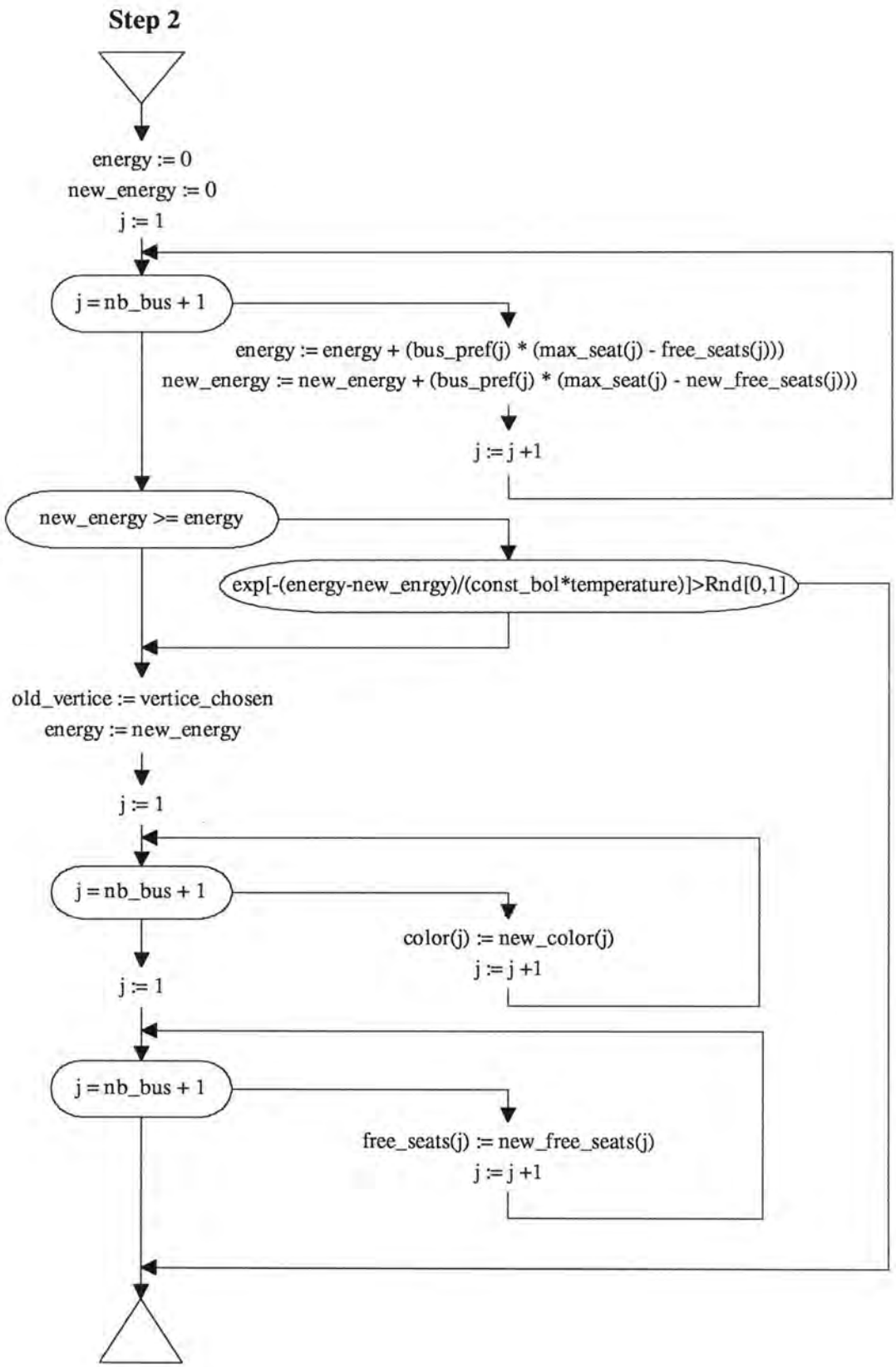

Map Creator
Mythos
Carte de la ville
Données sur les étudiants
Données sur les bus
Préoptimisation
Matrice d'adjacence
Supervertex
Optimisation
Impression de la solution


Optimisation
Création d'une solution initiale
Recuit Simulé
Essayer manuellement ?
Transfert manuel
Solution acceptée ?
Ajustement des préférences des bus
Impression de la solution

où
- Map Creator est le premier logiciel développé, pour la création de la carte d'une ville, et est présenté dans le paragraphe 4.1.1;
- Mythos est le logiciel principal, comprenant les modules d'encodage des étudiants, des bus, du calcul du plus court chemin, du recuit simulé et du générateur des résultats.
- Le module de préoptimisation de Mythos correspond au calcul du plus court chemin entre chaque paire d'étudiants.
- Le module d'optimisation de Mythos correspond au module du recuit simulé et du transfert des étudiants d'un bus vers un autre (partie interactive).




Step 1
j:=1
j=nb_vertex+1
v_mask(j):=FALSE
new_color(j):=color(j)
j:=j+1
j:=1
j=nb_bus+1
new_free_seats(j):=free_seats(j)
j:=j+1
vert_lowerbound:=1
vert_upperbound:=nb_vertex
v_counter:=0
sv_Counter:=0
neigh_found:=False
(v_counter<nb_vertex) AND (NOT neigh_found)
j:=1
j=Nb_bus+1
color_mask(j):=FALSE
j:=j+1
color_lowerbound:=0
color_upperbound:=Nb_bus
color_counter:=0
vertice_chosen:=Select_a_vertice
vertice_chosen<>old_vertice
color_found:=FALSE
color_counter<nb_bus
color_chose:=Select_a_color()
color_found:=Color_is_good(vertice_chosen,color_chosen)
color_found
color_found
color(vertice_chosen)=0
Uncolor_SV(vertice_chosen)
new_color(vertice_chosen):=color_chosen
new_free_seats(color_chosen):=free_seats(color_chosen)-1
neigh_found:=TRUE
color_chosen<>0
new_color(vertice_chosen):=color_chosen
new_free_seats(color_chosen):=free_seats(color_chosen)-1
new_free_seats(color(vertice_chosen)):=free_seats(color(vertice_chosen))+1
neigh_found:=TRUE
sv_length_vert_chosen:=Sv_length(vertice_chosen)
REDIM sv_mask(sv_length_vert_chosen) AS INTEGER
j:=1
j=sv_length_vert_chose
sv_mask(j):=FALSE
j:=j+1
sv_counter:=sv_length_vert_chosen
sv_counter=0
color_a_vertice_in_sv(vertice_chosen,sv_length_vert_chos
new_color(vertice_chosen):=color_chosen
new_free_seats(color(vertice_chosen)):=free_seats(color(vertice_chosen))+1
neigh_found:=TRUE


Step 2
energy := 0
new_energy := 0
j := 1
j = nb_bus + 1
energy := energy + (bus_pref(j) * (max_seat(j) - free_seats(j)))
new_energy := new_energy + (bus_pref(j) * (max_seat(j) - new_free_seats(j)))
j := j +1
new_energy >= energy
exp[-(energy-new_enrgy)/(const_bol*temperature)]>Rnd[0,1]
old_vertice := vertice_chosen
energy := new_energy
j := 1
j = nb_bus + 1
color(j) := new_color(j)
j := j +1
j := 1
j = nb_bus + 1
free_seats(j) := new_free_seats(j)
j := j +1

Function Color_a_vertice_in_SV (ByVal vertice As Integer, ByVal sv_length As Integer) As Integer

Spécifications : Cette fonction renvoie la valeur TRUE (0) si elle réussit à colorer UN sommet choisi au hasard parmi ceux qui appartiennent au même supervertex que ***vertice*** *(la longueur du supervertex étant* ***sv_length****), elle renvoie la valeur FALSE sinon. Le compteur* ***sv_counter*** *est décrémenté dans les deux cas.*

```
    Dim g, v As Integer
    Dim SV_lowerbound, SV_upperbound As Integer
    Dim color_lowerbound, color_upperbound, C As Integer
    ReDim Col_Mask(1 To nb_bus)
    Dim Col_Counter As Integer

    For g = 1 To nb_bus
      Col_Mask(g) = False
    Next g

    'Choose a vertice in the SV
    SV_lowerbound = 1
    SV_upperbound = sv_length - 1
    Color_a_vertice_in_SV = False
    Do
      g = Int((SV_upperbound - SV_lowerbound + 1) * Rnd +
              SV_lowerbound)
      v = vertice
      Do
          v = SuperVertex(v)
          g = g - 1
      Loop Until g = 0
    Loop Until SV_Mask(v) = False

    'chose a color
    color_lowerbound = 1
    color_upperbound = nb_bus
   Col_Counter = 0
   Do While Col_Counter < nb_bus
      Do
          C = Int((color_upperbound - color_lowerbound + 1) * Rnd +
                  color_lowerbound)
      Loop Until Col_Mask(C) = False

      Col_Mask(C) = True
      Col_Counter = Col_Counter + 1
      If Color_is_good(v, C) Then
          Color_a_vertice_in_SV = True
          New_Color(v) = C
          new_free_seats(C) = free_seats(C) - 1
          Exit Do
      End If
    Loop

    SV_Counter = SV_Counter - 1

End Function
```

Function Color_is_good (ByVal vert As Integer, ByVal Col As Integer) As Integer

Spécifications : Cette fonction renvoie la valeur TRUE si le sommet ***vert*** *peut être coloré avec la couleur* ***col****, et FALSE sinon. Le sommet* ***vert*** *peut être coloré avec la couleur* ***col*** *SSI*

- ***col*** *est différente de la couleur actuelle de* ***vert***
- *le bus correspondant à la couleur* ***col*** *n'est pas rempli*
- *parmi tous les sommets adjacents au sommet* ***vert****, il n'en existe aucun qui soit coloré avec la couleur* ***col***

```
Dim j As Integer

    Color_is_good = True
    If (Col = Color(vert) Or (free_seats(Col) = 0)) Then
      Color_is_good = False
    Else
      If Col <> 0 Then
          For j = 1 To nb_vertex
            If Col = Color(j) Then
                If Edge(vert, j) = 1 Then
                  Color_is_good = False
                  Exit For
                End If
            End If
          Next j
      End If
    End If
```

End Function

Static Function Select_a_color () As Integer

Spécifications : Cette fonction renvoie une valeur i comprise entre les bornes color_lowerbound et color_upperbound, telle que la valeur du masque Color_Mask(i) est FALSE (i.e. cette couleur n'a pas encore été essayée). Le masque Color_Mask(i) ainsi que la valeur du compteur Color_Counter sont mis à jour.

```
    Dim C  As Integer

     Do
         C = Int((color_upperbound - color_lowerbound + 1) * Rnd +
                                     color_lowerbound)
     Loop Until color_lowerbound = False

     Select_a_color = C
     Color_Mask(C) = True
     Color_Counter = Color_Counter + 1
```

End Function

Static Function Select_a_vertice () As Integer

Spécifications : Cette fonction renvoie une valeur i comprise entre les bornes vert _lowerbound et vert _upperbound, telle que la valeur du masque V_Mask(i) est FALSE (i.e. ce sommet n'a pas encore été essayé). Le masque V_Mask(i) ainsi que la valeur du compteur V_Counter sont mis à jour.

```
Dim s As Integer

Do
  s = Int((vert_upperbound - vert_lowerbound + 1) * Rnd +
                    vert_lowerbound)
Loop Until V_Mask(s) = False

Select_a_vertice = s
V_Mask(s) = True
V_Counter = V_Counter + 1
```

End Function

Function sv_length (ByVal vertice As Integer) As Integer

Spécifications : Cette fonction renvoie le nombre d'éléments du supervertex qui contient le sommet ***vertice****.*

```
Dim j, k As Integer

j = 0
k = vertice

Do
  k = SuperVertex(k)
  j = j + 1
Loop Until k = vertice

sv_length = j
```

End Function

```
Static Sub uncolor_SV (ByVal vertice As Integer)
```

Spécifications : Cette procédure donne la couleur 0 au sommet coloré (i.e. de couleur ≠ 0) du supervertex contenant le sommet ***vertice****, et met à jour la variable* ***new_free_seats***

```
  Dim j, k As Integer

    k = vertice

    For j = 1 To sv_length(vertice)
      If Color(k) <> 0 Then
          new_free_seats(Color(k)) = free_seats(Color(k)) + 1
          New_Color(k) = 0
      End If
      k = SuperVertex(k)
    Next j

End Sub
```

Kill Empty Bus

Cette procédure est exécutée une fois qu'au moins un bus s'est vidé. Il faudra traiter d'une manière différente le fait que le bus vide est un bus réel ou un bus virtuel.

Nous distinguons les cas suivants :

- Si le bus vide est un bus virtuel, alors il est supprimé de l'ensemble des bus disponibles.
- S'il n'existe aucun bus virtuel et que le bus vide est un bus réel, alors le bus réel est supprimé de l'ensemble des bus disponibles.

Remarque : Tant que la solution possède des bus virtuels, la procédure ne supprimera jamais de bus réels.

```
Static Sub KillEmptyBus (ByVal witch As Integer)
```

*Spécifications : Cette procédure trie les éléments des tableaux globaux **free_seats, max_seats, bus_name, bus_pref** de telle sorte que les éléments dont la valeur **free_seats** est nulle se trouvent en fin de tri.*

```
Dim g, d, s As Integer

If witch = virtual Then 'Only the virtual busses
    g = bus_maxrec
Else  'Virtual busses and real one !
    g = 1
End If

d = nb_bus

Do While Not (g >= d + 1)

    If free_seats(d) = max_seats(d) Then
      'the bus d is empty
      d = d - 1
    Else
      'the bus d is not empty
      If (free_seats(g) <> max_seats(g)) Then
          'the bus g is not empty
          g = g + 1
      Else
      'the bus g is empty
          Call permute(free_seats(g), free_seats(d))
          Call permute(max_seats(g), max_seats(d))
          Call permute(bus_name(g), bus_name(d))
          Call permute(bus_pref(g), bus_pref(d))
          For s = 1 To nb_vertex
            If Color(s) = d Then
                Color(s) = g
            End If
          Next s
          d = d - 1
          g = g + 1
```

```
      End If
    End If
Loop

nb_bus = d

End Sub
```

Preuve formelle de la correction de l'algorithme :

$$\{Pré\} \equiv \left\{ \begin{array}{l} nb\_bus \geq 1 \\ \forall\, i \in IN,\ 1 \leq i \leq nb\_bus : free\_seats[i] \geq 0 \end{array} \right\}$$

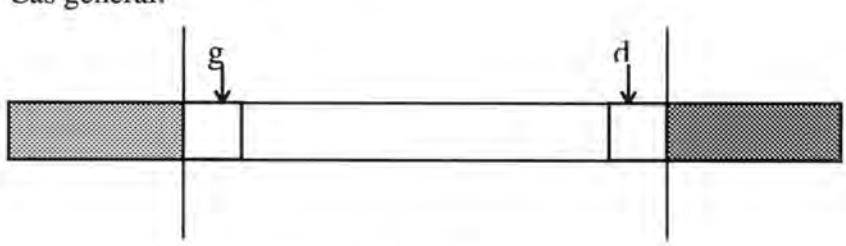
$$\{Post\} \equiv \left\{ \begin{array}{ll} & 1 \leq k \leq nb\_bus + 1 \\ \exists\, k \in IN \text{ tq} & \forall\, i \in IN,\ 1 \leq i \leq k-1 : free\_seats[i] > 0 \\ & \forall\, j \in IN,\ k \leq j \leq nb\_bus : free\_seats[i] = 0 \end{array} \right\}$$

$$\{IA\} \equiv \left\{ \begin{array}{ll} & g \leq d + 1 \\ & 1 \leq g \leq nb\_bus + 1 \\ \exists\, g,d \in IN \text{ tq} & 1 \leq d \leq nb\_bus + 1 \\ & \forall\, i \in IN,\ 1 \leq i \leq k-1 : free\_seats[i] > 0 \\ & \forall\, j \in IN,\ k \leq j \leq nb\_bus : free\_seats[i] = 0 \end{array} \right\}$$

$$\{B\} \equiv \{g = d + 1\}$$

$$\{Init\} \equiv \{g := 1;\ d := nb\_bus\}$$

Cas général:




remplis de zéros


remplis de non nuls


???

{Pré} ∧ {Init} ⇒ {I_A}

$$I_A \equiv \left\{ \exists\, g = 1, d = nb\_bus \in \mathbf{IN} \text{ tq } \begin{array}{l} 1 \le nb\_bus + 1 \Rightarrow g \le d + 1 \\ 1 \le 1 \le nb\_bus + 1 \Rightarrow 1 \le g \le nb\_bus + 1 \\ 1 \le nb\_bus \le nb\_bus + 1 \Rightarrow 1 \le d \le nb\_bus + 1 \\ \forall\, i \in \mathbf{IN},\ 1 \le i \le 0 : free\_seats[i] > 0 \\ \forall\, j \in \mathbf{IN},\ nb\_bus + 1 \le j \le nb\_bus : free\_seats[i] = 0 \end{array} \right\}$$

{IA} ∧ {B} ⇒ {Post}

$$\{I_A\} \wedge \{B\} \Rightarrow \left\{ \exists\, g, d \in \mathbf{IN} \text{ tq } \begin{array}{l} g = d + 1 \\ 1 \le g \le nb\_bus + 1 \\ 1 \le d \le nb\_bus + 1 \\ \forall\, i \in \mathbf{IN},\ 1 \le i \le d : free\_seats[i] > 0 \\ \forall\, j \in \mathbf{IN},\ d + 1 \le j \le nb\_bus : free\_seats[i] = 0 \end{array} \right\} \Rightarrow \{Post\}$$

{I_A} ∧ {¬B} ⇒ {I_A}

Cas 1 : free_seats[d] = 0

{Iter}≡{d:=d-1}

Cas 2 : free_seats[d] ≠ 0 et free_seats[g] ≠ 0

{Iter}≡{g:=g+1}

Cas 3 : free_seats[d] ≠ 0 et free_seats[g] = 0

{Iter}≡{permuter(g,d); d:=d-1; g:=g+1}

4. Compactage de la matrice d'adjacence

Rappelons que la matrice d'adjacence est une matrice carrée, symétrique, d'ordre *nb_vertex*. Les éléments de cette matrice sont des booléens.

Pour avoir une idée de la taille réelle d'une telle matrice, considérons un problème de taille moyenne.

Soit 70 étudiants avec, en moyenne, une longueur d'intervalle de l'ordre de 15 minutes. La précision du résultat est de l'ordre de la minute.

Par conséquent, le nombre total des noeuds dans les 70 supervertex créés est 70 x 15 = 1050 (noeuds).

La matrice d'adjacence a donc 1050 x 1050 = 1.102.500 éléments !

Supposons ensuite que le langage que nous allons utiliser pour implémenter cette matrice supporte directement le type de données "*bit*" (ce qui n'est pas le cas de Visual Basic). Calculons maintenant la place mémoire occupée par une telle matrice.

1.102.500 éléments x 1 bit = 1.102.500 bits ⇒

1.102.500 / 8 = 137.812,5 bytes =

137.812,5 / 1024 =

134,5 Kbytes !

En Visual Basic, le type de données le plus économique (en occupation place mémoire) est le type entier (*Integer*) codé sur 16 bits (2 bytes) ! Une variable booléenne en Visual Basic occupe donc 16 bits.

Calculons, pour le même problème, la taille de la matrice si elle est implémentée en Visual Basic :

1.102.500 éléments x 16 bits =

17.640.000 bits =

2.205.000 bytes =

2.153 Kbytes =

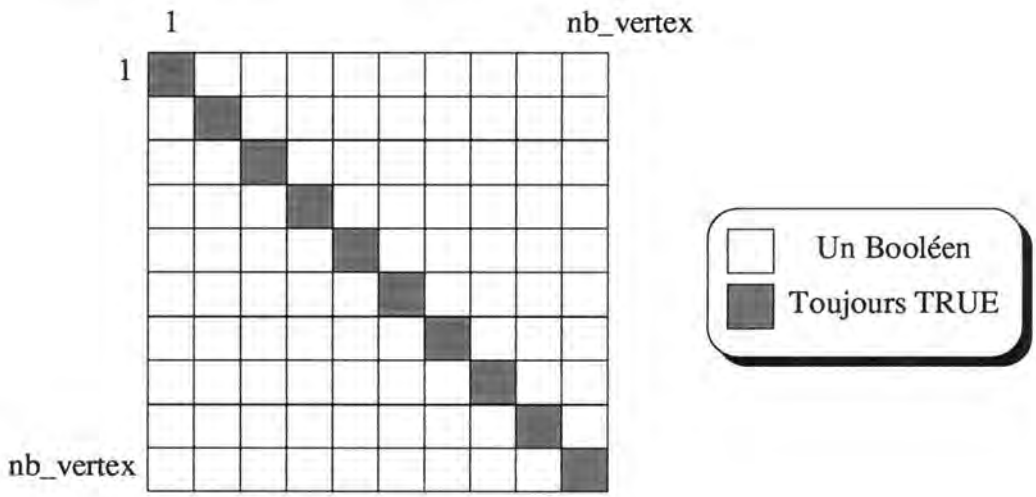
2,15 MegaBytes !

Rappelons que le système Dos/Windows 3.1 est fait pour tourner sur des ordinateurs personnels avec une mémoire RAM allant de 2 à 8 MegaBytes. Même sur

un système de 8 MegaBytes de RAM, il serait inacceptable de ne pas optimiser une telle structure de données. En plus, si la matrice d'adjacence occupe autant de place mémoire, le système devra très souvent utiliser le mécanisme de mémoire virtuelle pour la représenter. L'algorithme de recuit simulé devra accéder des milliers de fois aux éléments de cette matrice. Nous laisserons au lecteur le soin d'imaginer la perte au niveau des performances de l'algorithme, à cause des "*page fault*" fréquents!

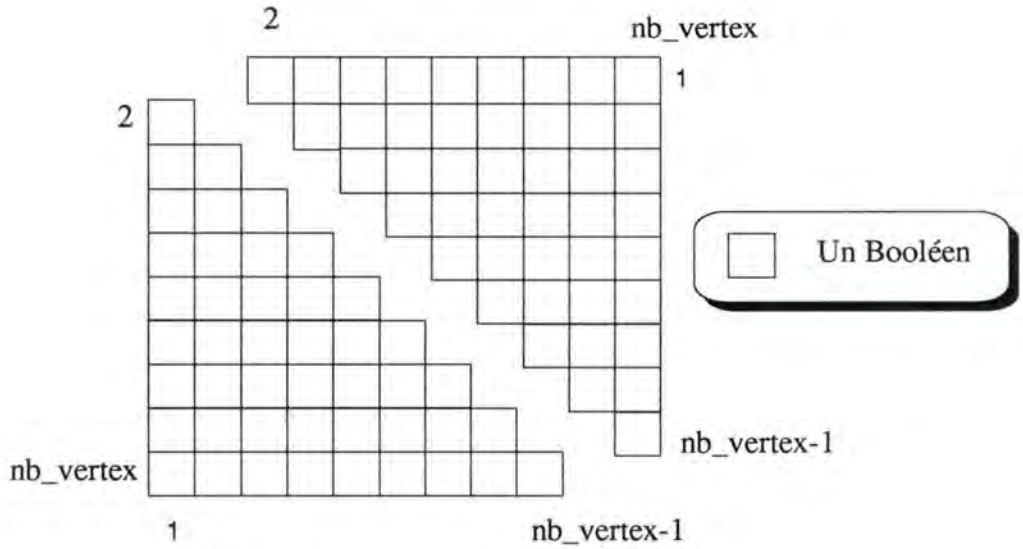
Un compactage de cette matrice est donc indispensable pour pouvoir travailler avec des problèmes de tailles raisonnables.

Dans la figure suivante nous avons représenté la matrice d'adjacence comme elle a été définie dans le paragraphe 1.3.4.



La diagonale de cette matrice est composée par des éléments dont la valeur est toujours égale à TRUE, car un étudiant à une heure précise peut toujours se trouver dans le même bus que lui-même à la même heure! La matrice suivante est donc équivalente à celle qui précède, en sachant que :

$$\forall\, i \in [1, \text{nb\_vertex}] \Rightarrow Adj_{ii} = \text{TRUE}$$

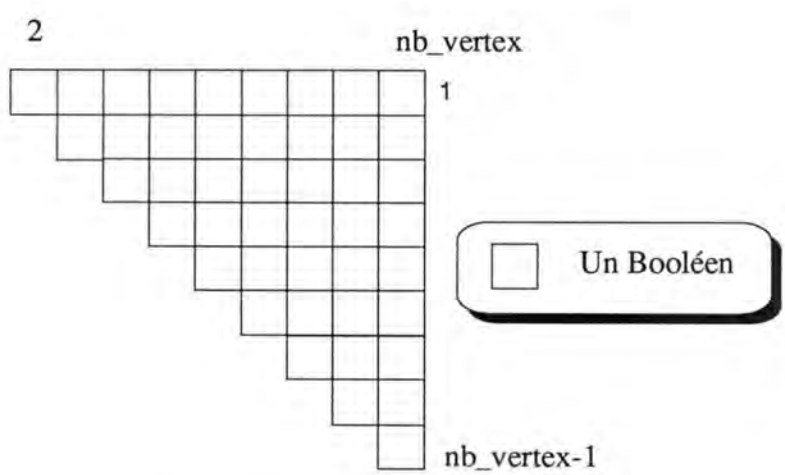


Au début de ce paragraphe nous avons remarqué que la matrice d'adjacence est une matrice symétrique i.e. $\forall$ i,j $\in$ [1, nb_vertex] $\Rightarrow$ $Adj_{ij} = Adj_{ji}$. Il est donc trivial que nous n'avons pas besoin de conserver les deux matrices triangulaires représentées par la figure précédente, mais seulement une de deux car ces deux matrices triangulaires sont redondantes. La matrice triangulaire, représentée par la figure suivante contient autant d'informations que la matrice d'adjacence initiale, en sachant que :

$$\forall i \in [1, nb\_vertex] \Rightarrow Adj_{ii} = TRUE$$

et

$$\forall i,j \in [1, nb\_vertex] \Rightarrow Adj_{ij} = Adj_{ji}.$$



Le nombre d'éléments de cette matrice est égal à :

$$\sum_{i=1}^{nb\_vertex-1} i = (nb\_vertex - 1)\frac{1+(nb\_vertex-1)}{2}.$$

Pour notre problème nous aurons donc une matrice triangulaire de 550.725 éléments. Nous avons dit précédemment que chaque élément de cette matrice est codé sur 16 bits, ce qui donne :

550.725 éléments x 16 bits =

8.811.600 bits =

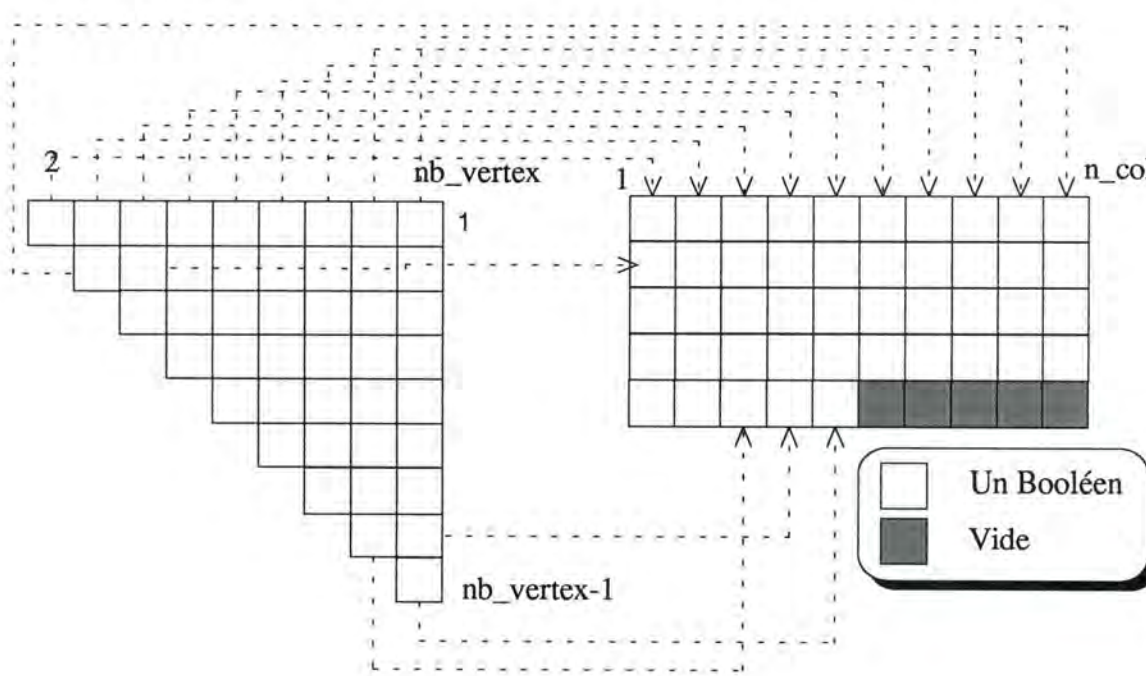
1.101.450 bytes =

1.075,6 Kbytes ≈

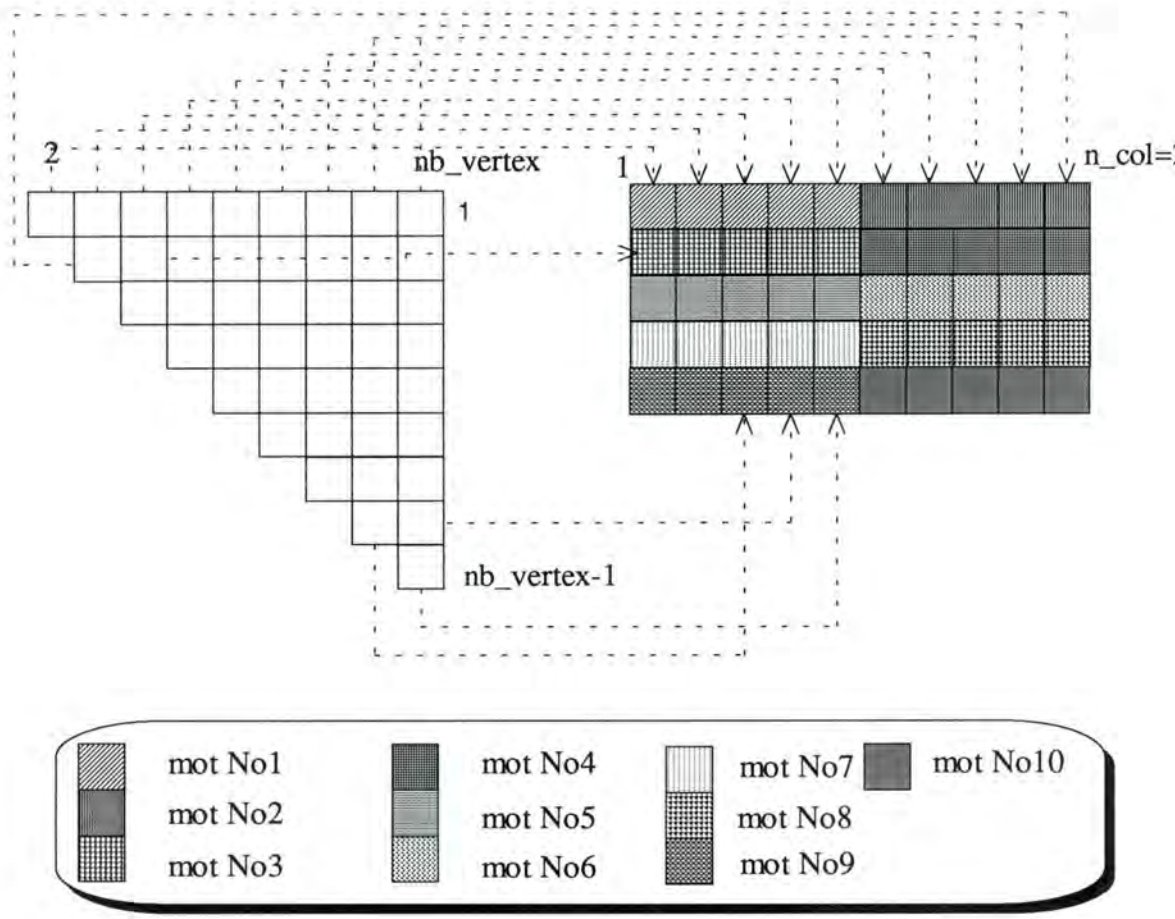
1 MegaByte !

Nous pouvons remarquer que, malgré les améliorations apportées, la taille de la matrice reste très grande. L'inconvénient principal est que nous occupons 16 bits pour une information qui pourrait être codée sur 1 bit.

Nous avons décidé d'implémenter une structure de données permettant la représentation d'une matrice triangulaire dont les éléments sont des booléens codés sur 1 bit. A cette fin, nous utilisons un tableau normal de *n_col* colonnes, où *n_col* est une constante à déterminer. Appelons *Virt_Matrix* (virtual matrix) ce tableau. Le nombre de lignes de *Virt_Matrix* sera variable, en fonction de la taille du problème traité. Pour expliquer comment les données sont représentées dans *Virt_Matrix,* supposons que chaque élément de ce tableau est de type "bit". La figure suivante schématise la manière dont la matrice triangulaire a été représentée :



Nous avons considéré que chaque élément de la matrice est du type *bit*. Comme ce type de donnée n'est pas directement implémentable en Visual Basic, nous avons implémenté un mécanisme d'accès aux bits d'un entier (type Integer ou Long). Au lieu de considérer un tableau dont les éléments sont de type *bit*, nous considérerons donc un tableau dont les éléments sont des entiers, mais que nous avons accès à chaque bit de ceux ci. Supposons que ces entiers sont codés sur 4 bits. La figure suivante représente un schéma équivalent au précédent, où les éléments du tableau sont de type entier (codés sur 4 bits).



Dans la pratique nous avons utilisé des entiers codés sur 32 bits (Long Integer). Le premier bit d'un entier correspond au signe de l'entier ('+' ou '-'). Nous n'avons utilisé que les 31 bits restants (variable *size*). La constante *n_col* a été fixée à 20. La fonction suivante calcule pour l'élément (i,j) de la matrice triangulaire, son correspondant dans la matrice finale (compactée).

```
Function calc (ByVal i As Integer, ByVal j As Integer, ByVal m_dim As
Integer, ByVal size As Integer, ByVal n_col As Integer, lign As Long,
Col As Integer) As Integer
```

Spécifications: i,j représentent l'élément (i,j) de la matrice triangulaire. m_dim représente le nombre des supervertex (l'ordre de la matrice d'adjacence). La constante size est fixée à 31 et représente les bits utiles dans chaque entier. La constante n_col est fixée à 20 et représente le nombre d'entiers par ligne dans la matrice virt_matrix. Cette fonction a comme objectif de donner la position (lign,col) de l'entier contenant le bit correspondant à l'élément logique (i,j) de la matrice triangulaire et la valeur de la fonction représente le numéro du bit correspondant dans cet entier.

```
Dim wor, inte As Long
 Dim bit As Integer
 Dim r As Double

 'inte:=trunc((i-1)*m_dim+j-((i+1)/2)*i)
  r = (m_dim - (i + 1) / 2)
  r = r * (i - 1)
  r = r + j - ((i + 1) / 2)
  inte = Int(r)
  wor = (inte \ size) + 1
  bit = inte Mod size

  If bit = 0 Then
   bit = size
   wor = wor - 1
  End If

  lign = (wor \ n_col) + 1
  Col = wor Mod n_col

  If Col = 0 Then
   Col = n_col
   lign = lign - 1
  End If
  calc = bit
```

End Function

Nous allons présenter ci-dessous les primitives implémentées pour l'accès à la matrice (logique) d'adjacence.

Static Function Edge (ByVal i As Integer, ByVal j As Integer) As Integer

Spécifications: i,j représentent l'élément (i,j) de la matrice d'adjacence (initiale). Cette fonction rend la valeur True s'il existe un arc entre l'élément (i,j) et la valeur False sinon.

```
 'Indicates if there is an edge from the vertex i
 'to the vertex j in the adjacence matrix of the graph

 Dim lign As Long
 Dim Col As Integer
 Dim bit As Integer

 If Not (i = j) Then
   If i > j Then
      Call permute(i, j)
   End If
   bit = calc(i, j, nb_vertex, size, n_col, lign, Col)
```

```
    If ith_bit_is_0(virt_matrix(lign, Col), bit) Then
        Edge = 0
    Else
        Edge = 1
    End If
  Else
    Edge = 0
  End If

End Function
```

```
Sub Remove_Edge (ByVal i As Integer, ByVal j As Integer)
```

Spécifications: i,j représentent l'élément (i,j) de la matrice d'adjacence (initiale). Cette procédure enlève l'arête entre les éléments i,j (s'il en existe une).

```
  'Remove the edge from the vertex i to the vertex j
  'in the adjacence matrix of the graph

  Dim lign As Long
  Dim Col As Integer
  Dim bit As Integer

  If Not (i = j) Then
    If i > j Then
        Call permute(i, j)
    End If
    bit = calc(i, j, nb_vertex, size, n_col, lign, Col)
    Call Set_ith_bit_to(virt_matrix(lign, Col), bit, 0)
  End If

End Sub
```

```
Sub Set_Edge (ByVal i As Integer, ByVal j As Integer)
```

Spécifications: i,j représentent l'élément (i,j) de la matrice d'adjacence (initiale). Cette procédure ajoute une arête entre les éléments i,j.

```
  'Set an edge from the vertex i to the vertex j
  'in the adjacence matrix of the graph

  Dim lign As Long
  Dim bit As Integer
  Dim Col As Integer

  If i <> j Then
    If i > j Then
        Call permute(i, j)
    End If
    bit = calc(i, j, nb_vertex, size, n_col, lign, Col)
    Call Set_ith_bit_to(virt_matrix(lign, Col), bit, 1)
  End If
End Sub
```

La librairie DLL Pascal suivante offre les deux fonctions de bas niveau pour l'accès à un bit d'un entier.

```
library matfunc;

function  ith_bit_is_0  (FourByte:LongInt;  i  :integer)  :integer;
export;
```

Spécifications: Cette fonction rend la valeur True si le $i^{éme}$ bit de l'entier FourByte est égal à 0 et False sinon.

```
begin
      FourByte:=FourByte shr (i-1);

    if odd(FourByte) then ith_bit_is_0:=0
    else ith_bit_is_0:=-1;

end;
```

```
procedure Set_ith_bit_to (var FourByte : LongInt; i :Integer; value:
Integer); export;
```

Spécifications: Cette procédure donne au $i^{éme}$ bit de l'entier FourByte la valeur 1.

```
var Mask : Longint;

{Attention ! Four bytes means 31 BITS i.e. 1<=i<=31}

begin
    If (value = 0) and (ith_bit_is_0(FourByte, i)=0)  Then
      begin
            Mask:=1;
              Mask := Mask shl (i-1);
          FourByte := FourByte - Mask
       end
    Else
      if (value = 1) and (ith_bit_is_0(FourByte, i)=-1) Then
            begin
                  Mask:=1;
                    Mask := Mask shl (i-1);
              FourByte := FourByte + Mask
            end

End;

exports
    ith_bit_is_0 index 1,
    Set_ith_bit_to index 2;

begin
end.
```

Calculons maintenant la taille finale de la matrice d'adjacence pour notre exemple :

550.725 éléments / 31 bits par mot =

17.765,3 bits reparties sur des mots de 32 bits =

17.765,3 mots de 32 bits =

71061 bytes =

70 Kbytes!

Notre matrice finale occupe donc approximativement 32 fois moins de place mémoire que la matrice initiale.

5. The Cooling Shedule

Dans le paragraphe 2.7 nous avons présenté les principes généraux du choix d'un *cooling shedule*. Dans ce paragraphe-ci nous allons justifier nos propres choix. Rappelons que la construction d'une politique de refroidissement revient à spécifier les paramètres suivantes :

- la valeur initiale du paramètre de contrôle (variable init_temperature);
- la valeur finale du paramètre de contrôle (variable thermal_equilibrium);
- la longueur des chaînes de Markov (variable nb_floor);
- la constante de Boltzmann (variable const_Bol);
- la règle utilisée pour la diminution du paramètre de contrôle.

Remarquons que la qualité de la solution trouvée par l'algorithme du recuit dépend fortement du choix de ces paramètres. Une mauvaise politique peut *piéger* le recuit à une solution très éloignée de la solution optimale. Il est évident que le choix des valeurs de ces paramètres dépend également de la taille du problème. Un problème de taille petite n'a pas besoin d'autant d'itérations qu'un autre de grande taille. Il est très difficile de trouver un mécanisme général permettant d'ajuster les valeurs de ces paramètres en fonction de la taille du problème traité. Il est encore plus difficile de justifier ce choix en se basant sur une théorie mathématique. Aarts et Van Laarhoven ont essayé de trouver un procédé automatique pour le faire à l'aide de la théorie des chaînes de Markov. Malgré leurs efforts, leurs résultats ne sont pas très satisfaisants et en plus ils sont encore moins applicables sur un espace de configurations comme le nôtre (c.f. 1.3.6). La voie empirique reste donc la plus utilisée. Pourtant il ne faudra pas oublier les études théoriques sur le comportement du recuit (c.f. chapitre 2) qui sont un outil indispensable dans le choix d'une bonne politique de refroidissement.

La température initiale doit être élevée car comme la solution initiale (c.f. 4.3.3) est très loin de l'optimum, il faudra visiter plusieurs configurations dans l'espace d'états pour "oublier" cette solution initiale. De l'autre coté il ne faudra pas lui donner une valeur extrêmement élevée, sinon le temps d'exécution sera augmenté considérablement. En plus il ne faudra pas oublier qu'au départ, presque toutes les configurations proposées doivent être acceptées, ce qui revient à une valeur pour la probabilité d'acceptation (c.f. 2.5.1) proche de l'unité. Cette probabilité d'acceptation est fonction de la température du système, de la constante de Boltzmann et de la

différence d'énergie entre les deux configurations. Il faudra donc tenir en compte tous ces éléments pour obtenir des résultats satisfaisants.

Il ne faudra pas oublier que notre algorithme de recuit est exécuté plusieurs fois et par conséquent, ce que nous en attendons n'est pas une solution très proche de l'optimalité, mais simplement une solution nettement meilleure que la précédente. C'est le choix de la condition d'arrêt du relancement du recuit (c.f. 4.3.4) qui détermine si la solution est assez proche d'un optimum ou pas.

En tenant en compte les considérations précédentes, nous avons choisi la politique de refroidissement suivante :

valeur initiale de la température	10
constante de Boltzman	30
valeur finale de la température	0,099

La température du système diminue suivant la règle *temp:=temp*cooling_rate*. La valeur de la variable cooling_rate a été fixée à 0,9. La longueur des chaînes de Markov est fonction de la taille du problème et du choix de l'utilisateur. La règle utilisée pour le calcul de cette longueur est la suivante :

- nb_floor:=(result_quality * nb_vertex) div paliers

  où paliers=(*Log*(thermal_equilibrium / init_temperature) div *Log*(cooling_rate))+1 (*)

Une option *Result Quality* dans le menu (où le *Toolbar*) du logiciel permet à l'utilisateur de fixer la valeur de la variable result_quality par rapport à ses besoins. Les options suivantes lui sont offertes :

- Normal → result_quality=9;
- High → result_quality=15;
- Very High → result_quality=25.

Remarquons que la variable *paliers* représente le nombre des paliers de température et nb_floor la longueur des chaînes de Markov pour ces paliers. En outre, cette longueur est identique pour chaque palier de température. Montrons que le nombre de paliers de température est bien celui calculé à l'aide de l'expression (*).

$$\text{init\_temperature}*(\text{cooling\_rate})^{\text{paliers}} = \text{thermal\_equilibrium} \Rightarrow$$

$$(\text{cooling\_rate})^{\text{paliers}} = \text{thermal\_equilibrium}/\ \text{init\_temperature} \Rightarrow$$

paliers**Log*(cooling_rate) = *Log*(thermal_equilibrium/ init_temperature) ⇒

paliers = *Log*(thermal_equilibrium/ init_temperature)/ *Log*(cooling_rate).

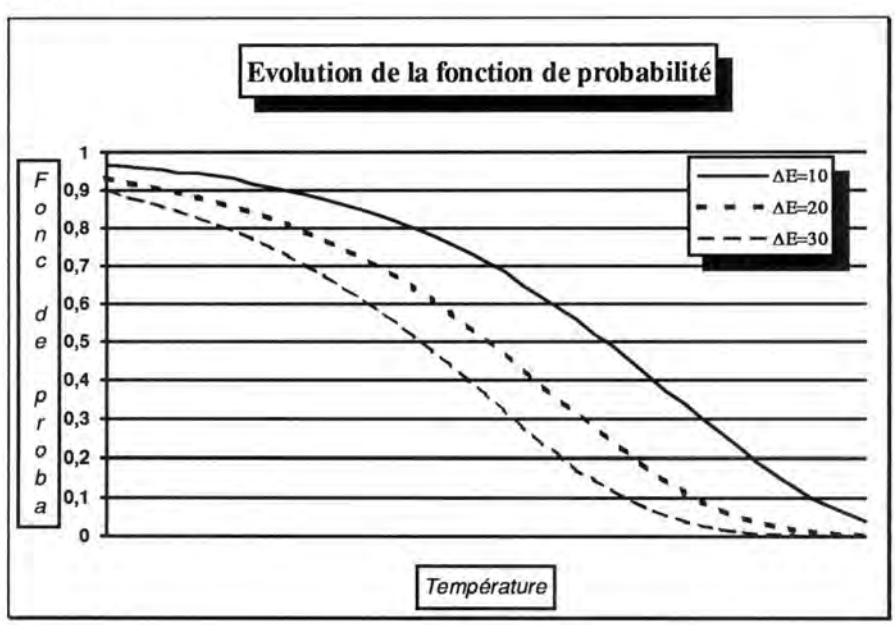
Rappelons que la probabilité d'acceptation d'une *mauvaise* solution est calculée par l'équation *exp(-ΔE/temperature * const_Bol)*. Nous devons vérifier que cette expression donne une valeur très proche de l'unité pour une valeur de température égale à init_temperature, très proche de zéro pour une valeur de température égale à thermal_equilibrium. Remarquons que le passage d'une solution quelconque vers une solution moins bonne se traduit par le transfert d'un étudiant vers un bus de préférence inférieure que celui où il se trouvait avant. Ce transfert provoque une diminution de l'énergie égale à:

30 si l'étudiant passe d'un bus de préférence *Absolute* vers un autre de préférence *Weak*;

20 si l'étudiant passe d'un bus de préférence *Absolute* ou *Strong* vers un autre de préférence *Medium* ou *Weak* respectivement;

10 si l'étudiant passe d'un bus de préférence *Absolute, Strong* ou *Medium* vers un autre de préférence *Strong, Medium* ou *Weak* respectivement.

En tenant compte cette dernier remarque, nous pouvons calculer la probabilité d'acceptation pour les différentes valeurs de différence d'énergie et de température pour notre politique de refroidissement. Le graphique suivant schématise ses valeurs et justifie nos choix.



Chapitre Cinq
Le logiciel Mythos

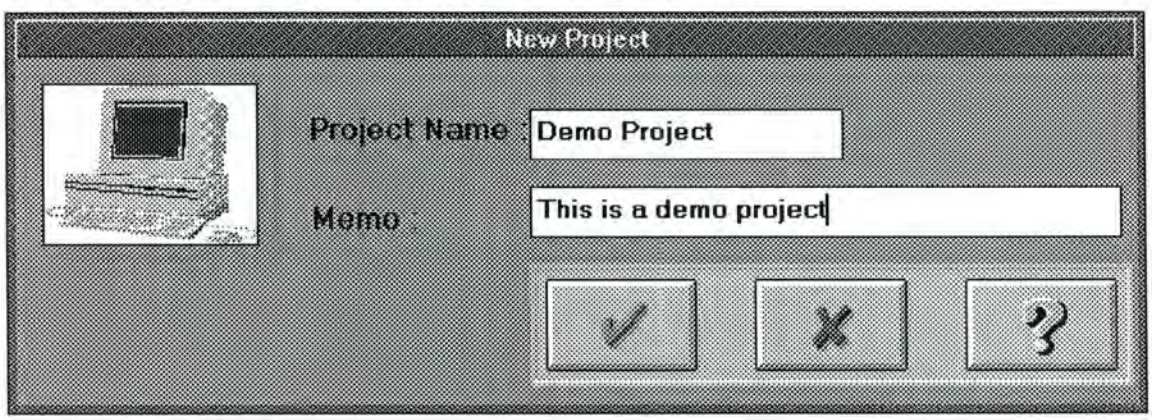
1. Description des différentes parties du logiciel

1.1 Introduction

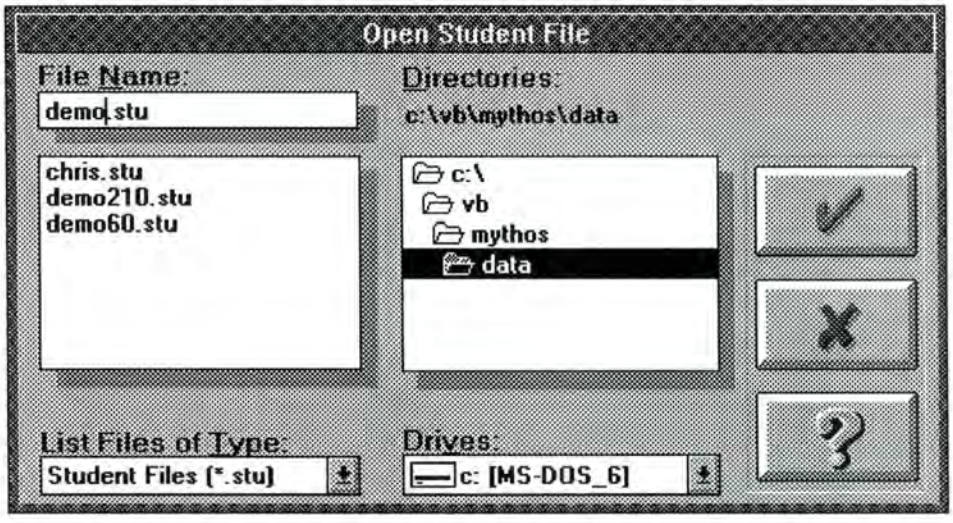
L'objet de ce paragraphe est de décrire les différentes parties du logiciel MYTHOS. Nous procéderons de la manière suivante. Nous supposerons qu'on dispose de la carte d'une ville déjà créée, et nous expliquerons pas à pas les opérations à effectuer pour obtenir une solution finale pour un problème extrêmement simple.

1.2 Création d'un nouveau projet

Pour créer un nouveau projet, nous devons choisir dans le menu *File* l'option *New Project*. La fenêtre suivante apparaît sur l'écran :



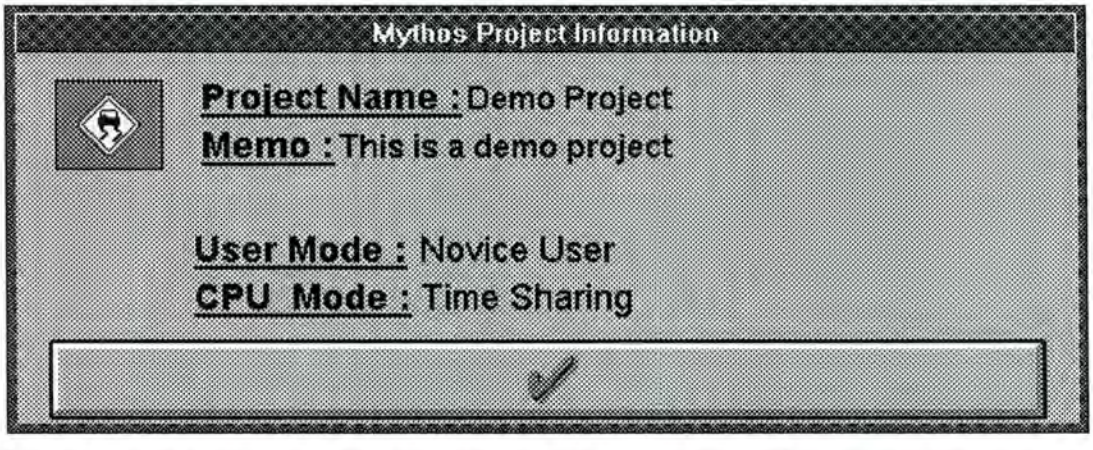
Le champ *Project Name* correspond au nom logique du projet. Le champ *Memo* correspond à une ligne de texte, servant à donner quelques explications supplémentaires sur le projet. Il est demandé à l'utilisateur de remplir au moins le champ *Project Name* et de cliquer sur le bouton 'OK'. La fenêtre suivante apparaît ensuite en demandant à l'utilisateur de donner le nom de la nouvelle base de données contenant des informations sur les étudiants. L'extension par défaut de la base de données correspondant aux étudiants est '.stu'. Nous appellerons notre nouvelle BD 'demo.stu'.



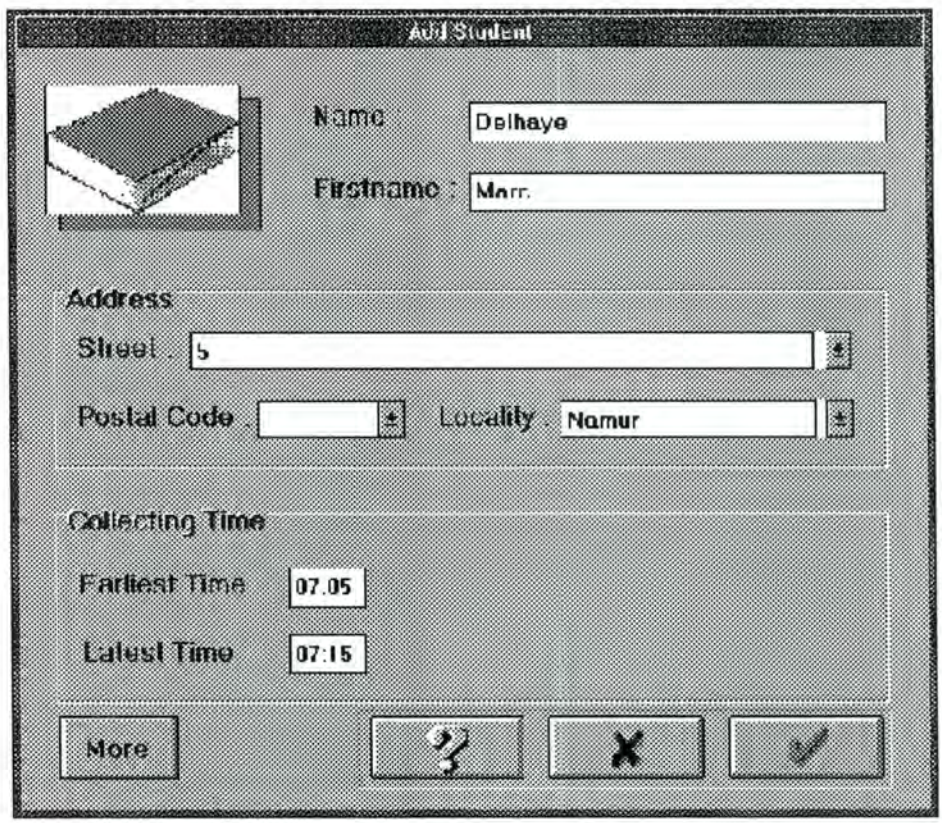
Le procédé précédent est répété une fois de plus pour la création de la BD correspondant aux bus. Ensuite, l'utilisateur doit *choisir* la BD correspondant à la carte de la ville en question. Dans ce cas nous ne pouvons pas mettre un nouveau nom, mais nous devons choisir une carte existante!

Finalement, l'utilisateur donne le nom *physique* du nouveau projet.

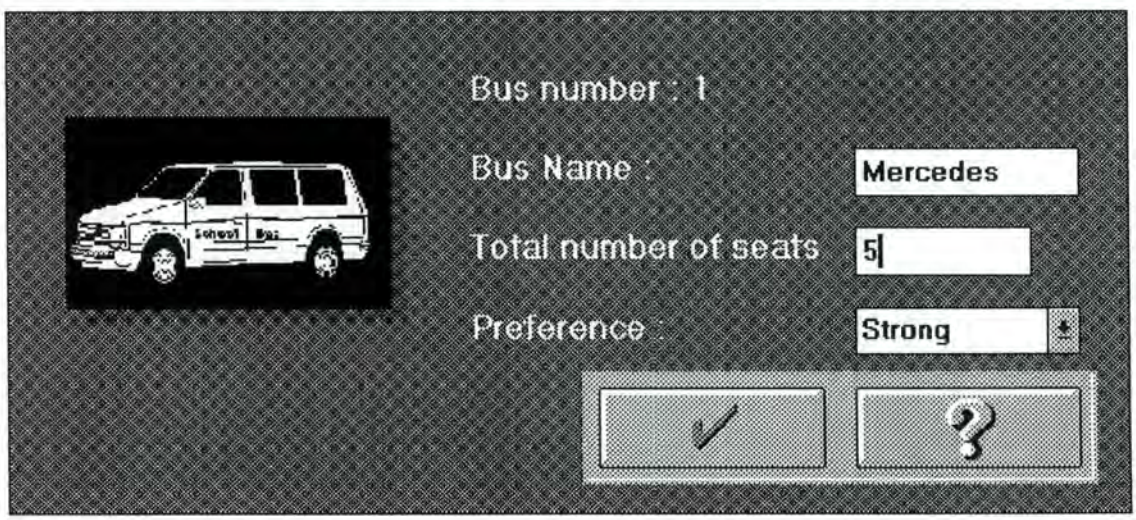
Une fois que tous les fichiers composant le nouveau projet ont été choisis, la fenêtre suivante apparaît à l'écran donnant des informations sur le nouveau projet.



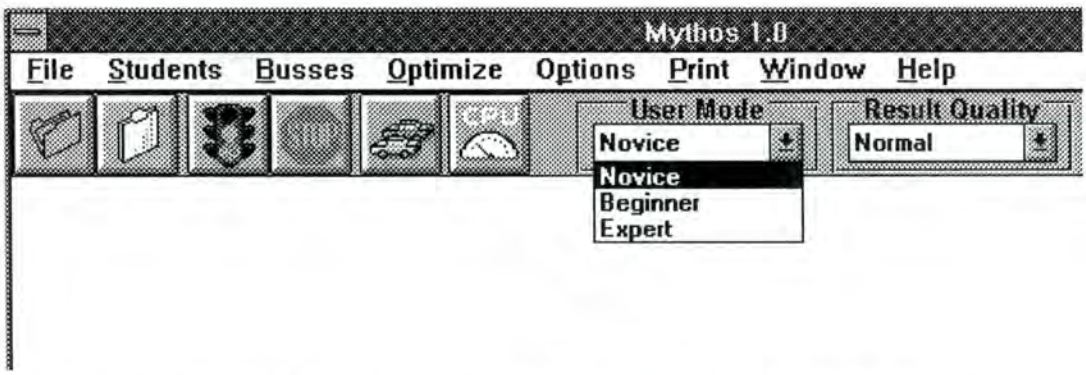
Si nous choisissons dans le menu *Student* l'option *Add*, l'écran suivant apparaît, nous permettant d'encoder les données sur les étudiants.



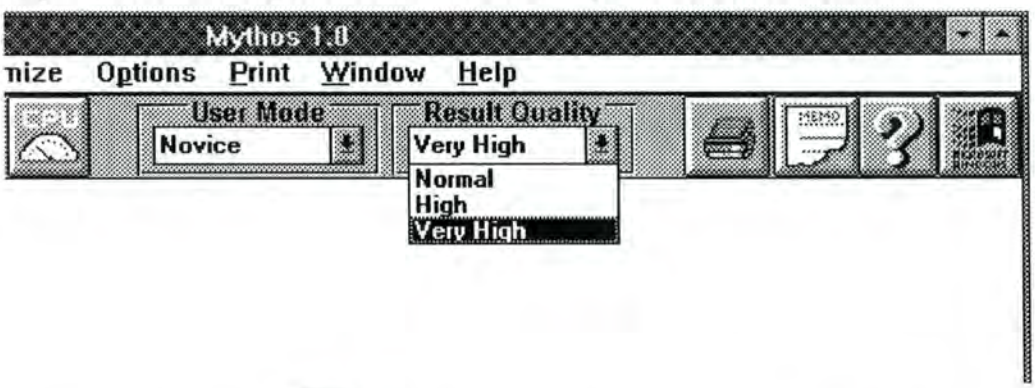
Une fois les étudiants encodés, nous devons encoder les données sur les différents bus. En choisissant dans le menu *Bus* l'option *Add*, l'écran suivant nous permet de le faire.



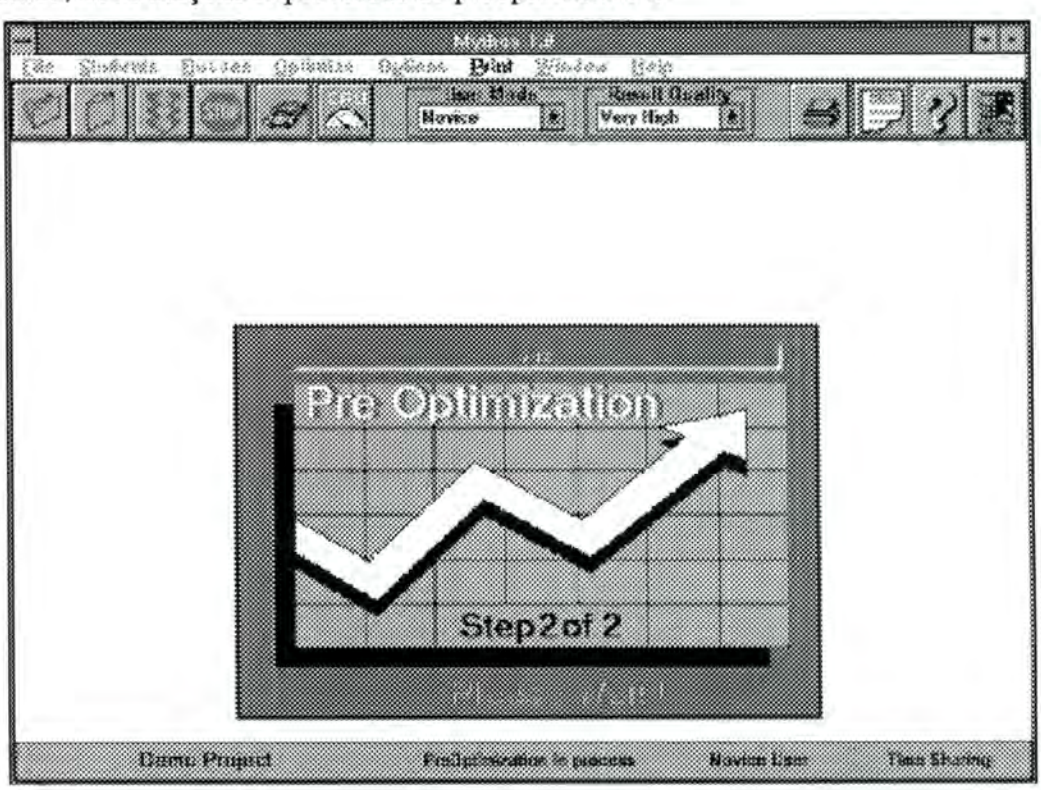
Une fois toutes les données encodées, il ne reste plus qu'à fixer les paramètres *User Mode* et *Result Quality* et à lancer la préoptimisation. Ces paramètres peuvent être fixés à l'aide de deux listes du *Toolbar*, ou en utilisant les sous-menus du menu *Optimize*. L'écran suivant montre comment modifier le paramètre *User Mode* :



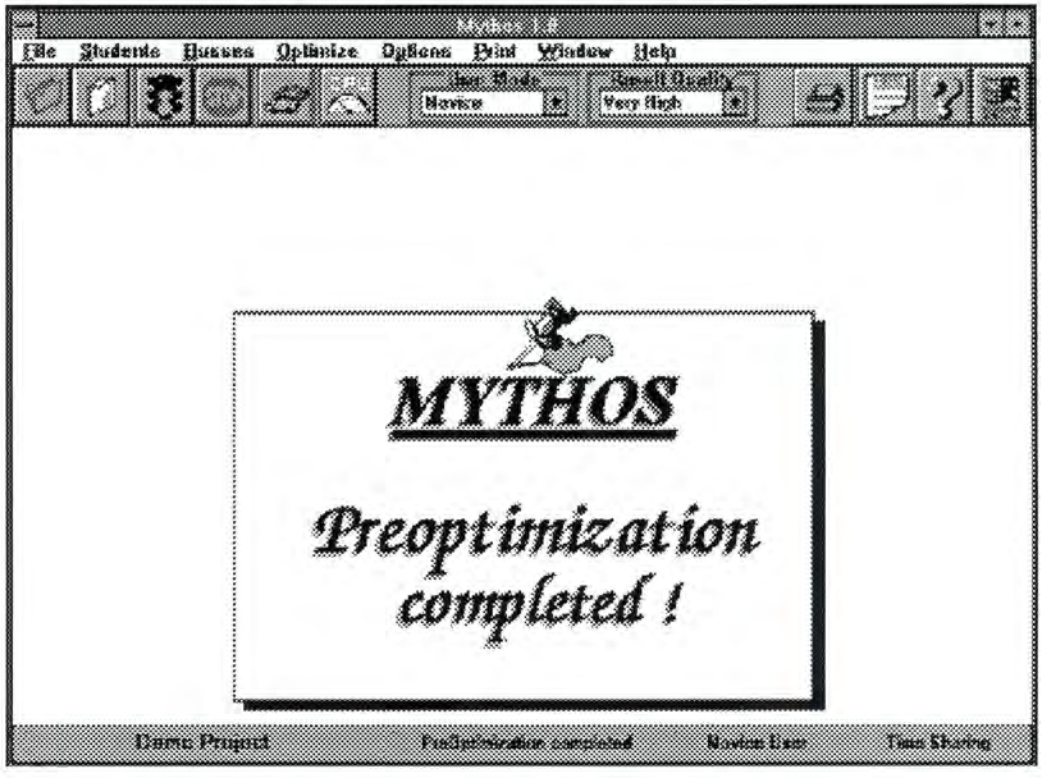
tandis que le suivant indique comment modifier le paramètre *Result Quality* :



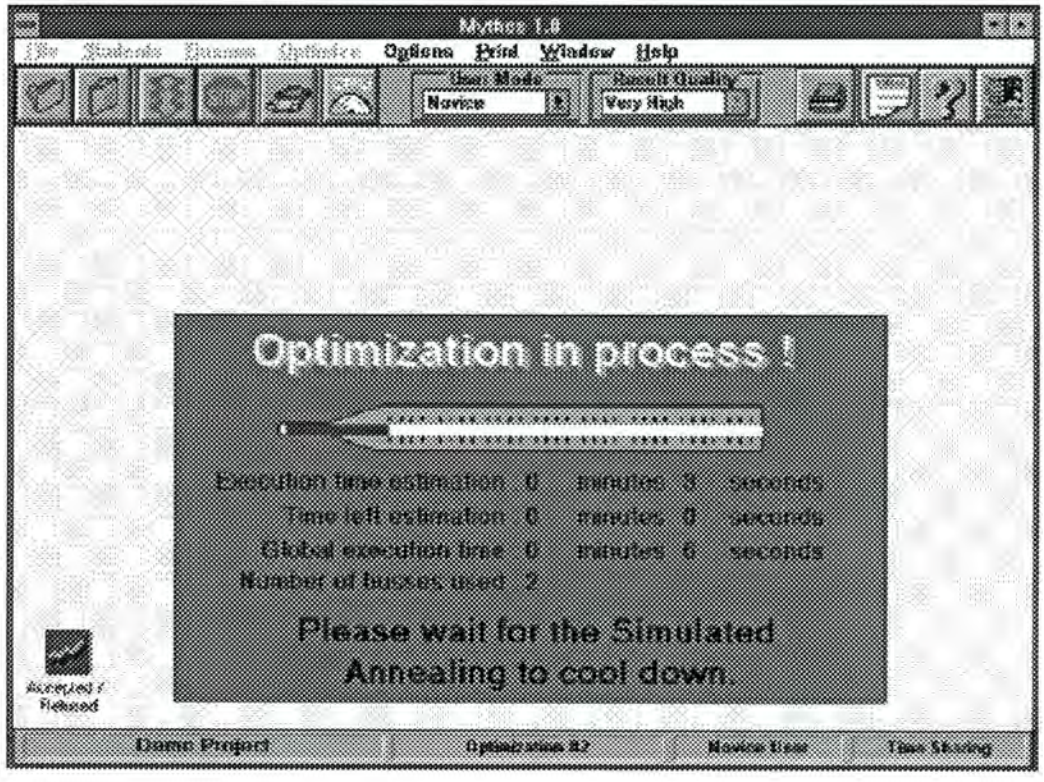
En poussant sur le bouton  ou en choisissant l'option *Preoptimization* dans le menu *Optimize*, nous lançons le processus de préoptimisation.



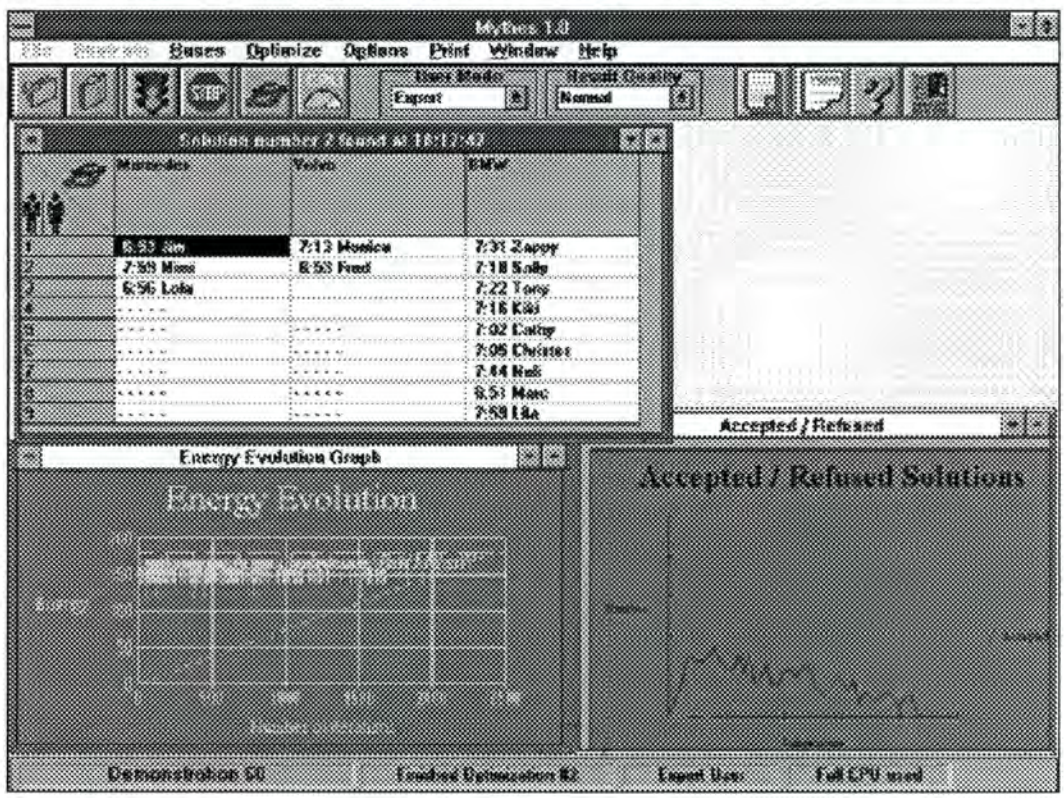
Quand le processus de préoptimisation est achevé l'écran suivant apparaît :



Nous pouvons maintenant commencer à optimiser. De nouveau en poussant sur le bouton  ou en choisissant l'option *Optimization* dans le menu *Optimize*, nous lançons le processus d'optimisation.



Une fois que le processus d'optimisation est terminé, nous obtenons une fenêtre avec la répartition des étudiants, avec l'heure de ramassage, dans les différents bus. L'utilisateur peut essayer manuellement de *transférer* un étudiant d'un bus vers un autre (si cela est possible) en utilisant le principe *drag & drop* de Windows, relancer une optimisation en changeant les préférences des bus restants et/ou l'option *Result Quality*, imprimer les résultats, sauver les résultats dans un fichier ou les prévisualiser en *format d'impression*.



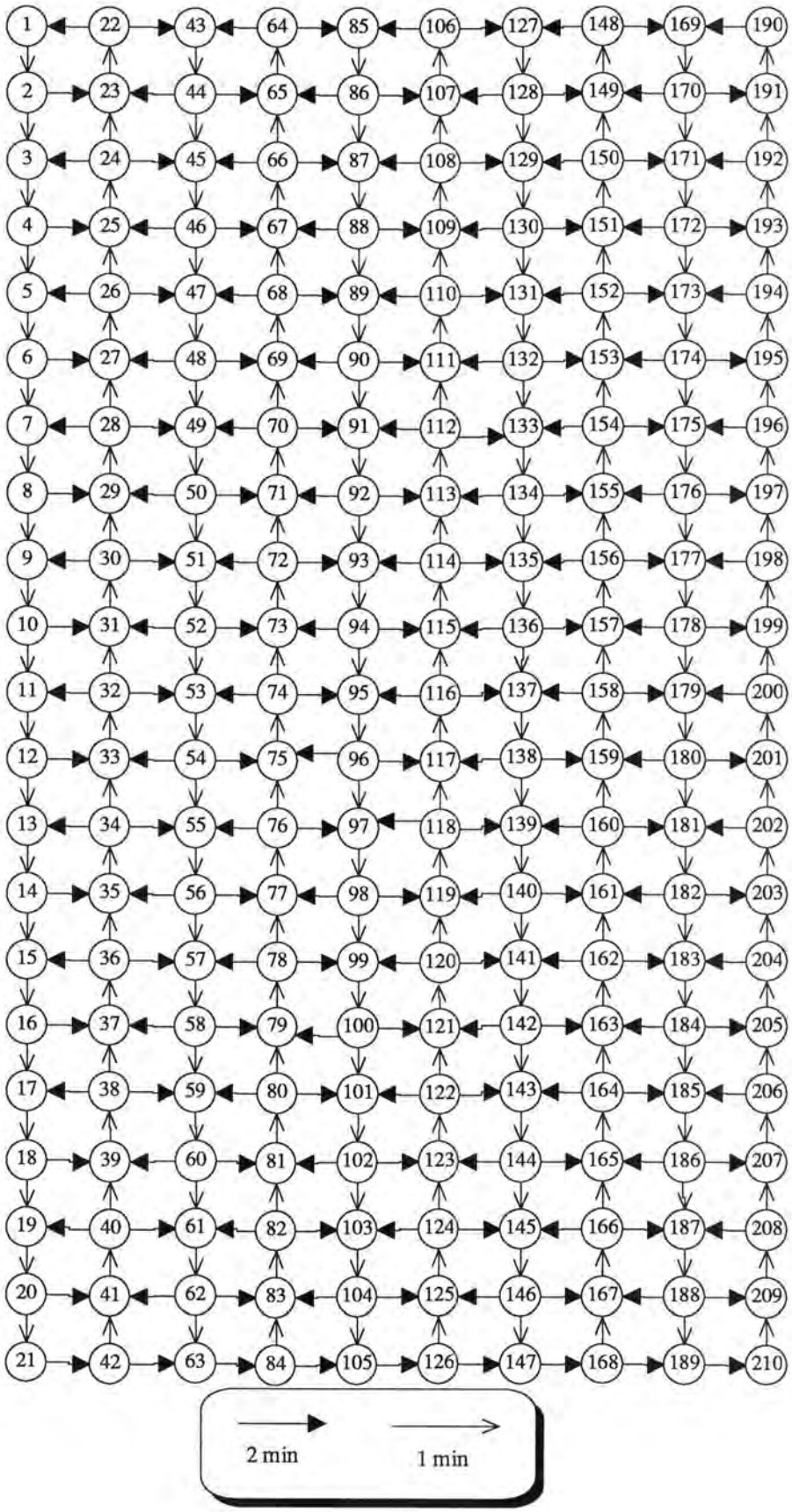
2. Mesure de Performances

Afin de donner une idée des performances du système global, nous allons d'abord décrire les données concernant un problème de ramassage scolaire et, ensuite, nous commenterons les résultats obtenus par le logiciel Mythos 1.0 lors de la résolution du problème décrit.

2.1 Le problème considéré

2.1.1 La carte du problème considéré

La carte représente un centre urbain d'une ville *américaine* se composant de 10 boulevards (à circulation rapide) disposés verticalement, et 21 rues (à circulation lente) disposées horizontalement traversant de part en part les 10 boulevards. Nous avons nommé les différents noeuds de ce graphe par des nombres compris entre 1 et 210. L'utilisation d'un tel type de ville a été motivé par la relative rapidité de générer automatiquement de telles villes. Ce choix ne devrait cependant pas influencer le degré de difficulté du problème pour l'ordinateur vu que celui-ci, contrairement à l'homme, ne tient pas compte de la *forme générale* du plan de la ville. Un nom plus cohérent pour le noeud 44 serait par exemple : *"3ème avenue, numéros impairs compris entre le numéro 37 et 83"*. Le graphe représentant cette ville est le suivant:


1 22 43 64 85 106 127 148 169 190
2 23 44 65 86 107 128 149 170 191
3 24 45 66 87 108 129 150 171 192
4 25 46 67 88 109 130 151 172 193
5 26 47 68 89 110 131 152 173 194
6 27 48 69 90 111 132 153 174 195
7 28 49 70 91 112 133 154 175 196
8 29 50 71 92 113 134 155 176 197
9 30 51 72 93 114 135 156 177 198
10 31 52 73 94 115 136 157 178 199
11 32 53 74 95 116 137 158 179 200
12 33 54 75 96 117 138 159 180 201
13 34 55 76 97 118 139 160 181 202
14 35 56 77 98 119 140 161 182 203
15 36 57 78 99 120 141 162 183 204
16 37 58 79 100 121 142 163 184 205
17 38 59 80 101 122 143 164 185 206
18 39 60 81 102 123 144 165 186 207
19 40 61 82 103 124 145 166 187 208
20 41 62 83 104 125 146 167 188 209
21 42 63 84 105 126 147 168 189 210
2 min
1 min

2.1.2 L'ensemble des étudiants

L'ensemble est composé de 45 étudiants dispersés à travers toute la ville. Chaque étudiant désire être ramassé par le bus scolaire à une heure située dans un intervalle dont le bornes sont comprises entre 6 h 30 et 8 h 35. Ces intervalles sont donc raisonnables si on considère que les cours commencent à 8 h 45. Les adresses ainsi que les intervalles sont répertoriés dans le tableau suivant:

Name	Address	MinTime	MaxTime
Adolf	173	06:45	07:20
Alice	68	07:35	07:40
Barbie	180	06:40	07:30
Bart	150	06:45	07:05
Ben	111	07:10	08:15
Bill	16	07:45	08:25
Bryan	8	07:55	08:05
Carl	14	07:55	08:20
Cathy	52	07:15	07:30
Chris	55	07:25	07:40
Cindy	36	07:40	08:15
Cyntia	128	06:50	07:20
Danny	78	07:55	08:15
Dolly	83	07:50	08:25
Fred	47	07:40	07:55
Georges	196	06:45	07:15
Jim	91	07:50	08:05
John	76	07:20	07:35
Jules	30	07:30	07:45
Keith	94	07:10	07:25
Kirk	188	07:40	08:35
Lola	74	07:30	07:40
Lolita	23	06:55	07:10
Lou	10	06:50	07:00
Luc	103	08:00	08:20
Marc	50	07:35	07:55
Max	33	07:30	07:40
Maxi	59	08:00	08:20
Micky	40	06:30	07:00
Minie	126	07:55	08:25
Nick	100	07:45	08:15
Pat	27	07:40	07:50
Piet	134	07:20	07:30
Pol	70	07:55	08:05
Roberto	43	06:50	07:20

Roger	182	06:50	07:40
Rudolf	130	07:15	07:45
Sally	117	07:30	07:40
Snoopy	87	06:50	07:25
Tonia	141	07:00	08:05
Tony	21	08:00	08:27
Tracy	114	07:10	07:25
Wim	136	07:10	07:25
Yann	144	07:40	08:10
Yves	164	07:40	07:55

2.2 Les résultats obtenus

Nous avons décrit ci-dessous les deux solutions obtenues pour le problème considéré en utilisant le logiciel Mythos 1.0 en mode *Very High* et *Normal*. Pour mesurer la difficulté du problème nous pouvons remarquer que le nombre des noeuds à colorer, tous supervertex confondus, est égal à 1092, et que 25% des éléments de la matrice d'adjacence sont des 1[1].

[1] Rappelons que la valeur 1 entre deux éléments signifie que deux étudiants (noeuds) à une heure précise, ne peuvent pas se trouver dans le même bus (avoir la même couleur). Par conséquent, plus il y a des 1 plus le problème devient difficile. Dans la pratique, la matrice ne dépassera jamais le 20% car les *time window* des étudiants seront de longueur minimum de 30 min.

2.2.1 Solution obtenue en mode *Very High*

Global Execution Time : *36 min*
Number of Optimizations : *1*
Result Quality : *Very High*
User Level : *Expert*
CPU Type : INTEL *486 DX-33*

Execution Time	Number of Buses
00:15	15
00:24	12
01:25	11
02:01	10
02:22	9
05:00	7
17:05	6
17:30	5
26:01	4
28:10	3

Mythos 1.0 - Report

DAF

20 seats **Filled : 16**

Strong preference

Departure time : 6:31

Arrival time : 8:27

Hour	Name	Address
6:31	Micky	40
6:51	Lou	10
7:02	Lolita	23
7:11	Roberto	43
7:23	Cathy	52
7:26	Chris	55
7:30	John	76
7:32	Lola	74
7:37	Max	33
7:45	Jules	30
7:48	Pat	27
7:58	Bryan	8
8:05	Carl	14
8:14	Cindy	36
8:21	Bill	16
8:27	Tony	21

Mercedes

20 seats **Filled : 20**

Absolute preference

Departure time : 6:41

Arrival time : 8:32

Hour	Name	Address
6:41	Barbie	180
6:48	Georges	196
6:53	Adolf	173
6:59	Bart	150
7:07	Snoopy	87
7:17	Cyntia	128
7:19	Rudolf	130
7:23	Piet	134
7:25	Wim	136
7:32	Ben	111
7:42	Alice	68
7:44	Fred	47
7:54	Marc	50
7:57	Pol	70
8:00	Jim	91
8:12	Danny	78
8:15	Nick	100
8:19	Luc	103
8:23	Minie	126
8:32	Kirk	188

VW

20 seats **Filled : 9**

Medium preference

Departure time : 7:05

Arrival time : 8:25

Hour	Name	Address
7:05	Roger	182
7:18	Tracy	114
7:24	Keith	94
7:31	Sally	117
7:41	Tonia	141
7:53	Yves	164
8:04	Yann	144
8:18	Maxi	59
8:25	Dolly	83

2.2.2 Solution obtenue en mode *Normal*

Global Execution Time : 12 *min*
Number of Optimizations : *1*
Result Quality : *Normal*
User Level : *Expert*
CPU Type : INTEL *486 DX-33*

Mythos 1.0 - Report

Virtual 1

2 seats Filled : 1
Weak preference
Departure time : 8:02
Arrival time : 8:02

Hour	Name	Address
8:02	Danny	78

DAF

20 seats **Filled : 16**

Strong preference

Departure time : 6:46

Arrival time : 8:30

Hour	Name	Address
6:46	Adolf	173
7:01	Bart	150
7:06	Cyntia	128
7:13	Snoopy	87
7:20	Rudolf	130
7:24	Piet	134
7:29	Ben	111
7:37	Alice	68
7:44	Fred	47
7:51	Marc	50
7:56	Pol	70
7:58	Jim	91
8:07	Nick	100
8:11	Luc	103
8:16	Minie	126
8:30	Kirk	188

Mercedes

20 seats **Filled : 15**

Absolute preference

Departure time : 6:47

Arrival time : 8:10

Hour	Name	Address
6:47	Georges	196
7:01	Barbie	180
7:06	Roger	182
7:16	Wim	136
7:19	Tracy	114
7:22	Keith	94
7:29	John	76
7:36	Chris	55
7:41	Max	33
7:45	Jules	30
7:50	Pat	27
7:56	Bryan	8
8:03	Carl	14
8:05	Bill	16
8:10	Cindy	36

VW

20 seats **Filled : 13**

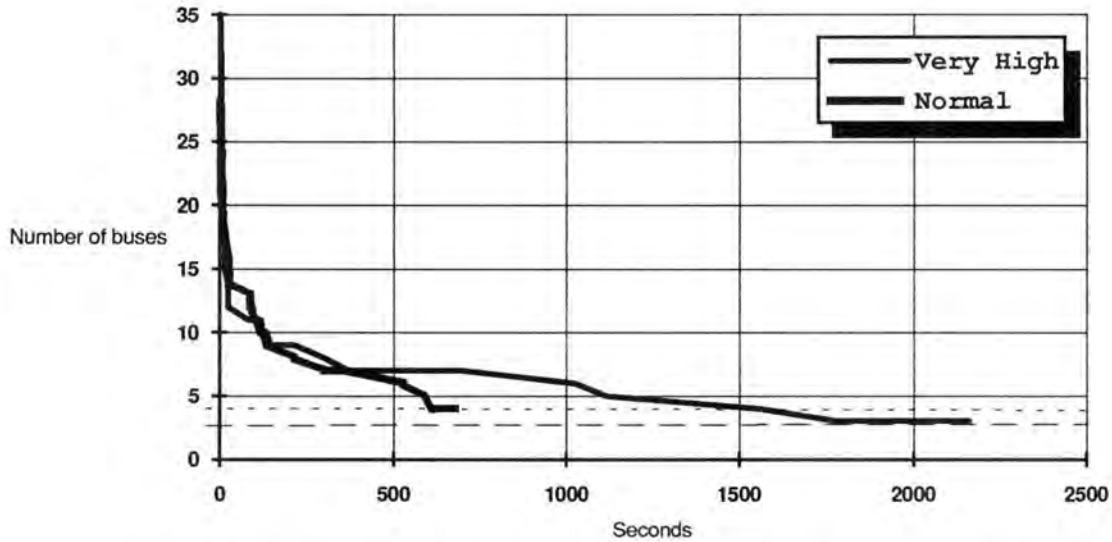
Medium preference

Departure time : 6:31

Arrival time : 8:23

Hour	Name	Address
6:31	Micky	40
6:51	Lou	10
7:04	Lolita	23
7:15	Roberto	43
7:26	Cathy	52
7:34	Lola	74
7:40	Sally	117
7:47	Tonia	141
7:53	Yves	164
7:56	Yann	144
8:06	Dolly	83
8:13	Maxi	59
8:23	Tony	21

2.2.3 Comparaison des deux modes d'exécution



Nous pouvons observer sur le graphe ci-dessus que, comme on pouvait s'y attendre, la qualité du résultat obtenu en mode *Very High* est meilleure qu'en mode *Normal* (3 bus au lieu de 4). Cependant, le prix à payer pour ce gain de qualité est le temps nécessaire pour l'optimisation (36 minutes au lieu de 12).

Comme nous pouvons également le remarquer sur ce graphe, après quelque minutes d'optimisation, la solution donnée par le programme en mode *Normal* est, en général, meilleure. Cette différence diminue au fur et à mesure que le temps s'écoule pour finalement s'inverser (la solution donnée par le programme en mode *Very High* devient meilleure). Cette différence peut s'expliquer par le fait que, lors de l'exécution du programme en mode *Normal*, les chaînes de Markov étant plus courtes, l'orientation des solutions se fait plus tôt, on obtient donc des meilleures solutions plus tôt, mais si cette orientation se fait trop tôt, il se peut que le programme n'atteigne qu'un extremum local (une solution avec 4 bus) au lieu d'un extremum global (une solution avec 3 bus).

Une dernière remarque que nous pouvons faire à propos de ces deux modes d'optimisation est la suivante. Non seulement la solution donnée en mode *Very High* nécessite moins de bus, mais en plus la répartition des étudiants dans ces différents bus reflète mieux les préférences qui leur ont été attribuées. Ceci s'explique par le fait que les chaînes de Markov de ce mode d'optimisation sont plus longues et donc, l'algorithme a plus de chances d'approcher un extremum global qui respecte les préférences des bus. Cependant notons que dans la solution obtenue en mode *Normal*, il ne fallait pas s'attendre à une répartition qui refléterait les préférences de l'utilisateur, mais plutôt à

une répartition uniforme! Etant donné que le recuit n'a pas réussi à supprimer tous les bus virtuels, les préférences de l'utilisateur n'ont pas pu être attribuées.

3. Critiques

3.1 Critiques des résultats obtenus

On peut constater sur l'exemple du paragraphe précédent que, pour un problème de taille moyenne, le temps d'exécution total pour obtenir une solution de qualité est raisonnable (moins d'une heure). Notre heuristique étant de complexité polynomiale (n^2), pour optimiser un problème de grande taille, le temps nécessaire sera sensiblement plus long. Cependant, ce temps reste nettement inférieur à ceux des algorithmes *classiques* proposés dans la littérature qui sont, eux, de complexité exponentielle.

Il faut souligner que l'approche choisie pour la résolution de ce problème possède un atout incontestable. Par le choix même d'une heuristique, l'utilisateur n'obtient pas seulement **une** bonne solution mais **plusieurs** (parmi lesquelles il est libre de choisir la plus appropriée).

3.2 Amélioration du temps d'exécution

Le choix d'un langage de prototypage nous a permis de concevoir et réaliser un logiciel de qualité sur une période de quatre mois. La contrepartie de ce choix est une inévitable lenteur. Il est indéniable que l'utilisation d'un langage plus performant tel que Visual C++, pour n'en citer qu'un seul, diminuerait le temps d'exécution d'un facteur 2 à 10. De plus, l'utilisation de structures de données plus appropriées améliorerait encore ce temps.

3.3 Extensions possibles du logiciel

Une première extension envisagée sera de permettre aux bus de passer par l'école, ceci afin de décharger les étudiants pour ensuite repartir à vide. Cette extension permettra d'envisager le problème dans son entièreté et, dans certains cas, d'aboutir à une solution nécessitant moins de bus.

Une seconde extension sera de considérer plusieurs fonctions-objectifs au lieu d'une seule comme c'est le cas actuellement. Notre problème deviendra alors multi-critère. Les fonctions-objectifs envisagées permettrons:

- un *centrage* des étudiants à l'intérieur de leurs contraintes horaires réduisant le nombre de *cas limites*[2] ;
- une minimisation de la distance totale parcourue par l'ensemble des bus;
- etc.

Une troisième extension sera de prendre en compte la variation du trafic dans la ville suivant le moment de la journée afin d'obtenir une solution plus proche de la réalité.

La modélisation originale du programme prenait en considération toutes ces extensions; seules les contraintes temporelles auxquelles nous étions astreins ne nous ont pas laissé le loisir de les mettre en oeuvre.

[2] Un cas limite correspond au ramassage d'un étudiant à son heure au plus tôt ou au plus tard.

4. Le manuel d'utilisation de Mythos

Dans ce paragraphe, le lecteur trouvera le manuel d'utilisation du logiciel *Mythos*. Ce manuel est entièrement écrit en anglais, étant donné que cette langue a été choisie pour l'interface homme-machine du logiciel.

User Manual for ...

Mythos 1.0

Requirements

The minimal configuration for Mythos is a 486DX-33 based computer, with 8 Mega of RAM and 6 Mega of free disk space. The minimal graphics mode is 800x600 with 16 colours.

About the Authors

This program has been developed by Christodoulidis Christos & Delbaye Marc.

The Principle of Virtual Buses

Mythos is trying to find a solution for a given problem in the following way. In the begining (of the optimization process) the number of buses considered is equal to the number of students of the current project !

That means that every student is attributed to a bus and that no other student is in it. Of course, the number of the **real buses** is lower than the number of the students. Since every real bus has one and only one student inside, Mythos will create virtual buses which also have one and only one student inside. Both the real & virtual buses will be the initial solution of the transport problem. The virtual buses are called *Virtual xxx* and their capacity is equal to two seats each.

During the iterations, the number of virtual buses is decreased as much as possible. If Mythos finds a solution with no virtual bus, then this will be accepted. On the contrary, if no solution excluding virtual buses is found, then Mythos will present to the user a solution with one or more virtual buses. In that case we suggest to the user to add a new real bus and restart the optimization process.

The Menu Bar

Files

New Project

Creates a new project file. Every project file has a **name** attribute and a **memo** attribute.

See also :

 Open Project

 Close Project

 Open Project

Opens an existing project file.

See also :

New Project

 Close Project

 Close Project

Closes the active project

Exit

Exit from program. All the files are closed before exiting.

Students

Add (Student)

This option adds a new student to the student database file. The following attributes have to be completed :

Name : The name of the student

Firstname : The firstname of the student

Address

Locality : The locality of the student's address

Street : The street in the locality of the student's address

Postal Code : Not yet implemented !

Collecting Time

Earliest Time : The earliest time that the student wishes to be collected

Latest Time : The latest time that the student wishes to be collected. The latest time has to be strictly greater than the earliest time.

See also :

Modify a Student

Visualize a Student

View (Student)

This option visualizes the attributes of a student already existing in the student database file.

The first window demands the user to enter the name of the student to be visualized. The following options are possible :

xxxxx : the exact name of the student

* : visualize all the students

A combination between some letters and the symbol *.

Example :

C*S will find all the students whose names start with the letter C and finish with the letter S

The following attributes are visualized :

Name : The name of the student

Firstname : The firstname of the student

Address

Locality : The locality of the student's address

Street : The street in the locality of the student's address

Postal Code : Not yet implemented !

Collecting Time

Earliest Time : The earliest time that the student wishes to be collected

Latest Time : The latest time that the student wishes to be collected.

See also :

Add a Student

Modify a Student

Modify (Student)

This option modifies the attributes of a student already existing in the student database file.

The first window demands the user to enter the name of the student to be visualized. The following options are possible :

xxxxx : the exact name of the student

* : visualize all the students

A combination between some letters and the symbol *.

Example :

C*S will find all the students whose names start with the letter C and finish with the letter S

The following attributes can be modified :

Name : The name of the student

Firstname : The firstname of the student

Address

Locality : The locality of the student's address

Street : The street in the locality of the student's address

Postal Code : Not yet implemented !

Collecting Time

Earliest Time : The earliest time that the student wishes to be collected

Latest Time : The latest time that the student wishes to be collected. The latest time has to be strictly greater than the earliest time.

See also :

Add a Student

Visualize a Student

Delete (Student)

Not yet implemented !

Buses

Add (Bus)

Adds a new bus in the opened project.

Every bus has a name, a capacity (maximum seats) and a preference given by the user. The preference of a bus represents the degree of the occupied seats in that bus.

The following preferences are offered :

- **Weak** : Try to empty the bus
- **Medium** : If there is another bus with a higher preference then it is preferred to the one with the medium preference.
- **Strong** :If there is another bus with an absolute preference then it is preferred to the one with the medium preference.
- **Absolute** : Try to fill the bus.

See also :

Visualize Bus

Modify Bus

Delete Bus

 Adjust Bus Preferences

View (Bus)

Visualize the bus attributes (name, capacity, preference) in the open project. No attribute can be changed.

See also :

Add Bus

Modify Bus

Delete Bus

 Adjust Bus Preferences

Modify (Bus)

Modifies the bus attributes.

Every bus has a name, a capacity (maximum seats) and a preference given by the user. The preference of a bus represents the degree of the occupied seats in that bus.

See also :

Add Bus

Visualize Bus

Delete Bus

 Adjust Bus Preferences

Delete (Bus)

Deletes a bus from the opened project.

See also :

Add Bus

Modify Bus

Visualize Bus

 Adjust Bus Preferences

Optimize

Result Quality

The following options are offered :

- **Normal** : Worst result quality but best execution time.
- **High** : Medium result quality and medium execution time..
- **Very High** : Best result quality but longer execution time.

 Adjust Bus Preferences

The following preferences are offered :

- **Weak** : Try to empty the bus
- **Medium** : If there is another bus with a higher preference then it is preferred to the one with the medium preference.
- **Strong** :If there is another bus with an absolute preference then it is preferred to the one with the medium preference.
- **Absolute** : Try to fill the bus.

See also :

Buses

User Level

The user level can be understood as the level of interactivity between Mythos and the user while searching for a solution.

Three user levels are offered:

- **Novice** (default)

  When the Novice mode is chosen, Mythos tries to find a good solution in a non interactive way. In the end of this execution the result will be shown on the screen and the interactive module will be enabled, but during the first execution Mythos tries to eliminate all the virtual buses.

- **Beginner**

When the Beginner mode is chosen, the interactivity level is higher than in Novice User.

- **Expert**

When the Expert mode is chosen, the interactivity level is the highest . The need to adjust the bus preferences and re-execute the optimiser is almost certain. The advantage of this User level is that the user can take almost all the critical decisions **between** two optimisations. On the other hand, a perfect knowledge of all the principles used by Mythos is necessary.

 PreOptimization / Run (Again) Optimizer

The first time you want to optimize, the only option available is PreOptimisation. Choosing this option Mythos is preparing the data for the optimization. Technically talking, Mythos is solving all the shortest path problems between every pair of students.

The second time that you want to optimize, the only option is Run Optimizer. Choosing this option Mythos tries to find the first solution to the given transport problem. The quality of the result and the execution time of this optimization depends of the given User Level.

After this second execution, the solution found is displayed on the screen and the interactive module of Mythos is enabled. That means that the user has different possibilities after the first optimization. These possibilities are:

-  Adjust Bus Preferences

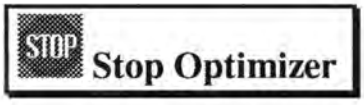
- **Run Again Optimizer**

Running again the optimizer gives another chance to Mythos to find a better solution, using the new bus preferences given by using the option Adjust Bus Preferences.

- **Try to find a solution interactivly**

The user has to use the drag & drop principle of Windows on the students. You have to double click on the student that you want to transfer, and after that to click on the destination bus. If this

transaction is possible, the transfert will be executed, otherwise an error message will appear.



Stopping the optimization process means that the user found an acceptable solution for his transport problem.

Options

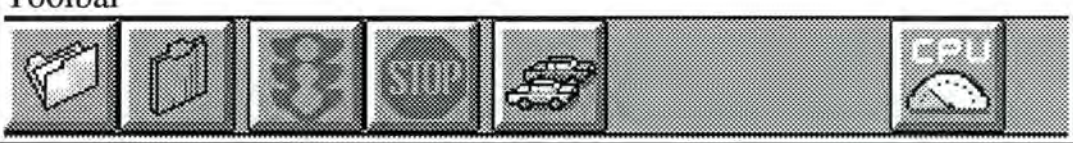
Full CPU use

When this option is checked  , the multitasking use of Microsoft Windows is not possible during the execution of the optimiser.

When this option is not checked  , the multitasking use of Microsoft Windows is possible during the execution of the optimiser, but the optimisation is slowed down.

Tip : When the option Full CPU use is checked and the optimiser is running, if you want to pass to multitasking mode you just have to click once the Full CPU use button and after a while the CPU mode will pass to multitasking CPU use.

Toolbar



The Toolbar permits a quick access to some functions of Mythos that are used often.

When this option is **checked** , the toolbar is visible.

When this option is **not checked**, the toolbar is not visible.

Window

Cascade

Arrange the windows of Mythos in the cascade way.

Tile Horizontal

Arrange the windows of Mythos in the tile horizontal way.

Arrange Icons

Arrange the icons in the MDI (Multiple Document Interface) of Mythos.

Help

Calls the Help file.

Bibliographie

AERTS, E., and KORST, J. [1989], *Simulated Annealing and Boltzmann Machines*, John Wiley & Son, New York.

AERTS, E., and VAN LAARHOVEN, P. [1987], *Simulated Annealing: Theory and applications*, D. Reidel Publishing Company.

ANANDALINGAM, G., FRIESZ, T., MEHTA, N., NAM, K., SHAH, S., TOBIN, R. [1993], *The multiobjective equilibrium network design problem revisited: A simulated annealing approach*, European Journal of Operational Research 65 44-57, North-Holland.

BENITO-ALONSO, M.A., DEVAUX, P. [1979], *L'accessibilité d'un service public: Un compromis entre plusieurs modes de transport*, Communication présentée à la 10ème réunion du Groupe EURO "Aide à la décision multicritère" LIEGE, 18 et 19 octobre 1979.

BAUSCH, R. [1992], *A multicriteria scheduling tool using a branch-and-bound algorithm*, European Journal of Operational Research 61 215-218, North-Holland.

BERGE, C. [1973], *Graphes et Hypergraphes*, DUNOD Paris-Bruxelles-Montréal.

BODART, F., PIGNEUR, Y. [1989], *Conception assistée des systèmes d'information*, MASSON Paris-Milan-Barcelone-Mexico.

BOOCH, G. [1992], *Conception orientée objets et applications*, ADDISON-WESLEY PUBLISHING COMPANY, France.

BRAD, J. COX [1987], *Object Oriented Programming. An Evolutionary Approach*, ADDISON-WESLEY PUBLISHING COMPANY.

BULFIN, L. R., and JEFFCOAT, E. D. [1993], *Simulated annealing for resource-constrained scheduling*, European Journal of Operational Research 70 43-51, North-Holland.

CHIANG, W.-C., FITZSIMMONS, J., and KOUVELIS, P. [1992], *Simulated annealing for machine layout problems in the presence of zoning constraints*, European Journal of Operational Research 57 203-223, North-Holland.

COUNTINHO-RODRIGUES, J., RODRIGUES, N., CLIMACO, J., *Solving an urban routing problem using heuristics; a successful study*, Belgian Journal of Operations Research, Statistics and Computer Science Vol. 33 (1,2).

CURRENT, J. [1993], *Multiple objectives in transportation network design and routing*, European Journal of Operational Research 65 1-5, North-Holland.

CURRENT, J., Marsh, M. [1993], *Multiobjective transportation network design and routing problems: Taxonomy and annotation*, European Journal of Operational Research 65 4-19, North-Holland.

DITTRICH, S. [1993], *Programmer en Microsoft Visual Basic 2 pour Windows*, MICRO APPLICATION.

DOWSLAND, K. [1993], *Some experiments with simulated annealing techniques for packing problems*, European Journal of Operational Research 68 389-399, North-Holland.

DREYFUS, G., et SIARY, P. [1988], *La méthode du recuit simulé: Théorie et applications*, I.D.S.E.T., Paris.

DUBOIS, N., and WERRA, D. [1993], *EPCOT: An efficient procedure for coloring optimally with tabu search*, Computers Math. Applic. Vol. 25, No. 10/11, pp. 35-45.

FRANTZ, G. [1993], *Visual Basic 3 pour Windows*, SYBEX.

HERAGU, S., ALFA, A. [1992], *Experimental analysis of simulated annealing based algorithms for the layout problem*, European Journal of Operational Research 57 190-202, North-Holland.

KAINEN, P., SAATY, T. [1977], *The four colour problem. Assaults & conquest*, Advanced Book Program, Great Britain..

KARKAZIS, J., *A minimax assignement problem on a linear communication network*, Belgian Journal of Operations Research, Statistics and Computer Science Vol. 33 (1,2).

KHOONG, C., M. [1993], *Shortest-Path Reconstruction algorithms*, The Computer Journal, Vol. 36, No. 6.

KRISHNAKUMAR, T.S., MURALIDHAR, R., RAGHAVENDRA, A., and RAGHAVENDRA, B.G. [1992], *A practical heuristic for a large scale vehicle routing problem*, European Journal of Operational Research 57 32-38, North-Holland.

LAURSEN, PER S. [1993], *Simulated Annealing for the QAP-Optimal tradeoff between simulation time and solution quality*, European Journal of Operational Research, North-Holland.

MARCHAND, M. [1989], *Mathématique discrète : Outil pour l'informaticien*, De Boeck Université.

NARSINGH, D. [1974], *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall.

NEMHAUSER, G., and WOLSEY, L. [1988], *Integer and Combinatorial Optimization*, WILEY INTERSCIENCE.

NIJENHUIS, A., and WILF, H. [1975], *Combinatorial Algorithms*, Academic Press.

PAPADIMITRIOU, C., and STEIGLITZ, K. [1982], *Combinatorial Optimization : Algorithms and Complexity*, Prentice-Hall.

PASCHOS, V., PEKERGIN, F., ZISSIMOPOULOS, V., *Approximating the optimal solution of some hard graph problems by a Boltzmann machine*, Belgian Journal of Operations Research, Statistics and Computer Science Vol. 33 (1,2).

POTVIN, J.-Y., and ROUSSEAU, J.-M. [1993], *A parallel route building algorithm for the vehicle routing and scheduling problem with time windows*, European Journal of Operational Research 66 331-340, North-Holland.

PROFILIDIS, V. [1991], *Evaluation Methods of Transport Projects*, Democritus Thrace University Greece.

ROSS, S. [1987], *Initiation aux probabilités*, Presses Polytechniques Romandes.

SAUL, B. GELFAND, and SANJOY, K. MITTER [1993], *Metropolis-type annealing algorithms for global optimization in* IR^d, SIAM J. CONTROL AND OPTIMIZATION Vol 31, No 1, pp. 111-131, January 1993.

Annexes

1. Problèmes NP-Complets

1.1 Complexité polynomiale

Un algorithme est polynomial ou de complexité polynomiale si T(n) étant son temps d'exécution maximum pour une donnée de taille n, alors il existe une constante c t.q. $\forall n\ T(n) \leq cn^k$ où k est une constante.

1.2 Réduction polynomiale

Soient P et Q deux problèmes de décision (c.à.d. deux problèmes demandant une réponse par OUI ou NON). P est polynomialement réductible à Q s'il existe une fonction totale calculable au moyen d'un algorithme polynomial, soit f, t.q. la réponse à l'exemplaire P(x) de P est OUI ou NON, selon que la réponse à l'exemplaire Q(f(y)) de Q est OUI ou NON (où x=f(y)).

Remarque :

Soient P et Q deux problèmes de décision et soit P polynomialement réductible à Q.

- S'il existe un algorithme polynomial qui résout Q, alors il en est de même pour P.
- S'il n'existe aucun algorithme polynomial qui résout P, alors il en est de même pour Q.

1.3 La classe NP (non-déterministe polynomial)

Il existe de nombreux problèmes pratiques pour lesquels les seuls algorithmes que nous avons pu trouver sont de complexité exponentielle, c'est-à-dire que leur temps d'exécution croît exponentiellement avec la taille des données. Comme une fonction exponentielle croît plus vite que tout polynôme, sans limitation du degré, la question

qui se pose naturellement est de savoir s'il ne serait pas dans la nature même de ces problèmes de ne pouvoir être résolus par un algorithme polynomial.

Considérons d'abord un problème de décision. Celui du cycle hamiltonien en est un : étant donné un graphe non orienté, a-t-il, oui ou non, un cycle hamiltonien ?

La question posée est la vérité ou la fausseté d'une proposition de la forme $\exists y\ R(x,y)$. Pour le cycle hamiltonien, x est un graphe non orienté, y est une permutation des sommets, et R(x,y) est la proposition : "y est un cycle hamiltonien du graphe x".

Tous les problèmes de cette forme, qui satisfont aux restrictions suivantes sont considérés comme étant de complexité non-polynomiale (NP).

> **Restriction 1.** Quand la proposition $\exists y\ R(x,y)$ est vraie, il existe un y_0 dont la taille est bornée supérieurement par un polynôme en la taille de x tel que $R(x,y_0)$.
>
> **Restriction 2.** Le calcul de la valeur de vérité de la proposition R(x,y) peut être fait par un algorithme polynomial en la taille totale de x et de y.

Les problèmes de cette catégorie sont dits de la classe NP, abréviation de "non-déterministe polynomial". La raison de cette dénomination est la suivante. On peut considérer qu'il existe un pseudo-algorithme comportant deux phases ;

1. une **phase de divination** : deviner y_0 et l'écrire;
2. vérifier (en utilisant l'algorithme de calcul de R) que $R(x,y_0)$ est vraie.

Les hypothèses faites montrent que le temps total est polynomial en la taille de x. Naturellement, on peut tirer de ce pseudo-algorithme (non-déterministe dans la phase de divination) un vrai algorithme en remplaçant la divination par une exploration exhaustive, ce qui est possible, puisque la taille de la "solution" y_0 est bornée supérieurement par un polynôme en la taille de x. Le problème est que l'algorithme construit de cette façon est toujours exponentiel.

Remarquons que nous n'avons considéré jusqu'à maintenant, que des problèmes de décision. Considérons un problème d'optimisation quelconque. On peut toujours lui associer un problème de décision de la forme $\exists y\ R(x,y)$, où x sont les données du problème d'optimisation et y une valeur quelconque. Dans ce cas R(x,y) est la question suivante : "pour les données x, la solution optimale (minimale) est elle plus petite que y ?". Ce problème de décision est polynomialement réductible au problème d'optimisation. En effet, pour résoudre le problème de décision pour y, il suffit de résoudre le problème d'optimisation, qui donnera une solution et puis comparer cette

solution à y. Il en résulte que, s'il n'y a pas d'algorithme polynomial pour le problème de décision, il n'y en a pas non plus pour le problème d'optimisation.

1.4 Les problèmes NP-complets (NP-Hard)

Définition : Un problème est NP-complet ou NP-Hard s'il appartient à la classe NP, et si **tous** les problèmes de la classe NP lui sont polynomialement réductibles.

Autrement dit, si un problème particulier est NP-complet, la découverte d'un algorithme polynomial pour ce problème équivaudrait à la découverte d'un algorithme polynomial pour tous les problèmes de la classe NP.

2. Développement sous Windows 3.1 à l'aide de Visual Basic 3.0

2.1 Introduction

Une application Visual Basic est constituée de fenêtres, comme la plupart des applications Windows. La fenêtre principale de l'application, les fenêtres secondaires ou les boîtes de dialogue sont appelées *feuilles* (en anglais *forms*). Sur ces feuilles viennent se placer des *contrôles* qui sont les éléments de l'interface d'utilisation : libellés, champs de saisie, boutons, listes...

L'environnement Visual Basic comprend un ensemble d'outils qui permettent le dessin des contrôles, leur placement, la modification de leurs dimensions... Tout cela se fait de façon à la fois intuitive et visuelle. Le résultat d'une action est immédiatement visible sur l'écran. Le concepteur de l'application voit donc à tout instant l'application telle qu'elle apparaîtra à l'utilisateur lorsqu'elle sera opérationnelle.

Chaque objet ainsi créé, feuille ou contrôle, dispose d'un jeu de propriétés qui lui est propre. Le code est constitué d'instructions utilisant le langage BASIC, avec des extensions particulières pour tenir compte de l'**approche orientée objet** de Visual Basic. Les instructions agissent sur des propriétés ou des variables. On peut par exemple utiliser une instruction pour changer la valeur d'une propriété d'un objet, ou encore pour tester l'état d'une variable.

Mais la structure générale d'un programme Visual Basic est très différente de celle d'une application classique. Le code est associé à chaque objet de l'application, feuille ou contrôle.

2.2 Types de programmation

La programmation des applications Visual Basic est dite *événementielle* par opposition à la programmation *linéaire* traditionnelle. Ces deux approches sont comparées ci-après.

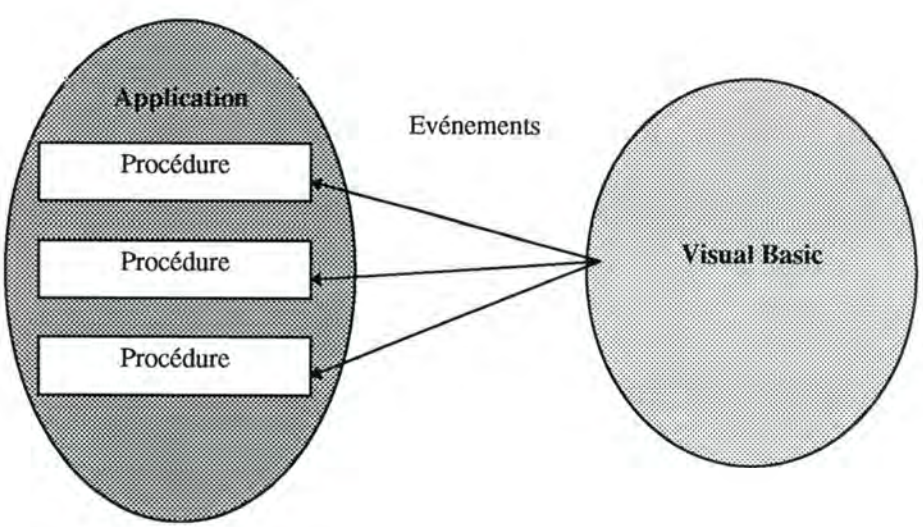
2.2.1 Programmation linéaire

La programmation linéaire est basée sur la structure suivante. Dans un programme traditionnel :

- Il y a un point d'entrée pour le programme.
- Il y a un point de sortie.
- Entre les deux il n'y a qu'une suite d'exécutions de procédures ou fonctions pour une exécution de l'application.
- Des appels à des procédures ou fonctions peuvent être faits.

2.2.2 Programmation sur événements

La programmation avec Visual Basic ne correspond plus du tout au schéma présenté précédemment, et c'est particulièrement déroutant pour le programmeur expérimenté. Une application est constituée d'un ensemble de *procédures indépendantes* les unes des autres.



Une procédure comprend des instructions écrites à l'aide du langage BASIC. Elle est associée à un *objet*, c'est-à-dire à un des éléments d'une feuille : la feuille elle-même ou bien un bouton, une liste, un champ de saisie... La procédure est appelée par Visual Basic lorsqu'il se produit un événement pour l'objet correspondant. S'il n'y a pas de code dans une procédure chargée de traiter un type d'événement pour un objet donné, il ne se passe rien de particulier lorsque l'événement est généré.

Pour écrire le code d'une application, il convient donc de déterminer les événements auxquels on souhaite réagir, et pour quels objets. Cela détermine les procédures dans lesquelles le code est écrit.

3. Les DLL (Dynamic Link Library)

Par DLL, on entend des bibliothèques de fonctions qui ne sont chargées en mémoire que lorsqu'on en a besoin. De plus, une fonction DLL chargée une fois est accessible à toutes les applications Windows, ce qui élimine la redondance des fonctions.

Nous avons développé un certain nombre de bibliothèques DLL dans le but d'augmenter la vitesse d'exécution des algorithmes. Le langage utilisé pour la création des DLL est *Pascal With Objects for Windows*.

4. Le fichier d'aide (Help File)

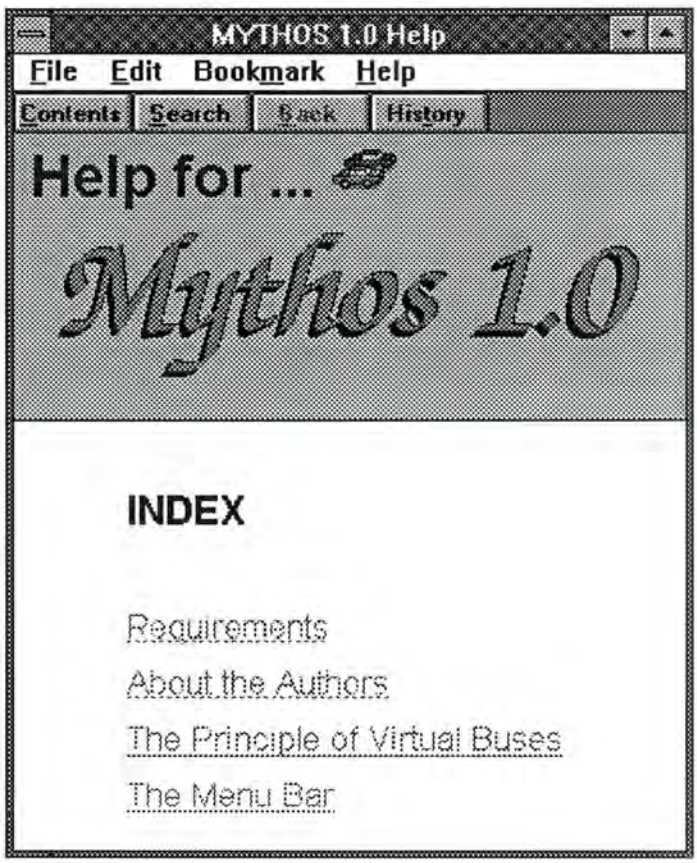
4.1 Introduction

Un fichier d'aide est un fichier qui contient du texte et des graphiques nécessaires pour communiquer à l'utilisateur des informations sur l'application qui est en train de s'exécuter. Chaque fichier d'aide est composé de plusieurs sujets que l'utilisateur peut sélectionner soit en cliquant sur les *hots spots* des sujets se trouvant dans l'*index* du fichier d'aide, soit en faisant une recherche à l'aide de *mots clés*, soit en appuyant sur la touche *F1* pendant l'utilisation de son application.

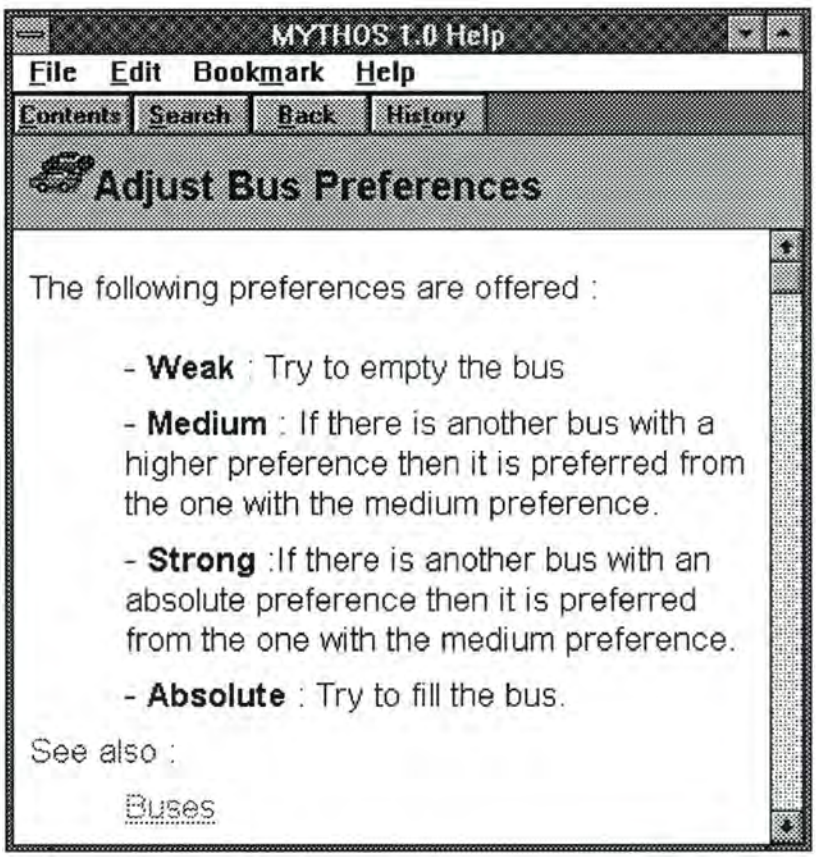
Sans entrer dans des détails techniques, nous allons exposer les éléments principaux nécessaires pour la création d'un tel fichier d'aide.

4.2 Quelques définitions

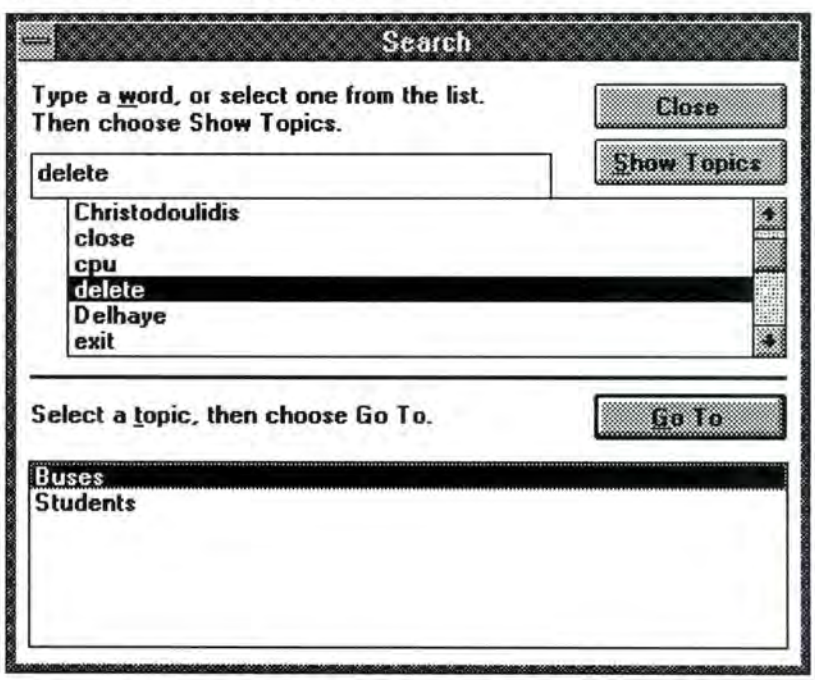
Le fichier d'aide peut être compris comme une arborescence d'informations. Au sommet de cette arborescence se trouve l'*index* du fichier d'aide. Intuitivement cet index correspond à l'index (table des matières) d'un livre. En cliquant sur un mot de l'index nous accédons au *sujet* concernant ce mot où à un sous-index (si le mot choisi représente une notion générale décomposée en plusieurs *sujets*). L'index que nous avons créé pour le logiciel Mythos est représenté par la figure suivante.



Un *sujet* est l'unité primaire d'un fichier d'aide. Cette unité primaire contient des informations sur un sujet bien délimité. Un exemple d'un *sujet* est donné dans la figure suivante :

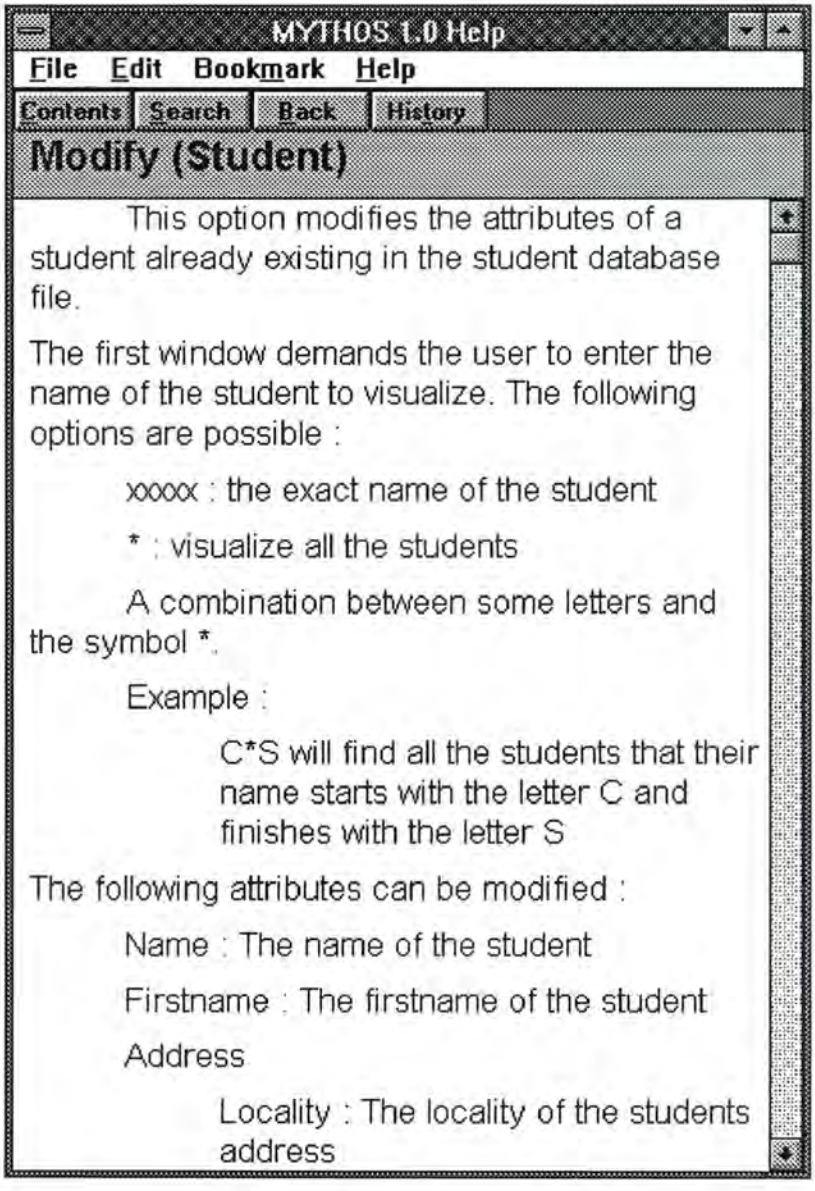


Un autre moyen pour accéder à un sujet (sans passer par l'index) est d'utiliser l'option *Search* qui donne lieu à la fenêtre suivante.



Le troisième moyen pour accéder à un *sujet* est de pousser sur la touche *F1* pendant l'utilisation du logiciel Mythos. L'aide a été programmée de telle manière que le sujet concernant l'écran courant de Mythos sera directement choisi! Donnons un exemple. Si l'utilisateur est en train de modifier les attributs des étudiants (*Modify*

Student) et si pendant cette modification, il pousse sur la touche *F1*, la fenêtre suivante lui sera proposée.



Le fichier d'aide développé, respecte à 100% les normes MS-WINDOWS 3.1 HELP FILE STANDARDS ver. 3.1. Pour plus d'informations et de détails, nous renvoyons le lecteur aux manuels spécialisés.

Listing

```
library RecSim;

uses wObjects;

Const n_col = 20;
      size = 31; (Working on 32 bit integers (Long Integer))

type ar=array[1..10000] of longInt;
     iar=array[1..10000] of Integer;
     car=array[1..n_col,1..800] of longInt;



procedure permute(var a :LongInt;var b :LongInt);
var c : LongInt;

begin
     c:=a;
     a:=b;
     b:=c;
end;

procedure Ipermute(var a :Integer;var b :Integer);
var c : Integer;

begin
     c:=a;
     a:=b;
     b:=c;
end;



function ith_bit_is_0 (FourByte:LongInt; i :integer) :Integer;


begin
        FourByte:=FourByte shr (i-1);

    if odd(FourByte) then ith_bit_is_0:=0
    else ith_bit_is_0:=-1;

end;


procedure Set_ith_bit_to (var FourByte : LongInt; i : Integer; value : Integer);

var Mask : Longint;

(Attention ! Four bytes means 31 BITS i.e. 1<=i<=31)

begin
    If (value = 0) and (ith_bit_is_0(FourByte, i)=0)  Then
       begin
                Mask:=1;
                Mask := Mask shl (i-1);
            FourByte := FourByte - Mask
       end
    Else
        if (value = 1) and (ith_bit_is_0(FourByte, i)=-1) Then
                begin
```

```
                    Mask:=1;
                    Mask := Mask shl (i-1);
                FourByte := FourByte + Mask
                end

End;


Function calc (i : Integer; j : Integer;  m_dim : Integer;  size : Integer;
               n_col : Integer; var lign : LongInt; var Col : Integer) : Integer;export;

    Var wor : LongInt;
        inte : LongInt;
        bit : Integer;

        k,l,m,p : LongInt;
Begin

    (inte:=trunc((i-1)*m_dim+j-((i+1)/2)*i))
(     r := (m_dim - (i + 1) / 2);
      r := r * (i - 1);
      r := r + j - ((i + 1) / 2);
      inte := Trunc(r);
)

      (if odd(i) then
      begin
         inte:=(m_dim-((i+1) div 2));
         inte:=inte * (i-1);
         inte:=inte +(j -((i+1) div 2));
      end
      else
         inte:=((m_dim-(i div 2)) * (i-1))-i+j;)

      if odd(i) then
      begin
         l:=(m_dim-((i+1) div 2));
         k:=(j -((i+1) div 2));
         inte:=longmul(l,(i-1)) ;
         inte:=inte + k;
      end
      else
        begin
          l:=(m_dim-(i div 2));
          k:=-i+j;
          inte:=longmul(l,(i-1)) ;
          inte:=inte + k;
        end;



      wor := (inte div size) + 1;
      bit := inte Mod size;

      If bit = 0 Then
      begin
         bit := size;
         Dec(wor);
      end;

      lign := (wor div n_col) + 1;
      Col := wor Mod n_col;

      If Col = 0 Then
```

```
    begin
       Col := n_col;
       Dec(lign);
    End;
    calc := bit;

End;


Function Edge (var virt_matrix:car;
               i : LongInt; j : LongInt;nb_vertex : LongInt) : LongInt;export;

{    'Indicates if there is an edge from the vertex i
   'to the vertex j in the adjacence matrix of the graph}

    var lign : LongInt;
        Col : Integer;
        bit : Integer;
Begin
    If Not (i = j) Then
    begin
        If i > j Then permute(i, j);
        bit := calc(i, j, nb_vertex, size, n_col, lign, Col);
        if lign<=800 then
        If ith_bit_is_0(virt_matrix[lign, Col], bit)=-1 Then
            Edge := 0
        Else
            Edge := 1
        Else
           if lign<=1600 then
             If ith_bit_is_0(virt_matrix[lign-800, Col], bit)=-1 Then
               Edge := 0
             Else
               Edge := 1
           Else
             if lign<=2400 then
               If ith_bit_is_0(virt_matrix[lign-1600, Col], bit)=-1 Then
                 Edge := 0
               Else
                 Edge := 1
             Else
               if lign<=3200 then
                 If ith_bit_is_0(virt_matrix[lign-2400, Col], bit)=-1 Then
                   Edge := 0
                 Else
                   Edge := 1
             Else Edge:=-20; {CATASTROPHE !!!}
    end
    Else
        Edge := 0;


End;




procedure uncolor_SV(var k : longint;var new_free_seats : ar;
                     var free_seats : ar;
                     var Color : ar;
                     var New_color : ar;
```

```
                     var SuperVertex : ar;
                         sv_1 : LongInt;
                        vertice : LongInt); export;

var j : longInt;
begin
    k := vertice;
    j := 0;

    Repeat
        If Color[k] <> 0 Then
        begin

            new_free_seats[Color[k]] := free_seats[Color[k]] + 1;
            New_Color[k] := 0;
            exit;
        End;
        k := SuperVertex[k];
        Inc(j);
    Until j = sv_1;

End;




Procedure KillEmptyBus (var free_seats:ar;var max_seats:ar;
                        var bus_pref : ar;var color : ar;
                        var nb_bus: LongInt;nb_vertex :LongInt;
                        bus_maxrec : LongInt; witch : longInt);export;

Const virt = 0;
      all = 1;

var g, d, s : LongInt;


Begin
If witch = virt Then {Only the virtual busses}
    g := bus_maxrec
Else  {Virtual busses and real one !}
    g := 1;
d := nb_bus;

While Not (g >= d + 1) do
  Begin
    If free_seats[d] = max_seats[d] Then
        {the bus d is empty}
        d := d - 1
    Else
        {the bus d is not empty}
        If (free_seats[g] <> max_seats[g]) Then
            {the bus g is not empty}
            g := g + 1
        Else
        {the bus g is empty}
        begin
            permute(free_seats[g], free_seats[d]);
            permute(max_seats[g], max_seats[d]);
            {strpermute(bus_name[g], bus_name[d]);}
            permute(bus_pref[g], bus_pref[d]);
            For s := 1 To nb_vertex do
                If color[s] = d Then
                    color[s] := g;
```

```
            d := d - 1;
            g := g + 1;
        End;
  End;

nb_bus := d;
End;


Function Select_a_vertice (var V_Mask:ar;var V_Counter:LongInt; var vert_upperbound : LongInt;
                           var vert_lowerbound : LongInt;nb_vertex:LongInt) : LongInt; export;
    var s : LongInt;
        i : LongInt;
Begin
    Repeat
        s := Random(vert_upperbound - vert_lowerbound + 1) + vert_lowerbound;
    Until V_Mask[s] = 0; (False)

    Select_a_vertice := s;
    V_Mask[s] := -1; (True)
    V_Counter := V_Counter + 1;

    If V_Counter < nb_vertex Then
    begin
        i := vert_lowerbound;
        While V_Mask[i] = -1 do
        begin
            Inc(vert_lowerbound);
            Inc(i);
        End;
        i := vert_upperbound;
        While V_Mask[i] = -1 do
        begin
            Dec(vert_upperbound);
            Dec(i);
        End;
    End;

End;

Function Select_a_color (var Color_Mask:ar;var color_counter:LongInt;
                         var color_upperbound : LongInt;
                         var color_lowerbound : LongInt;nb_bus:LongInt) : LongInt; export;
 var C : LongInt;
     i : LongInt;

Begin
        Repeat
            C :=  Random(color_upperbound - color_lowerbound + 1) + color_lowerbound;
        Until Color_Mask[C] = 0; (False)


        Select_a_color := C;
        Color_Mask[C] :=  -1; (True)
        Inc(color_counter);

        If color_counter < nb_bus Then
        Begin
            i := color_lowerbound;
            While Color_Mask[i] = -1 do (True)
            begin
                Inc(color_lowerbound) ;
                Inc(i);
            end;
            i := color_upperbound;
             While Color_Mask[i] =  -1 (True) do
```

```
            begin
                Dec(color_upperbound);
                Dec(i);
            end;
        End;

End;

procedure Init_New (var New_Color : ar; var Color : ar;nb_vertex :LongInt);export;

var j : LongInt;
begin
{    'Initilalisation of the new_colors}

        For j := 1 To nb_vertex do
        begin
          new_color[j] := color[j];
        end;
end;

procedure calc_energy (var bus_pref : ar; var max_seats : ar;
                       var Free_seats : ar; var New_Free_seats : ar;
                       var energy : longInt; var new_energy : longInt;nb_bus:longint);export;
var j: longint;
begin

    energy := 0;
    new_energy := 0;

    For j := 1 To nb_bus do
    begin
        energy := energy + (bus_pref[j] * (max_seats[j] - Free_Seats[j]));
        new_energy := new_energy + bus_pref[j] * (max_seats[j] - New_Free_seats[j]);
    end;
end;



exports
   uncolor_SV index 1,
   KillEmptyBus index 2,
   Select_a_vertice index 3,
   Select_a_color index 4,
   edge index 5,
   calc index 6,
   Init_New index 7,
   calc_energy index 8;

begin
end.
```

```
'Constants declarations
'-----------------------------------------------------------
Option Explicit

'Program Name
Global Const nom_prog = "Mythos 1.0"

'Busses preferences
Global Const Weak_pref = 0
Global Const Medium_pref = 10
Global Const Strong_pref = 20
Global Const Absolute_pref = 30

'Preference for each virtual bus
Global Const virtual_bus_pref = Weak_pref
Global Const virtual_bus_good_pref = Strong_pref

'Maximal number of seats for each virtual bus
Global Const virtual_bus_max_seats = 2

Global Const n_col = 20
Global Const sIze = 31 'Working on 32 bit integers (Long Integer)

'The precision of the time slice
Global Const inter = 1

'Message box constants
Global Const MB_OK = 0, MB_OKCANCEL = 1                 ' Define buttons.
Global Const MB_YESNOCANCEL = 3, MB_YESNO = 4
Global Const MB_ICONSTOP = 16, MB_ICONQUESTION = 32     ' Define Icons.
Global Const MB_ICONEXCLAMATION = 48, MB_ICONINFORMATION = 64
Global Const MB_DEFBUTTON2 = 256
Global Const IDOK = 1, IDCANCEL = 2, IDYES = 6, IDNO = 7' Define other.

'Arrange Method for MDI Forms
Global Const CASCADE = 0
Global Const TILE_HORIZONTAL = 1
Global Const TILE_VERTICAL = 2
Global Const ARRANGE_ICONS = 3

'The length of a record of the ADJMATRIX FILE in integer numbers (2 Bytes)
Global Const Mtx_Rec_Length = 100

'The length of a record of the SuperVertex FILE in integer numbers (2 Bytes)
Global Const SV_Rec_Length = 100


'Types declarations
'-----------------------------------------------------------

'The user defined data record for the Project file
Type ProRecord
    Project_Name As String * 40
    Student_File_Name As String * 60
    Bus_File_Name As String * 60
    Map_File_Name As String * 60
    InformatIon As String * 100
End Type

'The user defined data record for the SuperVertex file
Type SVRecord
      ligne(1 To SV_Rec_Length) As Integer
End Type

'The user defined data record for the Bus file
Type BusRecord
    BusNum As Long
    BusName As String * 20
    BusSeats As Long
    BusPreference As Long
End Type

Type MatAdjRecord
      ligne(1 To Mtx_Rec_Length) As Long
End Type

'Global variables declarations
'-----------------------------------------------------------

'Project's file variables
Global FileProNum
Global ProEntry As ProRecord

'Students'file variables
Global Stud_MaxRec As Integer

'SuperVertex's file variables
Global FileSVNum, SV_MaxRec, SV_Index
Global SVEntry As SVRecord


'Number of students in the open project
'-------------------------------------------------
Global nb_stu As Long

'Number of Vertex in the open project
Global nb_vertex As Long

'Bus' file variables
Global FileBusNum, Bus_MaxRec, Bus_Index
Global BusEntry As BusRecord

'Number of busses in the open project
Global nb_bus As Long

'The name of each bus
Global bus_name() As String

'The number of free seats in each bus
Global free_seats() As Long

'The level of preference for each bus
Global bus_pref() As Long

'The number of seats for each bus
Global max_seats() As Long

'The number of seats of the biggest bus
Global maximum_seats As Long

'Adj Matrix's file variables
Global FileMatNum
Global MatadjEntry As MatAdjRecord

'Virtual Adj Matrix
Global Virt_Matrix() As Long

'The number of integer in the Virtual Adj Matrix
Global nb_integer As Long
```

```
'Color Vertice
Global Color() As Long

'Super-Vertex tab
Global SuperVertex() As Long

'The number of optimizations made
Global nb_opti As Integer

'The 10 occurences of frmSolutions frame
Global frmSolutions(0 To 9) As New frmSolution

'Flag that incates if Cancel button has been pushed
Global Cancel_Flag As Integer

'Indicates where we come from
Global comefrom

Global FileName_Var As String

Global sleeper As Integer

Global FullCpu As Integer

Global ttest(1 To 10) As Integer

Function even (ByVal TwoByte As Integer) As Integer

    even = ((TwoByte Mod 2) = 0)

End Function

Sub permute (A, B)

    Dim C

    C = A
    A = B
    B = C

End Sub
```

```
VERSION 2.00
Option Explicit

    Const MaxInt = 32767

    Dim next_node() As Integer
    Dim prev_node() As Integer
    Dim first_node As Integer
    Dim Card_list As Integer   'cardinal of the list of temporary nodes
    Dim n As Long
    Dim EFG() As Integer 'an node is in E if we know the min time
                         '           in F       don't know the minimum time but
                         '                                  we know we can't reach it
                         '           in G       know NOTHING

    Const E = 1
    Const G = 2
    Const F = 3

    Dim tbFlow1() As Integer
    Dim tbFlow2() As Integer
    Dim lgtb As Integer

    Dim dico() As Integer   'Dictionary of the adjacents nodes

Sub cmdAdjPref_Click ()
    Call mnuAdjustBusPreferences_Click
End Sub

Sub cmdCloseProject_Click ()
    mnuCloseProject_Click
End Sub

Sub cmdCpuUse_Click ()
    Call mnuFullCpuUse_Click
End Sub

Sub CmdHelp_Click ()

    Dim helpval As Integer
    Dim dwData As Long

    dwData = 1
    helpval = WinHelp(hWnd, app.Path & "\Mythos.Hlp", HELP_CONTEXT, dwData)

End Sub

Sub CmdMemo_Click ()
    frmMemo.Show 1
End Sub

Sub cmdOpenProject_Click ()
    mnuOpenProject_Click
End Sub

Sub cmdOptimize_Click ()

    If mnuRunOptimizer.Enabled Then
       mnuRunOptimizer_Click
    Else
        If mnuRunAgainOptimizer.Enabled Then
            mnuRunAgainOptimizer_Click
        Else mnuPreOptimization_Click
        End If
    End If
```

```
End Sub

Sub cmdPrint_Click ()

    Dim k As Integer, l As Integer, ind As Integer
    Dim StTime As String, StHour As String


    If mnuToFile.Checked Then
        frmPrintToFile.Show 1
        DoEvents
    End If
    If comefrom = "Cancel" Then
        comefrom = ""
        Exit Sub
    End If

    panOptimizationInfo.Caption = "Printing in process"
    'Empty the Solution Data Base

    If Solution_Project_Table.RecordCount <> 0 Then
        Solution_Project_Table.MoveFirst
        Do Until Solution_Project_Table.EOF
            Solution_Project_Table.Edit
            Solution_Project_Table.Delete
            Solution_Project_Table.MoveNext
        Loop
    End If

    If Solution_Bus_Table.RecordCount <> 0 Then
        Solution_Bus_Table.MoveFirst
        Do Until Solution_Bus_Table.EOF
            Solution_Bus_Table.Edit
            Solution_Bus_Table.Delete
            Solution_Bus_Table.MoveNext
        Loop
    End If

    If Solution_Student_Table.RecordCount <> 0 Then
        Solution_Student_Table.MoveFirst
        Do Until Solution_Student_Table.EOF
            Solution_Student_Table.Edit
            Solution_Student_Table.Delete
            Solution_Student_Table.MoveNext
        Loop
    End If

    'Fill the Solution Data Base

    Solution_Project_Table.AddNew
    Solution_Project_Table("Name") = ProEntry.Project_Name
    Solution_Project_Table("Memo") = ProEntry.Information
    Solution_Project_Table.Update

    Bus_Index = 1
    Do Until Bus_Index = Bus_Maxrec
        Get FileBusNum, Bus_Index, BusEntry
        Solution_Bus_Table.AddNew
        Solution_Bus_Table("Name") = BusEntry.BusName
        Solution_Bus_Table("Capacity") = BusEntry.BusSeats
        Select Case BusEntry.BusPreference
        Case Weak_pref
            Solution_Bus_Table("Preference") = "Weak"
        Case Medium_pref
            Solution_Bus_Table("Preference") = "Medium"
        Case Strong_pref
```

```
            Solution_Bus_Table("Preference") = "Strong"
        Case Absolute_pref
            Solution_Bus_Table("Preference") = "Absolute"
        End Select
        Bus_Index = Bus_Index + 1
        Solution_Bus_Table.Update
    Loop

    For k = 1 To nb_bus

        For l = 1 To nb_vertex

            If Color(l) = k Then
                ind = 1
                StuInfoEntry = student_info(1)
                Do While StuInfoEntry.vertex_no < l
                    ind = ind + 1
                    StuInfoEntry = student_info(ind)
                Loop
                StuInfoEntry = student_info(ind)
                StTime = TimeSerial(Left(StuInfoEntry.MaxHour, 2), (Right(StuInfoEntry.MaxHour, 2) - (inter * (StuInfoEntry.vertex_no - 1))),
                StHour = Left(StTime, 5)
                If Right(StHour, 1) = ":" Then
                    StHour = " " & Left(StHour, 4)
                End If

                Solution_Student_Table.AddNew
                Solution_Student_Table("Bus_Name") = Trim$(bus_name(k))
                Solution_Student_Table("Name") = StuInfoEntry.name
                Solution_Student_Table("First_Name") = StuInfoEntry.firstname
                Solution_Student_Table("Time") = StHour
                Solution_Student_Table("Locality") = StuInfoEntry.locality
                Solution_Student_Table("Street") = StuInfoEntry.street
                Solution_Student_Table.Update

            End If

        Next l

    Next k

    report1.Action = 1
    panOptimizationInfo.Caption = "Printing completed"
End Sub

Sub cmdStopOptimize_Click ()
    mnuStopOptimize_Click
End Sub

Sub cmdSysinfo_Click ()

    Winfo.Show 1

End Sub

Sub ComRQuality_Click ()
        Select Case ComRQuality.ListIndex
        Case 0
            Call mnuNormal_Click
        Case 1
            Call mnuHight_Click
        Case 2
            Call mnuVeryHight_Click
        End Select

End Sub
```

```
Sub ComUserMode_Click ()
        Select Case ComUserMode.ListIndex
        Case 0
            Call mnuNovice_Click
        Case 1
            Call mnuBeginer_Click
        Case 2
            Call mnuExpert_Click
        End Select

End Sub

Sub create_dico ()

    Dim i As Integer, j As Integer
    Dim cur_No As Integer
    ReDim dico(1 To n, 9) As Integer

    Set snFlow = dbMap.CreateSnapshot("SELECT No1,No2,weigtht FROM Flow Order by No1")
    i = 1
    j = 1

    Do While Not snFlow.EOF
        cur_No = snFlow("No1")
        dico(i, j) = snFlow("No2")
        dico(i, j + 1) = snFlow("weigtht")
        snFlow.MoveNext
        j = j + 2
        If Not snFlow.EOF Then
          If snFlow("No1") <> cur_No Then
            Do While j <= 7
                dico(i, j) = -1
                dico(i, j + 1) = -1
                j = j + 2
            Loop
            i = i + 1
            j = 1
          End If
        End If
    Loop
    Do While j <= 7
        dico(i, j) = -1
        dico(i, j + 1) = -1
        j = j + 2
    Loop

End Sub

Static Sub init_controls_for_SA ()

mnuBusses.Enabled = False
mnuOptimize.Enabled = False
cmdOptimize.Enabled = False
cmdOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nfeuve.ico")
cmdStopOptimize.Enabled = False
cmdStopOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nstop.ico")


End Sub

Sub MDIForm_Load ()

' The form is horizontally and vertically centered when loaded.

    Top = Screen.Height / 2 - Height / 2
```

```
    Left = Screen.Width / 2 - Width / 2

' Go to the right directory

    FullCpu = False

    mnuCloseProject.Enabled = False
    cmdCloseProject.Enabled = False
    cmdCloseProject.Picture = LoadPicture(app.Path & "\ico\" & "nfilcl.ico")
    mnuStudents.Enabled = False
    mnuBusses.Enabled = False
    mnuOptimize.Enabled = False
    cmdOptimize.Enabled = False
    cmdOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nfeuve.ico")
    cmdStopOptimize.Enabled = False
    cmdStopOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nstop.ico")

    panUserLevel.Caption = "Novice User"
    panProjectName.Caption = nom_prog & "  Create / Open a Project "
    report1.ReportFileName = app.Path & "\data\mythos.rpt"
    ComUserMode.AddItem "Novice"
    ComUserMode.AddItem "Beginner"
    ComUaerMode.AddItem "Expert"
    ComUserMode.ListIndex = 0
    ComRQuality.AddItem "Normal"
    ComRQuality.AddItem "High"
    ComRQuality.AddItem "Very High"
    ComRQuality.ListIndex = 0
End Sub

Sub MDIForm_QueryUnload (Cancel As Integer, UnloadMode As Integer)

    Dim Response


   ' Get user response.
    Response = MsgBox("This will end-up your session with " & nom_prog, MB_ICONINFORMATION + MB_OKCANCEL, nom_prog)

    If (Response = IDOK) Then    ' Evaluate response

        Close
        Unload Me
        End
    Else
        Cancel = True
    End If

End Sub

Sub mnuAbout_Click ()

    'Dim Response

   ' Get user response.
    'Response = MsgBox(nom_prog & Chr$(10) & Chr$(10) & "Christodoulidis Christos" & Chr$(10) & "Delhaye Marc" & Chr$(10) & Chr$(10) & "199

    'If Response = IDOK Then    ' Evaluate response

    'End If
     frmAbout.Show 1

End Sub

Sub mnuAddBus_Click ()

    comefrom = "Add Bus"
```

```
    frmBusses.Caption = comefrom


    frmBusses.Show

End Sub

Sub mnuAddStud_Click ()

    comefrom = "Add Student"
    frmstudents.Caption = comefrom
    frmstudents.Show 1

End Sub

Sub mnuAdjustBusPreferences_Click ()
    frmBusAdjust.Show
End Sub

Sub mnuArrangeIcons_Click ()

    'Arrange all iconized child forms
    MdiMain.Arrange arrange_icons


End Sub

Sub mnuBeginer_Click ()

    panUserLevel.Caption = "Beginer User"

    mnuNovice.Checked = False
    mnuBeginer.Checked = True
    mnuExpert.Checked = False
    ComUserMode.ListIndex = 1

End Sub

Sub mnuBusses_Click ()

    If mnuRunOptimizer.Enabled = True Then
        If Bus_Maxrec = 1 Then
            mnuViewBus.Enabled = False
            mnuModifyBus.Enabled = False
            mnuDeleteBus.Enabled = False
        Else
            mnuViewBus.Enabled = True
            mnuModifyBus.Enabled = True
            mnuDeleteBus.Enabled = True
        End If
    End If

End Sub

Sub mnuCloseProject_Click ()

   Dim Response


    ' Get user response.
    Response = MsgBox("Do you really want to close your Project ?", MB_ICONQUESTION + MB_YESNO, nom_prog)

    If (Response = IDYES) Then    ' Evaluate response


        Close
```

```
        MdiMain!panProjectName.Caption = "There is no open project"
        Unload frmGraph
        Unload frmPoptOk

        mnuOpenProject.Enabled = True
        cmdOpenProject.Enabled = True
        cmdCloseProject.Enabled = False
        cmdOpenProject.Picture = LoadPicture(app.Path & "\ico\" & "filop.ico")
        cmdCloseProject.Picture = LoadPicture(app.Path & "\ico\" & "nfilcl.ico")
        mnuNewProject.Enabled = True
        mnuCloseProject.Enabled = False
        mnuAddStud.Enabled = True
        mnuStudents.Enabled = False
        mnuBusses.Enabled = False
        mnuOptimize.Enabled = False
        cmdOptimize.Enabled = False
        cmdOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nfeuve.ico")
        cmdStopOptimize.Enabled = False
        cmdStopOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nstop.ico")
        cmdmemo.Enabled = False
        cmdmemo.Picture = LoadPicture(app.Path & "\ico\" & "nmemo.ico")

        Load frmAlice
        panOptimizationInfo.Caption = " "
    End If
End Sub

Sub mnuDeleteStud_Click ()

    frmFindStu!lblAction.Caption = "Delete Student"
    frmFindStu.Show 1
    comefrom = "Delete Student"

End Sub

Sub mnuExit_Click ()

    Dim Cancel As Integer

    Cancel = False
    Unload Me

End Sub

Sub mnuExpert_Click ()

    panUserLevel.Caption = "Expert User"

    mnuNovice.Checked = False
    mnuBeginer.Checked = False
    mnuExpert.Checked = True
    ComUserMode.ListIndex = 2

End Sub

Sub mnuFindStud_Click ()

    comefrom = "Find Student(s)"
    frmstudents.Caption = comefrom
    frmFindStu.Show 1

End Sub

Sub mnuFullCpuUse_Click ()
    If mnuFullCpuUse.Checked Then
        mnuFullCpuUse.Checked = False
```

```
        FullCpu = False
        panCpuUse.Caption = "Time Sharing"
        cmdCpuUse.Picture = LoadPicture(app.Path & "\ico\" & "nmchip.ico")
    Else mnuFullCpuUse.Checked = True
        panCpuUse.Caption = "Full CPU used"
        FullCpu = True
        cmdCpuUse.Picture = LoadPicture(app.Path & "\ico\" & "mchip.ico")
    End If
End Sub

Sub mnuHelpIndex_Click ()
    Dim helpval As Integer
    Dim dwData As Long

    dwData = 1
    helpval = WinHelp(hWnd, app.Path & "\Mythos.Hlp", HELP_CONTEXT, dwData)
End Sub

Sub MnuHigh_Click ()

    Result_precision = Hight
    If mnuNormal.Checked Then
        mnuNormal.Checked = False
    ElseIf mnuVeryHight.Checked Then
        mnuVeryHight.Checked = False
    End If
    mnuHight.Checked = True

End Sub

Sub mnuHight_Click ()

    Result_precision = Hight
    If mnuNormal.Checked Then
        mnuNormal.Checked = False
    ElseIf mnuVeryHight.Checked Then
        mnuVeryHight.Checked = False
    End If
    mnuHight.Checked = True
    ComRQuality.ListIndex = 1
End Sub

Static Sub mnuMap_Click ()

Dim i As Integer

For i = 1 To nb_vertex - 1
    Call Set_Edge(i, i + 1)
    Call Set_Edge(i + 1, i)
Next i

Call Set_Edge(1, nb_vertex)
Call Set_Edge(nb_vertex, 1)



End Sub

Sub mnuModifyBus_Click ()

    comefrom = "Modify Bus"
    Bus_Index = 1
    frmBusses.Caption = comefrom
```

```
    frmBusses.Show

End Sub

Sub mnuModifyStud_Click ()

    frmFindStu!lblAction.Caption = "Modify Student"
    comefrom = "Modify Student"
    frmFindStu.Show 1


End Sub

Sub mnuNewProject_Click ()

    Dim comp, k As Integer

    nb_vertex = 0

    'Ask for a new project name
    frmNewProject.Show 1


End Sub

Sub mnuNormal_Click ()

    Result_precision = Normal
    If mnuHight.Checked Then
        mnuHight.Checked = False
    ElseIf mnuVeryHight.Checked Then
        mnuVeryHight.Checked = False
    End If
    mnuNormal.Checked = True
    ComRQuality.ListIndex = 0
End Sub

Sub mnuNovice_Click ()

    panUserLevel.Caption = "Novice User"

    mnuNovice.Checked = True
    mnuBeginer.Checked = False
    mnuExpert.Checked = False
    ComUserMode.ListIndex = 0

End Sub

Sub mnuOpenProject_Click ()

    Dim i As Integer
    Dim RecordLen As Integer
    Dim Confirm As Integer
    Dim filename As String

    Confirm = True

    Cancel_Flag = False


    'Open the file frame
    frmFile.Caption = "Open Project File"
    frmFile.Show 1
```

```
    If Not Cancel_Flag Then
        Unload frmAlice
        DoEvents

        Get FileProNum, 1, ProEntry

       'Open Student File"
        DbStudentOpener (Trim$(ProEntry.Student_File_Name))
        Set Name_Table = DbStu.OpenTable("SchoolBoy")
        i = 0    ' Keep track of records.
        Name_Table.MoveFirst
        Do While Not Name_Table.EOF
            i = i + 1
            Name_Table.MoveNext
        Loop
        Stud_MaxRec = i

       'Open SuperVertex File"
        SV_Index = 1    ' Keep track of records.
        RecordLen = Len(SVEntry)
        FileSVNum = FileOpener(Left$(Trim$(ProEntry.Student_File_Name), Len(Trim$(ProEntry.Student_File_Name)) - 4) & ".SVX", RandomFile, Re
        Get FileSVNum, SV_Index, SVEntry
        nb_vertex = SVEntry.ligne(1)


        '"Open Matrix File"
        RecordLen = Len(MatAdjEntry)
        FileMatNum = FileOpener(Left$(Trim$(ProEntry.Student_File_Name), Len(Trim$(ProEntry.Student_File_Name)) - 4) & ".MTX", RandomFile, R


        'Open Bus File
        Bus_Index = 0    ' Keep track of records.
        RecordLen = Len(BusEntry)
        FilebusNum = FileOpener(Trim$(ProEntry.Bus_File_Name), RandomFile, RecordLen, Confirm)
        Do While Not EOF(FilebusNum)
            Bus_Index = Bus_Index + 1
            Get FilebusNum, Bus_Index, BusEntry
        Loop
        Bus_Maxrec = Bus_Index

        'Open Map File
        DbMapOpener (Trim$(ProEntry.Map_File_Name))

        'Open Solution Data Base
        Set DbSolu = OpenDatabase(app.Path & "\data\solut.mdb", True, False)
        Set Solution_Project_Table = DbSolu.OpenTable("Project")
        Set Solution_Bus_Table = DbSolu.OpenTable("Bus")
        Set Solution_Student_Table = DbSolu.OpenTable("Student")


        mnuOpenProject.Enabled = False
        mnuNewProject.Enabled = False
        cmdOpenProject.Enabled = False
        cmdCloseProject.Enabled = True
        cmdOpenProject.Picture = LoadPicture(app.Path & "\ico\" & "nfilop.ico")
        cmdCloseProject.Picture = LoadPicture(app.Path & "\ico\" & "filcl.ico")
        mnuCloseProject.Enabled = True
        mnuStudents.Enabled = True
        mnuBusses.Enabled = True
        mnuOptimize.Enabled = True
        cmdOptimize.Enabled = True
        cmdOptimize.Picture = LoadPicture(app.Path & "\ico\" & "feuve.ico")
        cmdStopOptimize.Enabled = False
        cmdStopOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nstop.ico")
        mnupreoptimization.Enabled = True
```

```
        mnuRunOptimizer.Enabled = False
        mnuRunAgainOptimizer = False
        mnuStopOptimizer = False
        cmdmemo.Enabled = True
        cmdmemo.Picture = LoadPicture(app.Path & "\ico\" & "memo.ico")
        MdiMain!panProjectName.Caption = Trim$(ProEntry.Project_Name)
        frmMemo.Show 1
    End If
    Close FileProNum


End Sub

Sub mnuPreOptimization_Click ()

Dim it As Long
Dim it2 As Long
Dim Response


    Load frmPreOpt

    mnuOptions.Enabled = False
    mnuWindow.Enabled = False
    mnuHelp.Enabled = False
    mnuStudents.Enabled = False
    mnuFile.Enabled = False
    cmdCloseProject.Enabled = False
    cmdCloseProject.Picture = LoadPicture(app.Path & "\ico\" & "nfilel.ico")
    mnuBusses.Enabled = False
    mnuOptimize.Enabled = False
    cmdOptimize.Enabled = False
    cmdOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nfeuve.ico")
    panOptimizationInfo.Caption = "PreOptimization in process"

    DoEvents

    'Student number
    Set DsStudent = DbStu.CreateDynaset("Select * from SchoolBoy ORDER by AdressNo")
    nb_stu = 0
    DsStudent.MoveFirst
    Do While Not DsStudent.EOF
        DsStudent.MoveNext
        nb_stu = nb_stu + 1
    Loop

    ReDim student_info(1 To nb_stu) As Student_Info_type

    Dim Card_G As Integer

    '-------------------------------------------------
    '-----                                       ---------
    '-----      Creation of the SuperVertex      ---------
    '-----                                       ---------
    '-------------------------------------------------

    Dim comp As Integer
    Dim k As Long
    Dim ll As Long
    Dim diff As Integer
    Dim est_time As Integer

    frmPreOpt!labStep.Caption = "1"
    nb_vertex = 0
    comp = 1
    DsStudent.MoveFirst
```

```
    For k = 1 To nb_stu
        diff = Left(DsStudent("MaxHour"), 2) * 60 + (Right(DsStudent("MaxHour"), 2))
        diff = diff - (Left(DsStudent("MinHour"), 2) * 60 + Right(DsStudent("MinHour"), 2))
        diff = diff / inter
        nb_vertex = nb_vertex + diff
        ReDim Preserve SuperVertex(nb_vertex)
        Call Create_a_SV(comp, comp + diff - 1)
        comp = comp + diff
        DsStudent.MoveNext
        frmPreOpt!GauPreopt.FloodPercent = (k / nb_stu) * 80
    Next k

    'Call Save_Svert

    DsStudent.MoveFirst
    it = 1
    it2 = 1
    Do While Not DsStudent.EOF
        Do While SuperVertex(it) > it
            it = it + 1
        Loop
        StuInfoEntry.vertex_no = it
        StuInfoEntry.name = DsStudent("Name")
        StuInfoEntry.firstname = DsStudent("Firstname")
        StuInfoEntry.street = DsStudent("Street")
        StuInfoEntry.locality = DsStudent("Locality")
        StuInfoEntry.MaxHour = DsStudent("MaxHour")
        StuInfoEntry.MinHour = DsStudent("MinHour")
        StuInfoEntry.AdressNo = DsStudent("AdressNo")
        student_info(it2) = StuInfoEntry
        it2 = it2 + 1
        it = it + 1
        DsStudent.MoveNext
    Loop

    frmPreOpt!GauPreopt.FloodPercent = 100

    'ReDimension Virtual Adjacence Matrix
    nb_integer = ((((((nb_vertex) ^ 2) - nb_vertex) \ 2) + 1) \ size) + 1
    ReDim Virt_Matrix(((nb_integer \ n_col) + 1), n_col)

    'ReDimension Color Vertice
    ReDim Color(nb_vertex)  As Long

    'Initialise the virtual matrix with 1
    For k = 1 To ((nb_integer \ n_col) + 1)
       For ll = 1 To n_col
           Virt_Matrix(k, ll) = 2147483647 'means byte=111...
       Next ll
    Next k

    '-------------------------------------------------
    '-----                                       ---------
    '-----  Creation of the Adjacence Matrix     ---------
    '-----                                       ---------
    '-------------------------------------------------

    Dim Start, finish
    Dim i As Integer
    Dim j As Integer
    Dim l As Integer
    Dim Start_Node As Integer
    Dim MinTime_Start_Node As Integer
    Dim MaxTime_Start_Node As Integer
    Dim X As Integer
    Dim Y As Integer
```

```
    Dim Z As Integer
    Dim minimum As Integer
    Dim chosen As Integer
    Dim snStu1 As snapshot
    Dim snStu2 As snapshot
    Dim h1 As Integer, h2 As Integer, h2Init As Integer
    Dim trip As Integer
    Dim first_one_1 As Integer
    Dim first_one_2 As Integer
    Dim next_one_I As Integer
    Dim next_one_2 As Integer
    Dim weight As Integer
    Dim iter_number As Long
    Dim cur_node As Integer
    ReDim EFG(1 To nb_stu) As Integer
    Dim student2_No As Integer
    Dim student1_No As Integer
    Dim prev_chosen As Integer
    Dim MinTime_prev_chosen As Integer

    '----------------------------------------
    'Initialisation
    '----------------------------------------

    Set snStu1 = DbStu.CreateSnapshot("SELECT MinHour,MaxHour,AdressNo FROM Schoolboy ORDER by AdressNo")
    Set snStu2 = snStu1.Clone()

    'Count the number of nodes in the Map
    '----------------------------------------
    Street_Table.MoveFirst
    n = 0
    Do While Not Street_Table.EOF
        Street_Table.MoveNext
        n = n + 1
    Loop

Start = Timer

    create_dico

    iter_number = 0
    frmPreOpt!labStep.Caption = "2"
    frmPreOpt!GauPreopt.FloodPercent = 0
    DoEvents

    ReDim next_node(1 To n) As Integer
    ReDim prev_node(1 To n) As Integer

    ReDim MinTime(1 To n) As Integer
    'ReDim ComingFrom(1 To n) As Integer
    ReDim d(1 To n) As Integer

    first_one_1 = 1
    next_one_I = 1
    X = I

    snStu1.MoveFirst
    student1_No = 1

    Do
        'Start_Node = student1_No
        Start_Node = snStu1("AdressNo")
        MinTime_Start_Node = Left(snStu1("MinHour"), 2) * 60 + Right(snStu1("MinHour"), 2)
        MaxTime_Start_Node = Left(snStu1("MaxHour"), 2) * 60 + Right(snStu1("MaxHour"), 2)
```

```
        'Init EFG
        '   1. all students with adressNo=Start_Node are in E
        '   2. all students with minTime(Start_Node)>maxTime are in F
        '   3. the other ones are in G
        Card_G = nb_stu
        For I = 1 To nb_stu
            StuInfoEntry = student_info(i)
            If (StuInfoEntry.AdressNo <> Start_Node) Then '2
                h2 = Left(StuInfoEntry.MaxHour, 2) * 60 + Right(StuInfoEntry.MaxHour, 2)
                If (MinTime_Start_Node >= h2) Then
                    EFG(i) = F    'we don't know the min time but we know we can't reach them
                    Card_G = Card_G - 1
                Else
                    EFG(i) = G    'we don't know anything
                End If
            Else '1
            ' StuInfoEntry.AdressNo = Start_Node Then
                EFG(i) = E 'we know the min time: it is =0
                Card_G = Card_G - 1
            End If
        Next i

        'Init list of temporary nodes
        first_node = 1
        Card_list = n
        For j = 1 To n - 1
            next_node(j) = j + 1
            prev_node(j + 1) = j
            MinTime(j) = w(Start_Node, j)
        Next j
        next_node(n) = 0 'nil
        prev_node(1) = 0 'nil
        j = n
        MinTime(j) = w(Start_Node, j)
        MinTime(Start_Node) = 0
        Call Remove_node(Start_Node)

        'Find the best node out of the list of temporary nodes
        'minimum = MaxInt
        'cur_node = first_node
        'For l = 1 To Card_list
        '    Call Best_choice(MinTime(cur_node), minimum, cur_node, chosen)
            '----------------------------------------
            'If MinTime(cur_node) < minimum Then
            '    minimum = MinTime(cur_node)
            '    Chosen = cur_node
            'End If
            '----------------------------------------
        '    cur_node = next_node(cur_node)
        'Next l
        'Call Remove_node(chosen)

        Do While Card_G > 0
            If Not FullCpu Then
            'Give a chance to Windows
                DoEvents
            End If

            'Find the best node out of the list of temporary nodes
            minimum = MaxInt
            cur_node = first_node
            For l = 1 To Card_list
                Call Best_choice(MinTime(cur_node), minimum, cur_node, chosen)
                '----------------------------------------
                'If MinTime(cur_node) < minimum Then
                '    minimum = MinTime(cur_node)
```

```
            '    chosen = cur_node
            'End If
            '----------------------------------------

            cur_node = next_node(cur_node)
        Next l
        If MinTime(chosen) = MaxInt Then
            Response = MsgBox("Fuck You, Prickface !  DataBase is corrupted", MB_ICONSTOP + MB_OK, nom_prog)
            End
        End If
        Call Remove_node(chosen)

        'Upgrade Nodes
        l = 1
        Do While (l <= 7) And (dico(chosen, l) <> -1)
            'Call Compare_weight(, MinTime(cur_node), weight)

            If MinTime(chosen) + dico(chosen, l + 1) < MinTime(dico(chosen, l)) Then
                MinTime(dico(chosen, l)) = MinTime(chosen) + dico(chosen, l + 1)
            End If
            l = l + 2
        Loop


        'Put in the E set every student that the adress is = chosen
        'l = 1
        'Do While l <= nb_stu
        '    StuInfoEntry = student_info(l)
        '    If StuInfoEntry.AdressNo <= Chosen Then
        '        If StuInfoEntry.AdressNo = Chosen Then
        '            EFG(l) = E 'we know the min time
        '            Card_G = Card_G - 1
        '        End If
        '    Else Exit Do
        '    End If
        '    l = l + 1
        'Loop

        '-------------------------------------

    'Upgrade EFG
    l = 1
    Do While l <= nb_stu
    If EFG(l) = G Then
        StuInfoEntry = student_info(l)
        If (StuInfoEntry.AdressNo <> chosen) Then
                h2 = Left(StuInfoEntry.MaxHour, 2) * 60 + Right(StuInfoEntry.MaxHour, 2)
                If (MinTime_Start_Node + MinTime(chosen)) >= h2 Then
                    EFG(l) = F 'we don't know the min time but we know we can't reach them
                    Card_G = Card_G - 1
                End If
        Else
             EFG(l) = E 'we know the min time
             Card_G = Card_G - 1
        End If
    End If
    l = l + 1
    Loop

    frmPreOpt!GauPreopt.FloodPercent = 100 * ((iter_number * nb_stu) + (nb_stu - Card_G)) / (nb_stu * nb_stu)

    Loop 'Card_G=0
    iter_numbeF = iter_number + 1

    first_one_2 = 1
    next_one_Z = 1
```

```
    Y = 1

    snStu2.MoveFirst
    student2_No = 1
    Do
        trip = MinTime(snStu2("AdressNo"))
        h1 = Left(snStu1("MinHour"), 2) * 60 + Right(snStu1("MinHour"), 2)
        h2 = Left(snStu2("MinHour"), 2) * 60 + Right(snStu2("MinHour"), 2)
        h2init = h2

        Do
            h2 = h2init

            Do
                If EFG(student2_No) = E Then    'EFG(student)=E
                    If h1 + trip < h2 + inter Then 'fill the next cells with 0
                        Call Remove_Edge(X, Y)
                    End If
                End If
                next_one_2 = SuperVertex(next_one_2)
                h2 = h2 + inter
                Y = Y + 1
                Z = Y
            Loop Until next_one_2 = first_one_2

            Y = next_one_2
            next_one_1 = SuperVertex(next_one_1)
            h1 = h1 + inter
            X = X + 1

        Loop Until next_one_1 = first_one_1

        Y = Z
        If Y <= nb_vertex Then
            first_one_2 = Y
            next_one_Z = Y
        End If

        snStu2.MoveNext
        student2_No = student2_No + 1
        If Not snStu2.EOF Then X = first_one_1

    Loop Until snStu2.EOF
snStu1.MoveNext
student1_No = student1_No + 1
first_one_1 = X
next_one_I = X

Loop Until snStu1.EOF

Call save_matrix
'Call Load_Matrix

finish = Timer
'Debug.Print finish - Start

snFlow.Close
snStu1.Close
snStu2.Close

Unload frmPreOpt
frmPoptOk.Show

mnuFile.Enabled = True
mnuStudents.Enabled = True
mnuBusses.Enabled = True
```

```
    mnuOptimize.Enabled = True
    mnuOptions.Enabled = True
    mnuWindow.Enabled = True
    mnuHelp.Enabled = True
    cmdCloseProject.Enabled = True
    cmdCloseProject.Picture = LoadPicture(app.Path & "\ico\" & "filcl.ico")
    mnuBusses.Enabled = True
    mnuOptimize.Enabled = True
    cmdOptimize.Enabled = True
    cmdOptimize.Picture = LoadPicture(app.Path & "\ico\" & "feuve.ico")
    mnuRunOptimizer.Enabled = True
    mnupreoptimization.Enabled = False
    mnuAddStud.Enabled = False
    mnuModifyStud.Enabled = False
    mnuDeleteStud.Enabled = False
    panOptimizationInfo.Caption = "PreOptimization completed"
    DoEvents

End Sub

Sub mnuPrint_it_Click ()
    Call cmdPrint_Click
End Sub

Sub mnuRunAgainOptimizer_Click ()

    mnuAdjustBusPreferences.Enabled = False
    cmdAdjPref.Picture = LoadPicture(app.Path & "\ico\" & "ncars.ico")
    cmdAdjPref.Enabled = False
    mnuPrint_it.Enabled = False
    cmdPrint.Enabled = False
    cmdPrint.Picture = LoadPicture(app.Path & "\ico\" & "nprint.ico")

    'cascade child forms
    MdiMain.Arrange cascade
    'Give a chance to Windows
    DoEvents

    'Update the number of optimizations
    nb_opti = nb_opti + 1
    panOptimizationInfo.Caption = "Optimization #" & nb_opti

    Call init_controls_for_SA
    ComRQuality.Enabled = False

    DoEvents
    'Run the Simulated Annealing Greedy Algorithm
    Simulated_Annealing

    panOptimizationInfo.Caption = "Finished Optimization #" & nb_opti

    Call Reset_controls_after_SA
    ComRQuality.Enabled = True

End Sub

Sub mnuRunOptimizer_Click ()

Dim old_nb_col As Integer
Dim energy As Long, old_energy As Long
Dim color_to_attribute As Integer

'Load matrix
'Call Load_Matrix
```

```
Dim it As Long


'Unload all the grids
For it = 9 To 0 Step -1
    Unload frmsolutions(it)
Next it


'Disable things
mnuStudents.Enabled = False
mnuFile.Enabled = False
cmdCloseProject.Enabled = False
cmdCloseProject.Picture = LoadPicture(app.Path & "\ico\" & "nfilcl.ico")
mnuAddBus.Enabled = False
mnuModifyBus.Enabled = False
mnuDeleteBus.Enabled = False
mnuRunOptimizer.Enabled = False
mnupreoptimization.Enabled = False
mnuRunAgainOptimizer = True
mnuStopOptimizer = True
Unload frmPoptOk


'The name of each bus
ReDim bus_name(nb_stu) As String

'The number of free seats in each bus
ReDim free_seats(nb_stu) As Long

'The level of preference for each bus
ReDim bus_pref(nb_stu) As Long

'The number of seats for each bus
ReDim max_seats(nb_stu) As Long


'Load an initial solution
nb_bus = nb_stu


color_to_attribute = 1
For it = 1 To nb_vertex
    If SuperVertex(it) < it Then
        Color(it) = color_to_attribute
        color_to_attribute = color_to_attribute + 1
    Else
        Color(it) = 0
    End If
Next it


Call Init_bus_attribute
Call Init_free_seats

'Initialize the number of optimization
nb_opti = 0


Call init_controls_for_SA
```

```
'Result precision
If mnuNormal.Checked Then
    Result_precision = Normal
ElseIf mnuHight.Checked Then
    Result_precision = Hight
ElseIf mnuVeryHight.Checked Then
    Result_precision = Very_Hight
End If
ComRQuality.Enabled = False

'Can not show the results
sleeper = True
kill_bus = True
total_time = 0
Do
    nb_opti = nb_opti + 1

    If nb_bus <= Bus_Maxrec - 1 Then
        Call update_user_bus_preference
    Else
        Call update_bus_attribute
    End If

    panOptimizationInfo.Caption = "Optimization #" & nb_opti

    old_nb_col = nb_bus

    old_energy = 0
    For it = 1 To nb_bus
        old_energy = old_energy + (bus_pref(it) * (max_seats(it) - free_seats(it)))
    Next it

     DoEvents

    'Run the Simulated Annealing Greedy Algorithm
    Simulated_Annealing

    energy = 0
    For it = 1 To nb_bus
        energy = energy + (bus_pref(it) * (max_seats(it) - free_seats(it)))
    Next it

    'Give a chance to Windows
    DoEvents

Loop Until Stop_Condition(old_nb_col, energy, old_energy)

panOptimizationInfo.Caption = "Finished Optimization #" & nb_opti
ComRQuality.Enabled = True

'Show the final Solution
Call Visualize_Solution

'Set the user's preferences for the real busses
'Call update_user_bus_preference
'Set init preferences for the virtual busses
'For it = Bus_Maxrec To nb_bus
'    bus_pref(it) = virtual_bus_pref
'Next it

'Can show the results
sleeper = False
Call Reset_controls_after_SA

frmGraph.Graph1.DrawMode = 2
```

```
Call Show_msg_box(total_time, 0, nb_bus, energy)

End Sub

Sub mnuStopOptimizer_Click ()

    Dim Response


    ' Get user response.
    Response = MsgBox("Do you really want to end optimization ?", MB_ICONSTOP + MB_YESNO, nom_prog)

    If (Response = IDYES) Then    ' Evaluate response
        'Disable things
        mnuFile.Enabled = True
        cmdCloseProject.Enabled = True
        cmdCloseProject.Picture = LoadPicture(app.Path & "\ico\" & "filcl.ico")
        mnuStudents.Enabled = True
        mnuBusses.Enabled = True
        mnuOptimize.Enabled = True
        cmdOptimize.Enabled = True
        cmdOptimize.Picture = LoadPicture(app.Path & "\ico\" & "feuve.ico")
        cmdStopOptimize.Enabled = False
        cmdStopOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nstop.ico")
        mnuAddBus.Enabled = True
        mnuModifyBus.Enabled = True
        mnuDeleteBus.Enabled = True
        mnuRunOptimizer.Enabled = True
        mnuRunAgainOptimizer = False
        mnuStopOptimizer = False
        mnuAdjustBusPreferences.Enabled = False
        cmdAdjPref.Picture = LoadPicture(app.Path & "\ico\" & "ncars.ico")
        cmdAdjPref.Enabled = False
        mnuPrint_it.Enabled = False
        cmdPrint.Enabled = False
        cmdPrint.Picture = LoadPicture(app.Path & "\ico\" & "nprint.ico")


        panOptimizationInfo.Caption = "End of Optimization"


        'Unload all the grids
        Dim it As Long
        For it = 9 To 0 Step -1
            Unload frmsolutions(it)
        Next it

    End If


End Sub

Sub mnuStudents_Click ()

    If mnupreoptimization.Enabled Then
        If Stud_MaxRec = 0 Then
            mnuViewStud.Enabled = False
            mnuModifyStud.Enabled = False
            mnuDeleteStud.Enabled = False
        Else
            mnuViewStud.Enabled = True
            mnuModifyStud.Enabled = True
            mnuDeleteStud.Enabled = True
        End If
```

```
    End If

End Sub

Sub mnuToFile_Click ()
    report1.Destination = 2
    mnuToPrinter.Checked = False
    mnuToFile.Checked = True
    mnuToWindow.Checked = False
    If cmdPrint.Enabled Then
        cmdPrint.Picture = LoadPicture(app.Path & "\ico\" & "printf.ico")
    End If

End Sub

Sub mnuToolBar_Click ()
    If mnuToolBar.Checked Then
        mnuToolBar.Checked = False
        MdiMain!picToolBar.Visible = False
    Else
        mnuToolBar.Checked = True
        MdiMain!picToolBar.Visible = True
    End If
End Sub

Sub mnuToPrinter_Click ()
    report1.Destination = 1
    mnuToPrinter.Checked = True
    mnuToFile.Checked = False
    mnuToWindow.Checked = False
    If cmdPrint.Enabled Then
        cmdPrint.Picture = LoadPicture(app.Path & "\ico\" & "print.ico")
    End If

End Sub

Sub mnuToWindow_Click ()
    report1.Destination = 0
    mnuToPrinter.Checked = False
    mnuToFile.Checked = False
    mnuToWindow.Checked = True
    If cmdPrint.Enabled Then
        cmdPrint.Picture = LoadPicture(app.Path & "\ico\" & "printw.ico")
    End If
End Sub

Sub mnuVeryHight_Click ()

    Result_precision = Very_Hight
    If mnuHight.Checked Then
        mnuHight.Checked = False
    ElseIf mnuNormal.Checked Then
        mnuNormal.Checked = False
    End If
    mnuVeryHight.Checked = True
    ComRQuality.ListIndex = 2
End Sub

Sub mnuViewBus_Click ()

    comefrom = "Visualize Bus"
    Bus_Index = 1
    frmBusses.Caption = comefrom


    frmBusses.Show

End Sub

Sub mnuViewStud_Click ()

    frmFindStu!lblAction.Caption = "Visualize Student"
    comefrom = "Visualize Student"
    frmFindStu.Show 1


End Sub

Sub mnuWinCascade_Click ()

    'cascade child forms
    MdiMain.Arrange cascade


End Sub

Sub mnuWinTilHoriz_Click ()

    'Tile child forms (horizontal)
    MdiMain.Arrange Tile_horizontal

End Sub

Sub panUserLevel_DblClick ()
Dim it As Long
Dim it2 As Long
Dim Response


    Load frmPreOpt

    mnuOptions.Enabled = False
    mnuWindow.Enabled = False
    mnuHelp.Enabled = False
    mnuStudents.Enabled = False
    mnuFile.Enabled = False
    cmdCloseProject.Enabled = False
    cmdCloseProject.Picture = LoadPicture(app.Path & "\ico\" & "nfilci.ico")
    mnuBusses.Enabled = False
    mnuOptimize.Enabled = False
    cmdOptimize.Enabled = False
    cmdOptimize.Picture = LoadPicture(app.Path & "\ico\" & "nfeuve.ico")
    panOptimizationInfo.Caption = "PreOptimization in process"

    DoEvents

    'Student number
    Set DsStudent = DbStu.CreateDynaset("Select * from SchoolBoy ORDER by AdressNo")
    nb_stu = 0
    DsStudent.MoveFirst
    Do While Not DsStudent.EOF
        DsStudent.MoveNext
        nb_stu = nb_stu + 1
    Loop

    ReDim student_info(1 To nb_stu) As Student_Info_type

    Dim Card_G As Integer

    '-------------------------------------------------
```

```
    '-----                                              ---------
    '-----      Creation of the SuperVertex             ---------
    '-----                                              ---------
    '-------------------------------------------------------------

    Dim comp As Integer
    Dim k As Long
    Dim ll As Long
    Dim diff As Integer
    Dim est_time As Integer

    frmPreOpt!labStep.Caption = "1"
    nb_vertex = 0
    comp = 1
    DsStudent.MoveFirst
    For k = 1 To nb_stu
        diff = Left(DsStudent("MaxHour"), 2) * 60 + (Right(DsStudent("MaxHour"), 2))
        diff = diff - (Left(DsStudent("MinHour"), 2) * 60 + Right(DsStudent("MinHour"), 2))
        diff = diff / inter
        nb_vertex = nb_vertex + diff
        ReDim Preserve SuperVertex(nb_vertex)
        Call Create_a_SV(comp, comp + diff - 1)
        comp = comp + diff
        DsStudent.MoveNext
        frmPreOpt!GauPreopt.FloodPercent = (k / nb_stu) * 80
    Next k

    'Call Save_Svert

    DsStudent.MoveFirst
    it = 1
    it2 = 1
    Do While Not DsStudent.EOF
        Do While SuperVertex(it) > it
            it = it + 1
        Loop
        StuInfoEntry.vertex_no = it
        StuInfoEntry.name = DsStudent("Name")
        StuInfoEntry.firstname = DsStudent("Firstname")
        StuInfoEntry.street = DsStudent("Street")
        StuInfoEntry.locality = DsStudent("Locality")
        StuInfoEntry.MaxHour = DsStudent("MaxHour")
        StuInfoEntry.MinHour = DsStudent("MinHour")
        StuInfoEntry.AdressNo = DsStudent("AdressNo")
        student_info(it2) = StuInfoEntry
        it2 = it2 + 1
        it = it + 1
        DsStudent.MoveNext
    Loop

    frmPreOpt!GauPreopt.FloodPercent = 100

    'ReDimension Virtual Adjacence Matrix
    nb_integer = ((((((nb_vertex) ^ 2) - nb_vertex) \ 2) + 1) \ size) + 1
    ReDim Virt_Matrix(((nb_integer \ n_col) + 1), n_col)

    'ReDimension Color Vertice
    ReDim Color(nb_vertex)  As Long

    'Initialise the virtual matrix with 1
    For k = 1 To ((nb_integer \ n_col) + 1)
       For ll = 1 To n_col
          Virt_Matrix(k, ll) = 2147483647 'means byte=111...
       Next ll
    Next k
```

```
    '-------------------------------------------------------------
    '-----                                              ---------
    '-----  Creation of the Adjacence Matrix            ---------
    '-----                                              ---------
    '-------------------------------------------------------------
    load_matrix
    '-------------------------------------------------------------
    Unload frmPreOpt
    frmPoptOk.Show

    mnuFile.Enabled = True
    mnuStudents.Enabled = True
    mnuBusses.Enabled = True
    mnuOptimize.Enabled = True
    mnuOptions.Enabled = True
    mnuWindow.Enabled = True
    mnuHelp.Enabled = True
    cmdCloseProject.Enabled = True
    cmdCloseProject.Picture = LoadPicture(app.Path & "\ico\" & "filcl.ico")
    mnuBusses.Enabled = True
    mnuOptimize.Enabled = True
    cmdOptimize.Enabled = True
    cmdOptimize.Picture = LoadPicture(spp.Path & "\ico\" & "feuve.ico")
    mnuRunOptimizer.Enabled = True
    mnupreoptimization.Enabled = False
    mnuAddStud.Enabled = False
    mnuModifyStud.Enabled = False
    mnuDeleteStud.Enabled = False
    panOptimizationInfo.Caption = "PreOptimization completed"
    DoEvents

End Sub

Sub Remove_node (node As Integer)

    Dim bidon As Integer

    If (prev_node(node) <> 0) And (next_node(node) <> 0) Then 'in the list
        bidon = next_node(node)
        next_node(prev_node(node)) = bidon
        prev_node(bidon) = prev_node(node)
        next_node(node) = 0
        prev_node(node) = 0
        Card_list = Card_list - 1
    ElseIf (prev_node(node) <> 0) And (next_node(node) = 0) Then 'At the end of the list
        next_node(prev_node(node)) = 0
        prev_node(node) = 0
        Card_list = Card_list - 1
    ElseIf (prev_node(node) = 0) And (next_node(node) <> 0) Then 'At the begining of the list
        first_node = next_node(node)
        Card_list = Card_list - 1
        next_node(node) = 0
        prev_node(first_node) = 0
    ElseIf (prev_node(node) = 0) And (next_node(node) = 0) Then 'Only one element left
        Card_list = Card_list - 1
        If Not (first_node = node) Then Stop    ' a retirer plus tard    !!!!!!!!!!!!!
    End If

End Sub

Static Sub Reset_controls_after_SA ()

mnuBusses.Enabled = True
mnuOptimize.Enabled = True
cmdOptimize.Enabled = True
cmdOptimize.Picture = LoadPicture(app.Path & "\ico\" & "feuve.ico")
```

```
cmdStopOptimize.Enabled = True
cmdStopOptimize.Picture = LoadPicture(app.Path & "\ico\" & "stop.ico")
mnuAdjustBusPreferences.Enabled = True
cmdAdjPref.Enabled = True
cmdAdjPref.Picture = LoadPicture(app.Path & "\ico\" & "cars.ico")
mnuPrint_it.Enabled = True
cmdPrint.Enabled = True
If mnuToFile.Checked Then
    cmdPrint.Picture = LoadPicture(app.Path & "\ico\" & "printf.ico")
ElseIf mnuToWindow.Checked Then cmdPrint.Picture = LoadPicture(app.Path & "\ico\" & "printW.ico")
Else cmdPrint.Picture = LoadPicture(app.Path & "\ico\" & "print.ico")
End If



End Sub

Sub Timer1_Timer ()
    Timer1.Enabled = False
    Unload frmSlpatch
End Sub

Function w (i As Integer, j As Integer) As Integer

    Dim l As Integer

    If i = j Then
        w = 0
    Else
        l = 1
        Do While (l <= 7) And (dico(i, l) <> -1) And (dico(i, l) <> j)
            l = l + 2
        Loop
        If (l <= 7) And dico(i, l) <> -1 Then
            w = dico(i, l + 1)
        Else
            w = MaxInt
        End If
    End If

End Function

Function wbis (i As Integer, j As Integer) As Integer
    Dim l As Integer
    If i = j Then
'       w = 0
    Else
        l = 0
        Do
            l = l + 1
        Loop Until (l > lgtb) Or (tbFlow1(l) = j)
'        If (l <= lgtb) Then w = tbFlow2(l) Else w = MaxInt
    End If

End Function
```