République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université SAAD DAHLEB, BLIDA

USDB

Faculté des sciences

Département informatique

Mémoire pour l'obtention d'un diplôme de Master en informatique.

Option : Ingénierie du Logiciel

Thème :

Conception et Réalisation d'un Système de Génération Automatique des Emplois du temps

Organisme d'accueil : Ecole Technique de Blida IFEG

Promoteur :

Dr. BENNOUAR Djamel

Encadreur :

Mr MOKTAR Kamel

Réalisé par :

M^lle LAHIANI Nesrine

M^lle HARAZI Fatima-Zahra

Promotion 2010-2011

بسم الله الرحمن الرحيم

Mémoire de Master en informatique

Conception et Réalisation d'un Système de Génération Automatique des Emplois du temps

---

Présenté par :

LAHIANI Nesrine
HARAZI Fatima-Zahra

Remerciement

Nous tenons à remercier :

- *Allah le tous puissant ;*
- *Dr. Bennouar Djamel, notre promoteur, qui nous a dirigées, orientées, et aussi pour son entière disponibilité, son aide inestimable et ses conseils, sans lesquels ce travail n'aurait pu aboutir.*
- *Mr. Moktar Kamel, notre encadreur pour ses conseils.*
- *Mr. Messaoudi Ouechene Mohamed qui nous a vraiment aidé lorsque on avait besoin, nous lui somme donc très reconnaissant d'avoir bien voulu nous aider pour accomplir ce modeste travail.*
- *les enseignants de l'université qui ont contribué à notre formation, le long de ces cinq dernières années, trouvent ici l'expression de notre gratitude et de notre reconnaissance.*
- *Ainsi que tous qui ont été de prés ou loin de précieuse aide.*

Dédicaces

A mes très chers parents, sans qui je n'aurais vu la lumière du jour, ni devenu ce que je suis Puissant-ils trouver en ce modeste Travail tout la gratitude et la reconnaissance d'une fille dévouée,

A mes sœurs Siham, sabrine, aya.

A mes grands-parents et à toute ma famille.

A mon binôme et sa famille.

A tout mes amis

Fatima-Zahra

Dédicaces

Je dédie ce travail

- *A ma chère mère,*
- *A mon cher père,*
- *A mon cher frère,*
- *A ma chère sœur,*
- *A toute ma famille,*
- *A ma chère Binôme,*
- *A tous mes amis,*
- *Et A tous ceux qui m'aiment et que j'aime.*

Nesrine

Résumé

La gestion des emplois du temps est un problème difficile, notamment dans les universités, les établissements, …, et qui consomme de nombreuses ressources humaines et donc financières. C'est typiquement un problème de résolution de contraintes, NP-Complet, dont la solution n'est pas, a priori, connue dans le cas général. Le problème des emplois du temps consiste à répartir dans le temps des séances, pendant lesquelles s'effectue une activité pédagogique nécessitant des ressources, en l'occurrence des enseignants, groupes, salles et matériels. Cependant, les contraintes concernant les souhaits des responsables, des enseignants et des étudiants sont difficiles à exprimer. Notre solution est de faire une simulation à une réunion de négociation où les acteurs imposés dans l'emploi du temps négocient leurs souhaits.

Toutes les tentatives de modélisations mathématiques telles que la représentation par graphes (coloration de sommets ou d'arrêtes), programmation linéaire, théorie des ensembles … ont échouées et les exemples ont montré que ces méthodes exactes ne suffisent plus dans la plupart des cas réels. La raison est que la topologie de tels problèmes est de complexité très élevée. Les méta-heuristiques, par contre, sont dotées de mécanismes généraux leurs permettant une bonne investigation de l'espace de recherche.

Nous utiliserons les Algorithmes Génétiques et l'Algorithme de Recuit Simulé vu leurs grandes efficacités face aux problèmes d'optimisation combinatoire

Ce papier est relatif à l'analyse et la conception d'un outil interactif à la gestion automatique des emplois du temps capable de prendre en compte une organisation des données et de maintenir la cohérence des contraintes sur ces données.

Finalement, nous avons développé une application programmée en JAVA, capable d'assurer et de faciliter le déroulement de toutes les taches.

Abstract

The management of schedules is a difficult problem, particularly in universities, institutions....etc, which consumes so many resources and financial. This is typically a problem of constraint solving, NP-complete, the solution is not a priori known in the general case. The problem of timetabling is to allocate time sessions, during which an educational activity takes place in need of resources, namely teachers, groups, rooms and equipment. However, the constraints on the wishes of the officials, teachers and students are difficult to express. Our solution is to simulate a negotiation meeting where the players charged in the schedule negotiate their wishes.

All attempts to mathematical models such as representation by graphs (coloring of nodes or edges), linear programming, set theory have failed ... and the examples have shown that these exact methods are not enough in most real cases. The reason is that the topology of such complex problems is very high. Meta-heuristics, for against, have developed mechanisms allowing them a good general investigation of the search space.

We will use genetic algorithms and simulated annealing algorithm had their effective address combinatorial optimization problems

This paper concerns the analysis and design of an interactive tool for automatic management of schedules can be considered an organization of data and maintain consistency constraints on data.

Finally, we have developed an application programmed in Java, capable of ensuring and facilitating the progress of all tasks.

Liste des Abréviations

IA : Intelligence Artificielle

RO : Recherche Opérationnelle

ET : Emploi du Temps

AG : Algorithme Génétique

RS : Recuit Simulée

UML: Unified Modelling Language

SGBD : Système de Gestion de Base de Données

JDBC: Java Data Base Connection

ETB : Ecole Technique de Blida

STP : Service Technique et Pédagogique

SAF: Section D'administration et de Formation

CDD : Centre De Documentation

DR : Dispositif de Réalisation.

Table des figures

TABLE DES MATIERES

CHAPITRE II: ETUDE CONCEPTUELLE

I. CONTEXTE DU TRAVAIL

Chaque année, les responsables pédagogiques des universités et des écoles d'ingénieurs ont pour mission d'organiser les emplois du temps des différentes formations ou filières en essayant, au mieux, de satisfaire les contraintes « humaines » des enseignants et des étudiants, les contraintes pédagogiques imposées par la progression des enseignements et en tenant compte des contraintes « physiques » liées aux ressources matérielles (les salles, les équipements, etc.).

Jusqu'à présent certains responsables ont vainement tenté d'apporter des solutions « Informatisées » pour ce problème. Certains d'entre eux ont exploité des outils existants de génération automatique des emplois du temps. Cependant, ces responsables reprochent souvent la complexité et la lourdeur de ces outils. De plus, il est nécessaire d'avoir des compétences suffisantes en informatique pour comprendre les mécanismes d'utilisation et les machines sur lesquelles fonctionnent ces outils doivent généralement être assez puissantes. D'autres responsables ont élaboré eux-mêmes des outils qui leur permettent de dresser des bilans. Ces outils facilitent certaines tâches liées à la gestion des emplois du temps mais restent des solutions locales limitées et difficilement exploitables dans d'autres organisations.

Il existe bien des outils de génération des emplois du temps, mais ces outils souffrent d'un manque d'interfaces conviviales et accessibles.

Par exemple, certains outils peuvent prendre en compte différentes contraintes que doit exprimer l'utilisateur dans un langage spécifique. Cependant, les contraintes concernant les souhaits des responsables, des enseignants et des étudiants sont difficiles à exprimer et dans certaines situations, les contraintes des enseignants posent un grand problème, ce problème est que la satisfaction de tous les souhaits des enseignants rend l'élaboration d'un emploi du temps impossible, en revanche, les outils de génération automatique existants ne peuvent rien faire face à cette situation.

De plus, les responsables pédagogiques avouent que les difficultés ne viennent pas seulement de la génération des emplois du temps mais aussi de leur manipulation. On constate, en effet, que lors de la création d'un emploi du temps, les responsables partent généralement d'un emploi du temps existant auxquels ils apportent des modifications.

La méthode ne consiste donc pas à créer mais à adapter un emploi du temps. D'où la nécessité de disposer d'outils interactifs facilitant ces adaptations.

II. PROBLEMATIQUE

L'outil recherché doit donc avant tout être interactif, souple, ouvert et doit présenter des qualités ergonomiques. Ces raisons nous ont motivés à réfléchir sur certaines problématiques essentielles telles que :

- Création d'un emploi du temps en satisfaisant au maximum les contraintes des acteurs.
- Comment prendre en compte un environnement ouvert : ici, comment gérer l'ajout ou la disparition de contraintes en temps réel ?
- Comment gérer le fait que l'environnement est dynamique et que le système à mettre en œuvre doit être capable de s'adapter en conséquence ?
- Comment juger de la qualité de la solution ?

III. OBJECTIFS

Devant ce constat, il nous semble important de trouver une nouvelle solution plus adaptée qui aidera l'acteur dans la création d'emploi du temps ; cette solution devra être élaborée dans un contexte collaboratif, et suffisamment intelligente.

A travers cette solution, nous visons à atteindre les objectifs suivants :

- ✓ Fournir un outil léger et exploitable par des utilisateurs non informaticiens sur des ordinateurs de type « bureautique » disposant d'une faible puissance de traitement.
- ✓ Recenser les besoins en manipulation d'emplois du temps et de proposer une organisation générique pour que l'outil soit exploitable dans de nombreuses structures.
- ✓ Satisfaire les souhaits des acteurs en garantissant la création d'un emploi du temps complet.
- ✓ Optimisation de la solution ; créer un emploi du temps optimal le plus possible.
- ✓ Proposer un outil ouvert qui autorise de nombreuses évolutions.

IV. ORGANISATION DU MEMOIRE

Le manuscrit est organisé en cinq parties : génération d'emploi de temps, le problème de satisfaction de contraintes, la modélisation du system, la conception, la réalisation du système.

- **Chapitre I :** nous présentons le problème de génération de l'emploie du temps. nous l'avons devisé ce chapitre en trois parties :
  - PROBLEME D'EMPLOI DU TEMPS.
  - PRESENTATION DU PROBLEME DE SATISFACTION DES CONTRAINTES.
  - NOTRE CHOIX

- **Chapitre II :** étude conceptuelle

  Dans ce chapitre, ya deux parties la première, représentation de l'organisme d'accueil, et la deuxième on essaie de décrire de manière explicite les fonctionnalités de système, en suivant le modèle en cascade et en utilisant le langage de modélisation UML
- **Chapitre III :** Résolution de problème d'emploi du temps par l'Algorithme Génétique et le Recuit Simulé, en présentant les différentes étapes de ces deux algorithmes.
- **Chapitre IV :** Implémentation et Réalisation

  Dans le dernier chapitre, nous décrivons le développement de notre système, et cela présentant en premier lieu les choix effectués pour la réalisation, et nous abordons en deuxième lieu nous achèverons ce chapitre par la présentation de multiples interfaces de notre système (produit final).
- **Conclusion et Perspectives**

  Finalement, nous terminons par une conclusion générale qui récapitule les principales observations concernant l'évolution du travail et nous indiquons également comment les travaux réalisés tout au long de ce mémoire pourraient être améliorés, en donnant la perspective de nouvelle direction de recherche.
- **Annexe**

PROBLEME D'EMPLOI DU TEMPS

"Un emploi du temps réalisé peut jaillir sur les vies de mille étudiants et soixante-dix personnes pour les deux-cent jours de l'année scolaire, période par période cloche par cloche. Un outil si puissant peut facilement créer ou casser des situations, des étudiants et des enseignants ".

K.Johnson.Timetabling. [1980]

I. Définition du problème d'emploi du temps :

Définition1:

Un emploi du temps est une disposition des tâches d'un certains processus dans le temps, avec l'allocation des ressources a l'exécution de chacune de ces tâches, nécessitant certaines exigences, afin d'atteindre certains objectifs.

De cette définition, on peut extraire qu'un emploi peut être considérer comme une application injective. ***[09]***

Définition2:

Soient **T :** Ensemble des tâches **t** ;

P : Ensemble des périodes **p** ;

R : Ensemble des ressources **r** ;

H(t) : Ensemble des périodes possibles pour la tâche **t** ;

R_c(t) : Ensemble des ressources convenables a la tâche **t**;

On appelle un problème d'emploi du temps une application injective ***f*** de **T** dans P×R telle que $f(t) = (p, r)$ et vérifiant :

i/ $(p, r) \in H(t) \times R_c(t)$ **;**

ii/ si $f(t) = (p, r)$ et $f(t') \neq (p, r)$ alors t et t' sont deux tâches différentes. ***[10]***

II. Description du problème d'emploi du temps:

Une très grande classes des problèmes d'emploi du temps peut être décrite comme un ensemble d'affectation qui vérifient un ensemble de contraintes **C.** Où chaque affectation est un quadruplé ordonné de la forme (a, b, c, d), où :

a : appartient a un ensemble fini des tâches $E= \{e_1, e_2, ...e_r\}$, (où e_i peut représenter : un cours, un examen, un séminaire,... selon le type de problème)

b : appartient a un ensemble fini de temps de commencement des tâches $T= \{t_1, t_2, .. t_r\}$;

c : appartient a un ensemble fini des places dans lesquelles les tâches peuvent se dérouler $P= \{p_1, p_2, ...p_r\}$,

d : appartient a un ensemble fini d'agents qui jouent des rôles distingués dans ces tâches $A= \{a_1, a_2, ...a_r\}$. (Où a_i peut représenter un enseignant, un examinateur...)

L'affectation est représentée comme suit : « *une tache* ***a*** *commence a un temps* ***b*** *dans la place* ***c*** *avec l'agent* ***d.*** » **[17]**

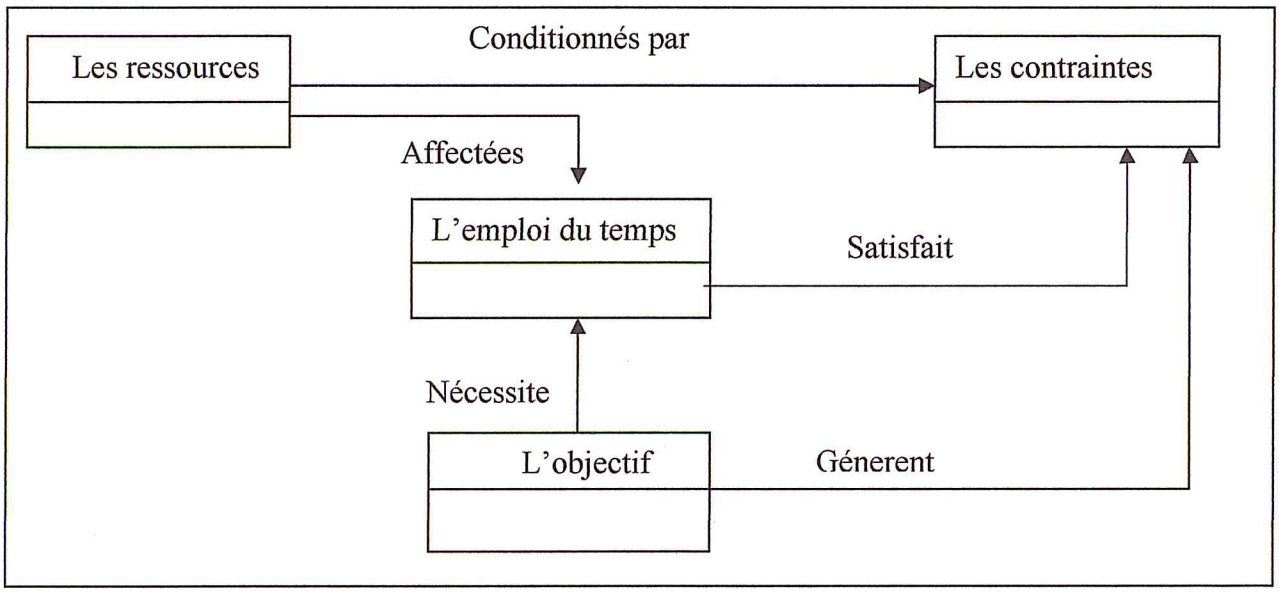


Figure 01: Description du problème d'emploi du temps

III. les modèles adoptés au problème d'emploi du temps :

III.1 le modèle de graphe :

Selon **[17]**, un problème d'emploi du temps peut être modélise de plusieurs façons et principalement en utilisant les graphes (graphes d'intervalle, graphe biparti,…), il peut se ramener à l'un de ces deux problème :

- ***Un problème de coloration des sommets d'un graphe :***

  Il consiste à attribuer une et une seule couleur à chaque sommet du graphe, tel que deux sommets adjacents ne reçoivent pas la même couleur.

- ***Un problème de coloration des arêtes d'un graphe :***
- Il consiste à attribuer une et une seule couleur à chaque arête du graphe, tel que deux arêtes adjacentes ne reçoivent pas la même couleur.

Critique :

Les modèles en graphes sont souvent utilisés pour représenter uniquement les modèles d'emploi de temps ou l'affectation doit généralement viser à minimiser un objectif qui a une forte relation avec le temps.

La complexité du problème d'emploi du temps demeure, et les difficultés liées à la taille de données et des contraintes demeurent aussi.

De plus ces modèles ne permettent pas de prendre en compte toutes les contraintes d'un emploi du temps, seule la contrainte de non chevauchement est considérée sans exprimer la cause qui provoque le non simultanéité.

III.2 le modèle de la programmation mathématique :

L'auteur dans **[17],** dit qu'à travers l'utilisation de la notion mathématique nous pouvons formellement représenter le problème de différentes façons (programme linéaire ou non linéaire, à variable bivalentes ou en nombres entiers). Le modèle le plus connu est la formulation en programmation linéaire à variables bivalentes **0** et **1**.

L'idée principale de la programmation mathématique est de résoudre un problème d'optimisation, avec un objectif qui mesure la qualité des solutions réalisables trouvées (par exemple pour l'emploi du temps d'horaire de cours, la meilleure configuration est celle qui utilise moins de salle, ou celle qui minimise les déplacements des classes entre les pavillons, ...).

Critique :

L'avantage de la méthode mathématique est que toutes sorte de contraintes peuvent être exprimé sous forme d'équation (ou inéquation). Mais la difficulté se présente lors de la formulation de certaines contraintes, ainsi que les critères d'optimisation sont flous et difficile à exprimer.

Cependant, à cause de l'augmentation de la taille des données, le nombre d'équations augmente avec la croissance du nombre de contraintes à respecter. Cela provoque a chaque croissance, une complication supplémentaire pour le problème, d'où toute tentative de résolution par les méthodes classique (la méthode simplexe, les méthodes arborescente, ...) serait trop couteuse en temps de calcul et en encombrement de mémoire. **[17]**

III.3 le modèle sous forme d'un CSP :

Le nombre énorme d'exigence représentés dans les problèmes ET rend la formulation de cet problème sous un CSP très intéressante. Ces exigences ne peuvent jamais être recensées, mais pour bien améliorer la qualité des configurations trouvées on distingue deux types de contraintes :

- ***Contraintes dures :*** qui ne doivent jamais être violées, car le non respect de cette contrainte implique que l'emploi du temps n'aura aucun sens.
- ***Contraintes souples :*** qu'il faut les respectées au maximum, plus ces contraintes sont respectées plus l'emploi de temps est meilleur.

Critique :

- Comparant avec les autres modélisations de problème d'ET, sa formulation sous forme d'un CSP a présenté beaucoup d'avantage, notamment une bonne adaptation avec le grand nombre de combinaison envisagées. Ainsi une souplesse de formulation de toute contrainte, plus ou moins meilleure que la programmation mathématique.
- La représentions comme CSP est souvent beaucoup plus proche du problème originale : les variables du CSP correspondent directement aux entités du problème, et les contraintes peuvent être exprimées sont devoir être traduites en inégalité linéaire. Ceci rend la formulation du problème plus au moins facile, la solution simple a comprendre et le choix de la bonne approche de la résolution plus claire.

- Bien que les algorithmes de CSP soient essentiellement très simple, peuvent parfois trouver la solution plus rapidement que les méthodes de programmation en nombre entier. Surtout dans le cas d'un programme non linéaire.
- Pour un programme mathématique, toutes les contraintes doivent être respectées, tandis que pour un CSP, certaines contraintes précises seront forcement satisfaites, et pour le reste, on souhaite les respectées pour l'amélioration de la solution.
- Mais il est important de marquer que les méthodes de résolution des CSPs manquent de rapidité et de nombre d'affectation peut croitre exponentiel en fonction du nombre des variables du problème et l'espace de recherche, c'est-à-dire la taille de problème.

PRESENTATION DU PROBLEME DE SATISFACTION DES CONTRAINTES

I. Le Problème de Satisfaction des contraintes :

De nombreux problèmes sont définis en termes de contraintes (de temps, d'espace, ... ou plus généralement de ressources):

- Les problèmes de planification et ordonnancement: planifier une production, gérer un trafic ferroviaire, ...
- Les problèmes d'affectation de ressources: établir un emploi du temps, alloué de l'espace mémoire, du temps CPU par un système d'exploitation, affecté du personnel à des tâches, des entrepôts à des marchandises, ...
- Les problèmes d'optimisation: optimiser des placements financiers, des découpes de bois, des routages de réseaux de télécommunication, ...
- Etc., ...

Ces différents problèmes sont désignés par le terme générique CSP (Constraint Satisfaction Problems), et ont la particularité commune d'être fortement combinatoires: il faut envisager un grand nombre de combinaisons avant de trouver une solution. La programmation par contraintes est un ensemble de méthodes et algorithmes qui tentent de résoudre ces problèmes de la façon la plus efficace possible. **[16]**

I.1 Qu'est ce qu'un CSP?

Un CSP est défini comme étant un ensemble de contraintes impliquant un **ensemble** de variables, dont chacune est définie sur son domaine propre. L'objectif consiste à trouver un ensemble de valeurs, choisies dans les domaines susmentionnés, à affecter à ces variables de sorte que toutes les contraintes soient satisfaites. **[14]**

I.1.1Concepts de base d'un CSP :

Plus formellement, un problème de satisfaction de contraintes est défini par le triplet (X, D, C) tel que :

- X = (X1, X2, ..., Xn) est l'ensemble des n variables du problème.

- D = (D1, D2,……, Dn) est un ensemble de n domaines finis dont chacun est associé à une variable de X. C'est à dire le domaine Di est associé à la variable Xi (leurs valeurs possibles).
- C = (C1, C2,……, Cm) est l'ensemble des contraintes. Chaque contrainte Cj est une relation entre certaines variables de X, qui limite les valeurs que peuvent prendre simultanément ces variables. **[14]**

I.2 Représentation des contraintes :

Une contrainte peut être représentée de différentes façons. Une contrainte portant sur des variables numériques peut être représentée par des équations d'égalité ou d'inégalité, un prédicat, une relation logique, une fonction booléenne, par exemple, xi ≠ xj. On qualifie ces contraintes de contraintes primitives. **[19]**

Dans la suite, nous définissons la notion de globalité dans un sens simple de combinaison de contraintes primitives. **[04]**

Définition 1 (contrainte globale) **[04]** : Une contrainte globale est une contrainte dont la sémantique est équivalente à une conjonction de plusieurs contraintes primitives. Exemple : soit le système de contraintes suivant
$(x_1 \neq x_2) \vee (x_2 \neq x_3) \vee (x_1 \neq x_3)$. Ces trois contraintes peuvent être modélisées par une unique contrainte globale *alldiff*(x1; x2; x3).

Définition 2 (méta-contrainte) **[04]** : On utilisera le terme de "méta-contrainte" pour qualifier une contrainte de contraintes, définie à l'aide d'opérateurs logiques. Soit c la contrainte suivante : si c1 est satisfaite alors c2 et c3 doivent être satisfaites. On écrira c : ٦c1 ∨ (c2 ∧c3).

Définition 3 (contrainte en intension) **[04]**: Une contrainte est représentée en intention lorsqu'elle exprime la relation entre plusieurs variables en utilisant les opérateurs arithmétiques ou logiques (par exemple x <= y, A∧ B => ٦ C).

Définition 4 (contrainte en extension) **[04]**: Une contrainte est représentée en extension lorsque tous les n-uplets autorisés (goods) ou bien non autorisés (nogoods) sont énumérés. Par exemple, si les domaines des variables x et y contiennent les valeurs 0, 1 et 2, alors on peut définir la contrainte "x est plus petit que y" en extension par "(x=0 et y=1) ou (x=0 et y=2) ou (x=1 et y=2)", ou encore par "(x, y) élément-de {(0,1), (0,2), (1,2)}".

Définition 5 (contrainte satisfaite) **[04]**: Une contrainte c = (x_{i1}..... x_{ik}) est satisfaite par un n-uplets $\mathcal{T} \in dx_{i1} \times ....\times dx_{ik}$ si et seulement si $\mathcal{T} \in r$, où r est la relation associée à c. On dit alors que $\mathcal{T}$ satisfait c. Dans le cas contraire on dit que $\mathcal{T}$ viole la contrainte c.

I.3 Complexité :

Un problème est dit *polynomial* s'il existe un algorithme permettant de trouver une solution optimale pour toutes ses instances en un temps polynomial par rapport à la taille de l'instance. Un tel algorithme est dit *efficace* pour le problème en question.

Cependant, pour la majorité des problèmes d'optimisation combinatoire, aucun algorithme polynomial n'est connu actuellement.

La difficulté de ces problèmes est bien caractérisée par la théorie de la NP-complétude. De nombreux problèmes d'optimisation combinatoire ont été prouvés NP-difficiles.

Cette difficulté n'est pas seulement théorique et se confirme hélas dans la pratique. Il arrive que des algorithmes exacts de complexité exponentielle se comportent efficacement face à de très grosses instances - pour certains problèmes et certaines classes d'instances. Mais c'est très souvent l'inverse qui se produit ; pour de nombreux problèmes, les meilleures méthodes exactes peuvent être mises en échec par des instances de taille modeste, parfois à partir de quelques dizaines de variables seulement. Pour traiter les grosses instances de ce type de problèmes, on se contente de solutions approchées obtenues avec une méthode heuristique. [16]

I.4 Formulation des problèmes CSP :

Etant donné un ensemble d'objets et de ressources, le problème consiste à affecter les objets aux ressources tout en satisfaisant un ensemble de contraintes et en minimisant une fonction objective donnée. Le modèle mathématique associé à un tel problème s'écrit de la manière suivante:

Max F(s) sous les contraintes :

C1, C2,..., Cn **[16]**

I.5 Emploi du temps et CSP :

Dans **[8]**, l'auteur utilise une approche énumérative pour résoudre le problème de l'emploi du temps. Ce problème d'optimisation d'emploi du temps se défini par six ensembles:

Un ensemble Professeurs = $\{P_1, \ldots, P_P\}$,

Un ensemble Formations = $\{F_1, \ldots, F_f\}$,

Un ensemble Enseignements = $\{E_1, \ldots, E_e\}$,

Un ensemble Salles = $\{S_1, \ldots, S_s\}$,

Un ensemble Créneaux = $\{C_1, \ldots, C_c\}$,

Et un ensemble Contraintes contenant l'ensemble des contraintes entre les variables des cinq ensembles précédents.

La connaissance du problème permet pour chaque cours de fixer quelques variables. Ainsi pour chaque professeur Pi de l'ensemble Professeurs, est fixé quel enseignement et à quelle formation il le dispensera. Il reste donc qu'à fixer pour chaque cours la salle et le créneau qui seront utilisés et qui satisfassent les contraintes. Le CSP proposé se présente de la façon suivante :

$V = \{P_1, \ldots, P_P, F_1, \ldots, F_f, E_1, \ldots, E_e, S_1, \ldots, S_s, C_1, \ldots, C_c\}$

$D = \{D(P_1) = \ldots = D(P_p) = [1,p], D(F_1) = \ldots = D(F_f) = [1,f], D(E_1) = \ldots = D(E_e) = [1,e], D(S_1) = \ldots = D(S_s) = [1,s], D(C_1) = \ldots = D(C_c) = [1,c]\}$

C= {les contraintes dures plus les contraintes faibles}. C1, C2,…, Cn.

II. Méthodes de résolution :

Un très grand nombre de méthodes de résolution existent en RO et en IA pour l'optimisation combinatoire et l'affectation sous contraintes. La figure 02 **[16],** met en parallèle les méthodes représentatives développées en RO et en IA, avec à titre indicatif la date approximative d'apparition de chaque méthode.

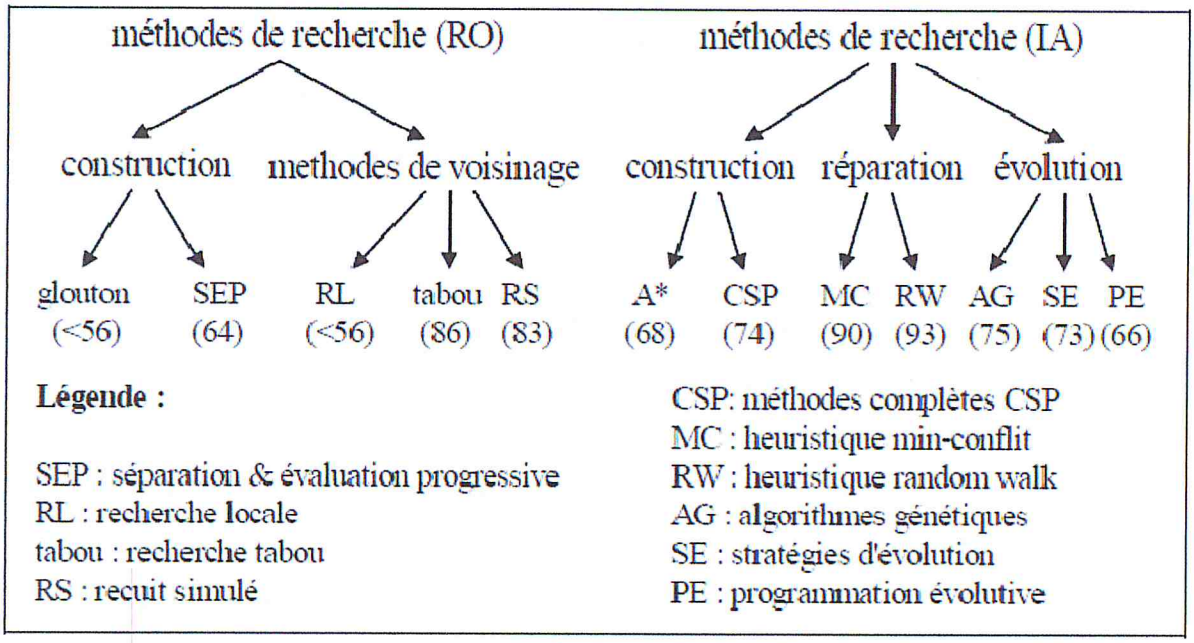


Figure 02: Classement des méthodes de résolution

D'une manière très générale, les méthodes de résolution suivent quatre approches différentes pour la recherche d'une solution : l'approche de construction, l'approche séquentielles, l'approche de voisinage et l'approche d'évolution.

L'auteur dans **[16],** devise ces méthodes en deux groupes de nature différente.

- **Le premier groupe,** comprend les méthodes exactes d'arborescence qui garantissent la complétude de la résolution: c'est le cas de SEP, A* et CSP. Le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre (dans le pire des cas).
- **Le second groupe,** comprend les méthodes approchées dont le but est de trouver une solution de bonne qualité en un temps de calcul raisonnable sans garantir l'optimalité de la solution obtenue. Les méthodes approchées sont fondées principalement sur diverses heuristiques, souvent spécifiques à un type de problème.

II.1.2 Les métaheuristiques :

Les méta-heuristiques sont des heuristiques modernes dédiées à la résolution des problèmes et plus particulièrement aux problèmes d'optimisation, qui visent d'atteindre un optimum global généralement enfoui au milieu de nombreux optima locaux.

1. Les méthodes de voisinage :

Ce sont des algorithmes à base de recherche itérative, en démarrant avec une solution initiale S_0, choisie aléatoirement ou obtenue par le biais d'une méthode constructive ou heuristique, elles essayent d'explorer l'espace de recherche **X** en se déplaçant d'une solution vers une autre solution voisine selon une stratégie distinctive (qui distinguent les solutions entre elles). Dans cette classe on trouve principalement la méthode de Descente, le Recuit Simulé, la recherche locale et la Recherche Tabou.

a. La méthode de descente ou recherche locale :

Cette méthode choisit toujours une solution S_j voisine de la solution courante S_i telle que : $f(S_j) < f(S_i)$ (cas d'un problème de minimisation). Pour cela, elle prend un échantillon **V*** du voisinage de la solution courante et compare la meilleure solution de **V*** avec la solution courante et puis prendre la décision portant sur le remplacement de la solution courante, s'il y a un apport au niveau de la qualité.

Son inconvénient majeur est qu'elle est incapable de progresser au delà du premier optimum local rencontré.

b. Le recuit simule (RS) :

Le recuit simulé trouve ses origines dans un principe de la thermodynamique. Cette méthode est issue d'une analogie entre le phénomène physique de refroidissement d'un corps en fusion, qui le conduit à un état solide. L'analogie exploitée par le recuit simulé consiste à considérer une fonction ***f*** à minimiser comme fonction d'énergie, et une solution **S** peut être vue comme un état donné de la matière dont ***f*** **(S)** est son énergie.

c. La recherche Tabou :

La méta-heuristique Recherche Tabou développée par F. Glover en 1990 permet de contrôler le problème des optimums locaux. L'idée consiste à garder la trace du cheminement passé dans une mémoire et de s'y référer pour guider la recherche.

Le principe de cette méthode est simple, en partant d'une solution initiale **S0** qui progresse itérativement à travers l'espace de recherche, en se déplaçant vers une solution **Sa**,

et en choisissant toujours la meilleure solution voisine même si cette dernière est moins bonne que la solution courante, ceci pour éviter de se bloquer dan un optimum local. Mais il induit un risque de tomber dans un cycle.

En effet, lorsque l'algorithme a quitté un optimum local quelconque par acceptation de la dégradation de la fonction objectif, il peut revenir sur ses pas, à l'itération suivante.

2. Les méthodes évolutives :

Ces méthodes se distinguent de celles déjà étudiées par le fait qu'elles opèrent sur une population de solutions, pour cela, elles sont souvent appelées des méthodes à base de population. Certaines d'entre elles ont des principes inspirés de la génétique et du comportement des insectes. La complexité de ces deux phénomènes biologiques a servi de modèle pour des algorithmes toujours plus sophistiqués ces vingt dernières années. Nous citons les algorithmes génétiques et les algorithmes de fourmis. Il existe d'autres méthodes évolutives indépendantes de la biologie, la recherche dispersée en est un bon exemple.

2.1 Algorithmes génétiques

Cette classe de méthodes est basée sur une imitation des phénomènes d'adaptation des êtres vivants. Les algorithmes génétiques fonctionnent sur une analogie avec la reproduction des êtres vivants.

De manière générale, les algorithmes génétiques utilisent un même principe.

Une population d'individus (correspondants à des solutions) évolue en même temps comme dans l'évolution naturelle en biologie. Pour chacun des individus, on mesure sa faculté d'adaptation au milieu extérieur par le fitness. Les algorithmes génétiques s'appuient alors sur trois fonctionnalités :

- **la sélection** qui permet de favoriser les individus qui ont un meilleur fitness (pour nous le fitness sera le plus souvent la valeur de la fonction objectif de la solution associée à l'individu).
- **le croisement** qui combine deux solutions parents pour former un ou deux enfants (offspring) en essayant de conserver les "bonnes" caractéristiques des solutions parents.
- **la mutation** qui permet d'ajouter de la diversité à la population en mutant certaines caractéristiques (gènes) d'une solution.

La représentation des solutions (le codage) est un point critique de la réussite d'un algorithme génétique. Il faut bien sûr qu'il s'adapte le mieux possible au problème et à l'évaluation d'une solution.

2.2 Algorithmes de colonies de fourmis

Cette nouvelle méta-heuristique imite le comportement de fourmis cherchant de la nourriture. A chaque fois qu'une fourmi se déplace, elle laisse sur la trace de son passage une odeur (la phéromone). Comme la fourmi est rarement une exploratrice isolée, avec plusieurs de ses congénères, elle explore une région en quête de nourriture. Face à un obstacle, le groupe des fourmis explore les deux côtés de l'obstacle et se retrouvent, puis elles reviennent au nid avec de la nourriture. Les autres fourmis qui veulent obtenir de la nourriture elles aussi vont emprunter le même chemin. Si celui-ci se sépare face à l'obstacle, les fourmis vont alors emprunter préférablement le chemin sur lequel la phéromone sera la plus forte. Mais la phéromone étant une odeur elle s'évapore. Si peu de fourmis empruntent une trace, il est possible que ce chemin ne soit plus valable au bout d'un moment, il en est de même si des fourmis exploratrices empruntent un chemin plus long (pour le contournement de l'obstacle par exemple). Par contre, si le chemin est fortement emprunté, chaque nouvelle fourmi qui passe redépose un peu de phéromone et renforce ainsi la trace, donnant alors à ce chemin une plus grande probabilité d'être emprunté.

NOTRE CHOIX

Le logiciel de génération d'emploi du temps que nous projetons réaliser pourrait être basé sur un nombre important d'algorithme qui mis en œuvre selon les objectifs de l'utilisateur. Dans notre cas, nous avons choisi de doter notre application pour un départ avec deux algorithmes : l'un appartenant aux algorithmes génétiques et l'autre appartenant aux algorithmes basé sur le recuit simule.

Nous présenterons dans ce qui suit de manière détaillés ces deux types d'algorithmes.

I. Les Algorithmes Génétiques:

Les algorithmes Evolutionnaire (AE) sont inspirés de la base de sélection naturelle élaborée par Charles Darwin. Le vocabulaire employé est directement calqué sur celui de la théorie de l'évolution et de la génétique.

Dans ce qui suit nous allons parler de l'origine des AG, leur but sera défini, ainsi que leur principe de fonctionnement.une vue détaillée sur les différents paramétrés d'un AG.

I.1 Historique :

Au siècle dernier, Charles Darwin observa les phénomènes naturels et fit les constatations suivantes :

- L'évolution n'agit pas directement sur les êtres vivants ; elle opère en réalité sur les chromosomes contenus dans leur ADN.
- l'évolution a deux composantes : la sélection et la reproduction.

- la sélection garantit une reproduction plus fréquente des chromosomes les plus forts.

- la reproduction est la phase durant laquelle s'effectue l'évolution.

Dans les années 60s, John H. Holland expliqua comment ajouter de l'intelligence dans un programme informatique avec les croisements (échangeant le matériel génétique) et la mutation (source de la diversité génétique). Il formalisa ensuite les principes fondamentaux des algorithmes génétiques :

- la capacité de représentations élémentaires, comme les chaînes de bits, à coder des structures complexes.

- le pouvoir de transformations élémentaires à améliorer de telles structures.

Et récemment, David E. Goldberg ajouta à la théorie des algorithmes génétiques les idées suivantes :

- un individu est lié à un environnement par son code d'ADN.

- une solution est liée à un problème par son indice de qualité **[18]**

I.2 Terminologie d'un algorithme génétique :

Avant d'aborder comment un problème soit résolu à l'aide d'un AG, nous devons également définir le vocabulaire employé. On rappelle ici que les sources d'informations consacrées au sujet se trouvent disséminées dans de nombreux articles et revues le plus souvent accessibles via un service payant ce qui nous a obligés de compter sur différents sites d'Internet et on ne peut donc garantir une meilleure couverture des informations exposées. La source, qui parait intéressante et pertinente, qu'on a utilisées est : **[13]**

- Chromosome : En biologie, il est défini comme le porteur de l'information génétique nécessaire à la construction et au fonctionnement d'un organisme. Dans le cadre des AG, il correspond à un élément représentant une solution possible d'un problème donné.
- Gène : En biologie, il représente une partie du chromosome, chaque chromosome est constitué d'un certain nombre de gènes. Pour un AG, chaque chromosome est divisé en un ensemble d'unités le constituant dites gènes.
- Individu : En biologie un individu est une forme qui est le produit de l'activité des gènes. Pour un AG, il est réduit à un chromosome et on l'appelle donc chromosome ou individu pour désigner un même objet.
- Population : Dans un système naturel, une population est simplement un ensemble d'individus. Par analogie, elle se définit comme l'ensemble des chromosomes. Elle est aussi appelée une génération.

- Parents : Dans un système naturel, les individus peuvent se reproduire en créant de nouveaux individus formant une nouvelle génération afin d'assurer la continuité de la vie. Dans le cadre d'un AG, les parents correspondent aux individus pouvant s'imposer ce qui donne naissance à de nouveaux individus descendants afin de former une nouvelle génération.
- Descendants : Dans un système naturel, la continuité de la vie se fait par la reproduction des individus de la population, il en résulte la naissance d'un certain nombre d'individus dits descendants et qui participent à la création de la nouvelle génération. Dans un AG, les descendants peuvent être considérés comme étant le résultat direct du processus de la reproduction des parents, chaque descendant hérite caractéristiques issues de ses parents.

Les autres concepts comme le croisement et la mutation seront discutés dans une section suivante.

I.3 Mécanismes de fonctionnement d'un (AG)

La mise en œuvre des algorithmes génétiques nécessite donc plusieurs étapes.

L'idée fondamentale est que: la population choisie contient potentiellement la solution, ou plutôt la meilleure solution, à un problème donné. Cette solution n'est pas exprimée car la combinaison génétique sur laquelle elle repose est dispersée chez plusieurs individus. Ce n'est que par l'association de ces combinaisons génétiques au cours de la reproduction que la solution pourra s'exprimer. Lors de la reproduction et de la recombinaison génétique associée, un individu hérite, par hasard, d'un des gènes de chacun de ses parents.

L'originalité des mécanismes repose en particulier sur le fait qu'il n'a pas considéré les seules mutations comme source d'évolution mais aussi et surtout les phénomènes de croisement. C'est en croisant les solutions potentielles existant que l'on peut se rapprocher de l'optimum **[06], [03].**

Les différentes étapes de fonctionnement des (**AG**) se résument à qui suit

I.3.1 Un principe de codage :

Cette étape associe à chacun des point de l'espace d'état une structure de données. Elle se place après une phase de modélisation du problème traité. Pour avoir un codage des individus qui n'est pas complexe pour minimiser le temps de calcule et une qualité de codage des données conditionne le succès des algorithmes.

Lors de la phase de sélection, les individus sont sélectionnés aléatoirement en respectant les probabilités **pi** associées pour former la population de la nouvelle génération. Cette méthode consiste à dupliquer chaque individu de la population proportionnellement à son milieu. Ainsi, les individus ayant la plus grande valeur de fitness auront plus de chance d'être choisis. Dans une population de *N* individus, la fonction de sélection est la suivante **[07]**:

$$P_s(x_i) = \frac{F(x)}{\sum_{i=1}^{N} F(xi)}$$

En utilisant cette probabilité de reproduction, on peut créer une roue de loterie biaisée. Chaque individu de la population occupe une section de la roue proportionnellement à son adaptation et qui indique aléatoirement quel individu peut se reproduire. Cette méthode n' assure pas la sélection des meilleurs individus et peut être une cause de la convergence prématurée **[07].**

- **Sélection par rang :**

La sélection précédente rencontre des problèmes lorsque la valeur d'adaptation des chromosomes varie énormément. C à d e les cases de la roulette ne sont plus proportionnelles à la fitness des individus, mais à leur rang dans la population. La sélection par rang trie d'abord la population par fitness. Ensuite, chaque chromosome se voit associé un rang en fonction de sa position. Ainsi le plus mauvais chromosome aura le rang **1**, le suivant le rang **2**, et ainsi de suite jusqu'au meilleur chromosome qui aura le rang ***m*** (pour une population de ***m*** chromosomes).

- **Sélection par tournoi** :

Sur une population de chromosomes, on forme paires de chromosomes. Dans les paramètres de l'AG, on détermine une probabilité de victoire du plus fort. Cette probabilité représente la chance qu'a le meilleur chromosome de chaque paire d'être sélectionné. Cette probabilité doit être grande (entre 70% et 100%). A partir des paires, on détermine ainsi individus pour la reproduction.

- **Elitisme :**

A la création d'une nouvelle population, il y a de grandes chances que les meilleurs chromosomes soient perdus après les opérations d'hybridation et de mutation. Pour éviter cela, on utilise la méthode d'élitisme. Elle consiste à copier un ou plusieurs des meilleurs chromosomes dans la nouvelle génération. Ensuite, on génère le reste de la population selon l'algorithme de reproduction usuel. Cette méthode améliore considérablement les algorithmes génétiques, car elle permet de ne pas perdre les meilleures solutions.

- **Le croisement :**

L'opérateur de croisement recompose les gènes d'individu existant dans la population, prend en entrée un couple d'individus parents C1et C2 et renvoie un couple d'individus enfants P1et P2 obtenus en choisissant un mécanisme de croisement.

- **La mutation :**

L'opérateur de mutation classique prend en entrée un individu P sélectionné pour la mutation et renvoie un individu mutant P' obtenu par transformation locale de l'un des caractéristique du P.

I.4 Un algorithme génétique simple :

Début
Générer une population initiale P_0. {D'une manière aléatoire}.
Calculer l'adaptation de chaque individu de P_0. {À l'aide de la fonction *f*}
$P_t=P_0$ /* P_t population courante
Tant que non-critère d'arrêt faire
POP-INTER=sélectionner (P_t) {Selon une méthode de sélection}
/* POP-INTER est la population intermédiaire.
P_c=Cross-Over (POP-INTER)
P_m=Mutation(P_c)
Constituer la nouvelle population (la génération P_{t+1})
Calculer l'adaptation de chaque individu de P_t.
Fait
Solution-optimal=Meilleure Adaptation(P_t)
Fin [2]

Figure 03 : Algorithme Génétique

Le critère d'arrêt peut être de nature diverse, par exemple :

- Un taux nombre maximum fixé qu'on désire atteindre
- Un certain temps de calcul à ne pas dépasser,
- Et si la solution ne change pas.

II. Le Recuit Simulé

La méthode de recuit simule (RS) fut introduire dans le but de pallier a l'inconvénient des méthodes ascendantes (descendante) qui arrêtent la recherche des qu'un premier optimum local a été trouve.

Tout comme les méthodes de recherche locale, le RS parcourt l'espace de recherche, solution par solution, à la recherche de meilleure. La nouveauté apporte par cette méthode est l'acceptation de solution pouvant détériorer la fonction objectif. La différence entre les deux types de méthodes réside dans la stratégie d'acceptation. Le RS accepte les solutions améliorant la fonction objectif sans aucune condition mais, contrairement à la recherche locale, il accepte aussi des solutions détériorant la fonction objectif ave une certaine probabilité. Avant de présenter l'algorithme du RS, donnons un petit aperçu de son origine.**[2]**

II.1 L'origine de Recuit Simulé :

Le recuit simule est dérivé des méthodes de physique statistique. Il est base sur une analogie faite avec le procédé du recuit physique utilise en métallurgie pour fabriquer des cristaux. Ce procédé permet d'améliorer la qualité d'un solide en cherchant un état d'énergie minimale correspondant à une structure stable du solide. Le principe consiste a chauffer un matériau a haute température, jusqu'à ce qu'il devienne liquide, puis le faire refroidir très lentement afin de laisser le cristal atteindre son énergie minimale, car un refroidissement rapide ou brusque entraine un blocage du système (apparition d'irrégularités).

Les premiers chercheurs à s'être intéressés à cette méthode sont Kirkpatrick et al (83) et Cerny (85). Ils ont proposé un algorithme base sur une analogie faite entre le processus du recuit des solides et des résolutions des problèmes d'optimisation combinatoires. Ils se sont inspires des travaux de Metropolis et al (53) qui ont simule le processus de recuit physique. Ces derniers ont utilise des méthodes stochastique pour simuler le comportement d'un système d'atome en mouvement avec une énergie E initialement en équilibre a une température T (le procédé de recuit). Partant de cet état initial, le système passe par une série d'états successifs. Le passage d'un état a l'autre se fait en créant une perturbation aléatoire

d'un atome du système créant une différence d'énergie ΔE. le nouvel état est accepte avec une probabilité P (ΔE, T)=exp (-ΔE/KT) ou T est la température du système et K une constante physique connue sous le nom de constante Boltzmann(99). La température est diminuée et le processus est répété un certain nombre de fois. La valeur de l'énergie du système, la température et la probabilité d'acceptation d'un nouveau déplacement diminuent au fil des itérations. Lorsque T tend vers zéro, le critère d'arrêt de l'algorithme est atteint et le système est dit « cristallise » dans un état quasi-stable qui correspond au minimum local de E. Dans cette analogie, les états du système correspondent aux solutions du problème, et les états d'énergie à la fonction objectif des solutions considérées. **[2]**

II.2 Principe du Recuit Simule :

Le principe de cette méthode est simple. A partir d'une solution courante S_c, une solution voisine S_n est générée. Si cette dernière améliore la fonction objectif (f), elle devient la solution courante. Sinon, S_n n'est retenue que selon la probabilité donnée par la formule suivante :

$$P(S\,n,\ T)=\exp\left\lfloor\frac{-(f(Sn)-f(Sc))}{T}\right\rfloor$$

II.3 Algorithme recuit simule :

Début
T=Température initial
S_c= solution initiale.
Tant que condition faire
i=0 ;
Tant que i<MaxIter //MaxIter nombre itérations prédéfinis
i=i+1
Générer une autre solution S_n a partir de S_c (dans son voisinage)
Si $F(S_c) < F(S_n)$ ***alors***
$S_c=S_n$
Sinon r=nombre aléatoire compris entre [0,1]
$P=\exp\left\lfloor\frac{-(f(Sn)-f(Sc))}{T}\right\rfloor$
Si r < P ***alors***
$S_c=S_n$
Finsi
Fait
Diminuer la température de T selon une stratégie
Fait.
Fin **[2]**

Figure 04 : Algorithme Recuit Simulé

Cette probabilité dépend de deux facteurs : la détérioration de la fonction objectif, et la valeur de la température courante. Plus la détérioration de la fonction objectif est petit, et la temperature elevee et plus les chances d'accepter la solution S η sont grandes. Ce procédé est retire un certain nombre de fois. Puis la température est diminuée (refroidissement) selon un système de refroidissement est relance. La figure 04 présente l'algorithme du RS. Il est clair qu'au fur et a mesure que la température diminue, les solutions générées détériorant la fonction objectif sont de moins en moins acceptées. En analysant le comportement du RS, on remarque qu'il passe par deux phases différentes :

- Tout d'abord, la température est initialisée à une grande valeur de manière a accepter, dans la première phase de l'exécution, tous les changements introduits. En d'autres termes beaucoup de solutions générées durant cette phase vont être acceptées. Cela va permettre de couvrir le plus possible l'espace de recherche car la probabilité d'accepter un changement tend vers 1. Pendant cette phase, le RS se comporte comme une recherche aléatoire de l'espace de recherche.
- Durant la deuxième phase de l'exécutable, et au fur et à mesure que la température diminue, la probabilité d'acceptation diminue et le RS refuse les solutions n'améliorant pas la fonction objectif. Le RS dans ce cas, devient une recherche descendante (ascendante) et peut donc converger vers un optimum local. Le mécanisme de refroidissement joue un rôle important et influe sur les performances du Recuit Simule

II.4 Mécanisme de refroidissement :

Un paramètre important pour le RS est la température. Comme nous l'avons dit plus haut, un refroidissement rapide ou brusque entraine du système dans un optimum local. De ce fait, les performances du Recuit Simulé dépendent fortement du refroidissement de la température et du critère de stabilité à une température donnée. Avec un "bon refroidissement", des solutions très proches de l'optimum peuvent être atteintes pour un bon nombre de problèmes NP-Complets. Un mécanisme de refroidissement de la température se base sur :

- ✓ Une température initiale (Ts) ;
- ✓ Une fonction de diminution de la température ;
- ✓ Le nombre d'itération à exécuter à une certaine température ;

Plusieurs approches pour le refroidissement de la température ont été étudiées et proposées. Elles peuvent être classées en 3 catégories :

- **Réduction continue** : dans ce cas de figure, pour chaque température, il y a une seule exécution de l'algorithme (figure 05). C'est le refroidissement le plus utilisé dans les travaux, car son implémentation est simple et il donne souvent de bons résultats. la diminution de la température doit être lente. La règle généralement utilisée est la suivante : $T_{k+1}=T_k/ (1+b*T_k)$ ou b est une petite valeur.

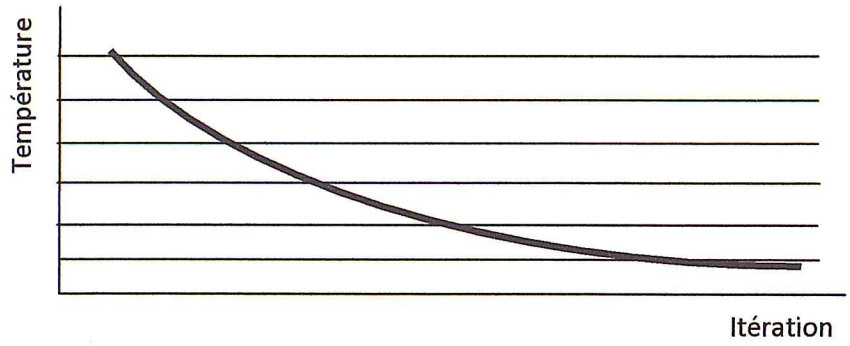


Figure 05: Réduction continue de la température

- **Réduction par palier** : dans ce cas de figure, la réduction de la température suit l'évolution du graphe de la figure 06

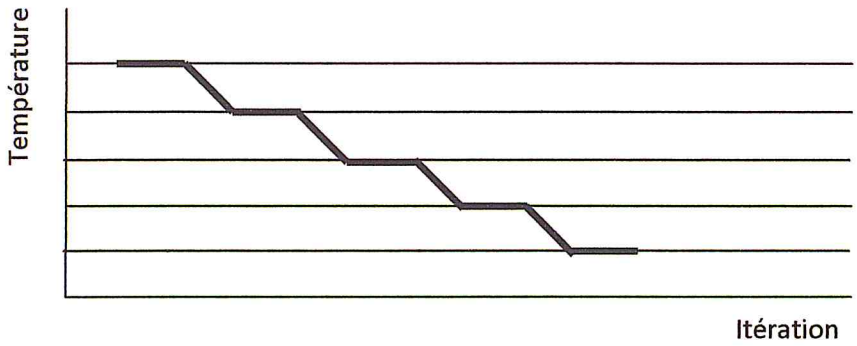


Figure 06 : Réduction par palier de la température

Pour une température donnée, l'algorithme du RS est exécuté un certain nombre d'itération, puis la température est réduite selon un certain facteur. En d'autres termes, pour chaque température, on essaie de trouver un équilibre. proposent différents règles pour la réduction de la température et pour déterminer le nombre de fois que doit s'exécuter l'algorithme à une température donnée. Une règle souvent utilisée pour réduire la température est la règle de refroidissement géométrique $T_{k+1}=T_k*c$, ou ''k'' est l'itération courante, et c un facteur de réduction de température $c \in [0.5, 0.99]$. Les meilleures performances sont pour les valeurs de c dans [0.8, 0.99]. Kirkpatrick et al proposant de prendre c=0.90 et L=nombre de variables du problème. Pour Johson L=m* |n| ou m est une constante et |n| la taille du voisinage. L'avantage de cette règle réside dans le fait que le nombre d'itération pour une température

donnée est proportionnel au nombre de voisin quelque soit la taille du voisinage. Quand à Benoni et al, ils utilisent un nombre fixe d'itération pour une température et la même règle pour la mis a jour de la température. En l'occurrence, le nombre de réduction de la température est de 50 et L est fixe à 9000 itération pour le problème du voyageur commerce visitant 400 villes.

- **Réduction non monotone de la température :** dans ce cas, la température est réduite d'une manière continue avec occasionnellement une incrémentation de la valeur de la température (figure 07). Si aucune solution n'est acceptée après un certain temps, on incrémente la valeur de la température, diminuer la température, diminue les chances d'accepter de nouvelles solutions, par conséquent la température est réajustée à une valeur supérieure pour permettre de s'échapper d'un optimum local.

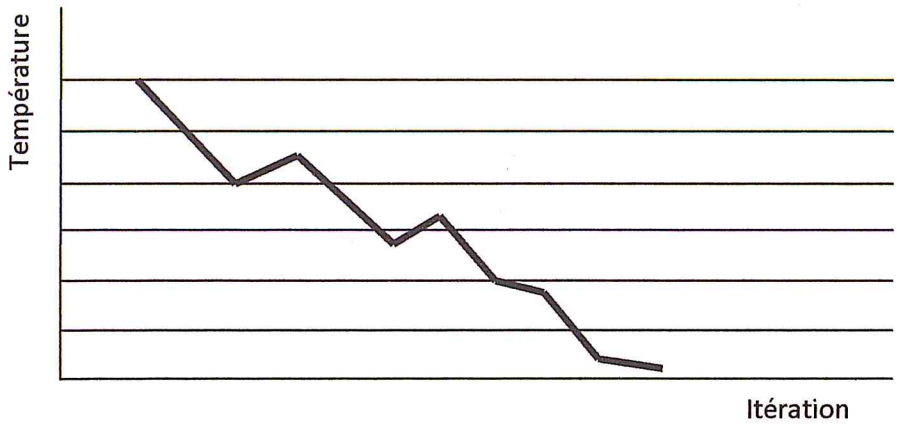


Figure 07: Réduction non monotone de la température

[2]

CONCLUSION

Les méthodes approchées constituent une classe des méthodes adaptable à très grand nombre de problèmes d'optimisation combinatoire et d'affectation sous contraintes. Elles permettent de trouver des solutions de bonne qualité en un temps d'exécution limité pour des problèmes réel de grande taille. Pour cette raison l'étude de ces méthodes est actuellement en plein de développement.

L'ORGANISME D'ACCUEIL ETB

I. présentation de l'organisme d'accueil:

L'étude de l'organisme d'accueil a pour but de bien comprendre la problématique et la quantité d'information circulant dans l'école ainsi les objectifs visés par ce travail.

I.1 Historique:

L'école Technique de Blida (ETB) est la plus importante à l'heure actuelle au sein de la société. Réparties sur 13 hectares, les installations de l'école permettent de faire face aux différents besoins en formation, d'une capacité d'accueil de 250 places pédagogiques, l'école dispose de :

- Salles de cours équipées de supports audio-visuels, et adaptées à la pédagogie.

- Laboratoires, ateliers spécialisés.

- Aires d'entraînements spécialisés Gaz et Electricité.

- Moyens informatiques importants à usage pédagogique.

- Des structures d'accueil, où la culture et le sport ont une large place, contribuant à la fois au travail, au confort et à l'épanouissement du stagiaire.

- Des équipements sportifs variés

- Un centre de documentation avec un cyberespace

- Des manifestations diverses –journée portes ouvertes, expositions…)

Les moyens : L'école dispose d' :

- Une équipe de formateur expérimenté spécialisés dans leur discipline, une équipe pluridisciplinaire spécialisée en
    - Identification et analyse des besoins
    - Conception des cahiers des charges pédagogiques

- Planification et évaluation des actions de formation
- Des capacités de montage des actions de formation à la carte

I.2 Présentation du champ d'étude:

Notre étude s'effectue au niveau de l'école Technique de Blida (SONELGAZ) dans l'organigramme ci-dessous présents ces différents secteurs qui nous permettre de savoir plus sur le centre de notre étude et le définir comme un service important

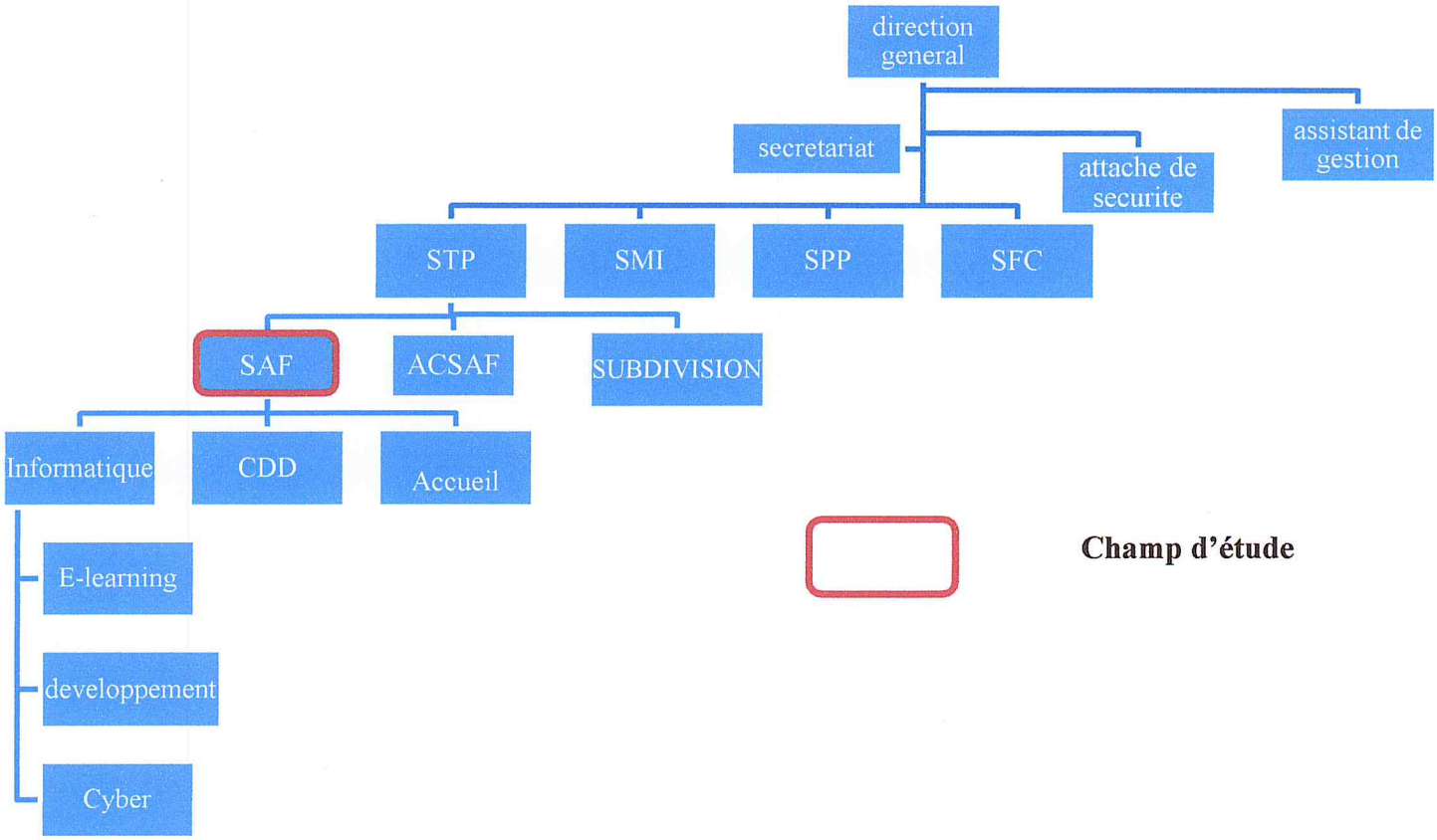


Figure 08: Organigramme de l'ETB

II. Cadre de l'étude :

A chaque arrivée d'une promotion, le responsable de la SAF a pour mission de concevoir les emplois du temps des différentes phase de cette promotion en essayant, au mieux, de satisfaire les contraintes « humaines » des formateurs et des stagiaires, les contraintes pédagogiques imposées par la progression des formateurs et en tenant compte des contraintes « physiques » liées aux ressources matérielles (les salles, les équipements, etc.).

C'est de là, qu'est née, l'idée de mettre en place un système de gestion des emplois du temps qui a pour mission de :

- Génération automatique des emplois du temps.
- Consultation et suivi des emplois du temps.

II.1 Infrastructures et moyens humains

Pour un fonctionnement de haut niveau, l'ETB dispose d'infrastructures et de moyens conséquents, à savoir :

- 33 salles de cours.
- 43 Ateliers dotées de matériels performants.
- 120 machines reliées et connectées au réseau local.
- Un cyber espace à l'usage des étudiants équipés de 50 machines.

Voici quelques chiffres :

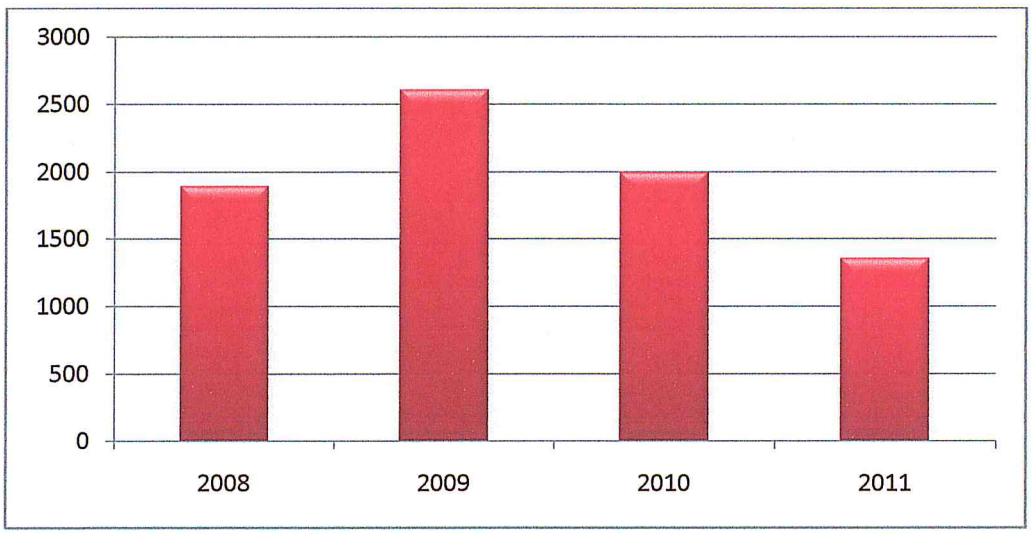


Figure 09 : Evolution des effectifs des stagiaires 2008-2011

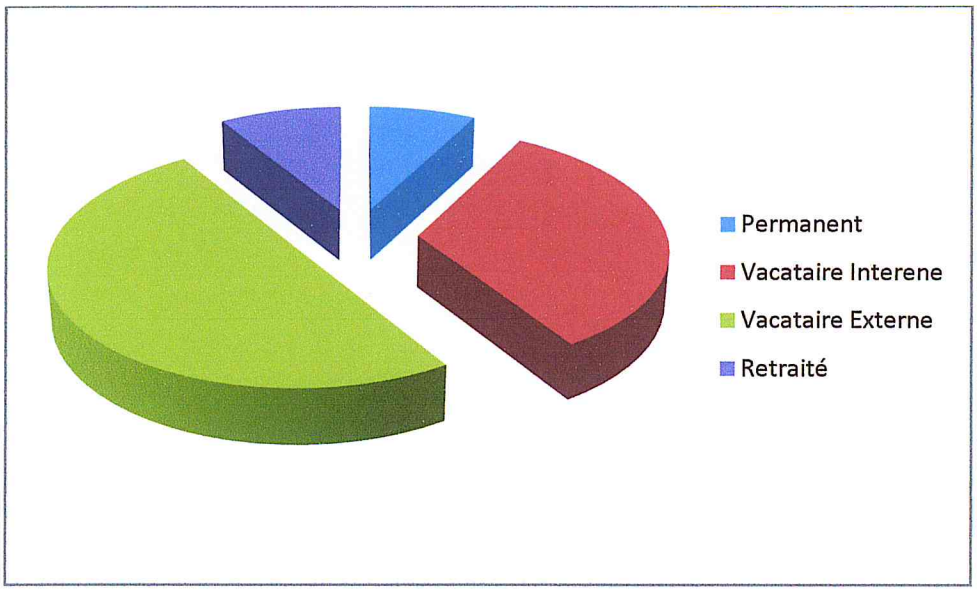


Figure 10: Distribution des formateurs de l'ETB selon leur type

II.2 Présentation du problème d'emploi du temps de l'ETB :

L'école technique de Blida ne fonctionne pas comme un établissement d'enseignement scolaire académique public, elle assure des périodes de formation qui se déroulent durant toute l'année, sauf le mois d'aout (période de congé fixe). La durée de formation est variable (entre 05 jours à 2 ans) par rapport a la spécialité dispensée et son mode de stage (une formation professionnel spécialisé ou un perfectionnement). Certaines peuvent avoir plusieurs phases, d'où son emploi du temps doit être établir plusieurs fois.

La date de début d'un stage de même mode et spécialité est à fixer des la première demande déposée qui devra être en harmonie avec la précédente. Elle sera lancée des qu'il est possible de le faire c'est à dire avoir suffisamment de formateurs et matériels libre.

Les gestionnaires étudient les différents données (différentes spécialités déjà programmées, la durée de formation de la spécialités voulue, le nombre de stagiaire, de formateurs, de salle de cours et des laboratoires nécessaires de la formation visée, ainsi que l'intendance : restauration et hébergement) nécessaires pour le lancement de la nouvelle promotion de stagiaire afin de fixer les dates adéquates de début de stage dans les meilleure délais et d'assurer le bon déroulement de toutes les formations de l'école.

Actuellement, ce travail de planification/programmation est préparé manuellement par des techniciens expérimentés de l'école. Cela leurs demandes beaucoup d'efforts et de concentration, et assez de temps.

CONCEPTION DU SYSTEME

INTRODUCTION

Dans les projets de gestion d'emploi du temps, l'optimisation de solution est un réel problème du fait que la recherche d'une solution est complexe car il s'agit d'un problème de résolution de contraintes, NP-Complet, dont la solution n'est pas, a priori, connue dans le cas général. Pour une telle gestion, notre système doit être capable de s'adapter pour réagir aux changements dynamiques de l'environnement.

I. Méthode de conception

Pour la conception de notre application, nous avons choisi d'utiliser une approche basé sur l'orienté objet. Une telle approche présente plusieurs avantages, à savoir :

- Le système développé est plus facile à maintenir du fait que les objets sont indépendants ils peuvent donc être modifiés. Mais, le fait de modifier l'implémentation d'un objet ou de lui ajouter des services ne doit pas affecter les autres objets du système.
- Les objets sont considérés comme des composants réutilisables appropriés vu leur indépendance. On peut alors développer des conceptions à l'aide des objets créés dans une autre conception.
- Pour certaines classes du système, il existe une correspondance claire entre les entités du monde réel (tels que les composants matériels) et les objets du système qui les contrôlent ce qui permet d'améliorer la compréhension de la conception.
- L'analyse orientée objet permet d'examiner un problème en mettant en évidence les classes et les objets correspondants sous forme de composants indépendants qui interagissent selon des modalités bien définies.

La démarche globale que nous avions suivie est constituée de quatre activités.

- Classification du monde réel : définition des classes et des objets du monde réel visé par notre application
- identification des diverses relations entre les diverses classes. (héritage, composition... etc.) ;
- Définition des attributs ;
- Définition des services offerts par chaque classe (les méthodes).

Pour la spécification dans les diverses phase du processus de conception (analyse, conception), nous avons choisi le langage UML (Unified Modeling Langages) qui est un langage de spécification mis en place pour supporter les divers concepts de l'orienté objet.

Le langage UML sert à **[01]** :

- ✓ Décomposer le processus de développement ;
- ✓ Mettre en relation les experts métiers et les analystes ;
- ✓ Coordonner les équipes d'analyse et de conception ;
- ✓ Séparer l'analyse de la réalisation ;
- ✓ Prendre en compte l'évolution de l'analyse et du développement ;
- ✓ Migrer facilement vers une architecture objet d'un point de vue statique et dynamique.

II. Les Diagrammes d'UML

II.1 Les diagramme de cas d'utilisation :

Un diagramme de cas d'utilisation capture le comportement d'un système, d'un sous système, d'une classe ou d'un composant tel qu'un utilisateur extérieur le voit. Il scinde la fonctionnalité du système en unités cohérentes, les cas d'utilisation, ayant un sens pour les acteurs. Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système, ils sont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique.

Il ne faut pas négliger cette première étape pour produire un logiciel conforme aux attentes des utilisateurs. Pour élaborer les cas d'utilisation, il faut se fonder sur des entretiens avec les utilisateurs. **[1]**

II.1.1 Les acteurs :

- ➢ **Chef de SAF :** C'est le responsable de la gestion des emplois du temps (création, modification et suppression).
- ➢ **ACESAF :** C'est la personne qui permet de créer les DRs (contient les modules affectes aux différentes phases).
- ➢ **Administrateur :** C'est la personne qui est responsable de la gestion des profils, gestion des promotions,… etc.
- ➢ **Externe :** Peut être un stagiaire, un formateur,…

La figure ci-dessous présente le diagramme de cas d'utilisation général de notre application

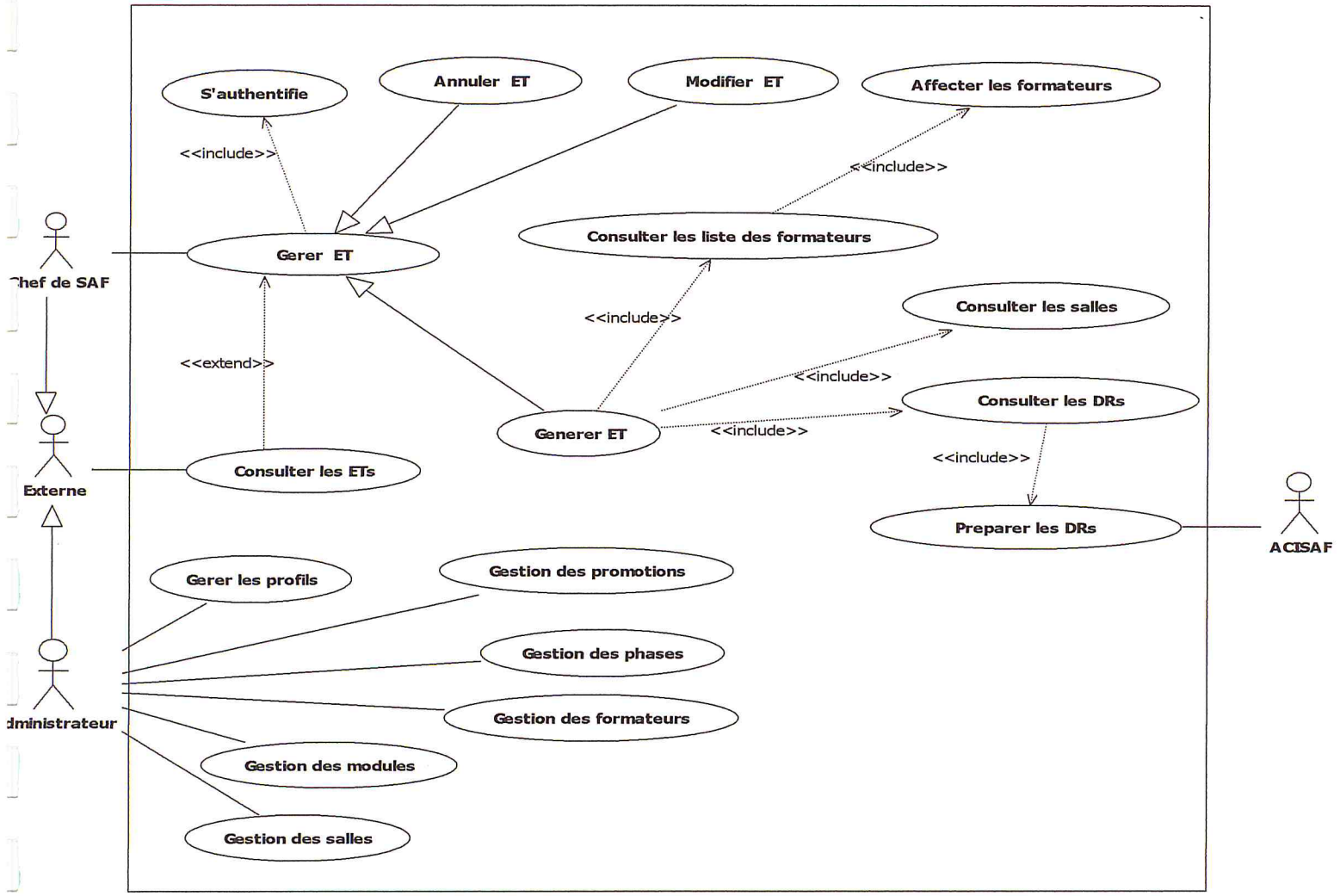


Figure 11 : Diagramme de cas d'utilisation général

II.1.2 Description préliminaire des cas d'utilisations

Voici une description préliminaire des cas d'utilisations énumérés précédemment :

Gestion promotion :

- Intention : gérer les promotions.
- Actions : créer une nouvelle promotion, supprimer une promotion, ou bien modifier une promotion.

Gestion des phases :

- Intention : gérer les phases
- Actions : créer une nouvelle phase, supprimer une phase, ou bien modifier une phase.

Gestion des formateurs :

- Intention : gérer les formateurs.
- Actions : ajouter, supprimer ou bien modifier un formateur.

Gestion des modules :

- Intention : gérer les modules.
- Actions : ajouter, supprimer ou bien modifier un module.

Gestion des salles :

- Intention : gérer les salles
- Actions : ajouter, supprimer ou bien modifier une salle.

Gérer l'emploi du temps :

- Intention : créer l'emploi du temps pour une promotion.
- Actions : créer un nouvel emploi du temps, modifier ou annuler l'emploi du temps.

Consulter l'emploi du temps :

- Intention : consulter l'emploi du temps d'une promotion.
- Actions : consulter l'emploi du temps.

Gérer les profils :

- Intention : créer les différents profils des utilisateurs du système.
- Actions : créer un nouveau profil, attribuer une fonction, attribuer des droits d'accès, modifier le profil, créer un mot de passe.

Préparer les DRs :

- Intention : préparer les dispositifs de réalisation.
- Actions : affecter a chaque phase d'une promotion, les modules avec ses volumes horaires hebdomadaires.

Remarque :

Notre objectif est seulement la gestion des emplois du temps, mais nous avons ajouté la gestion de la base de données (pour la portabilité de système) qui n'a pas été détaillés dans les diagrammes suivants.

II.2 La réalisation du diagramme de classe :

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet. Le diagramme de classes montre la structure interne. Il permet de fournir une représentation abstraite des objets du système qui vont interagir ensemble pour réaliser les cas

d'utilisation. Il est important de noter qu'un même objet peut très bien intervenir dans la réalisation de plusieurs cas d'utilisation.

Le diagramme de classes modélise les concepts du domaine d'application ainsi que les concepts internes créés de toutes pièces dans le cadre de l'implémentation d'une application. Le diagramme de classes permet de modéliser les classes du système et leurs relations indépendamment d'un langage de programmation particulier.

Les principaux éléments de diagramme de classes sont : les classes et leurs relations : association, généralisation et plusieurs types de dépendances. **[1]**

Voici une brève description des classes que nous allons utiliser dans notre diagramme de classes :

Classe	**Description**
Promotion	C'est une formation qui doit être réalisée conformément à un programme préétabli en fonction d'objectifs déterminés. Elle est composée de plusieurs phases
Phase	Une partie de promotion avec une période définit.
Formateur	désigne une personne pouvant assurer des enseignements
Module	Enseignement diffusé dans un domaine donné ou relatif à un niveau scolaire précis
Salle	est un lieu dans lequel sont assurés des enseignements. Peut être un atelier pour les TPs ou une salle permet des cours magistraux,
Groupe	C'est un ensemble d'étudiants
Emploi du temps	C'est un ensemble de séance
Séance	C'est une plage d'horaire avec un enseignant, un module, et une salle.
Disponibilité	C'est l'état libre d'un formateur de type vacataire dans des périodes bien définit.

Le Diagramme de classe général de notre application est le suivant :

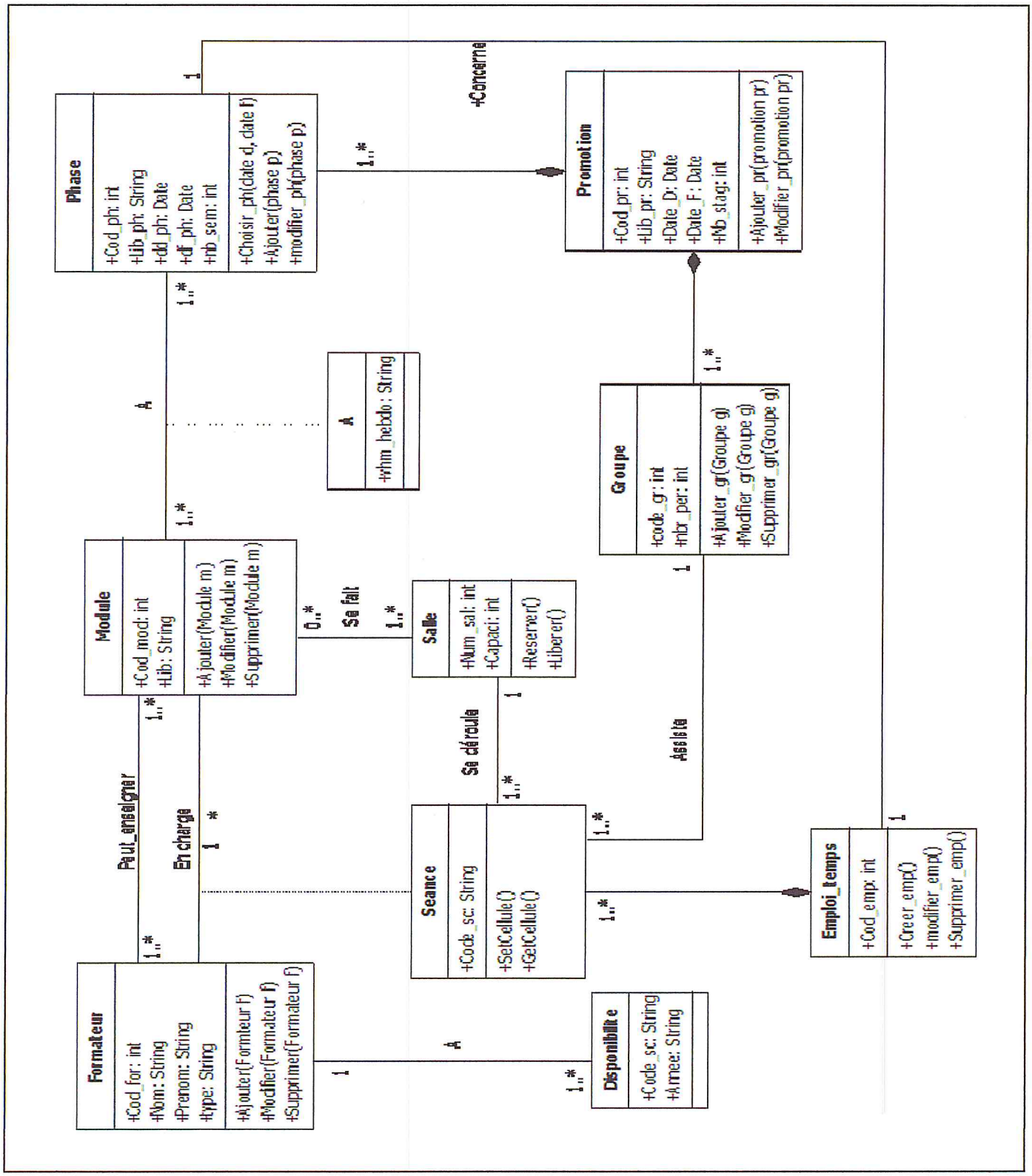


Figure 12 : Digramme de classe globale

II.3 La réalisation du diagramme de séquence :

Le diagramme de séquence permet d'établir un lien entre les diagrammes de cas d'utilisation et les diagrammes de classes : il montre comment des objets (i.e. des instances de classes) communiquent pour réaliser une certaine fonctionnalité. Il apporte ainsi un aspect dynamique à la modélisation du système.

Pour produire un diagramme de séquence, il faut focaliser son attention sur un sous-ensemble d'éléments du système et étudier leur façon d'interagir pour décrire un comportement particulier.

Les principales informations contenues dans un diagramme de séquence sont les messages échangés entre les lignes de vie, présentés dans un ordre chronologique. Ainsi, contrairement au diagramme de communication, le temps y est représenté explicitement par une dimension (la dimension verticale) et s'écoule de haut en bas. **[1].**

SCÉNARIOS DE « authentification »:

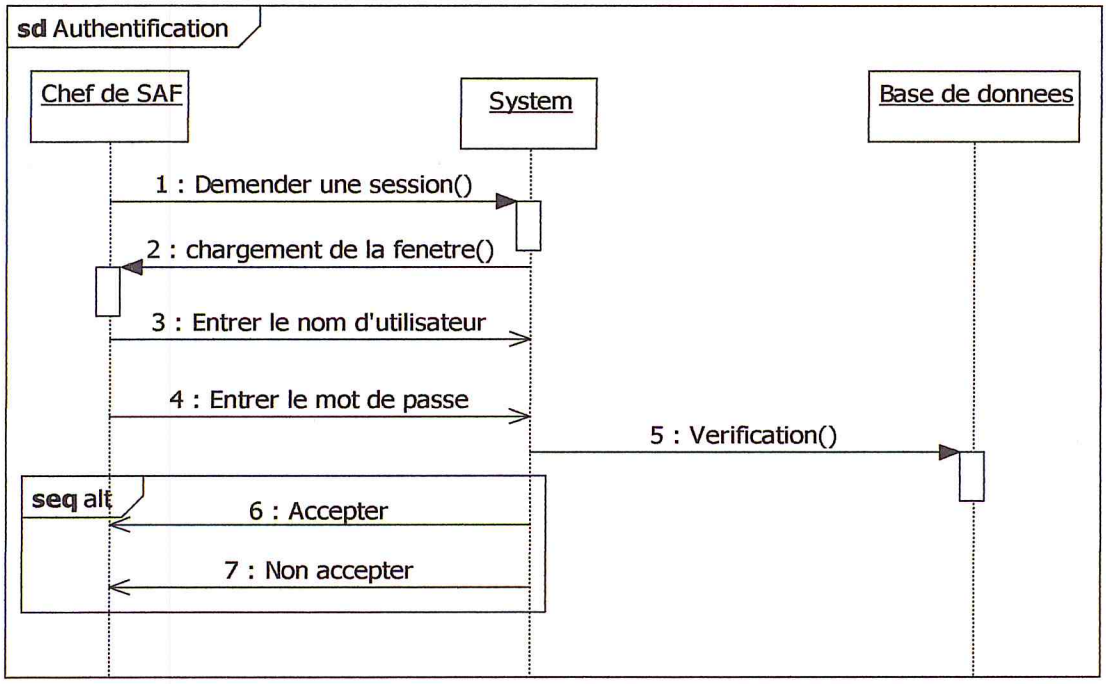


Figure 13: Diagramme de séquence modélisant l'authentification

Scénarios nominaux

- S'authentifier
  - L'utilisateur demande une session
  - Il entre son nom
  - Il entre son mot de passe
  - Le système vérifie et valide l'entrée

SCÉNARIOS DE « Créer l'emploi du temps »:

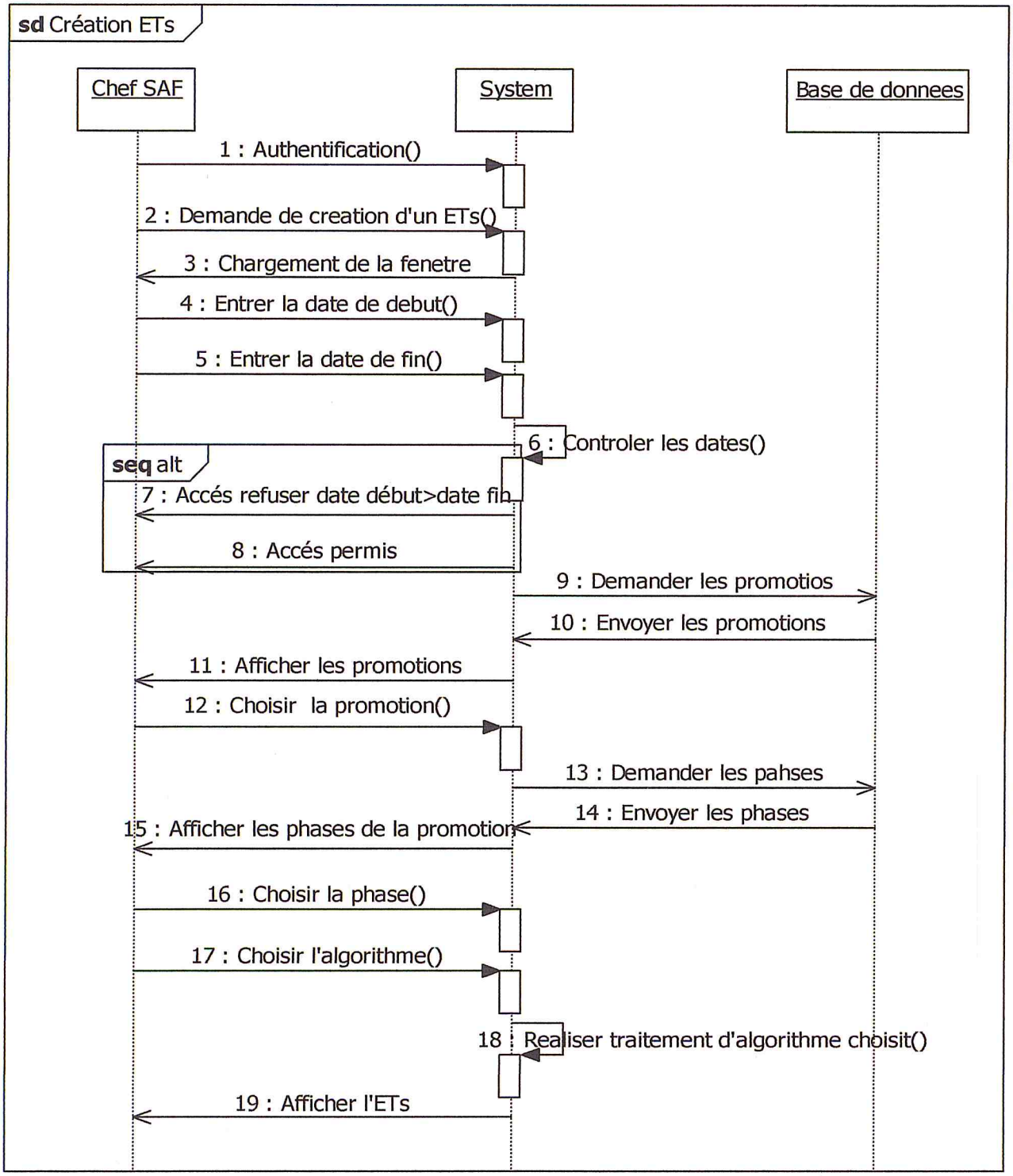


Figure 14: Diagramme de séquence modélisant la création d'un ET

Scénarios nominaux :

- créer un nouvel emploi du temps.
  - Le chef de la SAF donne un nom/mot de passe d'identification.
  - Entrer les dates début et fin.
  - Il choisit la promotion selon les dates entrées.
  - Il choisit la phase de la promotion choisie.
  - Il choisit la méthode pour la création automatique de l'emploi du temps.
  - Il sauvegarde l'emploi ou il l'annule.

SCÉNARIOS DE « la méthode d'Algorithme Génétique »:

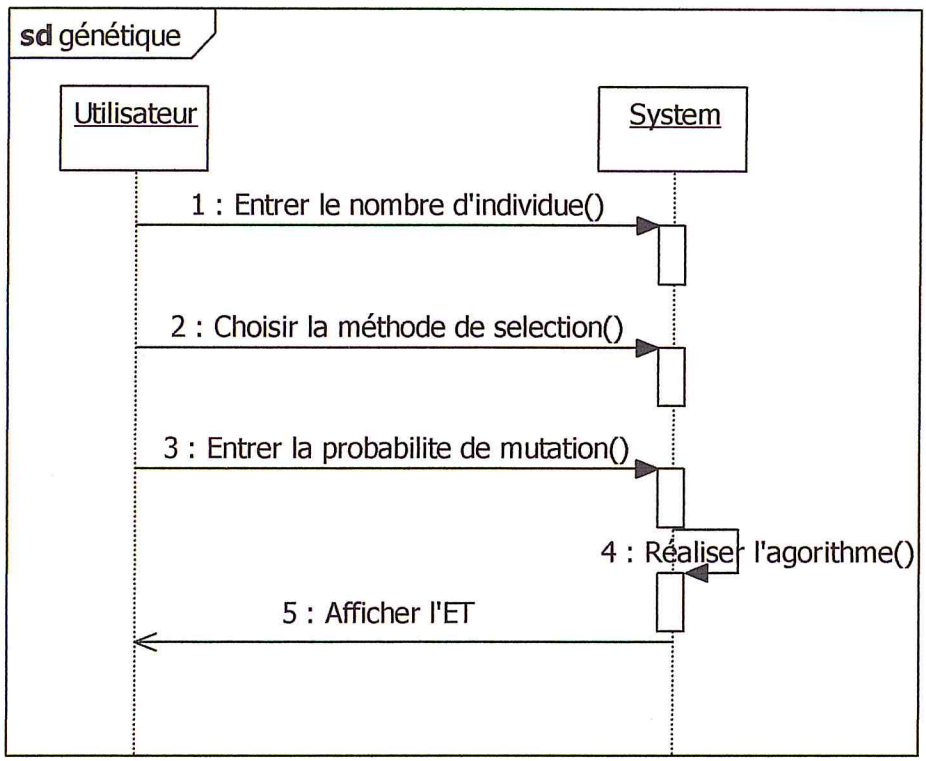


Figure 15 : Diagramme de séquence modélisant la méthode génétique

Scénarios nominaux

- L'algorithme génétique
  - L'utilisateur entre le nombre d'individus
  - Il choisit la méthode de sélection
  - Il entre la probabilité de mutation
  - Le système réalise l'algorithme
  - Et enfin le système affiche le résultat

SCÉNARIOS DE « la méthode de Recuit Simulé »:

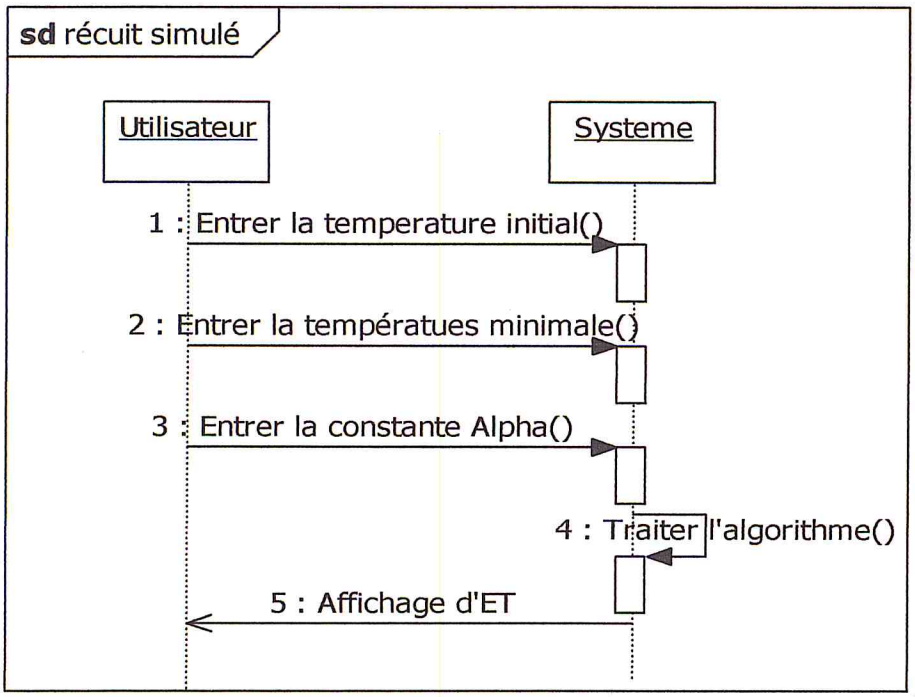


Figure 16: Diagramme de séquence modélisant le Recuit Simulé

Scénarios nominaux

- Le Recuit Simule
  - L'utilisateur entre la température initiale
  - Il entre la température minimale
  - Il entre la constante Alpha
  - Le système réalise l'algorithme
  - Et enfin le système affiche le résultat

II.4 La réalisation de diagramme d'états-transition:

Le diagramme d'états de transition d'UML décrit le comportement interne d'un objet à l'aide d'un automate à état finis. Ils présentent les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours se son cycle de vie en réaction à des événements discrets (de type signaux, invocation de méthode).

Le diagramme d'états-transition est le seul diagramme, de la norme UML, à offrir une vision complète et non ambigüe de l'ensemble des comportements de l'élément auquel il est attaché. En effet, un diagramme d'interaction n'offre qu'une vue partielle correspondant à un scénario sans spécifier comment les différents scénarios interagissent entre eux. **[1].**

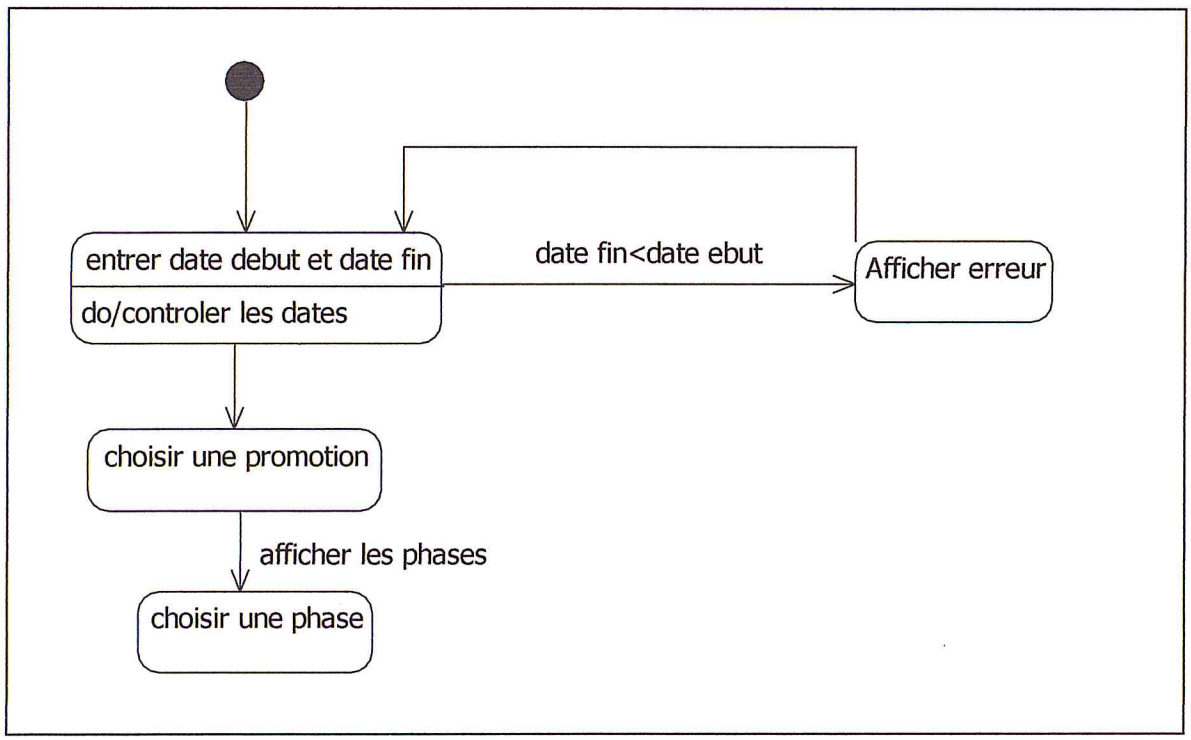


Figure 17 : Diagramme d'états de transition modélisant le choix d'une phase

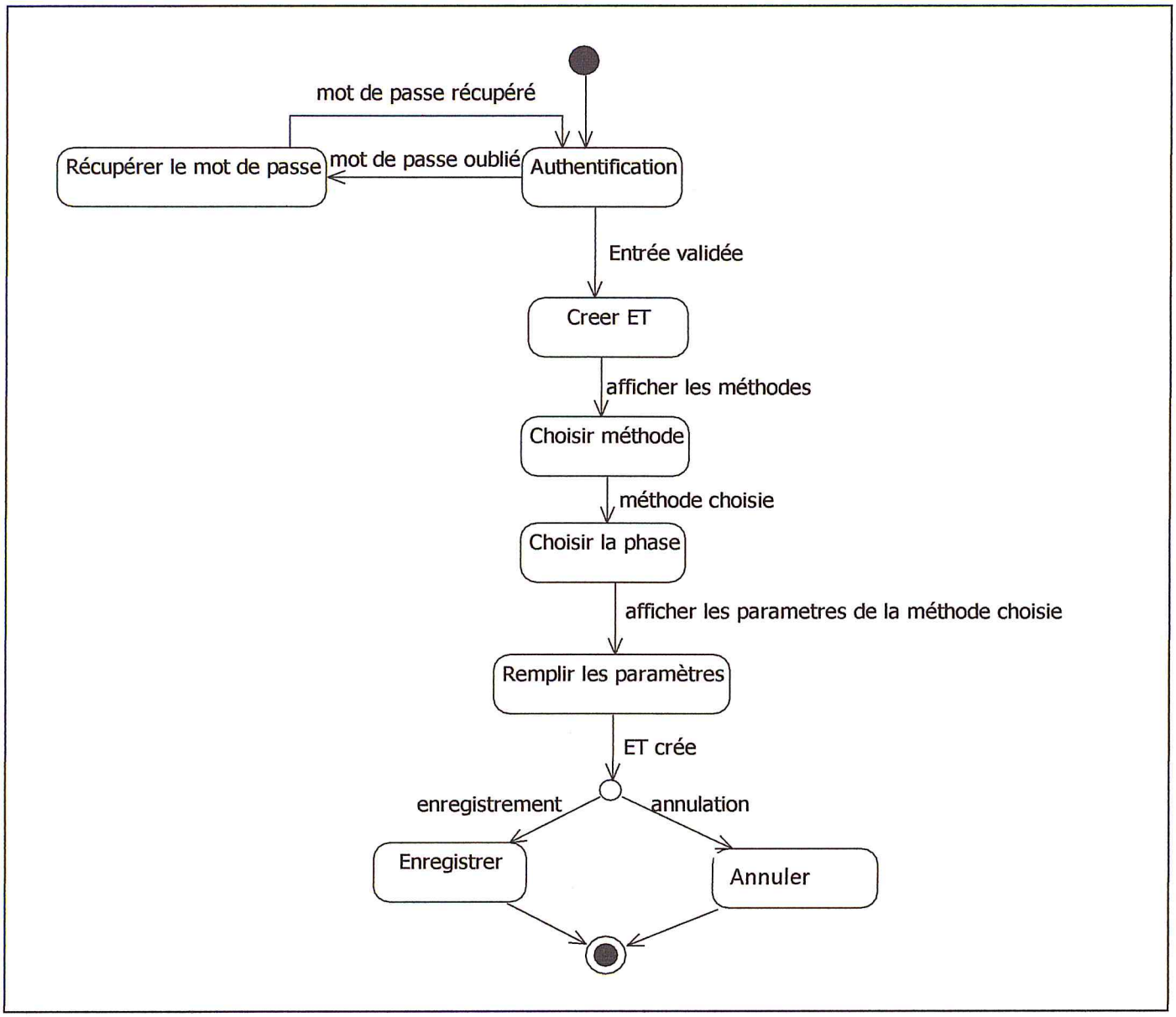
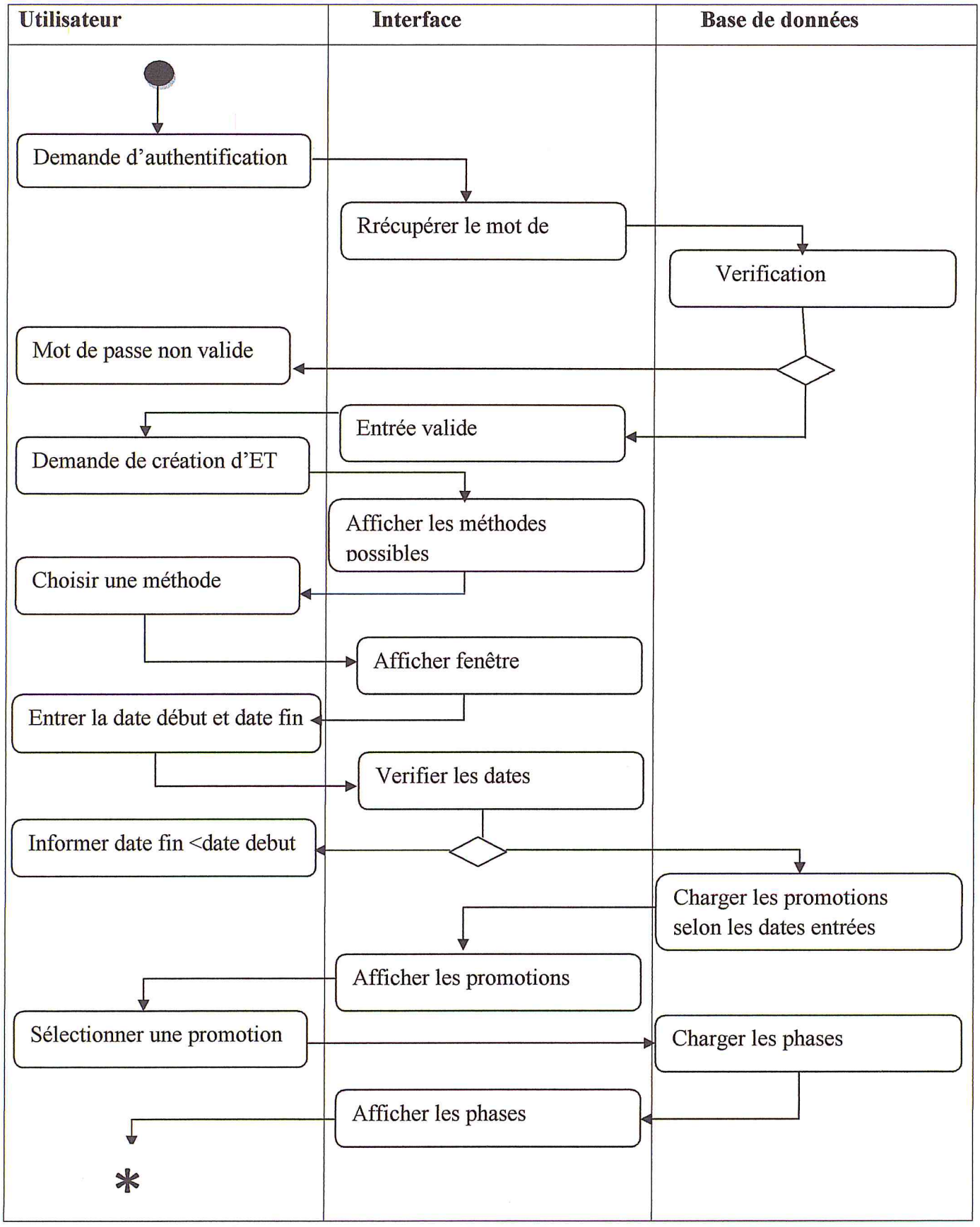


Figure 18 : Diagramme d'états de transition modélisant la création d'un ET

II.5 la réalisation de diagramme d'activités :

Les diagrammes d'activités permettent de mettre l'accent sur les traitements. Ils sont donc particulièrement adaptés à la modélisation du cheminement de flots de contrôle et de flots de données. Ils permettent ainsi de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation. **[1].**


Utilisateur
Interface
Base de données
Demande d'authentification
Rrécupérer le mot de
Verification
Mot de passe non valide
Entrée valide
Demande de création d'ET
Afficher les méthodes possibles
Choisir une méthode
Afficher fenêtre
Entrer la date début et date fin
Verifier les dates
Informer date fin <date debut
Charger les promotions selon les dates entrées
Afficher les promotions
Sélectionner une promotion
Charger les phases
Afficher les phases

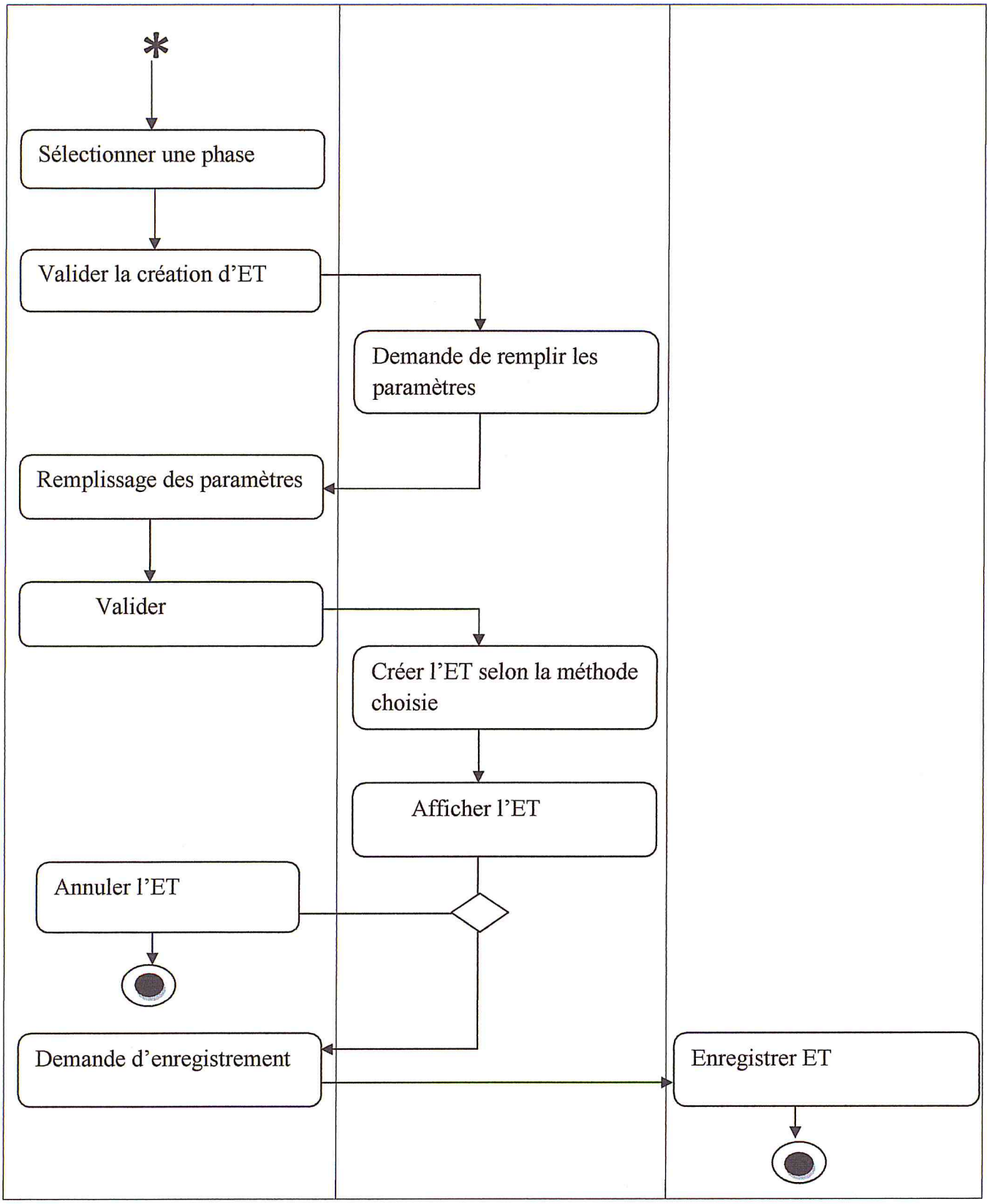


Figure 19: Diagramme d'activité modélisant la création d'un ET

III. Architecture générale de l'application :

Nous présenterons une architecture qui illustre les différents modules quand va réaliser, pour assurer un bon fonctionnement. Elle comporte trois niveaux qui sont :

- le niveau interface ;
- le niveau traitement ;
- le niveau base de données. La figure suivante détaille cette architecture.

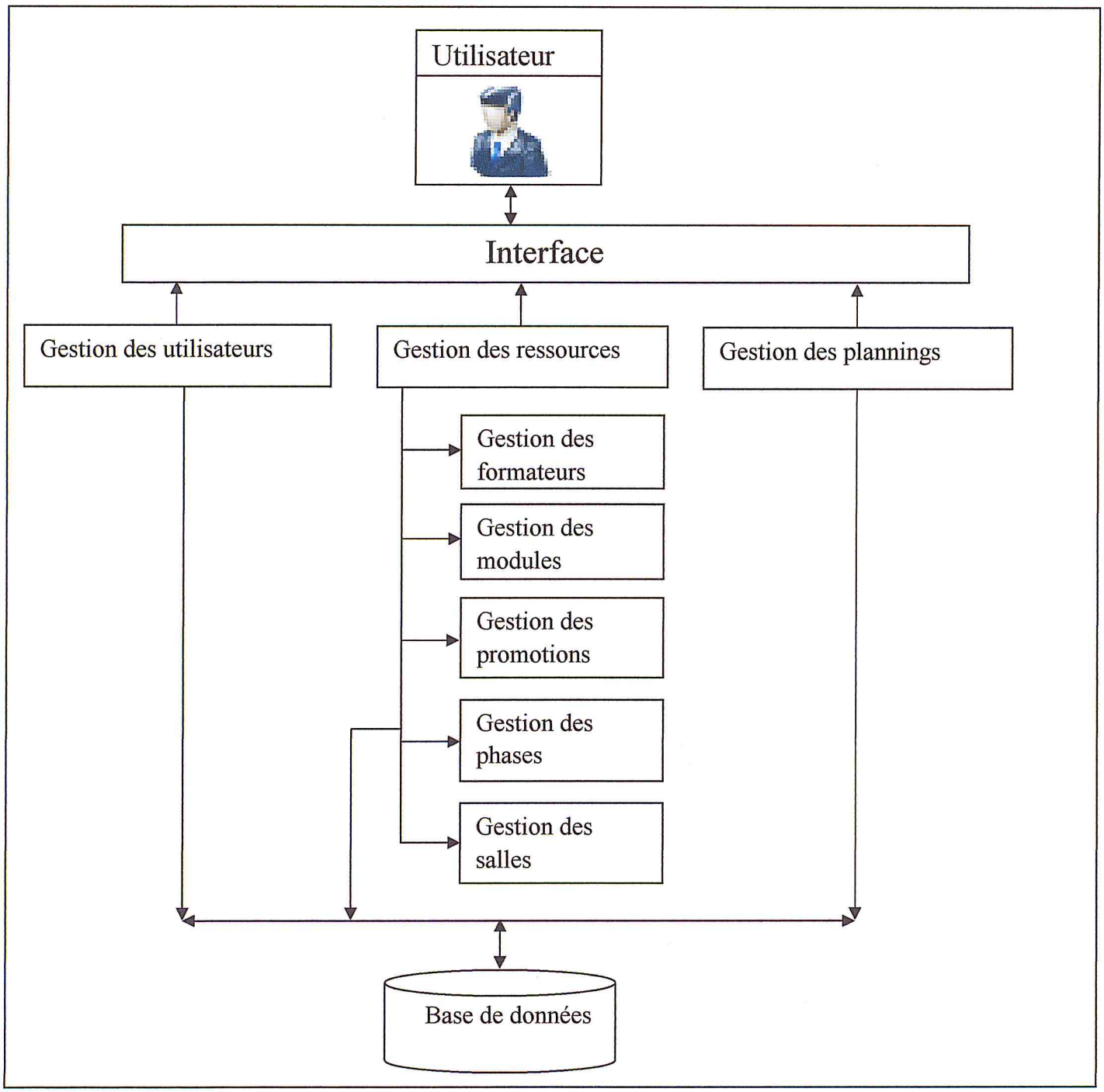


Figure 20: Architecture Modulaire de l'application

1. Niveau interface : ce niveau est essentiel dans la mesure où il représente la partie visible de l'application. Il est constitué d'un ensemble de fenêtre courante qui donne une image générale de l'application. Elle contient :

- Fenêtre d'authentification des utilisateurs.
- Fenêtre de Menu principale.
- Fenêtre de gestion des emplois du temps (la création, la modification et la suppression).
- Fenêtre de Mis a jour.

2. Niveau traitements : il comporte les modules suivants :

- **Gestion des utilisateurs :** son rôle est de créer et sauvegarder de nouveaux comptes utilisateurs (Inscription) dans la base de données prévue à cet effet et à modifier ou supprimer des comptes utilisateurs déjà existants.
- **Gestion des plannings (emplois du temps) :** cela consiste à affecter des ressources (enseignants, matières, salles, filières) à des créneaux horaires.
- **Gestion des ressources :** les ressources sont les enseignants, les matières, les salles, les promotions, les phases, pour chaque ressource on fera la création, la modification et la suppression.

3. Niveau données : est constituée de la base de données de l'application.

CONCLUSION:

Dans ce chapitre, nous avons mis en avant les phases nécessaires à la réalisation de notre application. Une conception détaillée des différents modules est réalisée, par le langage UML et ses différents diagrammes (cas d'utilisations, classes, séquence, états-transitions et activités).

Un modèle de données clair et solide est prêt pour être exécuté En suite nous avons décrire l'architecture modulaire de futur système.

Dans ce qui va suivre, nous entamerons la phase de modélisation où nous exposerons nos choix des modèles de résolution.

INTRODUCTION :

Lors de la résolution de tout problème pratique, sa modélisation sous l'un des problèmes classiques connus de la recherche opérationnelle ou l'intelligence artificielle, est une étape essentielle, car le choix du meilleure modèle facilite la résolution, ce choix est basée sur la nature de problème, une estimation de sa complexité (suivant la taille des données) et l'objectif voulu atteindre.

La définition de l'espace de solution admissible **S** et de la fonction objectif f a souvent une influence directe sur le choix de l'algorithme de résolution, ainsi que sur son efficacité.

Pour notre problème, nous avons adopté des modélisations et des techniques de résolutions très adaptables à sa nature combinatoire, afin de confectionner un bon emploi du temps qui arrange au mieux les formateurs, les stagiaires, et qui satisfait les règles posées par l'administration (les exigences pédagogiques).

I. Description du problème :

Le problème traité consiste à l'élaboration d'un emploi du temps de cours de l'ETB, dans la durée de planification est une semaine.

Il s'agit alors d'accorder à chaque cours la période, le formateur, et la salle qui assure le déroulement du cours sans aucuns conflits. Donc il revient à mieux exploiter et gérer les ressources que l'école dispose, a fin d'aboutir des bons horaires des cours.

Les conflits rencontrés dans ce type de problème sont situés sous l'appellation "contraintes". Ces contraintes se divisent en deux parties : dures qu'il faut respecter, et souples qu'il faut satisfaire au mieux.

II. Les contraintes du problème d'emploi du temps

II.1 Les contraintes dures :

a. Les contraintes de non chevauchement :

- Non chevauchement des enseignants : ne pas affecter deux cours enseignes par le même enseignant a la même période.
- Non chevauchement des classes : ne pas affecter deux cours d'une même classe a la même période.

- Non chevauchement des salles : ne pas affecter plus d'un cours a une salle au même temps

b. Les contraintes du domaine :
   - Certaines matières nécessitent les locaux précis (TP/TD).
   - Certaines matières nécessitent certains périodes (par exemple, les cours de mathématiques nécessitent les périodes de la matinée).
c. Respect de la capacité des locaux : le nombre des étudiants d'une classe ne doit pas dépasser la taille du local affecte a cette classe.
d. Pour une phase, la charge journalière maximale d'une matière est limitée,(par exemple :on ne dépasse pas deux heures de physique par jour).
e. Certains matières nécessitent un nombre de cours successifs,(par exemple :deux heurs successives de mathématiques).
f. Pour une phase ne pas avoir des heures creuses dans la journée.
g. Respecter la charge horaire totale d'un enseignant.

II.2 Les contraintes souples :

a. Eviter les emplacements de certains cours dans des certains périodes, pour des raisons d'incompatibilités entre les matières.
b. Eviter les séances creuses dans une journée pour un enseignant.
c. Respecter la charge horaire journalière maximale d'un enseignant.
d. Respecter la charge horaire journalière maximale d'une phase.
e. Eviter la programmation des cours durant les périodes (12h a 13h).

Pour une phase roulante éviter le déplacement des classes (dans une journée la phase se déplace entre le minimum de salles).

LA RESOLUTION PAR UN ALGORITHME GÉNETIQUE

I. Description de la stratégie de résolution:

Dans cette section, nous présenterons une application de l'algorithme génétique pour la résolution du problème d'ET. Cet algorithme fournit une solution de bonne qualité. La recherche de cette solution consiste à étendre l'algorithme génétique pour un ET réalisable.

I.1 Codage utilisé pour notre algorithme:

Puisque nous sommes dans le cas de la résolution d'un problème d'emploi de temps, le codage que nous avons retenu est de longueur égale au nombre de jours × séances (car la charge horaire de l'ETB est de 20 séances par semaine). L'individu est représenté par une matrice 5×4. La figure 21 présente le codage d'une solution quelconque.

j/S	8-10 h	10-12h	13-15h	15-17h
Samedi				
Dimanche				
Lundi				
Mardi				
Mercredi				

Figure 21: Codage d'un chromosome

I.2 Génération d'un Chromosome:

A partir d'une liste de module :

-choisir un module

-choisir un formateur parmi les formateurs concernés par ce module de manière aléatoire.

-affecter le module au formateur choisi.

-choisir une salle de sorte que la capacité de cette dernier soit supérieur ou égale au nombre de stagiaire de la promotion choisie et selon le type de module (TP ou bien Cours).

- affecter le module à la salle choisie.

-si la liste des modules n'est pas vide alors:

-reprendre les mêmes étapes avec touts les modules de la phase choisie.

-Sinon remplissage aléatoire de la Matrice d'emploi du temps

I.3 Génération d'une population initiale:

Dans cette étape, une population initiale doit être générée, où chaque chromosome représente une solution réalisable du problème. La procédure que nous avons utilisée pour générer la population initiale des individus, adaptée à notre problème est une génération aléatoire de **P** individus. Ce type de génération est le meilleur choix, car il permet l'hétérogénéité de la population. Le nombre d'individu est défini par l'utilisateur.

I.4 Fonction d'évaluation : *fitness*

Chaque chromosome (emploi du temps) doit avoir un score qui lui permet par la suite d'être classé, et ce score est calculé comme suit :

-initialiser le score de chaque chromosome a 20 car chaque matrice contient 20 séances.

-s'il y a un chevauchement des formateurs avec les autres emplois du temps déjà réalisé alors diminuer le score

-si le formateur est de type vacataire alors vérifier la disponibilité de ce dernier

Si le formateur n'est pas disponible durant cette séance alors diminuer le score.

- s'il y a un chevauchement des salles avec les autres emplois du temps déjà réalisé alors diminuer le score.

I.5 Opérateur de Sélection :

Selon la méthode choisie par l'utilisateur, on doit appliquer des différentes étapes telles que pour :

- Roulette : 1- Créer une roulette ou chaque individu est dupliqué selon son score
  2-Les individus sont sélectionnés aléatoirement et les individus ayant la plus grande valeur de score auront plus de chance d'être choisis.
  3- La suppression des individus non choisi
- Rang : 1- Trier d'abord la population par score.

2- Chaque chromosome se voit associé un rang en fonction de sa position.

3- Sélectionner aléatoirement les individus.

4- La suppression des individus non choisi

- Tournoi : 1-Former des paires de chromosomes

  2-le chromosome de paire qui a la plus grande valeur sera sélectionnée

  3- La suppression des individus non choisi

I.6 Opérateur de Croisement :

Pour obtenir de nouveaux individus (enfants) à partir d'une population initiale d'une itération, nous utiliserons l'opérateur de croisement décrit dans la section suivante :

- Prend en entrée un couple d'individus parents Soit P1 et P2 et renvoie un couple d'individus enfants E1et E2 tel que :
  1. pour chaque P1 [i1] [j1] = x et P2 [i2] [j2]=x' tel que : x et x' représentent des séances.
  2. Si x=x' alors : E1 [i1] [j2]=x et E2 [i2] [j1]=x avec i représente les jours et j les séances.

I.7 Opérateur de Mutation :

Les individus de la population issue du croisement vont ensuite subir un processus de mutation. L'opérateur de mutation utilisé est le suivant :

1. Transformer chaque individu X de la population en un individu X' tel que :

   X' est obtenue à partir d'un nombre de permutation des séances de X. (en respectant la contrainte des Tps qui doivent être déroulé dans la même journée selon le volume horaire de TP).

Critère d'arrêt

Le critère d'arrêt utilisé est le nombre de générations égal à IterMax. Après IterMax générations, l'algorithme génétique s'arrête et donne le meilleur chromosome qui possède le meilleur score.

II. Organigramme général d'un Algorithme Génétique :

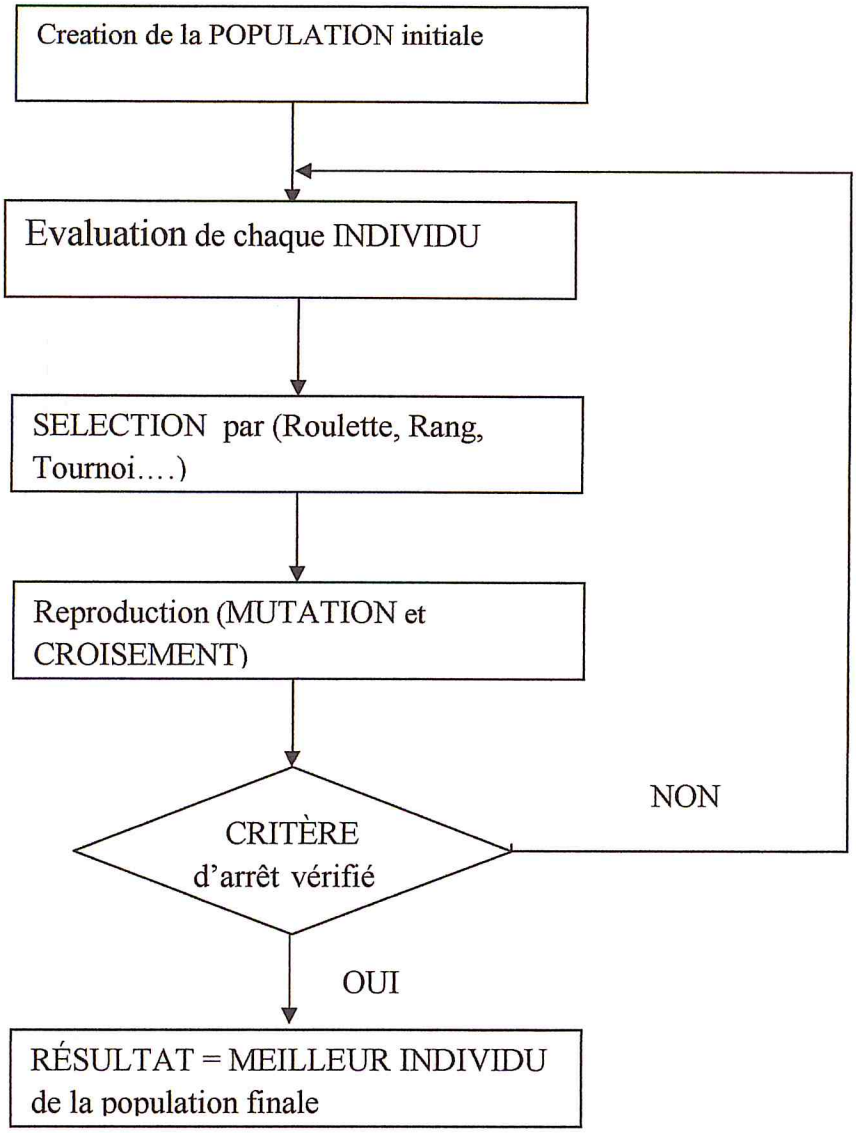


Figure 22 : Organigramme général d'un Algorithme Génétique

Notre algorithme génétique pour l'ET déterministe est initialisé par une taille de population P, une méthode de sélection, un croisement, une mutation, et par un critère d'arrêt (IterMax). Les étapes de cet algorithme sont présentées dans l'algorithme suivant.

III. Algorithme génétique :

L'algorithme de résolution se déroule comme suit :

Les données :

Max// représente le nombre maximum d'itération.

nbr_ind // nbr_ind représente le nombre d'individu, entrer par l'utilisateur

S_0 // la solution initiale.

Choix_méthode // méthode de sélection Roulette, Tournoi, Rang), entrer par utilisateur

Début

1: Initialiser : Max= nbr_ind /2, $\mu = 0$, Choix_méthode

2: Générer aléatoirement P (nbr_ind) séquences d'emploi du temps

3: ***Tant que*** μ <Max faire ***faire***

4 : Evaluer chaque individu de **P**

5 : $\mathbf{S_0}$ = Choisir parmi ces individus la meilleure solution

4: Sélectionner (**P**, Choix_méthode) les parents.

5: Supprimer les individus non sélectionnés.

6 : Croisement (indiv1, indiv2) deux a deux les individus de P, chaque croisement donnera deux fils.

7 : Ajouter les fils a la population P.

8 : **P'**=Mutation(**P**), **P'** est la nouvelle population

9 : Evaluer les individus de **P'**

10:Pour tout **X** de la population **P'**

11 : Si score(**X**) >score(S_0) alors

12 : $\mathbf{S_0}$=**X**

13 : $\mu = \mu + 1$

14 : ***Fait***

15 : Solution Optimale =$\mathbf{S_0}$.

Fin.

LA RESOLUTION PAR UN RECUIT SIMULE

I. Description de la stratégie de résolution:

Un modèle de recuit est défini par un jeu de réglages de différents paramètres. Les réglages diffèrent selon le problème traité et les résultats escomptés de l'utilisateur.

I.1 Configuration :

Est la même que le chromosome d'algorithme génétique, une matrice de taille 5×4 tel que les lignes représentent les jours et les colonnes les séances.

I.2 Solution initiale :

Son choix n'est pas primordial. On peut prendre une configuration au hasard

I.3 Fonction Voisinage :

A partir d'une solution en entrée, et avec quelque transformation de cette dernière, on obtient une nouvelle solution appelée «voisin ».

I.4 Température initiale :

Le paramètre T (température) est un réel positif et fixe. La température permet de contrôler l'acceptation des dégradations. Et elle doit être suffisamment grande pour que des transformations coûteuses soient acceptées

Selon [11] $\boldsymbol{T\,init = 10000}$ donne des résultats satisfaisants en termes de cout et de temps d'exécution.

I.5 Température Minimale :

C'est la température finale.

$$\boldsymbol{T\,min = 5000} \qquad [11]$$

I.6 Alpha :

Permettant de faire décroitre la température par la fonction suivante :

$$\boldsymbol{1 - (ln(T\,init) - ln(Tmin))/nbre\_iter} \qquad [11]$$

II. Organigramme général de Recuit Simulé:

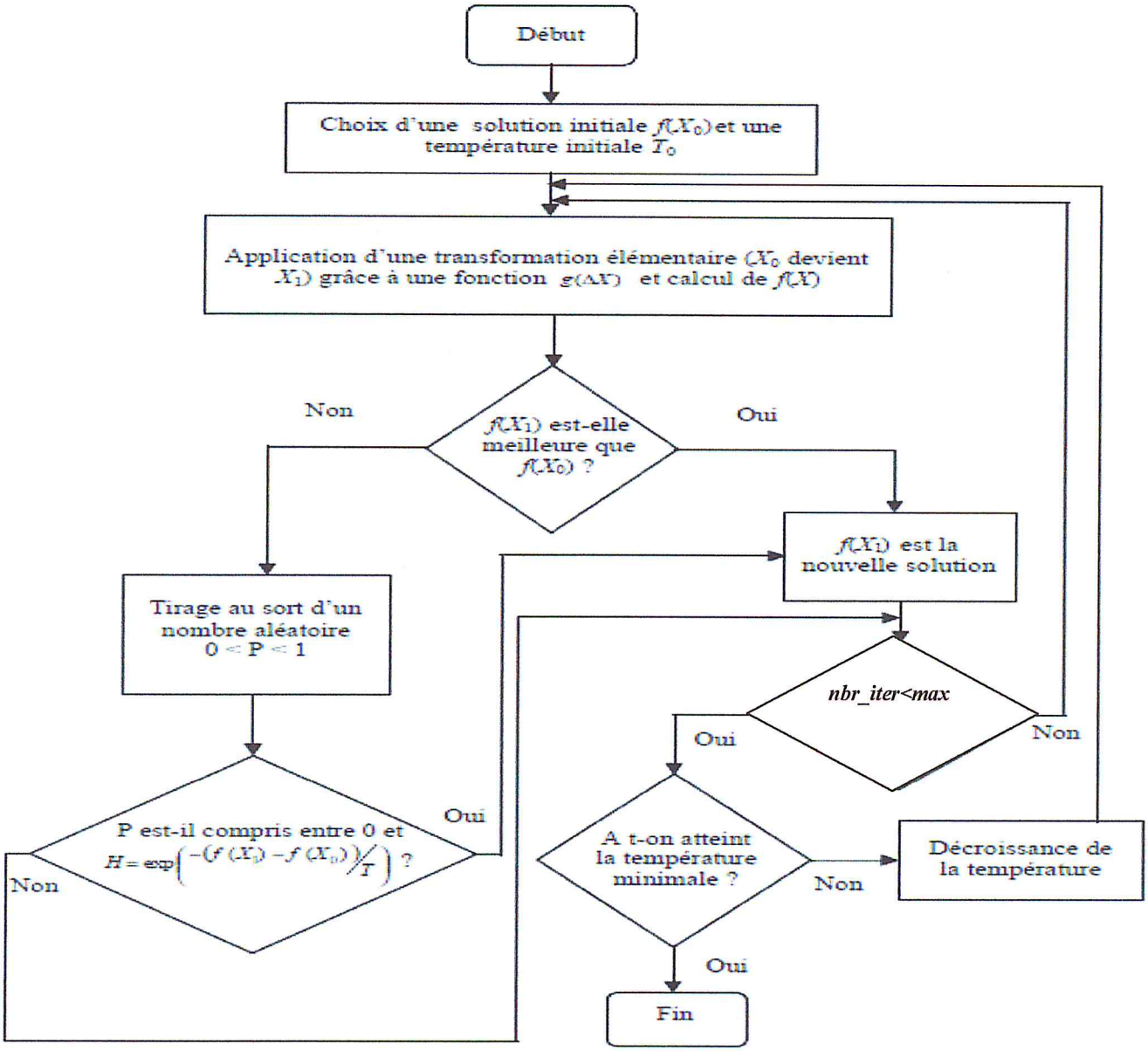


Figure 23 : Organigramme général de Recuit Simulé

III. Algorithme de Recuit Simulé:

L'algorithme de résolution se déroule comme suit :

Les données :

Tinit // Température initiale entrer par l'utilisateur.

Tmin// Température minimale entrer par l'utilisateur.

S_0 // la solution initiale.

Début

1 : Initialiser Tinit, Tmin

2 : ***Tant que*** Tinit >= Tmin ***faire***

3 : Alpha= $\mathbf{1} - (\boldsymbol{ln(T\ init)} - \boldsymbol{ln(Tmin)})/\boldsymbol{Max}$;

4 : nbre_iter =0

5 : ***Tant que*** nbre_iter<Max

6 : **S'**=voisin ($\mathbf{S_0}$)

7 : Delta= Evaluer **(S')**- Evaluer (S_0).

8 : p $\in$[0,1]

9 : Si Delta>0 alors

10: $\mathbf{S_0}$=**S'**

11: **Sinon**

12: **Si** p < exp $\left\lfloor \frac{-\text{Delta}}{\text{Tint}} \right\rfloor$ **alors**

13:$\mathbf{S_0}$=**S'**

14: **Fsi**

15: **Fsi**

16: nbre_iter =nbre_iter+1

17: **Fait**

18 : Tinit=Tinit×Alpha

Fin

ETUDE COMPARATIVE

Apres la résolution du problème d'ET avec deux algorithmes différents, nous allons faire une comparaison entre eux par rapport en temps d'exécution, la qualité de solution,…..

I. Avantage et Inconvénient :

Le tableau ci-dessous montre les avantages et les inconvénients des deux algorithmes utilisés lors de résolution.

	Algorithme Génétique	**Recuit Simulé**
Avantage	• On n'ait pas besoin de connaitre les caractéristiques de la solution du problème, mais seulement de déterminer parmi les solutions quelle est la meilleure. • Nombre de solutions important. • permet d'explorer N3 solutions possibles. • Explore tous l'espace des points en même temps, ce qui limite les risques de tomber dans des optimums locaux.	• simple à programmer et à paramétrer • On peut inclure facilement dés contraintes dans le corps du programme. • Pas couteux en temps d'exécution. • Facile a programmer.
Inconvénient	• coûteux en temps de calcul (l'algorithme est lent). • Paramètres difficiles à fixer (taille de population, nombre de génération à simuler…). • Difficile a programmer (les paramètres comme la taille de la population et la fonction d'évaluation sont difficiles à établir).	• qu'une fois l'algorithme piégé à basse température dans un minimum local, il lui est impossible de s'en sortir tout seul. • Il faut une diminution de la température « suffisamment lente », pour voir la solution s'améliorer

II. Analyse et résultat :

Pour apprécier la qualité des Algorithmes, il est important de comparer les résultats obtenus avec un emploi du temps optimal, ce qui est en général impossible à déterminer.

Apres un nombre d'exécution, nous avons eu les résultats montrés sur des courbes suivants :

Pour l'algorithme génétique, on a pris comme paramètres le nombre d'individus par rapport au temps d'exécution ce qui donne :

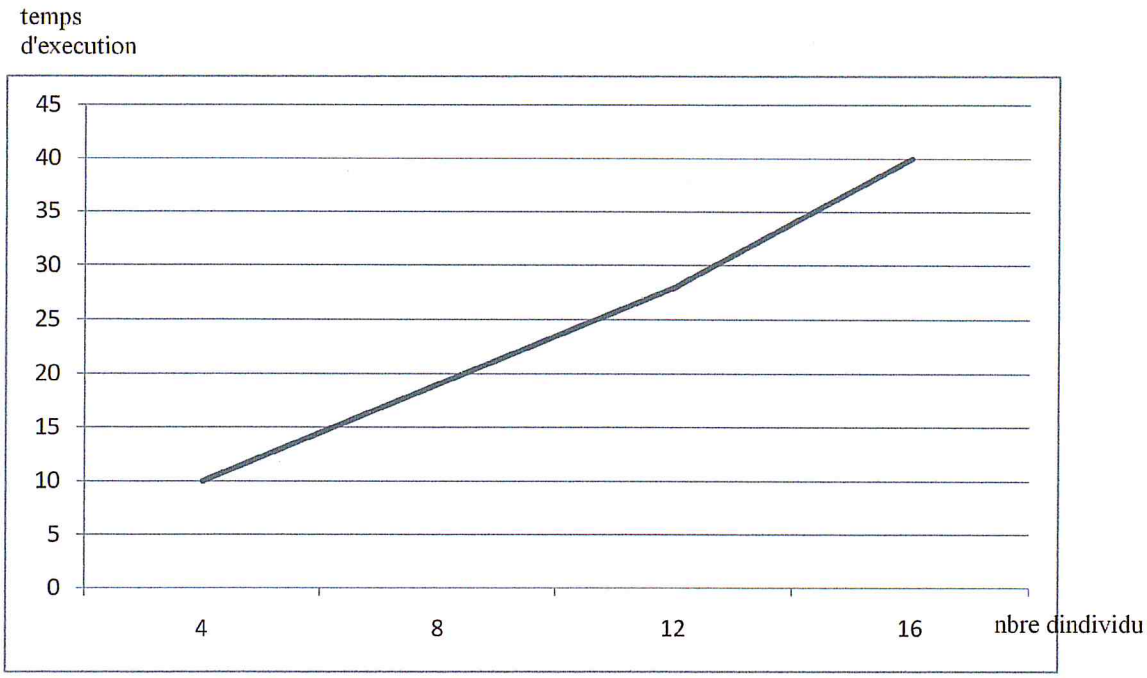


Figure 24 : Résultat obtenu en utilisant l'Algorithme Génétique

Et pour le Recuit Simulé nous avons pris comme paramètre le nombre d'itération par rapport au temps d'exécution:

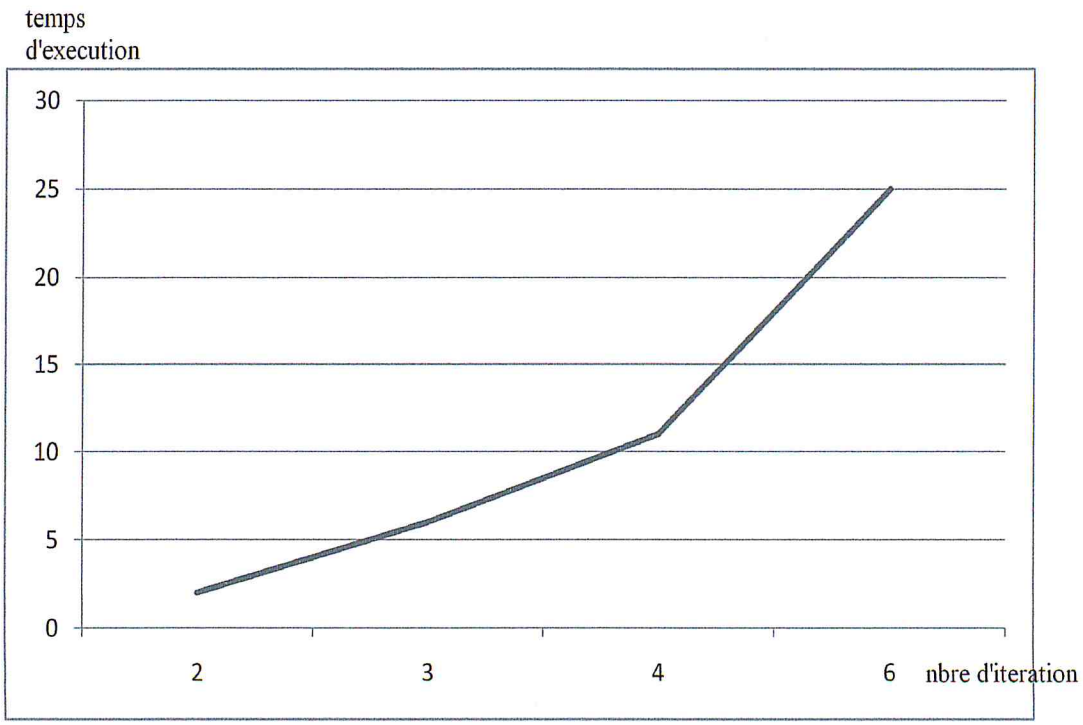


Figure 25 : Résultat obtenu en utilisant le Recuit Simulé

Et pour finir nous avons testé un nombre de problème avec les deux algorithmes et les résultats obtenus sont montrées sur les graphes suivants, qui montrent une comparaison entre ces algorithmes le premier par rapport au **temps d'exécution** et le deuxième par rapport à la **performance** calculé comme suit : Score(P)/ Score ($P_{aleatoire}$) où $P_{aleatoire}$: c'est le score d'une Solution aléatoire.

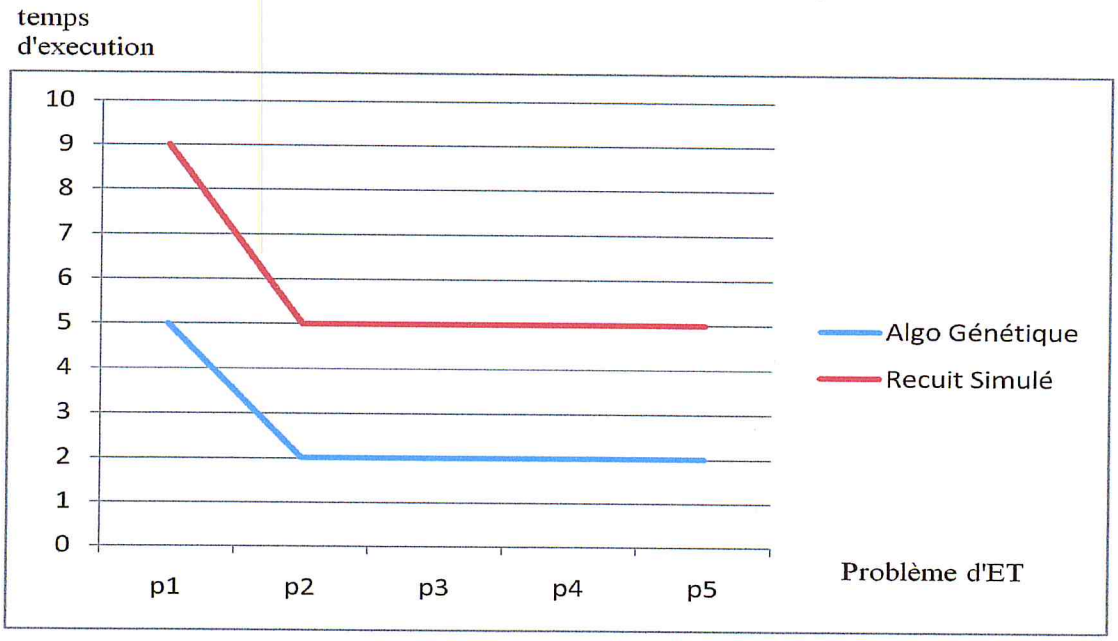


Figure 26 : Comparaison entre l'AG et le RS par rapport au temps

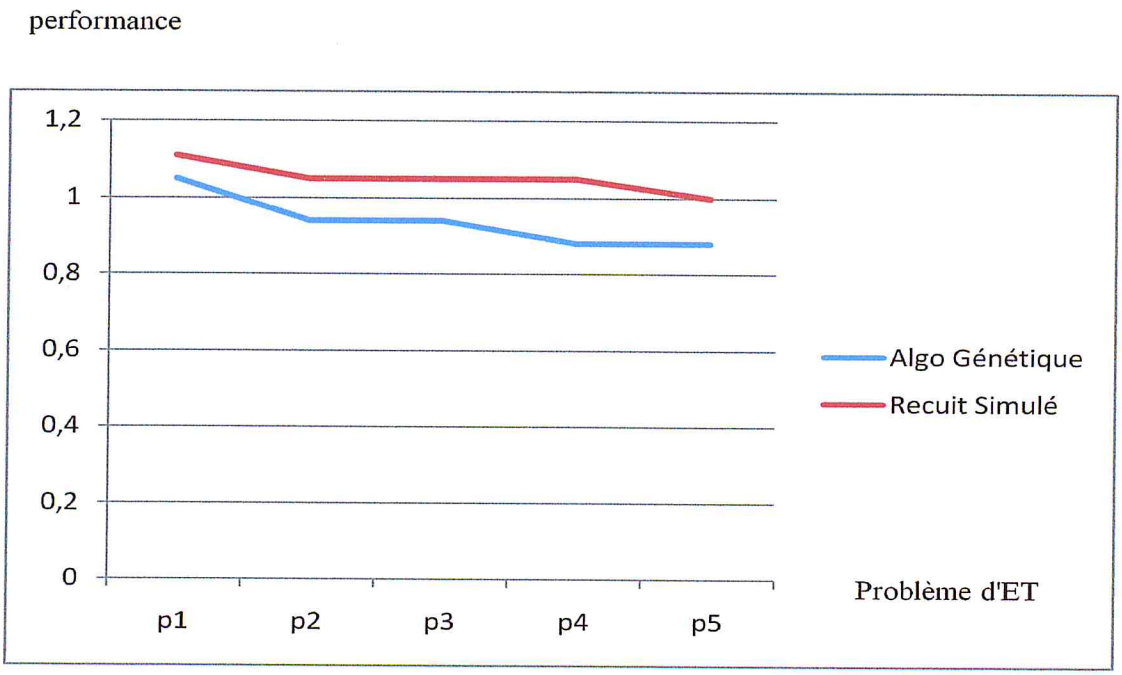


Figure 26 : Comparaison entre l'AG et le RS par rapport à la performance

D'après ces graphes, on remarque que le RS s'exécute plus rapidement par rapport à l'algorithme génétique, mais l'AG donne des résultats de bonne qualité.

CONCLUSION :

Dans ce chapitre on a pu :

- ✓ Définir une représentation des variables qui permet une exécution plus rapide au programme écrit sous langage JAVA.
- ✓ Régler les paramètres des deux algorithmes (Algorithme Génétique et Algorithme Recuit Simule) suivant les contraintes et le résultat estimé de la résolution de notre problème.
- ✓ Donner un organigramme d'exécution de l'algorithme appliqué sur les problèmes d'emploi du temps.
- ✓ En fin écrire un algorithme qui permet d'avoir la valeur d'adaptation de l'individu(ET) le plus performant de chaque génération pour un nombre définie de génération, et en fonction de deux valeurs d'entrée pour l'algorithme génétique :
    - Nombre d'individu.
    - Choix de méthode de sélection.

  Et trois valeurs d'entrée pour le recuit simulé :

    - La température initiale.
    - La température minimale
    - La constante Alpha.

Dans ce qui va suivre, nous entamerons la phase d'implémentation et de réalisation où nous exposerons nos choix concernant la technologie qui a servi à la mise en place de notre outil (environnement de développement, **SGBD**).

INTRODUCTION :

En se basant sur la conception élaborée dans le chapitre précédent, on a développé un outil permettant la gestion des emplois du temps.

Ainsi, dans ce chapitre, nous présentons le langage de programmation que nous avons utilisée dans le développement de l'application. Puis, nous donnerons un aperçu sur le langage de programmation et le système de gestion de bases de données intervenant dans le fonctionnement de notre outil.

Enfin, nous montrerons, à l'aide d'imprime écran, les résultats obtenus.

I. implémentation de la base de données :

La base de données de notre application est implémentée en se base sur le diagramme de classes UML qu'on a élaboré lors de notre application, suivant les règles de transformation suivantes :

- Chaque classe devient une relation.
- Les attributs de la classe devient des attribue de la relation.
- Si la classe possède un identifiant, il devient la clé primaire de la relation, sinon, il faut ajouter une clé primaire arbitraire.
- Pour les relations de type 0..1 à plusieurs ou 1à plusieurs, on prend la clé de la table qu'a comme cardinalité de la relation <<0..1>>ou<<1>> et l'ajout aux attributs de l'autre table.

Ainsi on obtient notre base de données va contenir les tables suivant.

Formateur (cod_fomat, nom_format, prén_format, type).

Module (cod_mod, lib_mod) .

Promotion (cod_promo, lib_promo, date_D, date_F).

Phase (cod_phase, lib_phase, date_D, date_F, cod_promo).

Salle (cod_salle, capacité).

Groupe (cod_gr, cod_prom, nbre_satgiere)

Emploi (cod_emp, cod_phase).

Séance (cod_emp, cod_sc, cod_mod, cod_fomat, cod_salle, cod_gr).

A (cod_phase, cod_mod, vhm_hebdo, nbre_ds).

Disponibilite(cod_format,cod_sc)

Peut-enseigner (cod_mod, cod_fomat).

Se_fait (cod_mod, cod_salle)

Table Formateur :

Attribut	**Signification**	**Type**	**Taille**
Cod fomat	Code formateur	N	4
Nom_format	Nom formateur	A	20
Prén_format	Prénom formateur	A	20
Type	Type formateur	A	20

Table Module :

Attribut	**Signification**	**Type**	**Taille**
Cod_mod	Code module	N	4
Lib_mod	Nom du module	A	20

Table Promotion :

Attribut	**Signification**	**Type**	**Taille**
Cod_promo	Code promotion	N	4
Lib_promo	Nom de la promotion	A	20
Date_D	Date début de promotion	N	15
Date_F	Date fin de promotion	N	15

Table Phase :

Attribut	Signification	Type	Taille
Cod_phase	Code phase	N	4
Lib_phase	Nom de la phase	A	20
Date_D	Code promotion	A	4
Date_F	Date début de phase	N	4
Cod_promo	Date fin de phase	N	15

Table salle :

Attribut	Signification	Type	Taille
Cod_ salle	Code formateur	N	4
Type_salle	Type Salle	A	10
specialite	Spécialité	A	10
capacite	Capacité	N	4

Table groupe :

Attribut	Signification	Type	Taille
Cod_gr	Code groupe	N	4
Cod_prom	Code promotion	N	4
Nbre_stagiere	Nombre de stagiaire	N	4

Table emploi :

Attribut	Signification	Type	Taille
Cod_ emp	Code emploi	N	4
Cod_ph	Code phase	N	4

Table Séance :

Attribut	Signification	Type	Taille
Cod_sc	Code phase	N	4
Cod_mod	Code module	N	4
Cod_fomat	Code formateur	N	4
Cod_salle	Code salle	AN	4
Cod_gr	Code groupe	N	4

Table A :

Attribut	Signification	Type	Taille
Cod_phase	Code phase	N	4
Cod_mod	Code module	N	4
vhm_hebdo	Volume du module	N	4
nbre ds	Nombre de semaine	N	4

Table disponibilite:

Attribut	Signification	Type	Taille
Cod_ format	Code formateur	N	4
Cod_sc	Code seance	A	10

Table Peut enseigner :

Attribut	Signification	Type	Taille
Cod_ format	Code module	N	4
Cod_mod	Code formateur	N	4

Table se _fait :

Attribut	Signification	Type	Taille
Cod_ mod	Code module	N	4

Cod_salle	Code salle	N	4

II. Environnement de développement :

Java est un langage de programmation à usage générale, évolué et orienté objet dont la syntaxe est proche du C. Il existe 2 types de programme en java : les applets et les applications, Une application autonome (stand alone program) est une application qui s'exécute sous le contrôle direct du système d'exploitation. Une applet est une application qui est chargée par un navigateur et est exécutée le contrôle de celui-ci.

III. Les outils :

La réalisation de notre logiciel nécessite les outils suivant :

- L'installation de **SGBD oracle 9I.**
- L'installation de java (JDK+un éditeur de texte qui nous permette de sauvegarder les fichiers au format de texte brute « Eclipse »), pour la programmation de notre application

IV. Les interfaces de l'application :

Dans ce qui suit, nous allons présenter le fonctionnement de notre système en présentant ses différentes interfaces.

IV.1. 1 Interface d'Authentification

En premier lieu, tout accès a la base de données est verrouillé alors l'utilisateur doit d'abord s'inscrit.

L'utilisateur s'identifie en saisissant, dans la zone d'identification (Figure 28), un login et un mot de passe qui lui sont propres après avoir sélectionner son rôle (Administrateur Responsable de SAF ou Invité). Le vérifier ces informations dans la base de données, voici les scénarios possibles :

- La connexion s'établie et l'interface appropriée s'affichera.
- La connexion est échouée (mot de passe ou le login est erroné).

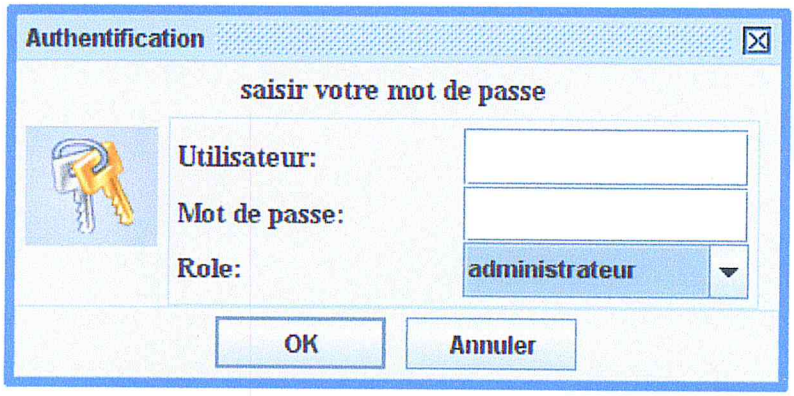


Figure 28: Fenêtre d'authentification

IV.2 Interface menu principale :

C'est la première page qui apparaît à l'administrateur après la validation de l'authentification après duquel il peut faire choisir plusieurs fonctionnalités selon son besoin.

Figure 29: Fenêtre principale

IV.3 Interface Choix de phase :

Cette fenêtre permet à l'utilisateur de choisir une phase pour la création d'un emploi du temps, en remplissant un nombre de champs (Figure 30).

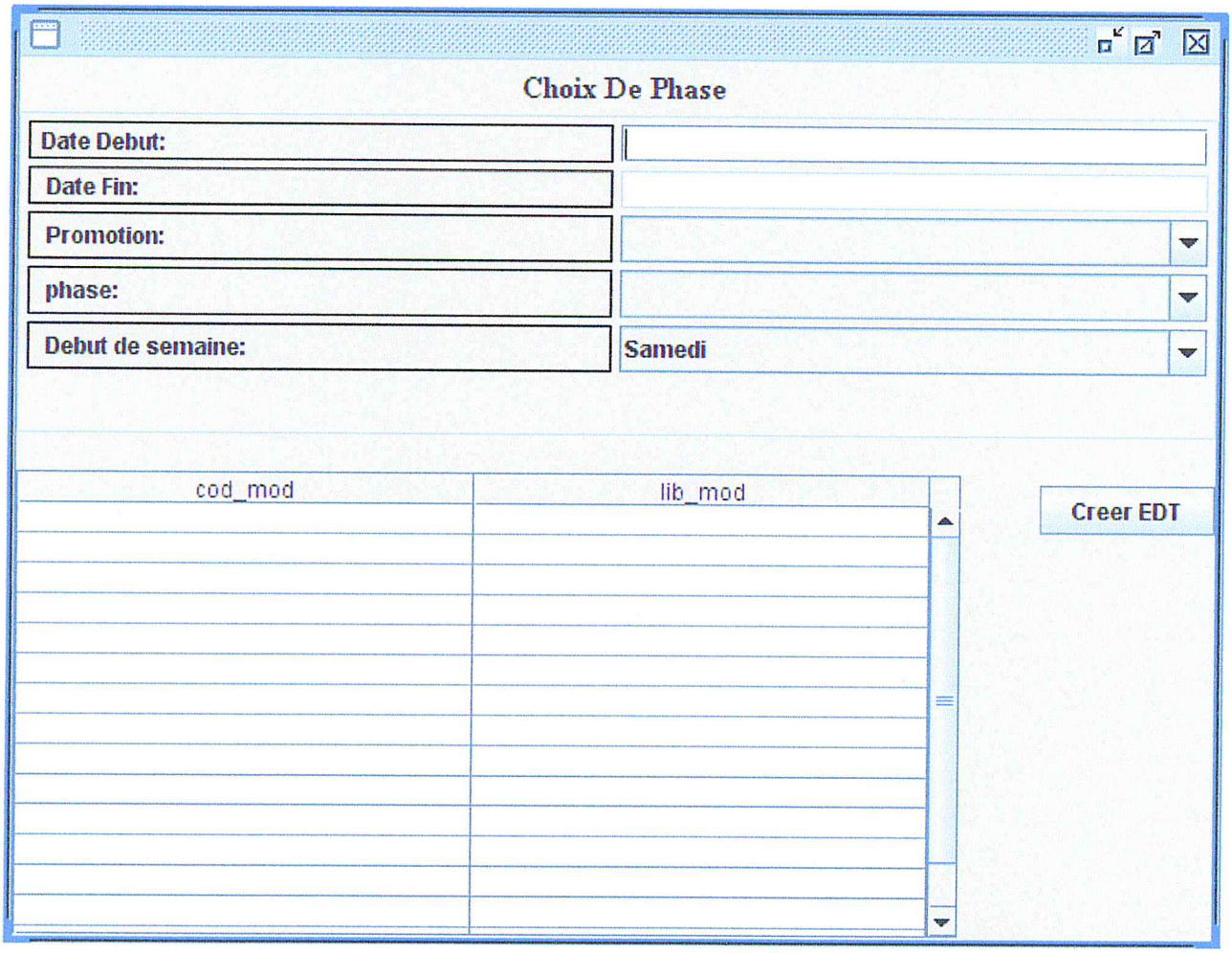


Figure30: Fenêtre Choix de phase

IV.4 Interface les paramètres d'Algorithme Génétique :

Elle apparait après le choix de la phase pour pouvoir créer un emploi du temps suivant les paramètres entres par l'utilisateur ou bien des valeurs par défaut (Figure 31)

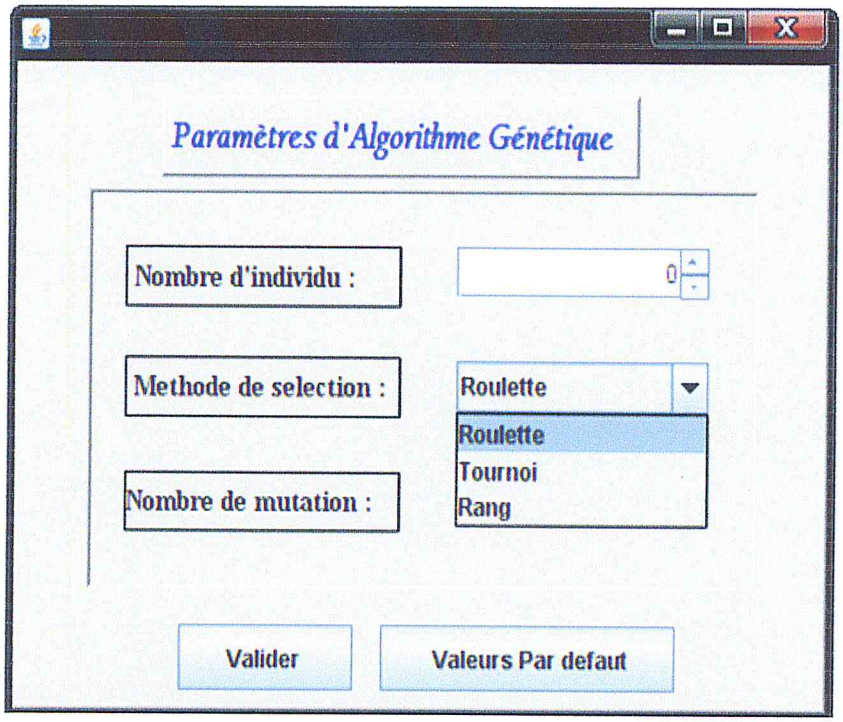


Figure 31: Fenêtre Paramètres d'Algorithme Génétique

IV.5 Interface les paramètres de Recuit Simulé :

Elle apparait après le choix de la phase pour pouvoir créer un emploi du temps suivant les paramètres entres par l'utilisateur ou bien des valeurs par défaut (Figure 32)

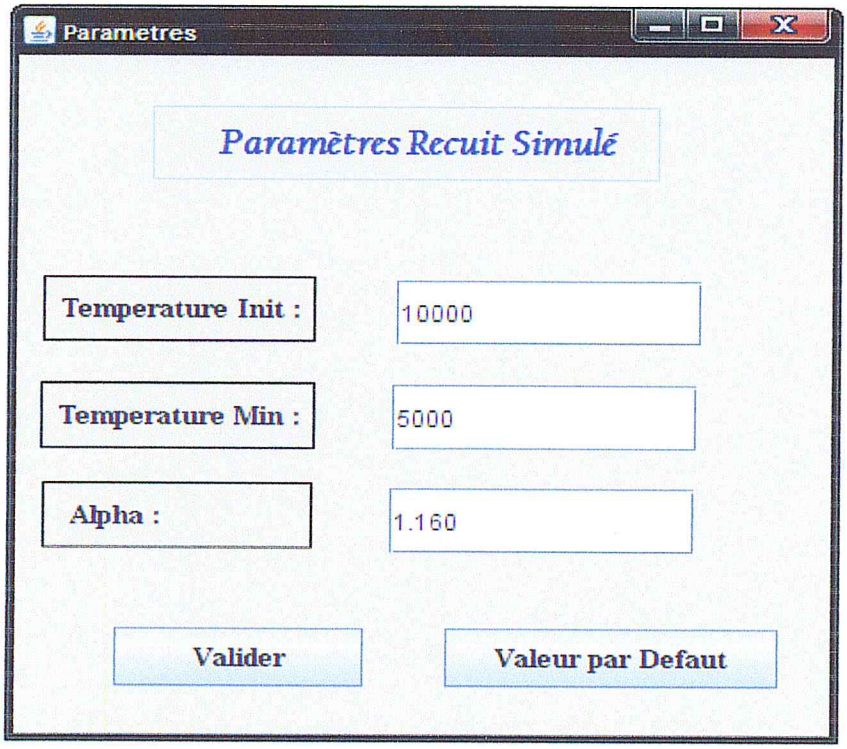


Figure 32: Fenêtre Paramètres de Recuit Simulé

IV.6interface Emploi du temps Automatique :

Apres le choix de la méthode de création d'ET par Algorithme génétique ou bien le Recuit Simulé (Figure 31 et Figure 32), la fenêtre (Figure 33) s'affichera un emploi du temps réalisé par la méthode choisi, puis l'utilisateur peut l'enregistrer ou bien l'imprimer:

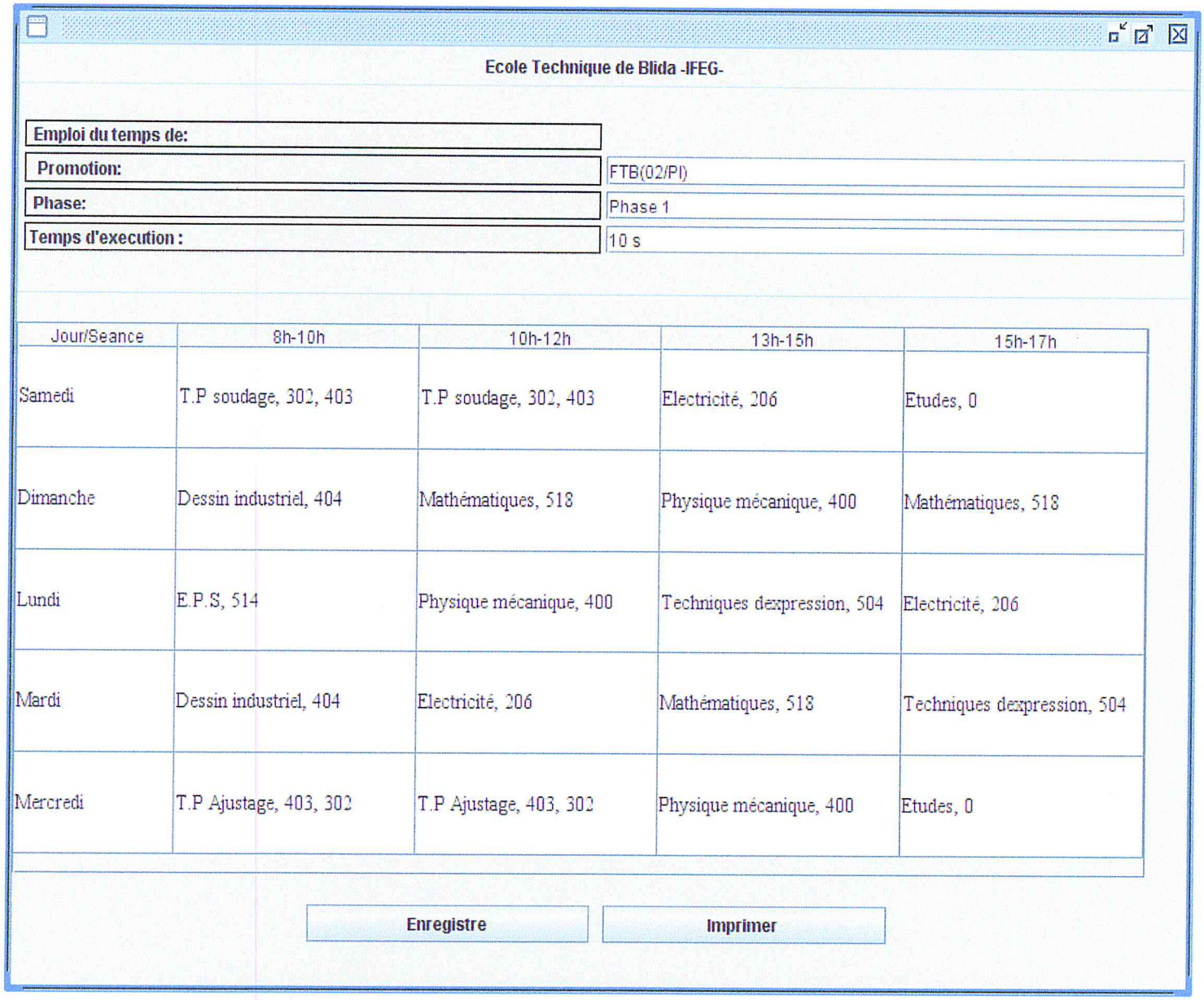


Figure 33: Fenêtre d'affichage Emploi du temps Automatique

IV.7 interface Emploi du temps manuel :

Cette fenêtre permet de créer un nouvel emploi du temps manuellement, voici la fenêtre:

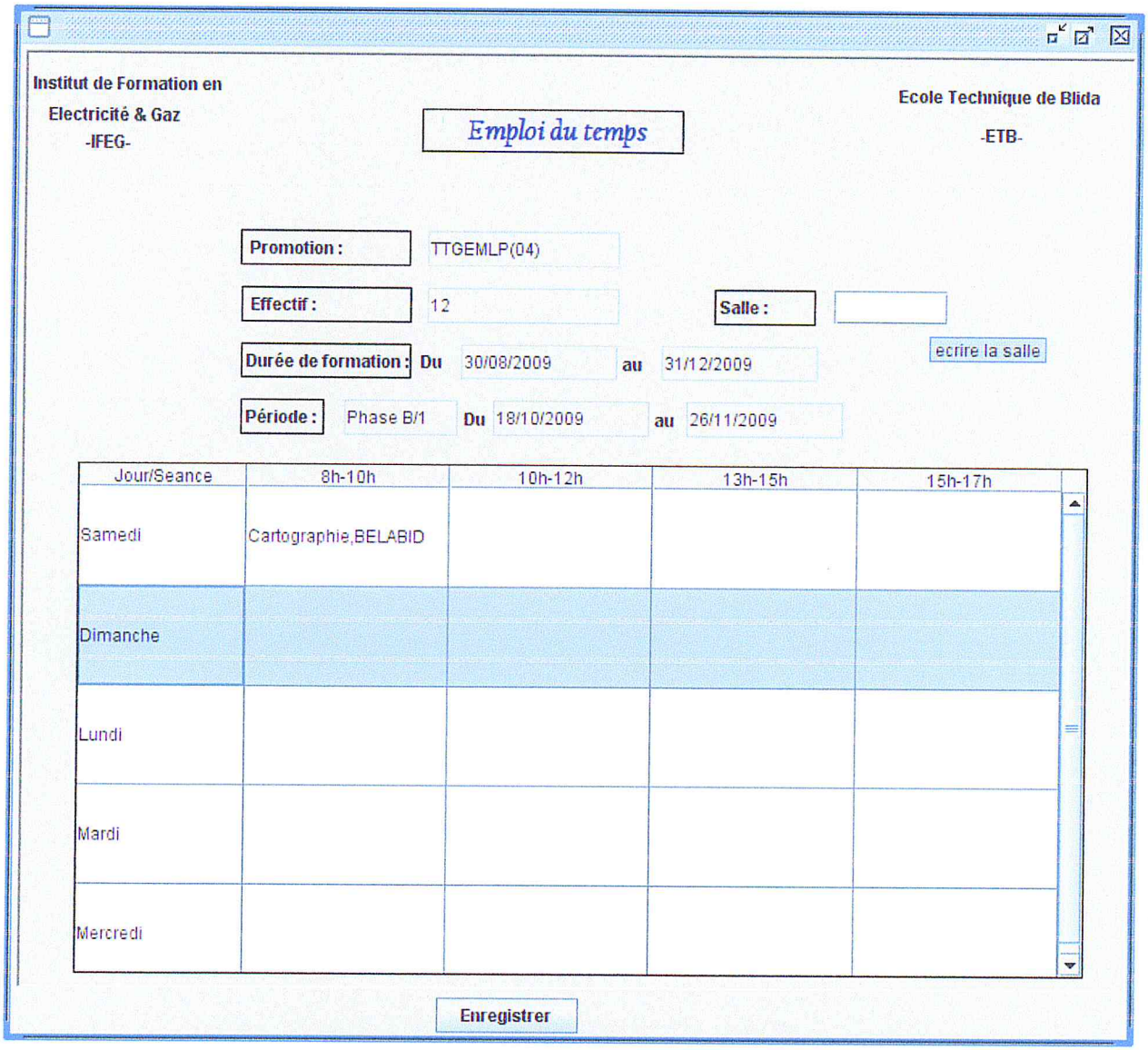


Figure 34: Fenêtre de création d'emploi du temps Manuel

IV.8 Interface les paramètres de Recuit Simulé :

Dans cette interface, on y trouve un menu dans lequel figure :

1- Ajout d'un formateur. 2-Ajout d'un module.

3- Ajout d'une promotion. 4-Ajout d'une phase.

5-Ajout d'une compétence. 6-Ajout d'une salle.

Figure 35: Fenêtre d'ajout

Voici un exemple (Figure 36) de commande d'ajout, l'utilisateur peut ajouter une promotion quelconque juste par un simple clique (1), une autre fenêtre s'affiche où l'utilisateur peut saisir toutes les informations qui concernent la promotion ajoutée, après la validation de l'ajout cette dernière s'ajoutera a données.

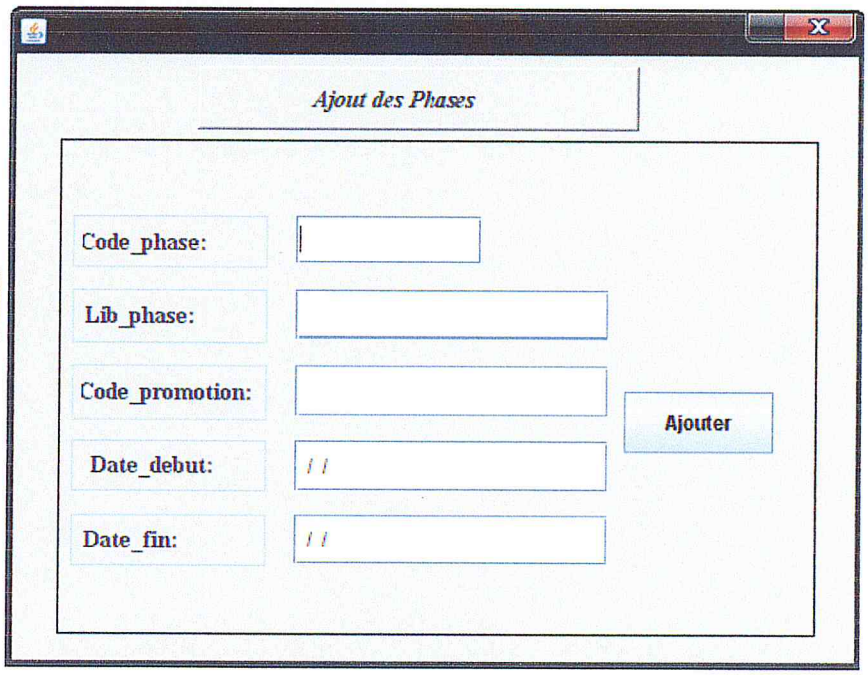


Figure 36: Fenêtre d'ajout d'une phase

IV.9 Les contraintes des enseignants

Dans cette page l'administrateur introduit les contraintes de disponibilités de chaque vacataire qui exprime les souhaits des enseignants.

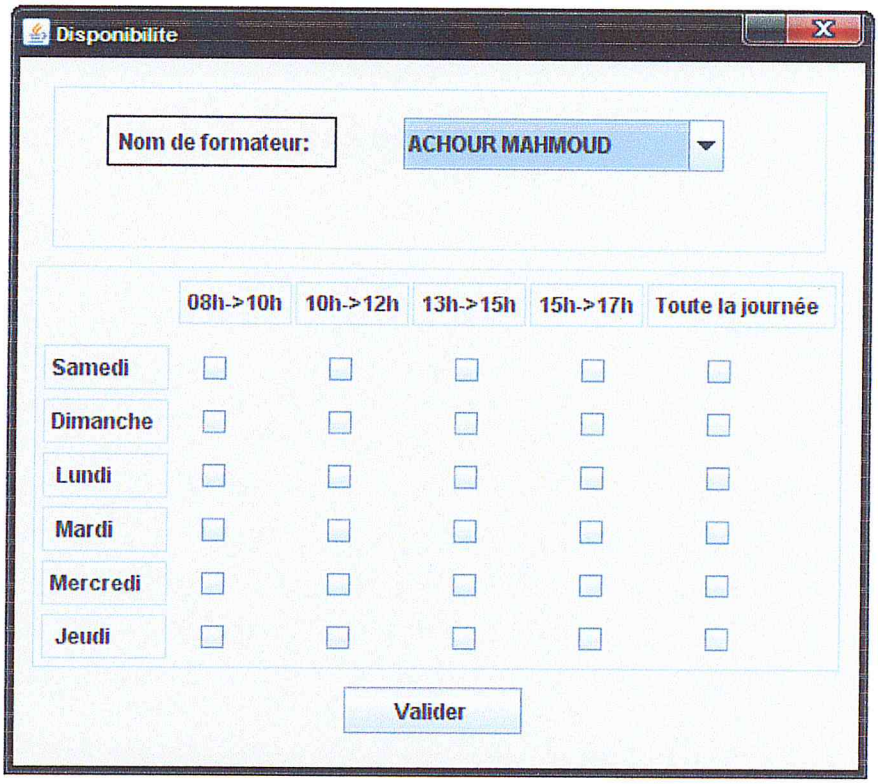


Figure 37: Fenêtre disponibilité

IV.10 Fenêtre de consultation des ETs :

Cette fenêtre (Figure 38) permet a l'utilisateur de faire une recherche et puis affiché emplois du temps correspond a la recherche pour faire une consultation, une modification, ou bien une annulation

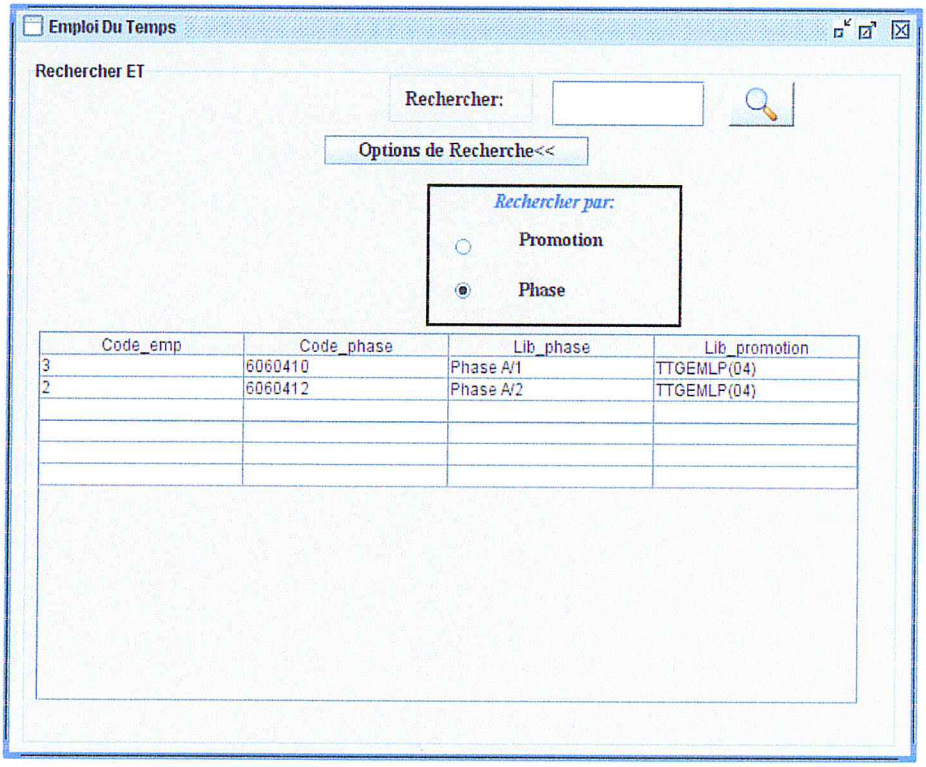


Figure 38: Fenêtre de Recherche d'un ET

CONCLUSION :

Dans le présent chapitre, nous avons proposé notre application. Dans la partie implémentation nous avons détaillé les outils et langages de programmation utilisés pour développer notre solution. Nous avons utilisé principalement le langage Java pour le développement de notre solution. Nous avons également utilisé le SGBD Oracle 9i, pour la gestion de base de données qui constitue notre solution. Et enfin, nous avons montré quelques fenêtres principales de notre application

CONCLUSION & PERSPECTIVES

L'organisation et la gestion des ressources remplissent un rôle très important dans le déroulement des différentes actions de toutes entreprises. Les établissements d'enseignements nécessitent une excellente organisation, afin d'assurer un bon fonctionnement.

Parmi les problèmes pratiques traitées par la recherche opérationnelle et l'intelligence artificielle le problème de génération de l'emploi du temps. Différents modèles et approche ont été proposés pour tenter de le résoudre.

Ce problème est caractérisé par le volume énorme des données et des contraintes, qui sont hétérogènes en nature et en importance, et ils régissent la complexité du problème. Par exemple l'ajout d'une nouvelle contrainte provoque une complexité beaucoup plus élevé, et également, sur le choix des méthodes de résolution. Pour cela, les méthodes exactes sont avérées inutiles, tandis que, en surmontant, le critère d'excellence de qualité, les méthodes approches apportent plus d'efficacité et plus de rapidité. A cette base, notre problème a été résolu.

La satisfaction de toutes les contraintes posées par les responsables des établissements, ainsi que leurs modélisations et la difficulté majeure de ce problème, d'où le modèle de problème de satisfaction de contraintes a prouvé une efficacité remarquable, dans le sens de réduire cet obstacle, en séparant ces contraintes en dures qui représentes les exigences et faibles qui traduisent les souhaits.

Comme la problématique abordée dans notre projet est récente et fait encore l'objet de recherches actives, nous avons été confrontées à quelques difficultés au niveau de l'implémentation de notre prototype, du fait qu'il devait mettre à contribution les technologies les plus récentes.

Ce projet nous a permis de mettre en pratique nos connaissance théorique, de présenter le problème d'emploi du temps, de lui adopter deux modèles Algorithme Génétique et Recuit Simulé.

Bien que notre logiciel soit opérationnel, et à travers lequel **nos objectifs ont été atteint**, il y'a toujours place pour son amélioration. Car les différents travaux effectués dans ce contexte, ouvrent au monde de la gestion des empois du temps de nouvelles perspectives.

Nous envisagerons d'étendre cette étude en améliorant l'emploi du temps obtenu, en incluant d'autre exigence et souhaits. A fin de pouvoir d'appliquer a l'université, pour cela nous espérons tester l'hybridation avec d'autre méta-heuristiques.

Et nous espérons avoir fournis un document de base aux prochains étudiants ayant un projet similaire et nous aimerons bien avoir une continuité à ce travail.

« Une réalité complexe reste toujours inachevée et incomplète. On ne peut jamais la comprendre ou la maîtriser entièrement ou définitivement. Dès qu'on pense avoir trouvé des schémas explicatifs globaux, ils se modifient, laissant place à d'autres modèles tout aussi éphémères. »

- Dominique Génelot, extrait de "Manager la complexité" -

Références bibliographiques :

Via livres et thèses :

[01]: Laurent AUDIBERT "UML 2" : Institut Universitaire de Technologie de Villetaneuse – Département Informatique Édition 2007-2008

[02] : BENATCHBA Karima« Modèle d'exécution pour l'aide a la résolution du problème MAX-SAT» .Institut National d'Informatique, Alger.

[03]: Melle : Terki Amel « Analyse des performances des algorithmes génétiques Utilisant différentes techniques d'évolution de la population », Thèse Magister En Electronique Option : contrôle, UNIVERSITE MENTOURI CONSTANTINE.

[04] : Belaïd SAAD « Intégration des problèmes de satisfaction de contraintes distribués et sécurisés dans les systèmes d'aide à la décision à base de connaissances ». Thèse de doctorat. UNIVERSITÉ DE PAUL VERLAINE - METZ

[05][10]: D.Beasley ET R. Martin, "An Overview of Genetic Algorithms": Part 2, department of Computing Mathematics, University ofWales College of Cardiff, CF2 4YN, 1993.

[06][11]: J.PhilippeRennard, "Genetic Algorithm Viewer: Démonstration d'un algorithme génétique", www.rennad.org/alif, Avril 2000.

[07][12]: N.Benahmed,"Optimisation de réseaux de neurones pour la reconnaissance de chiffres manuscrits isolés: sélection et pondération des primitives par algorithme génétique". Université du Québec ,2002.

[08][FRA04] : Francillette Remy, « Optimisation combinatoire», Université des Antilles et Guyane, Faculté des sciences exactes et naturelles, Rapport de T.E.R., 2004.

I. Notion de complexité :

D'une manière générale, pour résoudre un problème, on est appelé à trouver l'algorithme le plus efficace. Cette notion d'efficacité induit normalement toutes les ressources de calcul nécessaires pour exécuter un algorithme. Or, le temps d'exécution est généralement le facteur dominant pour déterminé si un algorithme est assez efficace pour être utilisé dans la pratique, pour cela on se concentre principalement sur cette ressource.

On appelle complexité en temps d'un algorithme le nombre d'instructions élémentaires mises en œuvre dans cet algorithme afin de résoudre un problème donné. Une instruction élémentaire sera une affectation, une comparaison, une opération algébrique, la lecture et l'écriture etc.... Mais comme le décompte précis de toutes les instructions d'un programme risque d'être assez pénible, et qu'entre deux exécutions du même algorithme avec un jeu de paramètres différent, le nombre d'instructions exécutées peut changer, on se contentera, en général, d'apprécier un ordre de grandeur de ce nombre d'instructions. C'est ce qu'on désigne sous le nom de complexité de l'algorithme. Donc pour mesurer la complexité temporelle d'un algorithme, on s'intéresse plutôt aux opérations les plus coûteuses :

- Racine carrée, Log, Exp, Addition réelle ...
- Comparaisons dans le cas des tris ...

II. Les différentes classes de complexité :

1. La classe P :

La classe **P** contient tous les problèmes relativement faciles c'est à dire ceux pour lesquels on connaît des algorithmes efficaces. Plus formellement, ce sont les problèmes pour lesquels on peut construire une machine déterministe (exp une machine de Turing) dont le temps d'exécution est de complexité polynomiale (le sigle **P** signifie « **P**olynomial time »).

2. La classe NP :

Les problèmes de la classe **NP** sont ceux pour lesquels on peut construire une machine de Turing non déterministe dont le temps d'exécution est de complexité polynomiale (le sigle **NP** provient de « **N**ondeterministic **P**olynomial time ») (et non de « Non Polynomial).

Contrairement aux machines déterministes qui exécutent une séquence d'instructions bien déterminée, les machines non déterministes ont la remarquable capacité de toujours choisir la meilleure séquence d'instructions qui mène à la bonne réponse lorsque celle-ci existe. Ce concept abstrait est en fait la base de toute la théorie de la **NP**-complétude.

3. La classe NP-complets :

Parmi l'ensemble des problèmes appartenant à **NP**, il en existe un sous-ensemble qui contient les problèmes les plus difficiles : on les appelle le problème **N P**-complets. Un problème **NP**-complets possède la priorité que tout problème dans **NP** peut être transformé (réduit) en celui-ci en temps polynomial. C'est à dire qu'un problème est **NP**-complets quand tous les problèmes appartenant à **NP** lui sont réductibles. Si on trouve un algorithme polynomial pour un problème **NP**-complets, on trouve alors automatiquement une résolution polynomiale de tous les problèmes de la classe **NP.**

4. La classe NP-difficiles :

Un problème est **NP**-difficile s'il est plus difficile qu'un problème **NP**-complet, c'est à dire s'il existe un problème **NP**-complet se réduisant à ce problème par la réduction de Turing. Ceci explique pourquoi, lors de l'étude d'un nouveau problème, on commence par chercher à classer ce problème. Si l'on parvient à montrer qu'il est polynomial, le problème sera résolu. Si par contre, on parvient à montrer qu'il est **NP**-complet, la recherche d'un algorithme exact pour résoudre un tel problème ne sera pas de première priorité, et il sera approprié de se concentrer sur des méthodes heuristiques que la plupart des spécialistes de l'optimisation combinatoire ont orienté leurs recherches pour les développer. Une méthode heuristique est souvent définie comme une procédure exploitant au mieux la structure du problème considéré, dans le but de trouver une solution de qualité raisonnable en un temps de calcul aussi faible que possible.

III. Heuristique :

Une heuristique est une technique qui améliore l'efficacité d'un processus de recherche, en sacrifiant éventuellement l'exactitude ou l'optimalité de la solution.

Pour des problèmes d'optimisation (NP-complets) où la recherche d'une solution exacte (optimale) est difficile (coût exponentiel), on peut se contenter d'une solution satisfaisante donnée par une heuristique avec un coût plus faible.

Certaines heuristiques sont polyvalentes (elles donnent d'assez bons résultats pour une large gamme de problèmes) alors que d'autres sont spécifiques à chaque type de problème.

Dans le backtracking, dans les jeux de stratégie (jeu d'échec), on a déjà utilisé les heuristiques (les fonctions d'estimations dans Mimmax).

Les heuristiques peuvent donner des solutions optimales, ce qui semble paradoxa

IV. Métaheurisique :

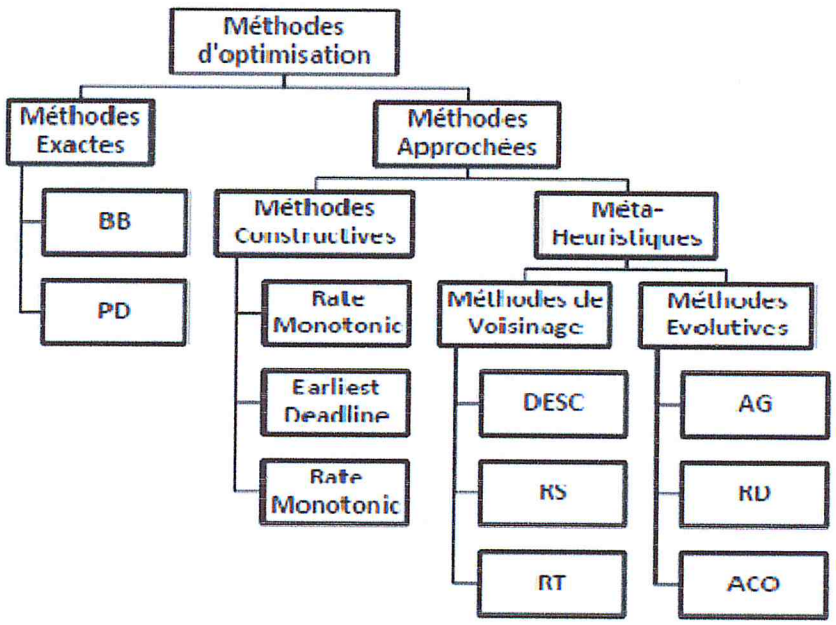
La **méta-heuristique**, elle, se place à un niveau plus général encore, et intervient dans toutes les situations où l'ingénieur ne connaît pas d'heuristique efficace pour résoudre un problème donné, ou lorsqu'il estime qu'il ne dispose pas du temps nécessaire pour en déterminer une.

En 1996, I.H. Osman et G. Laporte définissaient la méta-heuristique comme « un processus itératif qui subordonne et qui guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales ».

En 2006, le réseau Métaheuristics (metaheuristics.org) définit les méta-heuristiques comme « un ensemble de concepts utilisés pour définir des méthodes heuristiques, pouvant être appliqués à une grande variété de problèmes. On peut voir la méta-heuristique comme une « boîte à outils » algorithmique, utilisable pour résoudre différents problèmes d'optimisation, et ne nécessitant que peu de modifications pour qu'elle puisse s'adapter à un problème particulier ».

Elle a donc pour objectif de pouvoir être programmée et testée rapidement sur un problème.

Comme l'heuristique, la méta-heuristique n'offre généralement pas de garantie d'optimalité, bien qu'on ait pu démontrer la convergence de certaines d'entre elles. Non déterministe, elle incorpore souvent un principe stochastique pour surmonter l'explosion combinatoire. Elle fait parfois usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite du processus de recherche.



Classification des méthodes d'optimisation

Légendes

BB : Branch and Bound. **RS** : Recuit Simulé.

PD : Programmation Dynamique. **RT** : Recherche Tabou.

DESC : Méthode de Descente. **ACO** : Optimisation par Colonie de Fourmis.

RD : Recherche Dispersée.

IV.1 Méthodes exactes :

Les méthodes exactes sont des méthodes qui garantissent la complétude de la résolution autrement dit ces méthodes donnent à tous les coups la solution optimale. Le temps de calcul nécessaire de telles méthodes augmente en général exponentiellement avec la taille du problème à résoudre. On distingue dans ce cas l'approche constructive qui est probablement la plus ancienne et occupe traditionnellement une place très importante en optimisation combinatoire. Une méthode constructive construit pas à pas une solution de la forme s = ($<V_1,v_1>$ $<V_2,v_2>$…$<V_n,v_n>$) en partant d'une solution partielle initialement vide s = 0, elle cherche à étendre à chaque étape la solution partielle s = ($<V_1,v_1>$ … $<V_{i-1},v_{i-1}>$) (i <= n) de l'étape précédente. Pour cela, elle détermine la prochaine variable V_i, choisit une valeur v_i dans Di et ajoute $<V_i,v_i>$ dans s pour obtenir une nouvelle solution partielle s = ($<V_1,v_1>$…$<V_n,v_n>$ $<V_i,v_i>$). Ce processus se répète jusqu'à ce que l'on obtienne une solution complète.

IV.2 Les méthodes approchées :

Contrairement aux méthodes exactes, les méthodes approchées ne procurent pas forcément une solution optimale, mais seulement une bonne solution (de qualité raisonnable) en un temps de calcul aussi faible que possible.

Une partie importante des méthodes approchées est désignée sous le terme de Méta-heuristiques. Plusieurs définitions d'une méta-heuristique ont été proposées une d'eux :

- *« A metaheuristics is a set of concepts that can be used to define heuristic méthods that can be applied to a wide set of different problems ».*

Plusieurs classifications des méta-heuristiques ont été proposées ; la plupart distinguent globalement deux catégories : les méthodes à base de solution courante unique, qui travaille sur un seul point de l'espace de recherche à un instant donné, appelées méthodes à base de voisinage comme les méthodes de recherche locale(méthode de la descente), de recuit simulé et de recherche tabou, et les méthodes à base de population, qui travaillent sur un ensemble de points de l'espace de recherche, comme les algorithmes évolutionnaires et les algorithmes de colonies de fourmis.

I. L'approche orientée objet

L'approche orientée objet considère le logiciel comme une collection d'objets dissociés, identifié set possédant des caractéristiques. Une caractéristique est soit un attribut (i.e. une donnée caractérisant l'état de l'objet), soit une entité comportementale de l'objet (i.e. une fonction). La fonctionnalité du logiciel émerge alors de l'interaction entre les différents objets qui le constituent.

L'une des particularités de cette approche est qu'elle rapproche les données et leurs traitements associés au sein d'un unique objet.

Comme nous venons de le dire, un objet est caractérisé par plusieurs notions :

L'identité – L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. On construit généralement cette identité grâce à un identifiant découlant naturellement du problème (par exemple un produit pourra être repéré par un code, une voiture par un numéro de série, etc.)

Les attributs – Il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations sur l'état de l'objet.

Les méthodes – Les méthodes d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions (appelées opérations) que l'objet est à même de réaliser. Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets).

De plus, les opérations sont étroitement liées aux attributs, car leurs actions peuvent dépendre des valeurs des attributs, ou bien les modifier.

La difficulté de cette modélisation consiste à créer une représentation abstraite, sous forme d'objets, d'entités ayant une existence matérielle (chien, voiture, ampoule, personne, . . .) ou bien virtuelle (client, temps, . . .).

La Conception Orientée Objet (COO) est la méthode qui conduit à des architectures logicielles fondées sur les objets du système, plutôt que sur la fonction qu'il est censé réaliser.

II. Les concepts de base pour une modélisation objet :

Dans l'approche objet, le modèle est calqué sur la réalité physique du monde réel. Les objets se comportent comme des entités indépendamment, autosuffisantes qui collaborent par échange de messages, On réalise ainsi l'encapsulation {l'intérieur d'un objet des attributs décrivant l'état de cet objet mais également des méthodes de traitement des messages. Dans un but de généralisation, on introduit la notion de classe d'objet qui permet de partager les

propriétés communes à plusieurs classes (héritage), ainsi que de raffiner les méthodes de traitement pour les rendre plus spécifiques quand elles existent pour plusieurs classes (polymorphisme).

II.1. Le concept d'objet : On appelle objet un élément informatique regroupant les principales caractéristiques des éléments du monde réel. Il satisfait les principes suivants :

- Le principe de distinction : un objet a une identité.
- Le principe de permanence : l'évolution d'un objet ne remet pas en cause son identité et à un moment donné il est dans un seul état.
- Le principe d'activité : un objet possède un comportement.

II.2. Le concept de classe : Une classe est un groupe d'objets ayant les mêmes attributs et les mêmes opérations et une sémantique. Ainsi, une classe peut être considérée comme un moule, à partir du quel sont générés les objets.

Les messages :

Les objets sont considérés comme des entités totalement indépendantes, qui ne peuvent communiquer entre elles qu'au moyen de messages. La transmission d'un message d'un objet X vers un objet Y, se traduit par un appel, {partir de X, d'une méthode de Y. Bien entendu, la méthode utilisée appartient {l'interface de Y (partie
visible de l'objet Y). Ainsi, l'envoi de messages représente le seul moyen de communication entre objets.

L'héritage :

Il peut s'avérer nécessaire de créer une hiérarchie entre différentes classes. Cette classification consiste {factoriser les éléments communs d'un ensemble de classes dans une classe plus générale, que l'on nomme superclasse. L'héritage est un mécanisme de transmission des propriétés (attributs et méthodes) d'une superclasse vers une sous-classe (classe dérivée), il évite donc la duplication et encourage la réutilisation. La hiérarchie des classes ou classifications permet de gérer la complexité en ordonnant les objets au sein d'une arborescence de classes d'abstraction croissante.

III. Historique la programmation par objets :

Les premiers langages de programmation qui ont utilisé des objets sont Simula I (1961-64) et Simula 67 (1967), conçus par les informaticiens norvégiens Ole-Johan Dahl et Kristan Nygaard.

Simula 67 contenait déjà les objets, les classes, l'héritage, l'encapsulation, etc.

Alan Kay, du PARC de Xerox, avait utilisé Simula dans les années 1960. Il réalisa en 1976

Smalltalk qui reste, aux yeux de certains programmeurs, le meilleur langage de programmation par objets.

Bjarne Stroustrup a mis au point C++, une extension du langage C permettant la programmation orientée objets, aux Bell Labs d'AT&T en 1982. C++ deviendra le langage le plus utilisé par les programmeurs professionnels. Il arrivera à maturation en 1986, sa standardisation ANSI

/ ISO date de 1997.

Java est lancé par Sun en 1995. Comme il présente plus de sécurité que C++, il deviendra le langage favori de certains programmeurs professionnels.

IV. Présentation d'UML:

La description de la programmation par objets a fait ressortir l'étendue du travail conceptuel nécessaire : définition des classes, de leurs relations, des attributs et méthodes, des interfaces etc.

Pour programmer une application, il ne convient pas de se lancer tête baissée dans l'écriture du code : il faut d'abord organiser ses idées, les documenter, puis organiser la réalisation en définissant les modules et étapes de la réalisation. C'est cette démarche antérieure à l'écriture que l'on appelle modélisation ; son produit est un modèle.

Les spécifications fournies par la maîtrise d'ouvrage en programmation impérative étaient souvent floues : les articulations conceptuelles (structures de données, algorithmes de traitement) s'exprimant dans le vocabulaire de l'informatique, le modèle devait souvent être élaboré par celle-ci. L'approche objet permet en principe à la maîtrise d'ouvrage de s'exprimer de façon précise selon un vocabulaire qui, tout en transcrivant les besoins du métier, pourra être immédiatement compris par les informaticiens. En principe seulement, car la modélisation demande aux maîtrises d'ouvrage une compétence et un professionnalisme qui ne sont pas aujourd'hui répandus

IV.1. Histoire des modélisations par objets

Les méthodes utilisées dans les années 1980 pour organiser la programmation impérative (notamment Merise) étaient fondées sur la modélisation séparée des données et des traitements.

Lorsque la programmation par objets prend de l'importance au début des années 1990, la nécessité d'une méthode qui lui soit adaptée devient évidente. Plus de cinquante méthodes apparaissent entre 1990 et 1995 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD,

OOM, OOSE, etc.)Mais aucune ne parvient à s'imposer. En 1994, le consensus se fait autour de trois méthodes :

– OMT de James Rumbaugh (General Electric) fournit une représentation graphique des aspects statique, dynamique et fonctionnel d'un système ;

– OOD de Grady Booch, définie pour le Department of Defense, introduit le concept de paquetage (package) ;

– OOSE d'Ivar Jacobson (Ericsson) fonde l'analyse sur la description des besoins des utilisateurs (cas d'utilisation, ou use cases).

Chaque méthode avait ses avantages et ses partisans. Le nombre de méthodes en compétition s'était réduit, mais le risque d'un éclatement subsistait : la profession pouvait se diviser entre ces trois méthodes, créant autant de continents intellectuels qui auraient du mal à communiquer.

Événement considérable et presque miraculeux, les trois gourous qui régnaient chacun sur l'une des trois méthodes se mirent d'accord pour définir une méthode commune qui fédérerait leurs apports respectifs (on les surnomme depuis « the Amigos »). UML (Unified Modeling Language) est né de cet effort de convergence. L'adjectif unified est là pour marquer qu'UML unifie, et donc remplace.

En fait, et comme son nom l'indique, UML n'a pas l'ambition d'être exactement une méthode : c'est un langage.

L'unification a progressé par étapes. En 1995, Booch et Rumbaugh (et quelques autres) se sont mis d'accord pour construire une méthode unifiée, Unified Method 0.8 ; en 1996, Jacobson les a rejoints pour produire UML 0.9 (notez le remplacement du mot méthode par le mot langage, plus modeste). Les acteurs les plus importants dans le monde du logiciel s'associent alors à l'e_ort (IBM, Microsoft, Oracle, DEC, HP, Rational, Unisys etc.) et UML 1.0 est soumis à l'OMG5. L'OMG adopte en novembre 1997 UML 1.1 comme langage de modélisation des systèmes d'information à objets. La version d'UML en cours en 2008 est UML 2.1.1 et les travaux d'amélioration se poursuivent.

5 L'OMG (Object Management Group) est une association américaine à but non-lucratif créée en 1989 dont l'objectif est de standardiser et promouvoir le modèle objet sous toutes ses formes. L'OMG est notamment à la base des spécifications UML, MOF, CORBA et IDL. L'OMG est aussi à l'origine de la recommandation MDA

UML est donc non seulement un outil intéressant mais une norme qui s'impose en technologie à objets et à laquelle se sont rangé

IV.2 Les Principaux diagrammes utilisés dans le développent des systèmes d'information de gestion:

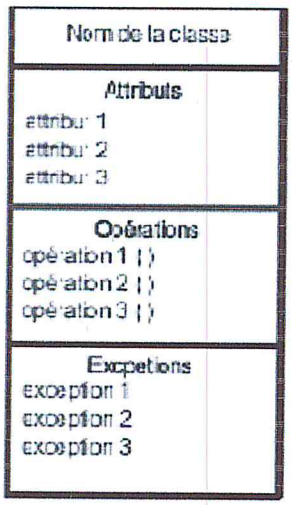
Un diagramme donne à l'utilisateur un moyen de visualiser et de manipuler des éléments de modélisation. Les différents types de diagrammes d'**UML** sont présentés dans l'extrait du Méta modèle suivant :

Diagrammes structurels ou diagrammes statiques (UML Structure)	Diagrammes comportementaux ou diagrammes dynamiques (UML Behavior)	Diagrammes d'interaction (Interaction diagram)
-diagramme de classes (Class diagram) -diagramme d'objets (Object diagram) -diagramme de composants (Component diagram) -diagramme de déploiement (Deployment diagram) -diagramme de paquetages (Package diagram) -diagramme de structures composites (Composite structure diagram)	-diagramme de cas d'utilisation (Use case diagram) -diagramme d'activités (Activity diagram) diagramme d'états transitions (State machine diagram)	-diagramme de séquence (Sequence diagram) -diagramme de communication (Communication diagram) -diagramme global d'interaction (Interaction overview diagram) -diagramme de temps (Timing diagram)

a) Diagramme des classes:

Le diagramme des classes est une représentation statique du système d'information. Il permet aux concepteurs de visualiser les relations interclasses dans un système. Une classe décrit un ensemble d'éléments. Chaque relation entre classes est définit par des associations. **Représentation d'une classe UML.** Les classes en **UML** sont représentées par un rectangle avec des séparations :

Exemple :



Exemple d'une classe UML

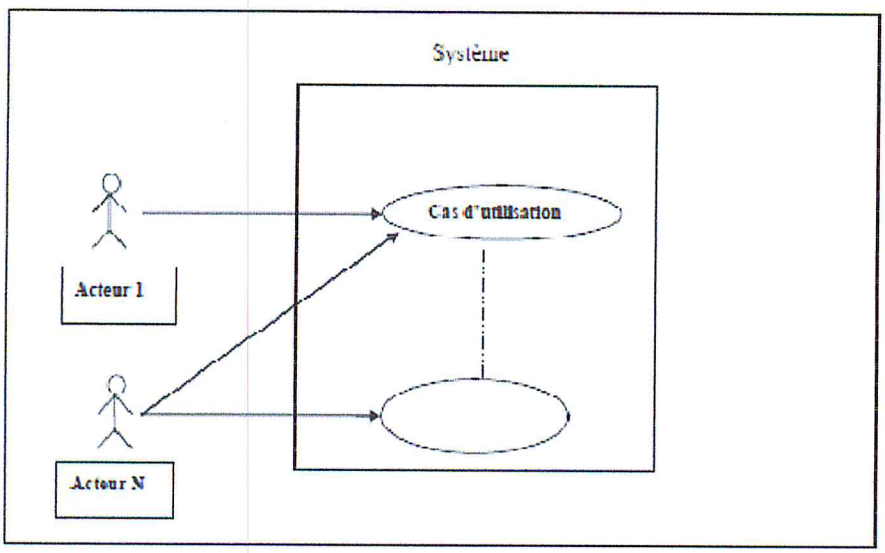
Les associations Un diagramme des classes est composé de plusieurs classes qui sont reliées entre elles. Des liens sont alors tracés afin de créer ces relations. On les appelle des associations. Représentation d'une association entre 2 classes

La multiplicité des relations : La multiplicité se précise sur les extrémités des associations :

La multiplicité associer {une terminaison d'une association indique le nombre d'objet apparaissant dans l'autre extrémité

a) Diagramme Cas d'utilisation :

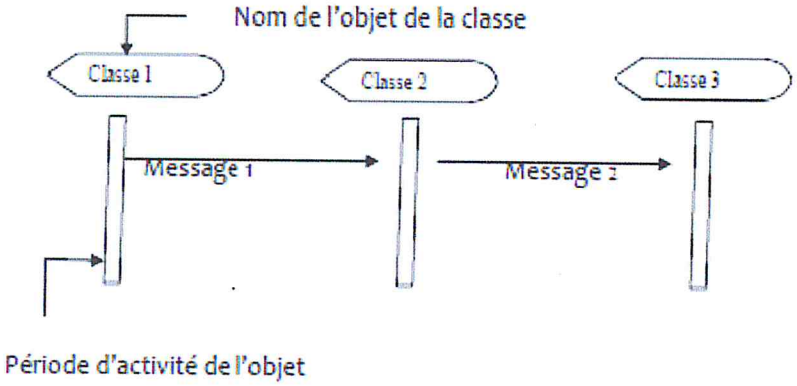
L'étude de cas d'utilisation a pour objectif de déterminer ce que chaque acteur attend du système. La détermination des besoins est basée sur la représentation de l'interaction entre l'acteur et le système. A ce prospect de la modélisation, les interactions représentent les principaux événements qui se produisent dans le domaine de l'application. Plus tard, ces événements sont traduits en messages qui déclenchent des opérations.



b) Diagramme séquence :

IL permet d'étudier les interactions entre les objets et constitue une partie dynamique de système d'information.

IL montre d'une façon séquentielle les envois des messages qui interviennent entre les objets, il peut également montrer les flux des données :



c) Diagramme d'état transition :

Le diagramme d'état transition décrit la vie d'un objet en termes d'événement déclenchant des modifications de l'état de celui-ci, il identifie à la fois les événements externe et interne. Le passage d'un état {un autre s'appelle transition

d) Les diagrammes de déploiement :

Qui représentent le déploiement des composants sur les dispositifs matériels,

e) Les diagrammes d'activités :

Qui représentent le comportement d'une opération en termes d'action.

f) Les diagrammes de communication :

Qui donne une représentation spatiale des objets, des liens des interactions, il est équivalent au diagramme au diagramme de séquence

g) Les digrammes d'objets :

Qui représentent les objets et leur relation et correspondent à des diagrammes de communication simplifiés, sans représentation des envois de message.

h) Les diagrammes de composants :

Qui représentent les physiques d'une application.

[09]: ATISH Chand « A heuristic approach to constraint optimization in timetabling» University of the south of Pacific, Suva, FIJI.2002

[10]: CHERGHI Mohamed El Amine « modèle structure d'un algorithme d'affectation des enseignements aux locaux». Thèse de magister. Université de Houari BOUMEDIENE. 1991.

[11]: E.Aycan and T.Ayav « Solving the course scheduling problem using Simulated Annealing». Departement of computer engeneering, Izmir Institute of Technology.

[12]:F. Glover & M. Laguna « Tabu search». Livre, Kluwer Academic Publishers, Boston.1997

[13]: LABOUDI Zakaria « Evolution d'automates cellulaires par algorithmes génétiques quantiques sur un environnement parallèle ». Thèse de magister. Université Mentouri de Constantine.2008/2009

[14] : TROUDI Fatiha« Résolution du problème de l'emploi du temps : Proposition d'un algorithme évolutionnaire multi objectif ». Thèse de Magister. Université Mentouri – Constantine.2006

[15] : BOULMERKA Aissa «Adaptation des métaheuristiques à l'ordonnancement hors-ligne des tâches temps réel à contraintes strictes en environnement monoprocesseur» ECOLE NATIONALE SUPERIEURE D'INFORMATIQUE.2009

[16] : BEN-DELHOUM.S, HADJI.B« Gestion des emplois du temps par les Systèmes Multi-Agents». INSTITUT NATIONAL D'INFORMATIQUE.2008

[17] : AZZOUNE.M, BENAICHOUBA.Z « La génération d'un outil de confection automatique des emplois du temps». Thèse d'ingéniorat. Université SAAD DAHLAB-BLIDA.2006

Via internet :

[18] : http://www.a525g.com/intelligence-artificielle/algorithme-genetique.htm

[19] : http://www710.univ-lyon1.fr/~csolnon/Site-PPC/session1/e-miage-ppc-sess1.htm