REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE IBN KHALDOUN - TIARET

MEMOIRE

Présenté à :

FACULTÉ MATHEMATIQUES ET INFORMATIQUE
DÉPARTEMENT D'INFORMATIQUE

Pour l'obtention du diplôme de :

MASTER

Spécialité : Réseaux et télécommunications

Par : **Mekid Asmaa Hayat**

Sur le thème

Application des algorithmes génétiques pour la résolution de problèmes d'ordonnancement dans les environnements de simulation.

Soutenu publiquement le 27 / 09 / 2021 à Tiaret devant le jury composé de :

NASSANE Samir	Grade	Université MCA	Président
MEBAREK Bendaoud	Grade	Université Prof	Encadreur
BENGHENI Abdelmalek	Grade	Université MCB	Examinateur

2020-2021

Remerciements

*A*l'issue de ce travail, nous tenons à remercier tous ceux qui nous ont aidé à réaliser ce travail dans des conditions acceptables au vue de la situation sanitaire qui prévaut actuellement.

En particulier nous tenons à remercier vivement :

*L*e Docteur **Mebarek Bendaoud,** notre encadreur, pour son aide, ses encouragements et la sympathie qu'il a eu à notre égard pendant tout ce travail. Grace à ses précieux conseils, nous avons pu achever nos travaux dans, relativement, de bonnes conditions.

*T*ous nos enseignants pour leur dévouement et les efforts qu'ils ont fourni pendant tout notre cursus universitaire.

Tout le personnel du département Informatique pour leur aide.

*N*os amis de la promotion avec qui nous avons partagé des souvenirs gais et tristes durant ces cinq années passées ensemble à l'Université de Tiaret.

*E*nfin, un grand merci et toutes nos reconnaissances à nos chers parents, nos frères et sœurs et tous les membres de nos honorables familles qui nous ont énormément encouragé et soutenu tout au long de notre scolarité et pendant cette effroyable pandémie.

Résumé :

L'ordonnancement du processeur implique que les travaux ou les tâches doivent être assignés à un processeur particulier pour être exécutés à un moment particulier. Le problème de la recherche d'un ordonnancement optimal pour un ensemble de tâches est NP-complet. L'algorithme qui implémente l'ordonnancement nécessite un temps exponentiel et ou polynomial pour atteindre une solution optimisée. Diverses techniques d'optimisation peuvent être utilisées pour trouver des solutions optimales pour l'ordonnancement des processus. L'algorithme génétique est l'une des solutions optimisées pour l'ordonnancement des processus.

Dans ce travail, j'évaluerais la performance et l'efficacité des algorithmes les plus populaires dans ce domaine notamment : FCFS, SJF, LBE, SLB et MAX-MIN avec l'algorithme génétique.

Mot clés : ordonnancement, NP-complet, algorithme génétique, technique d'optimisation

الملخص :

تشير جدولة المعالج إلى أنه يجب تعيين الوظائف أو المهام إلى معالج معين ليتم تنفيذها في وقت معين. مشكلة العثور على جدول أمثل لمجموعة من المهام هي NP كاملة. تتطلب الخوارزمية التي تنفذ الجدولة وقتًا أسيًا و / أو متعدد الحدود للوصول إلى حل أمثل. يمكن استخدام تقنيات التحسين المختلفة لإيجاد الحلول المثلى لجدولة العمليات. تعد الخوارزمية الجينية أحد الحلول المثلى لجدولة العمليات.

في هذا العمل ، سأقيم أداء وكفاءة الخوارزميات الأكثر شيوعًا في هذا المجال بما في ذلك: FCFS و SJF و LBE و SLB و MAX-MIN بالمقارنة مع الخوارزمية الجينية.

الكلمات الرئيسية: الجدولة ، NP كاملة ، الخوارزمية الجينية ، تقنية التحسين

List des abbreviations

GANTT	Generalized Activity Normalization Time Table.
IaaS	Infrastructure as a Service.
PaaS	Platform as a Service.
SaaS	Software as a Service.
Mekespan	Le temps d'exécution.
LBE:	Loading Balancing Exchange.
SLB:	Simple Loading Balancing.
SJF	Shortest job first.
AG	Algorithme génétique.
FIFO	First In First Out.
UML	Unified Modeling Language.
Lo	Low.
Hi	High.
THMH	Tache High Machine High.
THML	Tache High Machine Low.
TIMI	Tache Low Machine Low.
TC	Complétion Time.
UC	Unité Centrale.

Table de matière:

Chapitre –III- Les algorithmes génétiques

Chapitre -IV-Implémentation, résultat et discussion

Liste Des Tableaux :

Liste Des Figures :

Introduction générale

Depuis quelques années le Cloud computing a pris une place importante dans le domaine de calcul scientifique surtout avec l'apparition des services de cloud comme 'Google Cloud,'AWS',etc… qui fournit à ces utilisateurs des ressources importantes, mais malgré l'avancement qu'on a reconnu dans ces dernière années au niveau de puissance ordinateurs , le problème de ressource reste primordiale à résoudre d'où interviennent les algorithmes d'ordonnancement.

L'ordonnancement des taches dans un environnement Cloud permet d'optimiser à la fois l'allocation des ressources et la gestion des taches ce qui conduit à réduire le cout et augmenter la qualité de service mais ce n'est pas assez simple que ça dans les Cloud ou il y a plusieurs enjeux et facteurs prises en compte, notamment le temps d'exécution de taches et leur cout correspondants.

Dans ce travail je vais présenter le problème d'ordonnancement et les différentes approches utilisées pour le résoudre, ensuite les comparer et arriver à une conclusion sur la base de ces expérimentations équitables pour tous les algorithmes implémenté de ce mémoire notamment l'algorithme génétique.

Dans ce mémoire je vais présenter quatre chapitres :

Dans le premier chapitre, je vais présenter d'une façon générale l'ordonnancement et le calcul dans un environnement distribué.

Dans le deuxième chapitre, je présente les techniques et les différentes approches heuristiques et méta-heuristique utilisées pour établir l'ordonnancement.

Le troisième chapitre est dédié entièrement aux algorithmes génétiques pour l'ordonnancement

Enfin, le quatrième chapitre est consacré à l'implémentation des algorithmes et aux expérimentations

Chapitre -I -

Ordonnancement des tâches dans un environnement de calcul distribué

1 -Introduction :

Les systèmes distribués à grande échelle comme les Grilles ou les Nuages (Cloud) sont fondamentalement dynamiques et instables, et il est également réaliste de considérer que certaines ressources vont subir des défaillances pendant leur utilisation [01]. La panne d'une ressource peut affecter l'entière exécution des applications qui nécessitent la disponibilité de plusieurs ressources en même temps. Afin de pouvoir gérer des plates-formes dynamiques à grande échelle, il faut se tourner vers des algorithmes d'ordonnancement et d'équilibrage de charge décentralisés, de telle sorte que le système puisse passer à l'échelle, sans que les performances de la plate-forme soient limitées par celle du nœud en charge de l'ordonnancement.

2- L'ordonnancement :

On peut considérer l'ordonnancement de taches comme la gestion et la prise en charge d'un ensemble de taches de leur émission jusqu'à l'exécution. L'ordonnancement est un mécanisme de négociation entre deux objets, l'un représentant l'utilisateur (ou l'application) et l'autre les ressources. Les objets coté utilisateur sont les ordonnanceurs. [02] Une autre définition, un ordonnancement constitue une solution au problème d'ordonnancement. Il est défini par le planning d'exécution des taches (« ordre » et « calendrier «) et d'allocation des ressources et vise à satisfaire un ou plusieurs objectifs. Un ordonnancement est très souvent représenté par un diagramme de GANTT.

2.1 Problème d'ordonnancement :

Le but de l'ordonnancement des tâches est de trouver un plan d'exécution optimal des tâches qui prend en considération leurs contraintes : les ressources, le budget, ladate de fin, la performance, etc. En général, un problème contraint se compose de : tâches, ressources, conditions contraintes et une ou plusieurs fonctions objectives. Les problèmes d'ordonnancement peuvent être classes en deux grandes catégories :

1- Les problèmes d'ordonnancement en ligne (online) :

Pour lesquels la date d'arrivée (release date) des jobs n'est pas connue à l'avance.

2- Les problèmes d'ordonnancement hors ligne (offline) :

Pour lesquels les dates d'arrivées des jobs (généralement ils sont tous prêts à t=0 et toutes leurs caractéristiques sont connues avant l'ordonnancement). Ces problèmes ont été très largement étudiés pour les jobs séquentiels et pour les jobs parallèles.

Le processus d'ordonnancement se composent de tout ou partie des étapes suivantes: priorisation des taches, allocation des ressources , allocation et enfin ordonnancement

La phase priorisation des taches: établit l'ordre des tâches de départ leurs propriétés et leurs contraintes. Après cette phase, on a une liste ordonnée.

- **La phase allocation des ressources:** réserve ou alloue un ensemblede ressources, c'est-à-dire qu'elle calcule le nombre de machines virtuelles pour l'ordonnancement des tâches.
- **La phase ordonnancement :** sélectionne les ressources parmi celles précédemment alloué qui permettent d'exécuter les tâches selon l'ordre prédéfinis. Quelle fait l'ordonnancement de chaque tâche à des ressources qui lui sont optimales.

2.2 - Les types d'ordonnancement :

Différents types de planification sont basés sur des critères, tels que l'environnement statique ou dynamique, centralisé vs distribué, etc. bien qu'ils puissent se chevaucher et ne pas être clairement distincts :

Planification statique : Pré-planification des tâches, toutes les informations sur les ressources et les tâches disponibles dans l'application doivent être connu et de plus une tâche est assignée une fois à une ressource, afin de faciliter l'adaptation en fonction du planificateur.

Planification dynamique : Plus flexible que statique où les travaux sont disponibles dynamiquement pour planification au fil du temps par le planificateur sans problème, pour être en mesure de déterminer le temps d'exécution à l'avance. C'est hautement essentiel d'inclure l'équilibre de la charge comme facteur principal pour obtenir un algorithme de planification stable et efficace.

Planification centralisée : Comme mentionné dans la planification dynamique, un ordonnanceur centralisé / distribué est responsable de la prise de décision globale. En utilisant planification centralisée ; Facilité de mise en œuvre, efficacité et plus de contrôle et de surveillance des ressources sont des avantages acquis. D'autre part ; tel planificateur manque d'évolutivité, de tolérance aux pannes et d'efficacité performances, il n'est donc pas recommandé pour les applications à grande échelle.

Planification décentralisée/distribuée : Ce type de la planification est plus réaliste pour les réseaux réels malgré son faible efficacité par rapport à la planification centralisée, dans laquelle les planificateurs locaux demandent à gérer et maintenir l'état de la file d'attente des travailleurs car il n'y a plus d'entité centrale de contrôle.

Planification coopérative: Dans ce cas, le système a déjà beaucoup d'ordonnanceurs, chacun est responsable d'effectuer certaines activités dans le processus de planification vers système commun large échelle basée sur la coopération de procédures, règles, données et utilisateurs actuels du système .

Planification préventive : Ce genre de planification permet à chaque tâche à interrompre pendant l'exécution et une tâche peut être migrée vers une autre ressource en laissant son originale ressource allouée inutilisée pour être disponible pour d'autres tâche. Elle est plus utile s'il y a des contraintes comme priorité d'être considéré [3].

Planification non préventive : Dans laquelle les ressources ne sont pas autorisées à être réaffecté jusqu'à ce que le travail planifié termine son exécution [3].

Mode immédiat / en ligne : dans lequel le planificateur planifie toute tâche récemment arrivé dès qu'elle arrive sans en attente du prochain intervalle de temps sur les ressources disponibles à ce moment [3].

Mode batch / hors ligne : le planificateur conserve les travaux entrants comme groupe de problèmes à résoudre dans le temps intervalles, de sorte qu'il est préférable de catégoriser une tâche pour des ressources en fonction de ses caractéristiques [3].

2.3 Le rôle d'ordonnanceur :

Un ordonnanceur devra trouver la machine la plus appropriée pour traiter les taches qui lui sont soumises. Les ordonnanceurs peuvent aller du plus simple (allocation de type round-robin) au plus compliqué (ordonnancement à base de priorités . . .). Les ordonnanceurs réagissent à la charge de la grille. Ils déterminent le taux de charge de chaque ressource afin de bien ordonnancer les prochaines taches. Ils peuvent être organisés en hiérarchie avec certains interagissant directement avec les ressources, et d'autres (méta ordonnanceurs) interagissant avec les ordonnanceurs intermédiaires. Ils peuvent aussi superviser le déroulement d'une tache jusqu'à sa terminaison, la soumettre à nouveau si elle est brusquement interrompue et la terminer prématurément si elle se trouve dans une boucle infinie d'exécution [04].

2.4. Différences entre l'ordonnancement préemptif et non préemptif :

Un ordonnanceur préemptif peut interrompre une tâche au profit d'une tâche plus prioritaire Un ordonnanceur non préemptif n'arrête pas l'exécution de la tâche courante. [05]

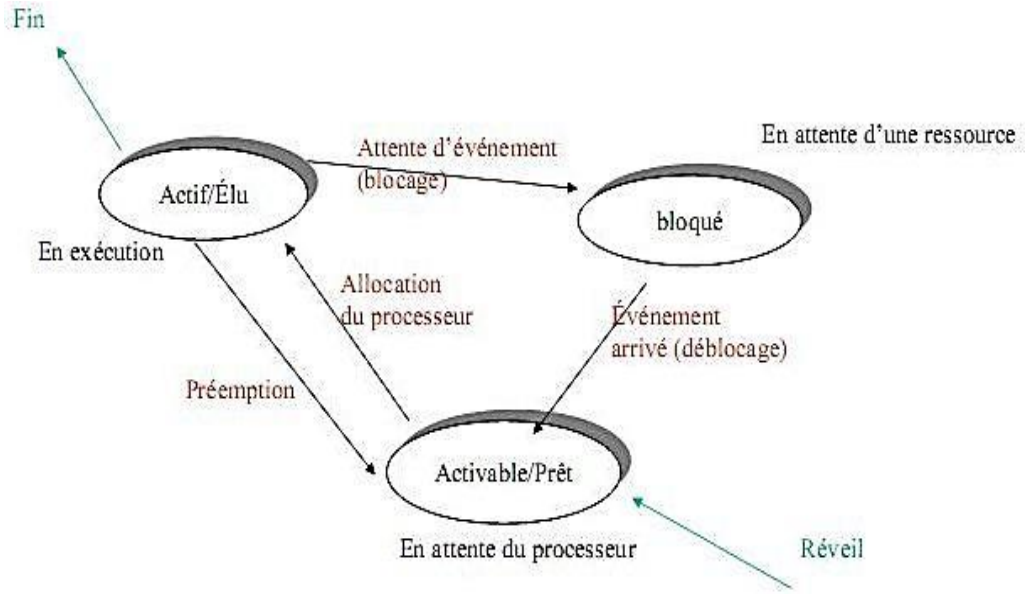


Figure 1 : Comportement d'une tâche

3 –Tâche :

Une tâche est une unité logique qui progresse vers un objectif unifié d'un système. Un tel ensemble de tâches pourrait être des images sur lesquelles un modèle distribué est en train d'apprendre ou de prédire. Un autre ensemble de tâches de ce type pourrait être un ensemble de calculs déchargés sur plusieurs nœuds en petits morceaux et envoyés aux travailleurs pour être calculés. Une tâche peut se terminer au nœud de travail - peut-être un changement d'état - ou une réponse peut être communiquée au principal. Il existe deux types de tâches : dépendantes et indépendantes.

Les tâches dépendantes s'appuient sur la réponse d'une tâche avant elle-même avant de pouvoir être exécutées. Ces tâches sont mieux adaptées à un système synchrone qui passe d'une tâche à l'autre en fonction des dépendances. Le système progresse à un rythme plus déterminable grâce à l'analyse prédictive de la succession des tâches.

Les tâches indépendantes peuvent être exécutées en parallèle et / ou simultanément. Ces tâches s'exécutent mieux dans un système asynchrone. Le problème à prendre en compte est la contention de ressources si un ensemble particulier de tâches nécessite la même ressource réseau (par exemple, accéder à une valeur de base de données) - cette situation a des solutions de contournement : des bases de données / serveurs cohérents éventuels. Le système fait une progression unifiée à un rythme indéterminé car les tâches peuvent s'exécuter simultanément vers l'objectif du système.

4 -Les ressources :

La ressource est un moyen technique ou humain destiné à être utilisé pour la réalisation d'une tâche et disponible en quantité limitée, On trouve plusieurs types de ressources :

- **Une ressource est renouvelable :** si après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible en même quantité (les hommes, les machines etc.) ; la quantité de ressources utilisée à chaque instant est limitée. Dans le cas contraire, elle est consommable (matière première, budget, etc.) ; la consommation globale (ou cumul) au cours du temps est limitée.
- **Les ressources disjonctives (ou non partageables) :** qui ne peuvent exécuter qu'une tache à la fois (machine-outil, robot manipulateur) et les ressources cumulatives (ou partageables) qui peuvent être utilisées par plusieurs tâches simultanément (équipes d'ouvriers, poste de travail).

5 Les contraintes :

Suivant la disponibilité des ressources et suivant l'évolution temporelle, deux types de contraintes peuvent être distingués [06] : contraintes de ressources et contraintes temporelles.

- **Les contraintes de ressources [07]** : plusieurs types de contraintes peuvent être induits par la nature des ressources. A titre d'exemple, la capacité limitée d'une ressource implique un certain nombre, à ne pas dépasser, de tâches à exécuter sur cette ressource. Les contraintes relatives aux ressources peuvent être disjonctives, induisant une contrainte de réalisation des tâches sur des intervalles temporels disjoints pour une même ressource, ou cumulatives impliquant la limitation du nombre de tâches à réaliser en parallèle.
- **Les contraintes temporelles [08] :** elles représentent des restrictions sur les valeurs que peuvent prendre certaines variables temporelles d'ordonnancement. Ces contraintes peuvent être [8] :

– Des contraintes de dates butoirs, certaines tâches doivent être achevées avant une date préalablement fixée,

– Des contraintes de précédence, une tâche *i* doit précéder la tâche *j*,

– Des contraintes de dates au plus tôt, liées à l'indisponibilité de certains facteurs nécessaires pour commencer l'exécution des tâches.

6 Les critères :

Généralement, l'objectif d'un problème d'ordonnancement consiste à optimiser un ou plusieurs critère(s) :

– L'utilisation maximale de l'UC.

– La capacité de traitement maximale.

– L'utilisation maximale des périphériques.

– La diminution du temps moyen de restitution : le temps mis par un processus pour aller de l'état ``nouveau'' à l'état ``terminé''.

– La diminution du temps moyen d'attente : le temps passé dans l'état ``prêt''.

– La diminution du temps moyen de réponse : dans un processus interactif, le temps entre une fin d'entrée et le début d'une sortie de résultat (pas la fin). C'est une mesure de ``réactivité''.

– L'équité.

7 -Allocation des tâches via le planificateur :

Le processus de gestion de l'attribution des tâches, à où et à qui, est la responsabilité d'un planificateur. Le planificateur d'un système distribué fonctionne comme le planificateur de processus sur n'importe quel système d'exploitation.

La planification des tâches du principal aux travailleurs peut se produire à plusieurs étapes. Il peut y avoir un planificateur global qui dirige toutes les tâches vers tous les travailleurs connectés. Il peut également y avoir des planificateurs locaux qui gèrent les tâches/ réponses entrantes et sortantes sur les nœuds principaux et de travail. L'organisation de cette échelle dynamique d'événements est une composante majeure de l'efficacité des systèmes. Dans un système distribué centralisé, il est plus courant de voir un planificateur global gérant l'allocation des ressources et des tâches entre le principal et les travailleurs.

Cependant, dans un système distribué décentralisé, il peut exister plusieurs planificateurs qui sont responsables d'une partie du système global. La gestion de cet état décentralisé peut être répartie sur les ressources. Chaque planificateur d'un système décentralisé peut être considéré comme le planificateur d'un système centralisé.

8 -Planification basée sur le cloud :

Infrastructure en tant que service (IaaS), le logiciel en tant que service (SaaS) et la plateforme en tant que service (PaaS) sont des modèles courants de services aux utilisateurs et chacun a ses propres exigences en matière de planification.

IaaS - le plus populaire dans les environnements cloud - offre des environnements physiques et virtuels pour déployer des calculs et / ou des applications à grande échelle dans un environnement public ou privé. Certaines de ces fonctionnalités sont limitées par des contrôles d'accès soumis à un coût par utilisation ou à un abonnement. Pour IaaS, le planificateur est responsable de «l'allocation des requêtes de la machine virtuelle sur [l'architecture] physique». Pour PaaS et SaaS, le planificateur est responsable de l'allocation

des applications et / ou des blocs de calcul susmentionnés aux travailleurs afin d'être exécutés / servis.

Une partie de l'optimisation des objectifs de la planification basée sur le cloud comprend l'accord de niveau de service (SLA) des utilisateurs avec leurs clients finaux (travailleurs). Le débit et la latence d'un système affectent directement les premiers utilisateurs et les accords SLA tiers - ces accords SLA peuvent être synonymes d'attentes de qualité de service (QoS).

Pour atteindre un SLA / QoS élevé, un ordonnanceur doit être robuste afin de réduire le temps et les coûts d'exécution. Un planificateur robuste doit être capable de gérer un changement de topologie d'application (de nouvelles instances étant ajoutées et supprimées de manière dynamique du pool), les configurations de ressources / logiciels, les données d'entrée et les sources de données. Il ne s'agit pas d'une liste complète des dynamiques possibles auxquelles un ordonnanceur devrait être robuste. L'une des plus grandes nuances des systèmes distribués est la capacité d'un programmateur à résister aux événements de panne.

La gestion des incidents de panne est un partisan majeur des systèmes distribués. Le planificateur doit être en mesure de récupérer, de redistribuer et / ou de ré implémenter ces occurrences afin de maintenir la qualité de service du système. Certes, si une occurrence de panne est suffisamment importante, il y aura des interruptions de service. Cependant, si une occurrence de panne est à petite échelle et gérée correctement, un utilisateur final peut même ne jamais la remarquer.

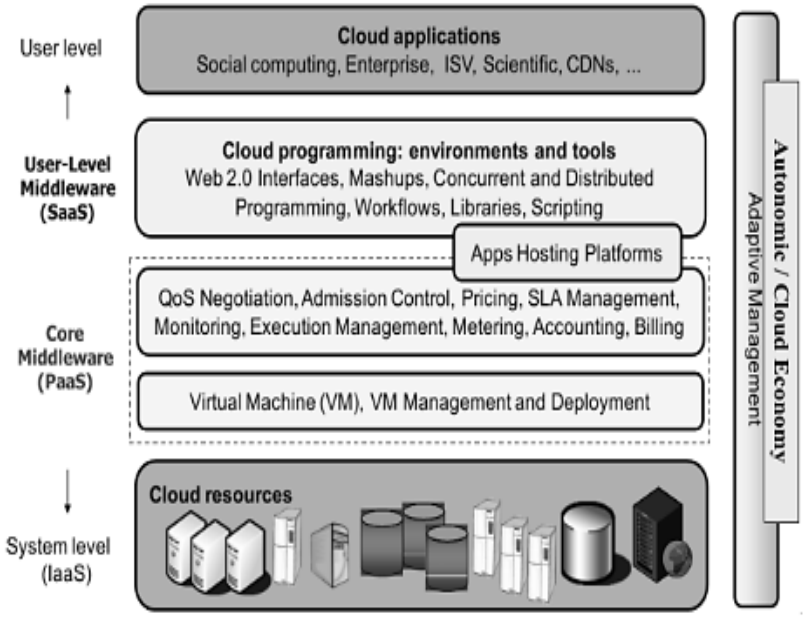


Figure 2 : Structure d'un Cloud

Chapitre -II -

Les Techniques D'ordonnancement Des Taches

(Les Heuristique et les Métaheuristique, Les Algorithmes)

1- Introduction :

La réalisation d'un projet d'informatique de gestion nécessite souvent une succession de tâches auxquelles s'attachent certaines contraintes :

- **De temps :** délais à respecter pour l'exécution des tâches ;
- **D'antériorité :** certaines tâches doivent s'exécuter avant d'autres ;
- **De production :** temps d'occupation du matériel ou des hommes qui l'utilisent.

Les techniques d'ordonnancement dans le cadre de la gestion d'un projet ont pour objectif de répondre au mieux aux besoins exprimés par un client, au meilleur coût et dans les meilleurs délais, en tenant compte des différentes contraintes.

L'ordonnancement se déroule en trois étapes :

- **La planification** : qui vise à déterminer les différentes opérations à réaliser, les dates correspondantes, et les moyens matériels et humains à y affecter.
- **L'exécution :** qui consiste à la mise en œuvre des différentes opérations définies dans la phase de planification.
- **Le contrôle :** qui consiste à effectuer une comparaison entre planification et exécution, soit au niveau des coûts, soit au niveau des dates de réalisation.

Il faut bien retenir qu'un seul algorithme d'ordonnancement universel, qui apporterait les meilleures performances pour tout type d'application ou matériel, n'existe pas.[09].

À travers ce chapitre nous allons présenter, les principaux algorithmes d'ordonnancement des tâches

2-Les Heuristiques :

Une méthode heuristique est une technique utilisée pour trouver des solutions prés à être optimales pour un coût de calcul raisonnable. Les heuristiques ne garantissent pas la faisabilité ou l'optimalité des solutions qu'ils produisent. En exploitant la structure d'un problème, des solutions raisonnables peut être décelés. Une approche heuristique devrait être utilisée lorsque des méthodes de résolution exactes nécessitent des calculs énormes impraticables. Les problèmes NP-complet sont généralement abordés avec des techniques heuristiques.

Le théorème Wolpert connu sous le nom de *no free lunch*, montre qu'il n'existe pas d'heuristique qui soit meilleure qu'une autre : pour toute heuristique, il existe au moins une instance du problème pour laquelle une autre méthode lui est supérieure. L'importance et la portée pratique de ce théorème suscitent des avis contradictoires [10]. Néanmoins du point de vue pratique, pour atteindre de bonnes performances, il est le plus souvent nécessaire

d'introduire dans les méthodes de résolution une certaine connaissance ou information sur l'instance ou le problème à résoudre.

Les heuristiques sont également des règles ou des stratégies utilisées pour améliorer l'efficacité d'autre méthodes ou même d'autres approches heuristiques. Braun et al [11], fournit une comparaison de l'heuristiques statiques d'ordonnancement. les étapes qu'elle suit chaque heuristique, on peut les regrouper dans des catégories suivantes [11]

- La construction : les solutions finales sont générées par l'ajout d'une composante de la solution à la fois.
- L'amélioration : une séquence de transformations est appliquée à partir d'une solution de départ pour l'améliorer. Cela comprend la classe importante des heuristiques de recherche locale.
- Le partitionnement : Le problème est divisé en sous-problèmes qui sont résolu de façon indépendante. Les solutions fragmentées sont combinées ensemble pour former une solution finale au problème original.
- La détente : Certaines contraintes sont assouplies afin de rendre le problème plus facile à résoudre, des transformations doivent être réalisées sur des solutions irréalisables pour les rendre possibles.
- La restriction : L'espace de solution est limitée pour rendre le problème plus facile à résoudre.

3 -Les Meta-heuristiques :

Les méta-heuristiques constituent l'une des trois classes des méthodes d'ordonnancement (les autres étant les méthodes exactes et les heuristiques). Elles sont adaptables à un grand nombre de problèmes sans changement majeur de l'algorithme. Il existe un grand nombre de métas heuristiques différents, leur permettant d'être adaptées à une large gamme de problèmes différents [12]. Une des façons de classer les méta-heuristiques est de distinguer celles qui utilisent des méthodes de recherche locale, qui ne manipulent qu'une seule solution à la fois, de celles qui exploitent une population de solutions. Les méta-heuristiques à recherche locale font évoluer à chaque itération une seule solution sur l'espace de recherche. C'est pourquoi la notion de voisinage est primordiale. Parmi les méta-heuristiques avec recherche locale, nous pouvons citer le recuit simulé et la recherche tabou.

L'autre approche de cette classification manipule en parallèle plusieurs solutions à chaque itération. Il existe principalement deux types de méta-heuristique à base de population : les colonies de fourmis et les algorithmes génétiques.

4-Équilibrage de charge :

Le problème de l'équilibrage de charge consiste à allouer des données au départ (fractions d'une application donnée), puis éventuellement à les redistribuer sur un ensemble de processeurs afin de minimiser leur temps de traitement. Ce problème peut être classifié en deux sous-catégories, l'équilibrage de charge dynamique et l'équilibrage de charge statique.[13]

4-1 Approche dynamique :

La distribution dynamique prend en compte la charge actuelle des nœuds dans un système distribué lors d'une répartition de taches. S'il existe un changement important de charge pendant l'exécution, un transfert de taches d'un nœud vers un autre est alors envisageable. L'algorithme utilisé dans ce type de distribution doit être capable de connaitre la charge des différents nœuds. Le but de ce transfert de taches est de trouver un nœud, qui permettra d'obtenir les meilleures performances si un ensemble de taches lui est soumis. Pour cela, il est nécessaire de connaitre l'état de charge de tous les nœuds d'un système, d'où la nécessité de disposer d'un système d'informations relatifs aux nœuds d'un système distribué.

Ce système d'informations est compliqué à mettre en œuvre et il devra garder toute son intégrité lors des opérations de transfert de taches dans le système. Cette approche est beaucoup plus souple que l'approche statique, puisqu'elle permet une exploitation multi-utilisateur (plusieurs applications exécutées en même temps).

D'autre part, cette approche nécessite un surcout de calcule pour analyser continuellement les états des nœuds, superviser l'échange inter-nœuds, calculer la charge de travail et la distribuer.[14]

4.2 Approche statique :

La distribution statique affecte les taches aux ressources, par des allocations uniques et définitives, en faisant abstraction de tout événement pouvant se produire en cours d'exécution. Les décisions d'affectation de tâches sont prises avant leurs exécutions.

L'algorithme utilisé doit donc pouvoir estimer les charges a priorité, pour permettre un équilibrage de charge effectif (la charge de chaque ressource du système est prédéfinie avant d'entamer l'exécution). Aussi, les différentes capacités des ressources doivent être connues à l'avance. C'est une méthode qui se prête mieux aux applications pour lesquelles on dispose de connaissances préalables et qui sont conçues pour s'exécuter sur un système à comportements prédéfinis. Les diverses tâches sont traditionnellement réparties au moment de la compilation.

L'avantage de ce type d'équilibrages est qu'il n'exige aucun temps d'exécutions supplémentaire (sur cout). Par contre, l'inconvénient majeur de telles solutions réside dans le fait qu'elles ne prennent pas en compte le caractère dynamique des machines et des taches.[15]

5 Les principaux algorithmes d'ordonnancement :

5.1 la Recherche tabou :

Des approches basées sur l'heuristique Tabou ont prouvé leur efficacité pour résoudre des problèmes d'ordonnancement. L'idée de la recherche Tabou est la suivante : à partir d'une donnée, on explore le voisinage et on choisit la position dans ce voisinage qui minimise la fonction objectif

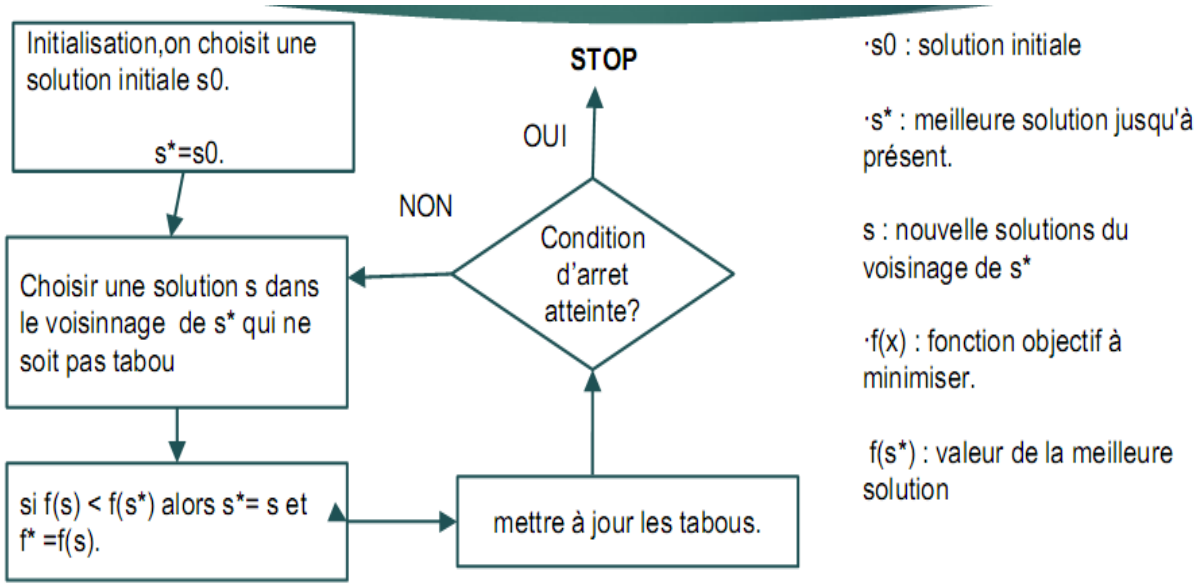


Figure 3 : L'algorithme TABOU.

5.2- Les algorithmes de colonie de fourmis (ANT) :

Un algorithme de colonies de fourmis est un algorithme itératif à population où tous les individus partagent un savoir commun qui leur permet de guider leurs futurs choix et d'indiquer aux autres individus des directions à suivre ou au contraire à éviter.

Fortement inspirée du déplacement des groupes de fourmis, cette méthode a pour but de construire les meilleures solutions à partir des éléments qui ont été explorés par d'autres individus. Chaque fois qu'un individu découvre une solution au problème, bonne ou mauvaise, il enrichit la connaissance collective de la colonie. Ainsi, chaque fois qu'un nouvel individu aura à faire des choix, il pourra s'appuyer sur la connaissance collective pour pondérer ses choix.

Les algorithmes de colonies de fourmis sont nés à la suite de constatations faites surle

comportement des fourmis qui sont capables de trouver le chemin le plus court, du nid à une source de nourriture, et de s'adapter aux changements de l'environnement. Les biologistes ont, ainsi, étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation des sources de nourriture. Les fourmis sortent de leur nid pour chercher de la nourriture et déposent une substance chimique appelée phéromone tout au long de leur parcours. Les premières fourmis ayant trouvé la nourriture reviennent au nid et marquent deux fois leurs parcours.

Cette substance guide le choix des trajets que vont effectuer les autres fourmis par deux mécanismes qui sont le renforcement et l'évaporation de la phéromone, les fourmis sont capables de dépasser certains obstacles en faisant des ponts (constituéde plusieurs fourmis) et elles choisissent les branches les plus courtes, comme le montre ***la Figure 4.***.

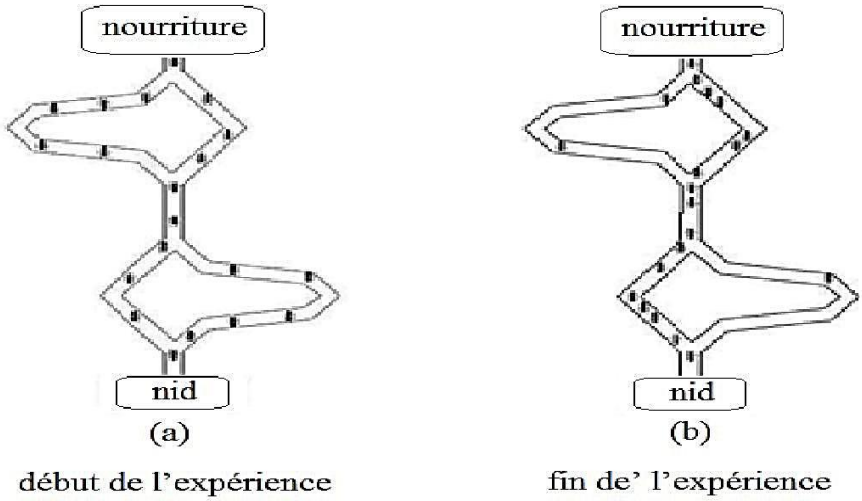


Figure 4- Sélection des branches les plus courtes par une colonie de Fourmis

Ce qui constitue une propriété très importante en optimisation qu'est celle de la convergence. Ces algorithmes ont donc pour but de reproduire le comportement naturel des fourmis pour retrouver le meilleur chemin possible vers un objectif donné. Cette approche est une méthode de diversification, elle consiste à attribuer à chaque fourmi une charge électrostatique. L'adaptation au domaine continu a été présentée par Krzysztof Socha **[16]**. L'algorithme itère sur une population de fourmis finie et constante de k individus, où chaque fourmi représente une solution ou une variable.

5.3 La descente stochastique avec la méthode KANGOUROU :

La descente stochastique est une heuristique qui part d'un principe basé sur le voisinage d'une solution X. Le voisin Y obtenu est acceptée si et seulement si Y est meilleur que X. Le seul paramètre important d'un algorithme de ce type est la méthode de sélection ou de génération d'un voisin.

La méthode de kangourou est publiée la première fois par Fleury en 1995 [17]. La méthode Kangourou offre une solution par une descente stochastique et une transition dans le voisinage de l'état actuel pour trouver une meilleure solution de la solution courante. La méthode donne un optimum local dans un temps acceptable, basé sur le recuit simulé, il permet l'étude de problèmes à forte combinatoire. Contrairement à la recherche taboue et aux algorithmes évolutionnistes, la méta-heuristique « méthode Kangourou » n'a besoin que d'une seule évaluation du critère de performance à chaque itération, ce qui est intéressant du point de vue du temps de calcul. L'intérêt de cette méthode est qu'elle est facile à mettre en œuvre, elle peut être couplée sans difficulté avec un modèle pour l'évaluation des performances et on dispose a tout instant d'une solution réalisable. L'algorithme du kangourou a beaucoup d'avantages car il permet la recherche globale ainsi que le réglage de paramètres du recuit simulé. Il présente plusieurs inconvénients comme le nombre de stationnements et de sauts nécessaire pour la recherche global.

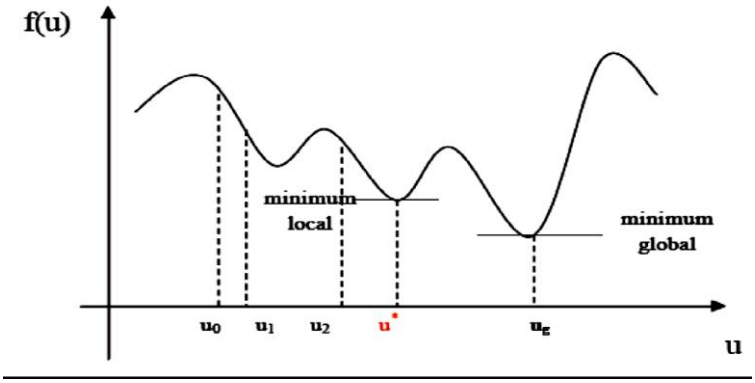


Figure 5 : Principe de KANGOUROU pour trouver un minimum local

6-Stratégies d'ordonnancement Max-Min, SLB, SJF, LBE, FIFO :

6.1- L'heuristique Max-Min : L'heuristique Max-Min commence par sélectionner la tâche non ordonnancé pour laquelle le temps d'exécution est minimal sur l'ensemble des machines est le plus long parmi toutes les taches non ordonnancées. Max-min essaye de réduire au minimum les pénalités encourues d'exécuter des taches avec des temps d'exécution plus longs. Par exemple si l'application étant ordonnancé a beaucoup de tâches avec des temps d'exécution très courts, et une tâche avec du temps d'exécution très long, alors Max-Min permet d'abord à cette tâche d'être exécutée en même temps que les tâches restantes (avec des temps d'exécution plus courts).

Pseudo-Code for Max-Min Algorithm

```
While (Critère d'arrêt n'est pas satisfait) do
{
//le temps d'exécution de chaque machine égale à la somme des temps d'exécution des taches qu'elles lui sont
affectées.
For toutes les Temps T
{
For toutes les machines mi {
For toutes les tâches tj
{
Chercher max ij la valeur maximum de TC ; // le maximum temps d'exécution de la
Tâche tj dans la machine mi.
}
For toutes les machines mj
{
temp_exe i = temp_exe i +TC ij. // le temps d'exécution de la tâche qui a
min ij dans la machine mi.
}
For toutes les machines mi
{
Trouver la machine qui a temp_exe i minimum.
}
Affecter cette tâche tj à la machine qui a le minimum de qui temp_exe i.
If (toutes les taches sont traitées)
Critère d'arrêt est satisfait ;
}
```

Figure 6 - pseudo code de l'algorithme Max-Min.

6.2 Algorithme FIFO/FCFS : L'algorithme FIFO (First In First Out) ou FCFS (First Come First Served) est l'un des algorithmes les plus simples qu'il soit. L'idée est d'ajouter chaque tâche et ressource disponible dans une file et d'exécuter chaque tâche et ressource par ordre d'arrivée.

6.3 Shortest Job First (SJF)/Plus court d'abord : L'algorithme SJF ressemble au FIFO, mais au lieu d'exécuter dans l'ordre d'arrivée, on choisit d'exécuter celui qui sera le plus court.

6.4-SLB (Simple Loading Balancing) :

La méthode d'équilibre de charge SLB (Simple Loading Balancing) préconise la réduction du temps d'exécution en transférant les tâches affectées sur la machine qui possède le temps d'exécution maximum à une autre machine possédant le temps d'exécution minimum afin de minimiser le makespan.

Algorithme 1 : ***Simple Algorithme d'équilibrage de charge***

While (critère d'arrêt n'est pas satisfait) do

{ Initialiser un auxiliaire à 0 ;//pour faire le test ;

Trouver machine m_{max} avec le maximum temps d'exécution : ct_{max} ;

Trouver machine m_{min} avec le minimum temps d'exécution : ct_{min} ;

For toutes les tâches t_j affectées sur m_{max} do

{Affecter tâche t_j sur m_{min} ;

$C_{m_{min}j} = w_{m_{min}j} + ETC_{m_{min}j}$; // Estimer temps d'exécution sur m_{min};

$C_{m_{max}j} = w_{m_{max}j} - ETC_{m_{min}j}$; // Estimer temps d'exécution sur m_{max} quand t_j est déplacée de m_{max};

If (($C_{m_{min}j} < ct_{max}$) and ($C_{m_{max}j} <$ auxiliaire) // pour avoir le minimum de $C_{m_{max}j}$

{

Affecter $C_{m_{max}j}$ à l'auxiliaire ;

Sauvegarder j ;//sauvegarder la tâche qui donne le minimum temps d'exécution

} }

Affecter la tâche t_j qui donne le minimum temps d'exécution parmi toutes affectées sur la machine m_{max} // la tâche j que nous avons déjà sauvegardé ;

Else

Critère d'arrêt est satisfait ; //la valeur du makespan n'est pas changée.

}

Figure 7- pseudo code de l'algorithme SLB

6.5 L'équilibrage de charge avec échange LBE (Loading Balancing withExcange) :

Pour améliorer les performances de l'heuristique SLB, une nouvelle technique d'équilibrage de charge est proposée, c'est l'équilibrage de charge avec échange LBE (Loading Balancing with Excange). Cette technique généralise l'ordonnancement SLB. Son principe est de minimiser le temps d'exécution maximal obtenue par l'ordonnancement, en réattribuant les tâches dans le but d'améliorer le temps nécessaire de la solution d'ordonnancement Min-Min, tout en minimisant le temps d'exécution maximum.

LBE considère la machine ayant le temps d'exécution maximum pour le réduire, par l'échange des tâches avec les autres machines, et par la réaffectation des tâches entre elles.

Pour chaque tache tj de la machine ayant le temps maximal d'exécution, et pour chaque tâche t_k des autres machines, on effectue un échange de la tâche tj et de la tâche t_k. Nous réaffecterons une seule paire d'échanges des tâches en même temps qui donne le makespan minimum parmi toutes les affectations de tâches entretoutes les machines.

Chapitre -III -

Les algorithmes génétiques

1-Introduction :

L'algorithme génétique Introduit par J.H. Holland et son équipe du Michigan, Ses travaux ont commencé en 1960 et en 1975, il publia le premier ouvrage ayant comme titre « Adaptation in Natural and Artificiel System », Objectifs de L'algorithme génétique : améliorer la compréhension des processus naturels d'adaptation, et concevoir des systèmes artificiels possédant des propriétés similaires aux systèmes naturels.

L'algorithme génétique (AG) est un algorithme de recherche basé sur les mécanismes de la sélection naturelle et de la génétique. Il combine une stratégie de "survie des plus forts" avec un échange d'information aléatoire mais structuré. Pour un problème pour lequel une solution est inconnue, un ensemble de solutions possibles est créé aléatoirement. On appelle cet ensemble la population. Les caractéristiques (ou variables à déterminer) sont alors utilisées dans des séquences de gènes qui seront combinées avec d'autres gènes pour former des chromosomes et par après des individus. Chaque solution est associée à un individu, et cet individu est évalué et classifié selon sa ressemblance avec la meilleure, mais encore inconnue, solution au problème. Il peut être démontré qu'en utilisant un processus de sélection naturelle inspiré de Darwin, cette méthode convergera graduellement à une solution.

Comme dans les systèmes biologiques soumis à des contraintes, les meilleurs individus de la population sont ceux qui ont une meilleure chance de se reproduire et de transmettre une partie de leur héritage génétique a la prochaine génération. Une nouvelle population, ou génération, est alors créée en combinant les gènes des parents. On s'attend à ce que certains individus de la nouvelle génération possèdent les meilleures caractéristiques de leurs deux parents, et donc qu'ils seront meilleurs et seront une meilleure solution au problème. Le nouveau groupe (la nouvelle génération) est alors soumis aux même critères de sélection, et par après génère ses propres rejetons. Ce processus est répété plusieurs fois, jusqu'à ce que tous les individus possèdent le même héritage génétique. Les membres de cette dernière génération, qui sont habituellement très différents de leurs ancêtres, possèdent de l'information génétique qui correspond à la meilleure solution au problème. L'algorithme génétique de base comporte trois opérations simples qui ne sont pas plus compliquées que des opérations algébriques :

– Sélection
– Reproduction
– Mutation [18]

2-Pourquoi les Algorithme génétique ?

Les Algorithme génétique ne permettent pas d'obtenir assurément une solution optimale exacte, mais plutôt une solution de qualité et ça en peu d'effort.

- Espaces de recherche importants
- Pas d'algorithmes déterministes adaptés [19]

3- la terminologie de l'algorithme génétique : Comme les algorithmes génétiques ont leurs racines à la fois dans la biologie et l'informatique, la terminologie utilisée est empreintée aux deux domaine nous allons passer en revue le lien entre les termes utilisés et leurs équivalents naturels pour ainsi nous aligner sur la littérature des algorithmes génétiques, en plein développement, et aussi pour nous permettre dans la suite de définir quelques analogies terminologiques.

Terme	Algorithme génétique	Signification biologique
Gène	trait, caractéristique	une unité d'information génétique transmise par un individu à sa descendance
Locus	position dans la ébahie	l'emplacement d'un gène dans son chromosome
Allèle	valeur de caractéristique	une dam différentes formes que peut prendre un gène, les allèles occupent le même locus
Chromosome	chaine	une structure contenant les gènes
Génotype	structure	l'ensemble des allèles d'un individu ponts par l'ADN d'une cellule vivante
Phénotype	ensemble de paramètres ou une structure décodé	aspect physique et physiologique observable de l'individu obtenu à partir de son génotype
Epistasie	non-linéarité	terme utilisé pour définir les relations entre deux gènes "distincts", lorsque la présence d'un gène empéche la présence d'un autre gène non-allèle.

Tableau 1: Résumé de la terminologie utilisée en AG

4 -Les avantages de L'AG :

Par rapport aux algorithmes classiques d'optimisation, l'algorithme génétique présence plusieurs points forts comme :

– Le fait d'utiliser seulement l'évaluation de la fonction objectif sans se soucier de sa nature. En effet, nous n'avons besoin d'aucune propriété particulière sur la fonction à optimiser (continuité, dérivabilité, convexité, etc.), *ce* qui lui donne plus de souplesse et un large domaine d'applications.

– Génération d'une forme de parallélisme en travaillant sur plusieurs points en même temps (population de taille N) au lieu d'un seul itéré, dans les algorithmes classiques.

– L'utilisation des règles de transition probabilistes (probabilités de croisement et de mutation), contrairement aux algorithmes déterministes où la transition entre doux itérations successives est imposée par la structure et la nature de l'algorithme. Cette utilisation permet dans certaines situations aux algorithmes génétiques d'éviter des optimums locaux et de se diriger vers un optimum global.[20]

5 -Les inconvénients de L'AG :

Les AG ne sont encore actuellement pas très efficaces en cout (ou vitesse de convergence), vis à vis de méthodes d'optimisation plus classiques.[21]

– Parfois les AG convergent très vite vers un individu particulier de la population dont la valeur d'adaptation est très élevée.

– Le respect des contraintes de domaine par les solutions codées sous forme de chaines de bits pose parfois problème. Il faut bien choisir le codage, voire modifier les opérateurs.

– En pratique l'efficacité d'un AG dépend souvent de la nature du problème d'optimisation. Selon les cas, le choix des paramètres et des opérateurs sera souvent critique, mais aucune théorie générale ne permet de connaitre avec certitude la bonne para-métrisation. Il faudra faire plusieurs expériences pour s'en approcher.[22].

6 -Principes de base *des* AG :

Indépendamment de la problématique traitée, les algorithmes génétiques sont basés sur six principes :

1. Choisir le codage des solutions.

2. Générer une population initiale de taille fixe *N,* formée d'un ensemble fini de solutions, dite génération initiale.

3. Définir une fonction d'évaluation (fitness) permettant d'évaluer une solution et la comparer aux autres.

4. Choisir les solutions par un mécanisme de sélection qui choisit pour un éventuel couplage.

5. Générer de nouvelles solutions à l'aide des opérateurs génétiques en utilisant :

- Opérateur de croisement : il manipule la structure des chromosomes des parents afin de produire dos individus meilleurs ou différents. Cet opérateur est effectué selon une probabilité $\boldsymbol{P_x}$.
- Opérateur de mutation : il évite d'établir des populations uniformes incapables d'évoluer. Il consiste à modifier les valeurs des gènes de chromosomes selon une probabilité de mutation ***P***m.

6. Établir un compromis entre les solutions produites (progénitures) et les solutions productrices (Les parents) en utilisant un mécanisme. d'insertion. En d'autres termes, et suite à des informations précises, décider ce qui doit rester et ce qui doit disparaître. Tout ceci, on sauvegardant à chaque génération une taille do la population N fixe.

7 -Fonctionnement des AG :

L'algorithme génétique, présenté dans la figure 8, débute par une génération d'une population initiale de N individus, pour lesquels, nous calculons les valeurs de leur fonction objectif et nous sélectionnons los individus par une méthode de sélection. Les individus, sujets de croisement par l'opérateur de croisement, sont choisis selon une probabilité $\boldsymbol{P_x}$**.** Leurs résultats peuvent être mutés par un opérateur de mutation avec une probabilité de mutation ***Pm.*** Les individus issus de ces opérateurs génétiques seront insérés par une méthode d'insertion dans la nouvelle population dont nous évaluons la valeur de la fonction objective de chacun de ses individus. Un test d'arrêt sera effectué pour vérifier la qualité des individus obtenus. Si ce test est vérifié alors l'algorithme s'arrête avec une solution optimale, sinon en réitéré le processus pour la nouvelle génération[23].

7.1 -Codage du chromosome :

Le choix du codage des données dépend de la spécificité du problème traité. Il conditionne fortement l'efficacité do l'algorithme génétique. Un chromosome (une solution particulière) a différentes manières d'être codé selon l'alphabet utilisé. Nous distinguons trois types de codages :

- Numérique si l'alphabet est constitué de chiffres.
- Symbolique si l'alphabet est un ensemble de lettres alphabétiques ou des symboles.
- Alpha-numérique si nous utilisons un alphabet combinant les lettres et las chiffres.[24]

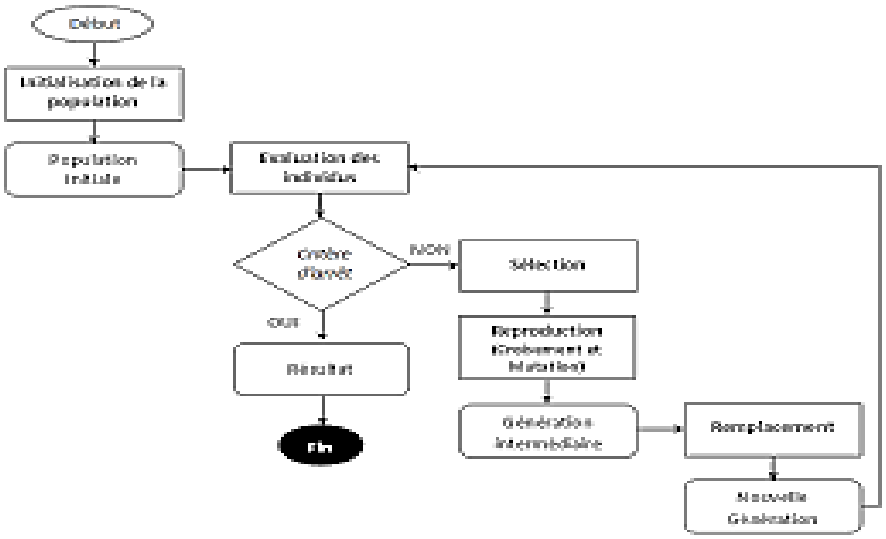


Figure 8 -l'organisation de l'algorithme génétique.

7.2 -Génération de la population initiale :

Dans les problèmes d'optimisation, une connaissance de "candidats de bonne qualité" comme points d'initialisation conditionne la rapidité de la convergence vers l'optimum. Si la position de l'optimum dans l'ensemble des solutions réalisables est totalement inconnue, il est naturel de générer aléatoirement des individus. Les tirages sont réalisés en respectant las cnntrainta4 et d'une manière uniforme dans chacun des domaines associés aux composantes de l'ensemble de solutions. Des connaissances â priori sur le problème traité permettent de générer los individus dans un domaine particulier afin d'accélérer la convergence de l'algorithme génétique. Les opérateurs de croisement et de mutation permettent d'entretenir la diversité d'une population non homogène au cours des générations afin de parcourir l'ensemble de solutions le plus largement possible. [25]

7.3 -Méthodes de sélection :

La sélection permet d'identifier les individus susceptibles d'être croisés dans une population. Nous trouvons dans la littérature plusieurs principes de sélection :

Sélection par rang : Il consiste à attribuer à chaque individu son classement par ordre d'adaptation.

Pour un problème de maximisation, nous classons les individus selon l'ordre croissant des valeurs de la fonction objectif. Ainsi, le plus mauvais individu (c'est-à-dire celui qui possède la plus petite valeur de la fonction objectif) prendra le numéro 1 et ainsi de suite (voir Tableau 2). Pour un problème de minimisation, nous ordonnons les individus selon l'ordre décroissant. On prélève ensuite une nouvelle population à partir de cet ensemble d'individus ordonnés, en utilisant dos probabilités indexées sur les rangs des individus :

Probabilité de sélection (Parent i) $= \frac{Rang(Parent\ i)}{\sum_{j \in population} Rang(Parent\ j)}$

Cette procédure est très simple et exagère le râle du meilleur élément au détriment d'autres éléments potentiellement exploitables. Le second, par exemple, aura une probabilité d'être sélectionné plus faible que le premier, bien qu'il soit peut âtre situé dans une région d'intérêt.

	Chromosome	Fitness	Rang	Prob de sélection $\frac{Rang}{total(Rang)}$
	Parent 1	30	2	33.33%
	Parent 2	60	3	50 %
	Parent 3	10	1	16.67%
Total		100	6	100%

Tableau 2 Sélection par rang pour un problème de maximisation.

Sélection par la roulette : Dans un problème d'optimisation de maximisation, on associe à chaque individu i une probabilité de sélection, noté **Prob i**. proportionnelle à sa valeur *F*, de la fonction objectif Fi de la fonction objectif :

$$\text{Prob i} = \frac{Fi}{\sum_{j\ population} (Fj)}$$

Chaque individu est alors reproduit avec la probabilité *Prob i*. Certains individus (les "bons") seront alors "plus' reproduits et d'autres (les "mauvais', éliminés *(voir* Tableau 3).

	Chromosome	Fitness	Prob de sélection $\frac{Fitness}{total(Fitness)}$
	Parent 1	30	30 %
	Parent 2	60	60 %
	Parent 3	10	10 %
Total		100	100%

Tableau 3: Sélection par la roulette pour un problème de maximisation.

Pour un problème de minimisation, on utilise une probabilité de sélection pour un individu i égala à $\frac{1-Prob\ i}{N-1}$

Sélection aléatoire : La sélection se fait aléatoirement., uniformément et sans intervention de la valeur d'adaptation. Chaque individu a donc une probabilité uniforme (1/N) d'être sélectionné. En général, la convergence de l'algorithme génétique est *lente* en utilisant cette méthode.

Sélection par Tournoi : Cette méthode de sélection augmente les chances des individus de mauvaise qualité par rapport à leur fitness, de participer à l'amélioration de la population. En effet, c'est une compétition entre les individus d'une sous-population de taille M (M ≤ N) prise au hasard dans la population. Le paramètre M est fixé à, priori par l'utilisateur. L'individu de meilleure qualité par rapport à la sous-population sera considéré comme vainqueur et sera sélectionné pour l'application de l'opérateur de croisement. Le paramètre *M* joue un rôle important dans la méthode du tournoi.

Dans le cas où *M = N* avec N est la taille de la population. Le résultat par la sélection de la méthode du tournoi donne à chaque fois un seul individu qui *est* le meilleur individu par rapport à la valeur de la fonction objective. Ce qui réduit l'algorithme génétique a un algorithme de recherche local travaillant sur une seule solution à la fois. Ce type d'algorithmes a pour inconvénient de converger parfois rapidement vers un optimum local.

Dans le cas M = 1, la méthode do sélection du tournoi correspond à la sélection aléatoire. .[25]

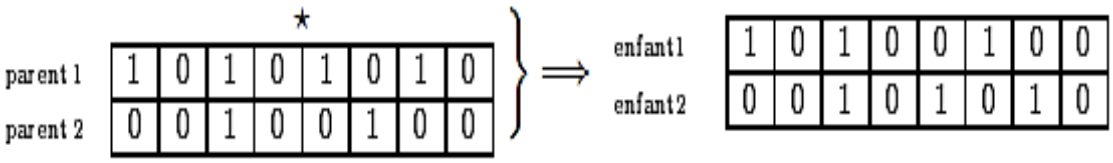
7.4 Opérateurs de croisement :

Le croisement a pour but d'enrichir la diversité de la population en manipulant les composantes des chromosomes. Classiquement, les croisements sont envisagés avec deux parents et génèrent deux enfants. Il est appliqué avec une probabilité P_x ,communément appelée probabilité de croisement. Après l'utilisation de la méthode de sélection pour le choix de deux individus, nous générons un nombre aléatoire a [0,1].

Si *(a < Px)*, nous appliquons l'opérateur do croisement sur le couple. Les plus anciens opérateurs de croisement utilisés sont l'opérateur de croisement à un point et à deux points sur deux chromosomes à codage binaire. Ils constituent la base des opérateurs *de* croisement.

L'opérateur à un point de croisement consiste à diviser chacun des deux parents en deux parties à la même position, choisie au hasard. L'enfant 1 est composé de la première partie du premier parent et de la deuxième partie du deuxième parent alors que l'enfant 2 est

constitué de la première partie du deuxième parent et de la deuxième partie du premier parent (tab 4). .[24]

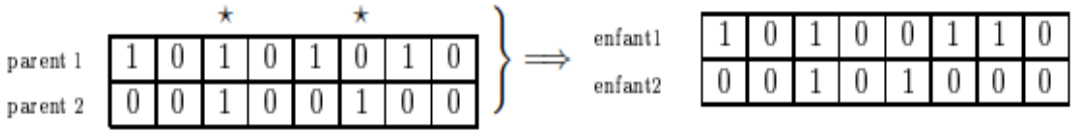


1	0	1	0	1	0	1	0
0	0	1	0	0	1	0	0

1	0	1	0	0	1	0	0
0	0	1	0	1	0	1	0

Tableau 4 -l'opérateur de croisement a 1 point .

L'opérateur à doux points do croisement est illustré dans le tableau 5. Il consiste à fixer deux positions. L'enfant 1 sera la copie du parent 1 en remplaçant sa partie entre les deux positions par celle du parent 2. On effectuera la môme opération pour déterminer l'enfant 2 en intervertissant les rôles des parent 1 et parent 2.



1	0	1	0	1	0	1	0
0	0	1	0	0	1	0	0

1	0	1	0	0	1	1	0
0	0	1	0	1	0	0	0

Tableau 5 -L'opérateur de croissement a 2 points.

7.5 Opérateurs de mutation :

L'opérateur de mutation apporte aux algorithmes génétiques l'aléa nécessaire à une exploration efficace de l'espace. Cet opérateur nous garantit que l'algorithme génétique sera susceptible d'atteindre la plupart des points du domaine réalisable. Cerf en 1994 [25] a démontré théoriquement que l'algorithme génétique converge en probabilité en utilisant l'opérateur de mutation et *sans* croisement. Les propriétés de convergence des algorithmes génétiques sont donc fortement dépendantes do *cet* opérateur.

Cet opérateur de mutation est utilisé avec une probabilité (P_m) nommée probabilité de mutation. Dans les algorithmes génétiques à codage binaire, cette probabilité s'effectuait sur les gènes en échangeant sa valeur de 0 à 1 ou de 1 à 0 et non sur le chromosome tout entier.[25]

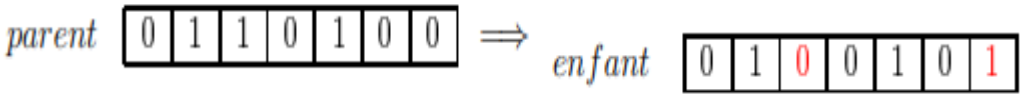


Tableau 6 Opérateur de mutation d'un bits.

7.6 -Méthode d'insertion :

Après l'étape de mutation, on utilise une méthode d'insertion peur générer une nouvelle population. Lors de la construction de cette population, on se trouve devant un vrai problème : faut-il garder lm enfants ou les parents nu bien un certain pourcentage des deux en respectant que la taille de la population (N) reste constante

Il s'agit do concevoir une stratégie d'évolution do la population. Nous distinguons dans la littérature deux stratégies :

– La première stratégie, notée (N. Nf), consiste à choisir les N individus à partir de Nf enfants déjà créés par les opérateurs de croisement et de mutation. Dans cette stratégie, on suppose que Nf ≥ N. Quand Nf = N, nous parlerons de la méthode générationnelle qui remplace les parents par les enfants.

– La seconde, notée (N + Nf), consiste à choisir les N individus à partir des N parents de la population précédente et de Nf nouveaux enfants. Un cas particulier de cette stratégie, appelé méthode d'état d'équilibre, a pour principe de sauvegarder une grande partie de la population dans la génération suivante. A chaque itération quelques chromosomes (parents) ayant les meilleurs coûts seront sélectionnés afin de créer des chromosomes lita qui remplaceront les plus mauvais parents. Ln reste de la population survie et sera copié dans la nouvelle génération.

L'élitisme est une stratégie complémentaire de la première stratégie. il consiste à copier quelques meilleurs chromosomes dans la nouvelle population. Il accroit l'efficacité de l'algorithme génétique basé sur la méthode d'insertion générationnelle. L'objectif est d'éviter que les meilleurs chromosomes soient perdus après les opérations de croisement et de mutation. Cette méthode améliore considérablement les algorithmes génétiques, car elle permet de conserver, à une itération k, le meilleur individu trouvé dans toutes les populations générées antérieurement. [25]

7.7 -Test d'arrêt :

Le test d'arrêt joue un rôle primordial dans le jugement de la qualité des individus. Son but est de nous assurer l'optimalité, de la solution finale obtenue par l'algorithme génétique.
Les critères d'arrêts sont de deux natures :

1. Arrêt après un nombre fixé a priori de générations. C'est la solution retenue lorsqu'une durée maximale do temps do calcul est imposée.
2. Arrêt lorsque la population cesse d'évoluer ou n'évolue plus suffisamment. Nous sommes alors en présence d'une population homogène dont on peut penser qu'elle se situe à la proximité de l'optimum. Ce test d'arrêt reste le plus objectif et le plus utilisé.

Il est à noter qu'aucune certitude concernant la bonne convergence de l'algorithme n'est assurée. Comme dans toute procédure d'optimisation l'arrêt *est* arbitraire, et la solution "en temps fini" ne constitua qu'une approximation de l'optimum. [25]

8 –Conclusion :

Dans ce chapitre, nous avons établi les fondations nécessaires à la compréhension des algorithmes génétiques. Nous avons exposé en détail les différentes étapes qui constituent la structure générale d'un algorithme génétique : Codage, méthode de sélection, opérateurs de croisement et de mutation avec leurs probabilités, méthode d'insertion et le test.

Chapitre -IV-

Implémentation, résultat et discussion

1-Introduction :

Dans la première partie de ce chapitre, nous allons faire la conception de notre application en utilisant le langage UML. Cependant notre application n'est pas d'une grande complexité afin d'utiliser la totalité des diagrammes d'UML, nous n'utiliserons que les diagrammes que nous trouverons nécessaires. La deuxième partie de chapitre consacré à la présentation de la mise en œuvre de notre application, nous commençons tout d'abord par une présentation du langage de programmation choisi. Ensuite nous montions les détails de notre application. On se termine ce chapitre par une synthèse de nos résultats obtenus.

Partie 1 :

2. La conception d'application :

Nous avons choisi l'UML pour modéliser notre application grâce à ces caractéristiques. Simplifie et efficace, pour comprendre la démarche de notre application, elle est présentée par des diagrammes d'une manière bref et claire.

3. L'objectif de notre application :

Notre objectif dans cette application est de minimiser la pénurie de ressources et d'assurer l'équité entre les parties utilisant les ressources. L'ordonnancement traite du problème de la décision sur laquelle des demandes en suspens doivent être allouées.

4. Présentation d'UML :

UML, c'est l'acronyme anglais pour « Unified Modeling Language ». On le traduit par « Langage de modélisation unifié ». La notation UML est un langage visuel constitué d'un ensemble de schémas, appelés des diagrammes, qui donnent chacun une vision différente du projet à traiter. UML nous fournit donc des diagrammes pour représenter le logiciel à développer : son fonctionnement, sa mise en route, les actions susceptibles d'être effectuées par le logiciel.

4.1 Utilité de l'UML

Fournir aux concepteurs de systèmes, ingénieurs logiciels et développeurs de logiciels des outils pour l'analyse, la conception et la mise en œuvre de systèmes logiciels, ainsi que pour la modélisation de processus métier et d'autres processus similaires. Faire progresser l'industrie en permettant l'interopérabilité des outils de modélisation visuelle orientés objet. Toutefois, pour permettre un échange significatif d'informations de modèles entre outils, il est nécessaire de trouver un accord sur la sémantique et la notation.

5. Implémentation de l'application

Présentation du langage de programmation C# est un langage orienté objet élégant et de type sécurisé qui permet aux développeurs de créer une variété d'applications sécurisées et fiables qui s'exécutent dans l'écosystème .NET. L'écosystème .NET est constitué de toutes les implémentations de .NET, y compris mais non limitées à .net Core et .NET Framework. Le

C# sa syntaxe ressemble beaucoup au langage Java de Sun Microsystems et au C++, vous pouvez l'utiliser pour créer des application clientes Windows, services Web XML, composants distribué, applications client-serveur, application de base de données et bien plus encore.

5.2 Environnement de développement :

L'environnement de développement intégré Visual Studio est une zone de lancement de création que vous pouvez utiliser pour modifier, déboguer et créer du code, puis publier une application. Un environnement de développement intégré (IDE) est un programme riche en fonctionnalités qui peut être utilisé pour de nombreux aspects du développement logiciel. Au-delà de l'éditeur et du débogueur standard fournis par la plupart des IDE, Visual Studio comprend des compilateurs, des outils de complétion de code, des concepteurs graphiques et de nombreuses autres fonctionnalités pour faciliter le processus de développement logiciel.

5.3 Diagramme de cas d'utilisation :

Identification des acteurs : Dans notre application on a un seul acteur humain. Identification des cas d'utilisation : Après avoir défini l'acteur principal de l'application, nous passerons à la détermination de ces cas d'utilisation qui illustrent :

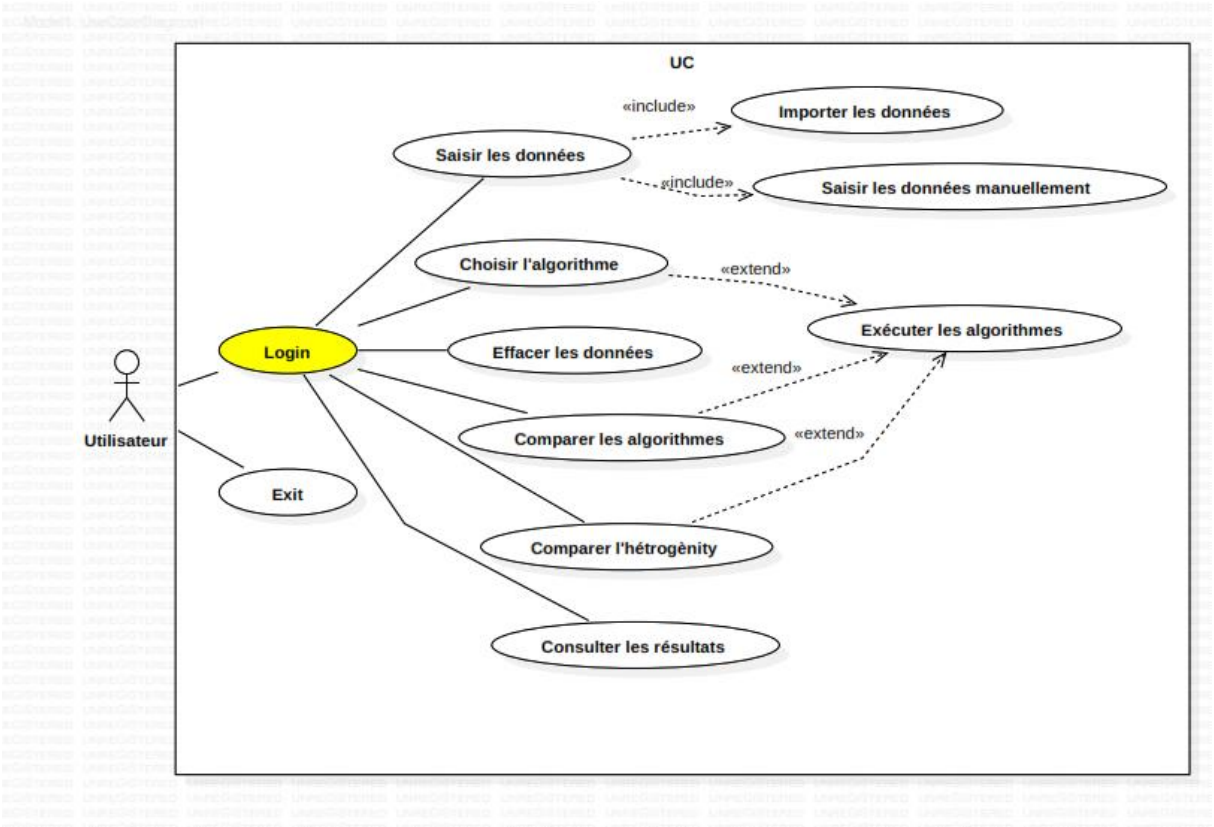


Figure 9 :Diagramme de cas d'utilisation

5.4 Diagramme de séquence :

Cas d'utilisation : Exécuter l'algorithme**.**

Acteur : Utilisateur.

Pré conditions : Saisir les données.

Post conditions : Affichage les résultats d'exécution

Scénario nominal :

L'utilisateur clique sur le bouton «**login** »

L'assistant affiche la fenêtre d'exécution

L'utilisateur clique le bouton « import data file »

L'assistant affiche la matrice de données

L'utilisateur choisit l'algorithme.

L'assistant affiche les résultats

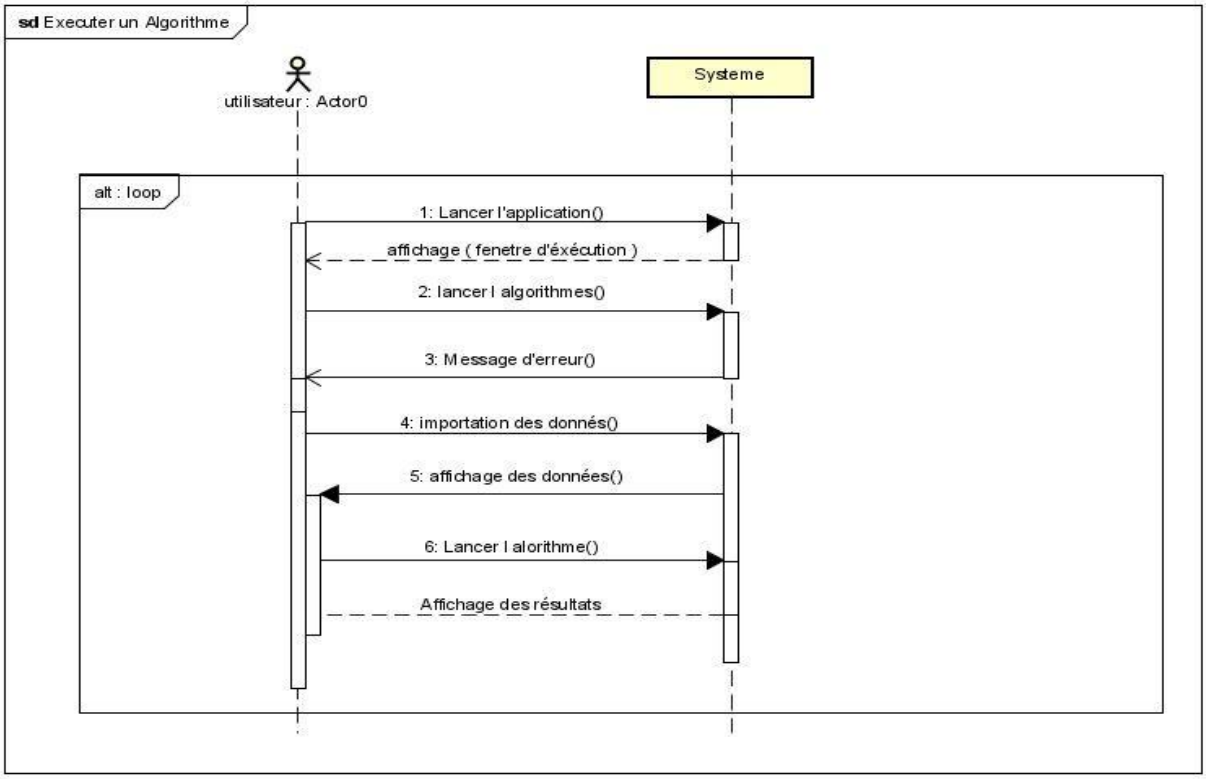


Figure 10 *DDS exécution de l'algorithme*

Alternatives

L'utilisateur peut à tout moment quitter.
L'utilisateur peut modifier ses données.
L'utilisateur peut choisir un autre algorithme.
L'utilisateur peut importer les données en cliquant sur le bouton (import data file).

Cas d'utilisation : Saisir les données.

Acteur : Utilisateur.

Pré conditions : L'utilisateur lance l'application et atteint la fenêtre « Données ».

Post conditions : Affichage de la fenêtre des données.

Scénario nominal :

L'utilisateur clique sur bouton « import data file ».

L'assistant affiche la matrice des données.

2 -ème scenario

L'utilisateur introduit les données

Alternatives :

L'utilisateur peut à tout moment quitter.
L'utilisateur peut modifier ses données. L'utilisateur peut importer les données en

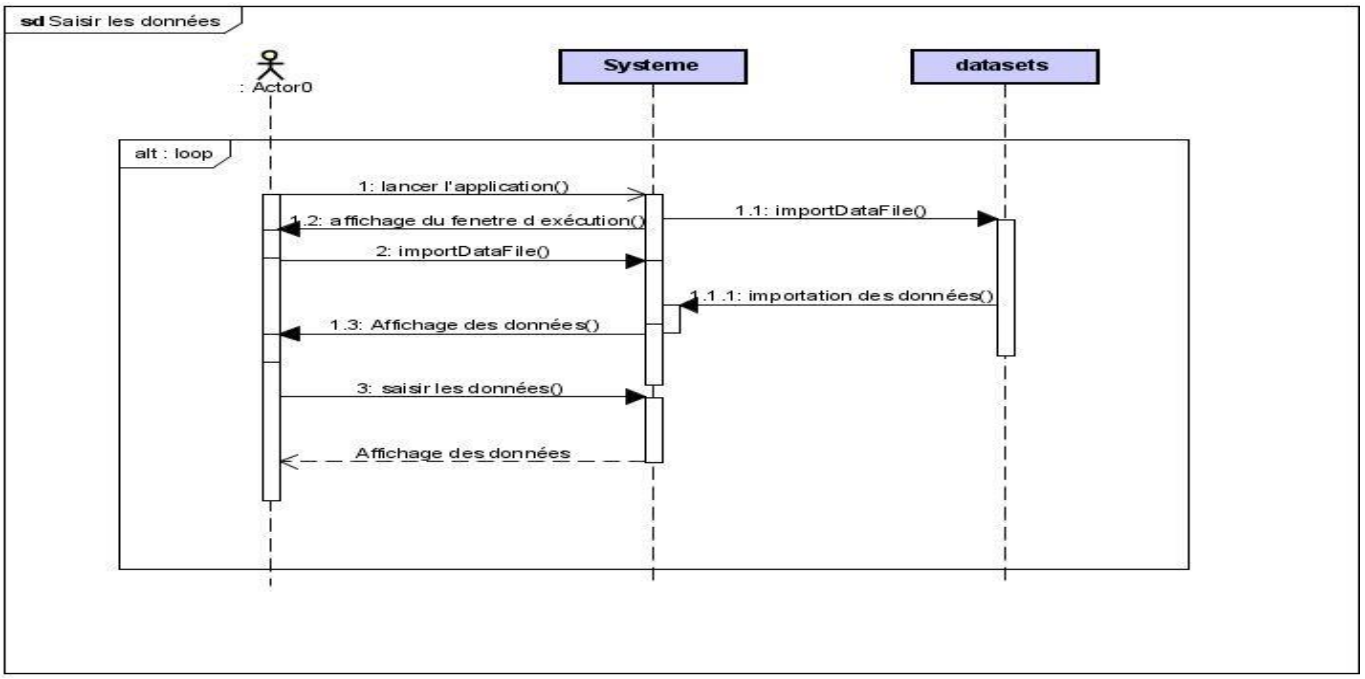


Figure 11 -DDS saisir les données

cliquant sur le bouton (importer fichier).

Cas d'utilisation : Consulter l'aide

Acteur : Utilisateur.

Pré conditions : L'utilisateur lance l'application et atteint la fenêtre « Aide ».

Post conditions : Consulter l'aide

Scénario nominal :

L'utilisateur clique sur l'onglet « about ». L'assistant affiche la fenêtre de l'aide.

L'utilisateur consulte les différentes sous catégories de l'aide.

Alternatives :

L'utilisateur peut à tout moment quitter.

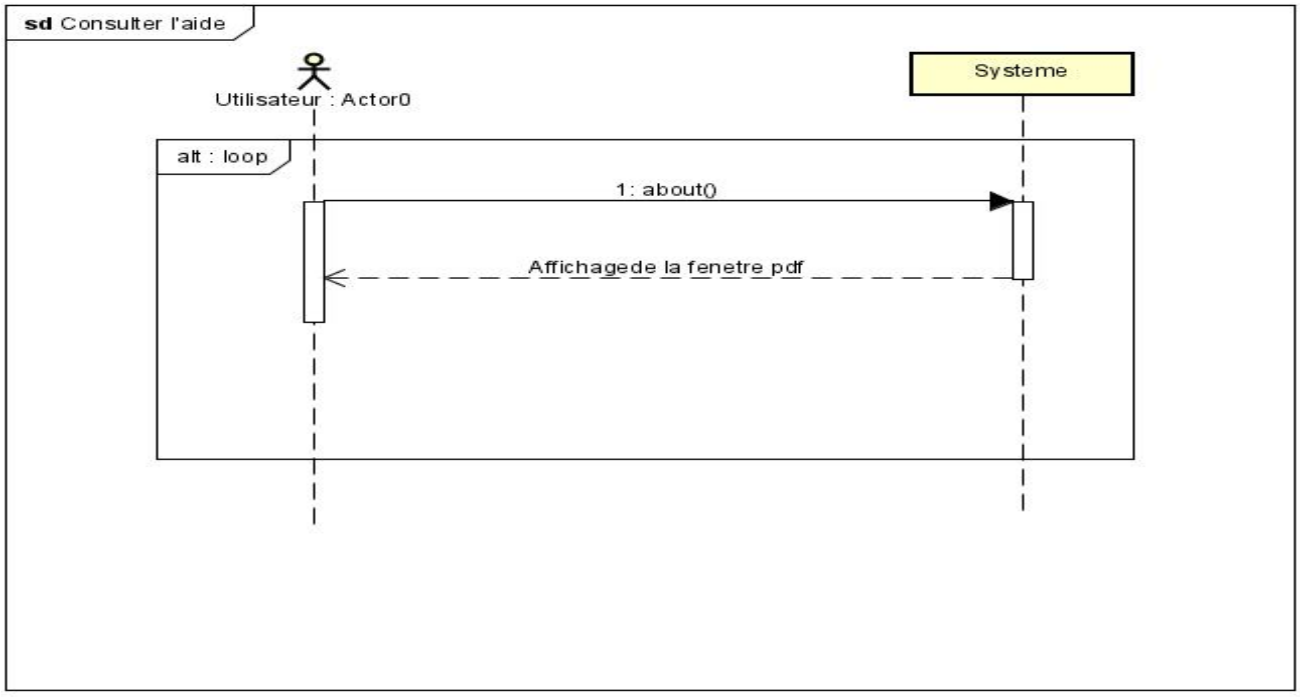


Figure 12: DDS consulter l'aide

Partie 2 :

6. Présentation de l'application :

L'interface Login : représente la fenêtre d'accueil de notre application.

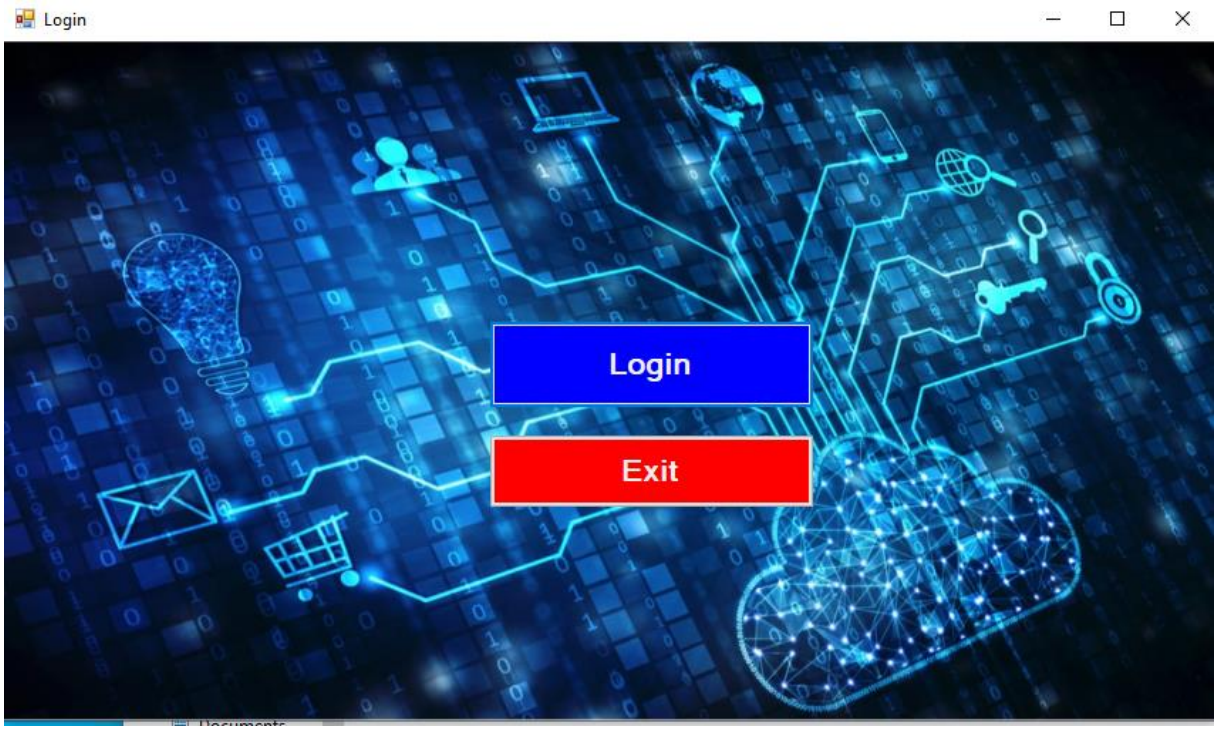


Figure 13- fenêtre d'accueil

Cette fenêtre contient 02 bottons :

Botton Exit : pour quitter l'application.

Botton Login : ouvrir la fenêtre d'ordonnancement suivante :

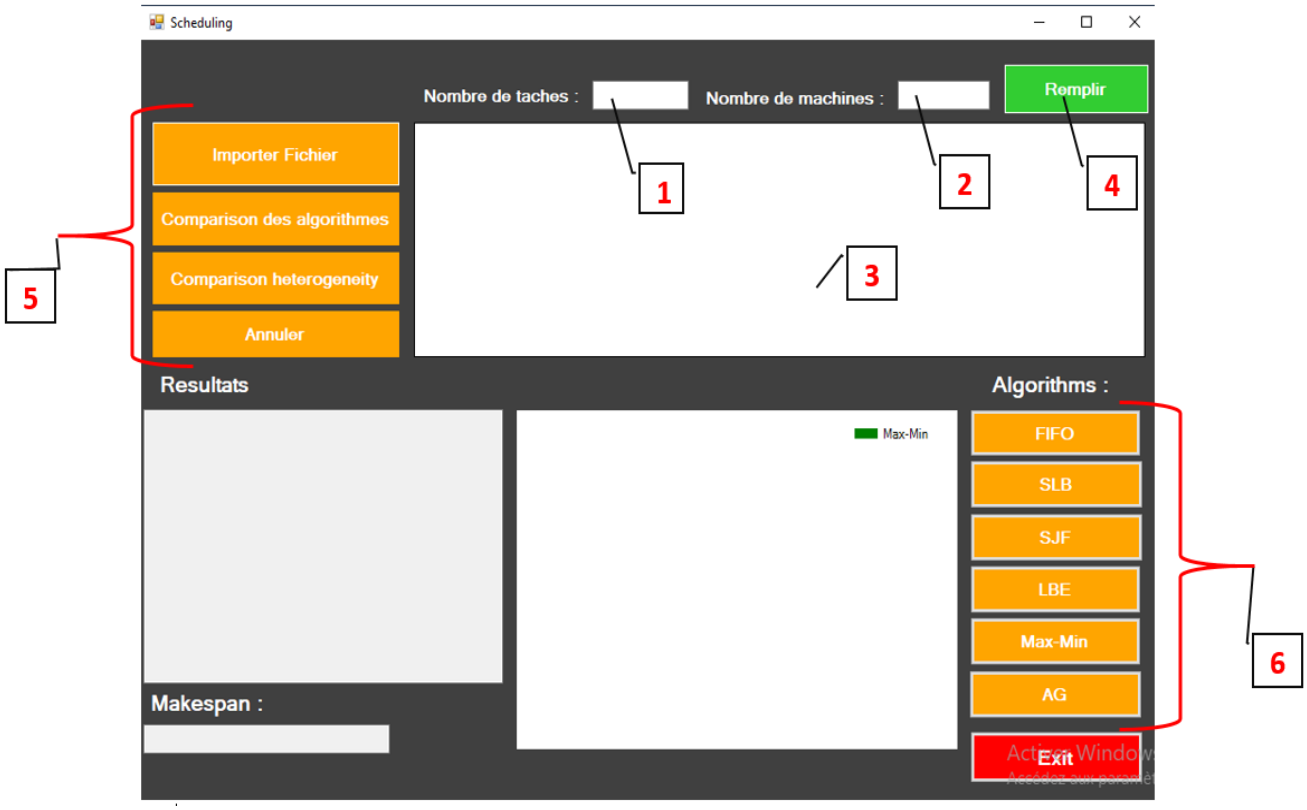


Figure 14 - fenêtre d'ordonnancement.

La page (fenêtre d'ordonnancement) permet à l'utilisateur de saisir ses données, à savoir, le nombre de tâches (1), le nombre de machines (2) et le temps d'exécution de chaque tâche dans chaque machine (3). Cette interface donne le choix à l'utilisateur de saisir les données manuellement par le Botton Remplir (4) ou par l'importation des données à partir d'un fichier en utilisant le Botton Importer Fichier (5). Dans la figure 14 un exemple de remplissage de Tableau manuellement, pour 8 machines et 20 taches. (L'utilisateur doit saisir le nombre de taches et de machines dans les champs Nombre de taches et Nombre de machines.

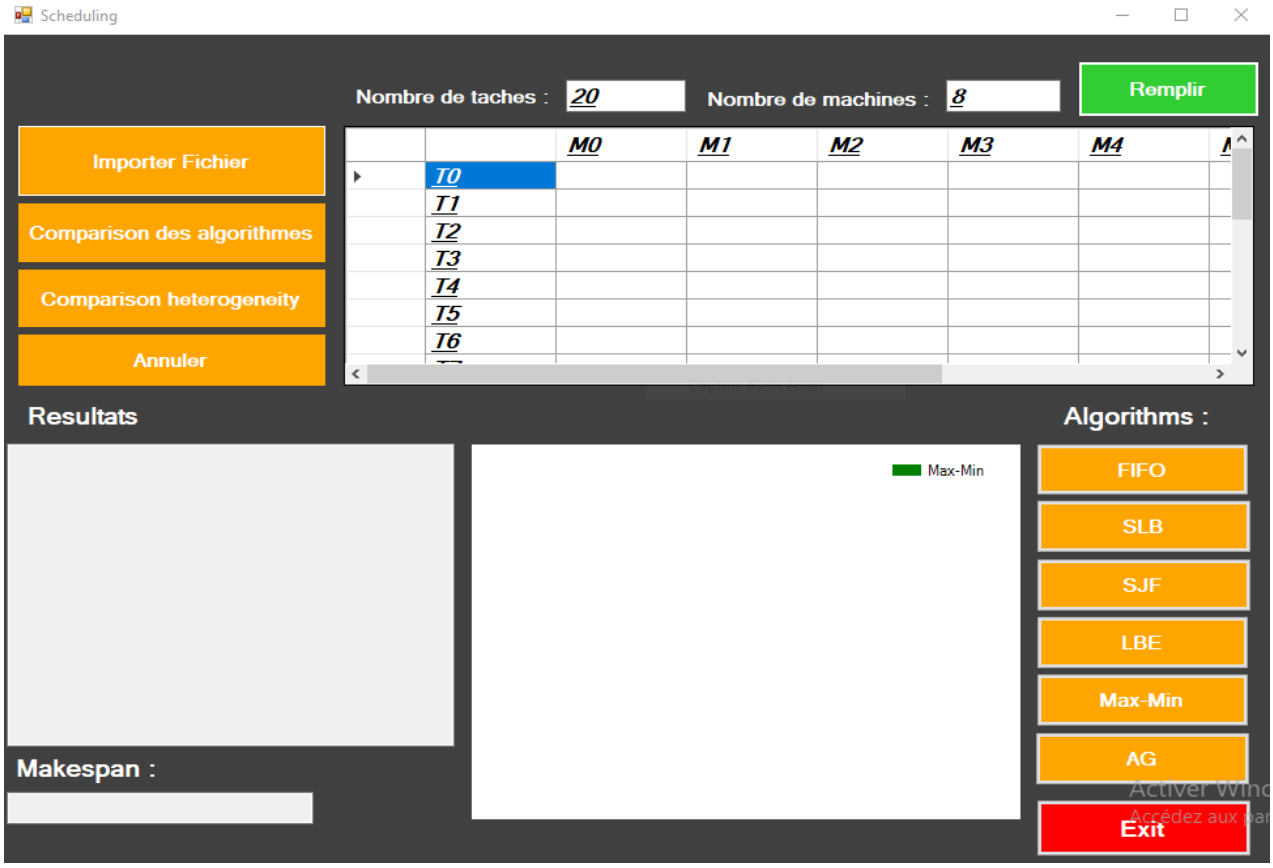


Figure 15- remplissage de Tableau

L'interface ordonnancement des taches est composée d'un menu qui permet l'accès aux différents algorithmes à exécuter (6).

Apres l'exécution d'un algorithme, les résultats sont affiches comme

l'exemple montré ci-dessous :

6.1-L'exécution de l'algorithme génétique:

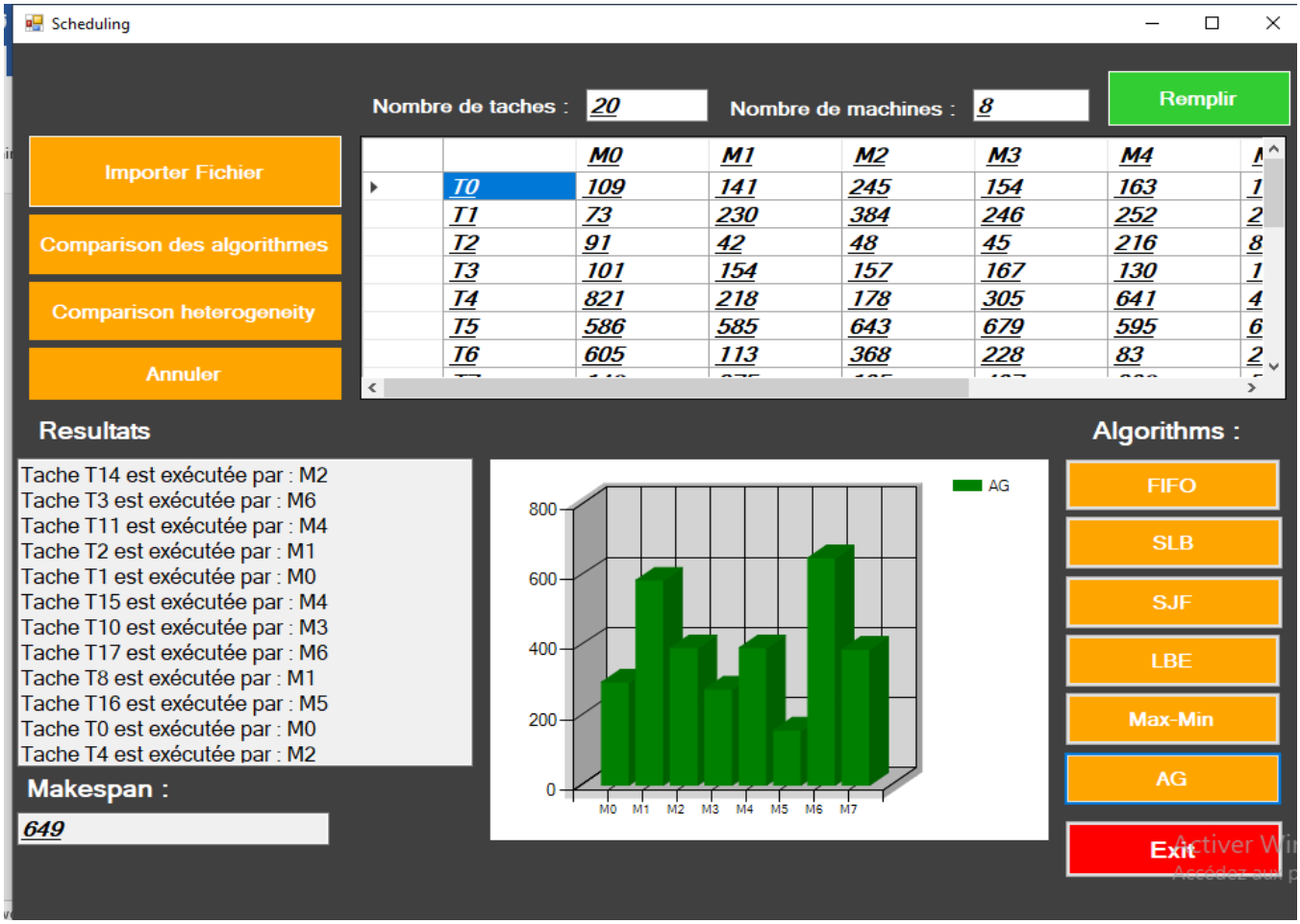


Figure 16- L'exécution de l'algorithme Génétique.

6.2 -L'exécution de l'algorithme FIFO (First in First out):

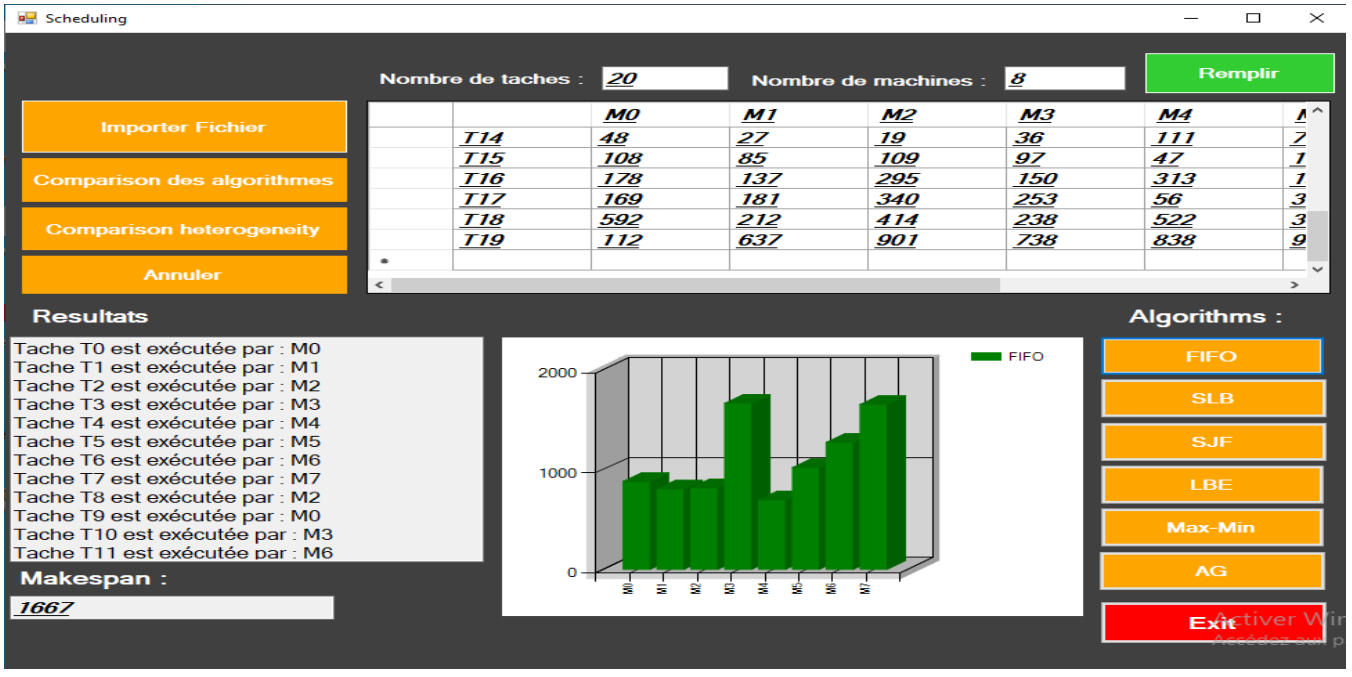
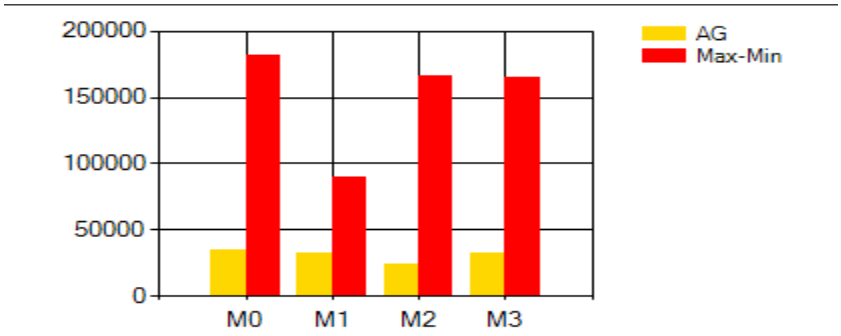


Figure 17- L'exécution de l'algorithme FIFO.

6.3 Comparaison entre AG et MAXMIN :



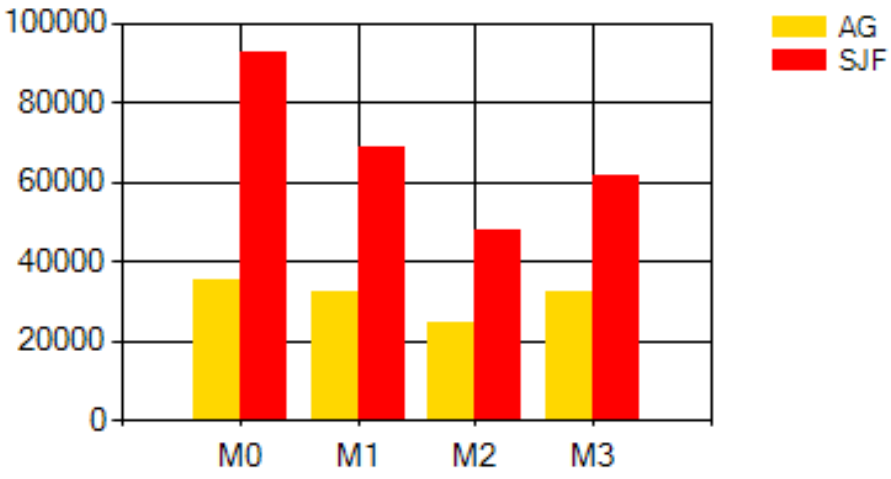
Makspan :

AG	35199

Max-Min	181156

Dans ce graphe, on remarque bien que l'algorithme génétique est bien meilleur que l'algorithme maxmin selon le critère de 'Makespan'.

6.4 Comparaison entre AG et L'algorithme SJF :



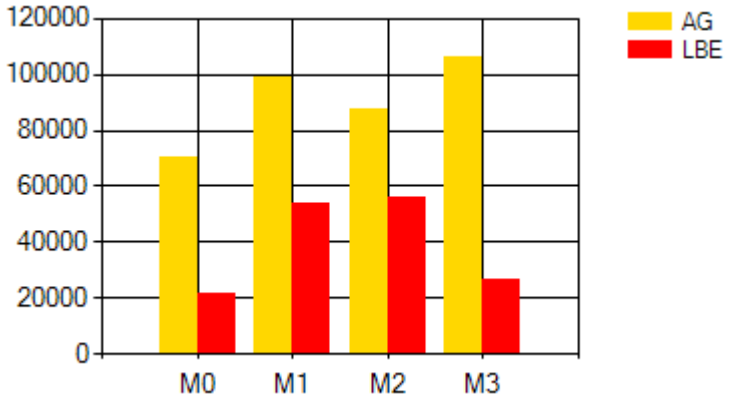
Dans ce graphe, on remarque bien que l'algorithme génétique est bien meilleur que l'algorithme SJF selon le critère de 'Makespan'.

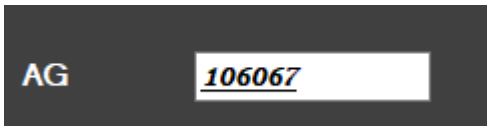
Makspan :

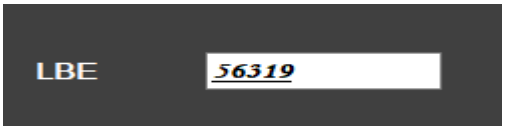
AG	35199

SJF	92413

6.5 Comparaison entre AG et L'algorithme LBE :







Dans ce graphe, on remarque bien que l'algorithme LBE est bien meilleur que l'algorithme génétique selon le critère de 'Makespan'.

7- Fenêtre des résultats (Comparisons of algorithms) :

Cette fenêtre permet d'afficher les résultats des algorithmes en affichant le makespan obtenu pour chaque algorithme.

Cette fenêtre permet de comparer les algorithmes développés et estimer le meilleur algorithme qui en calculant le temps d'exécution optimal (makespan).

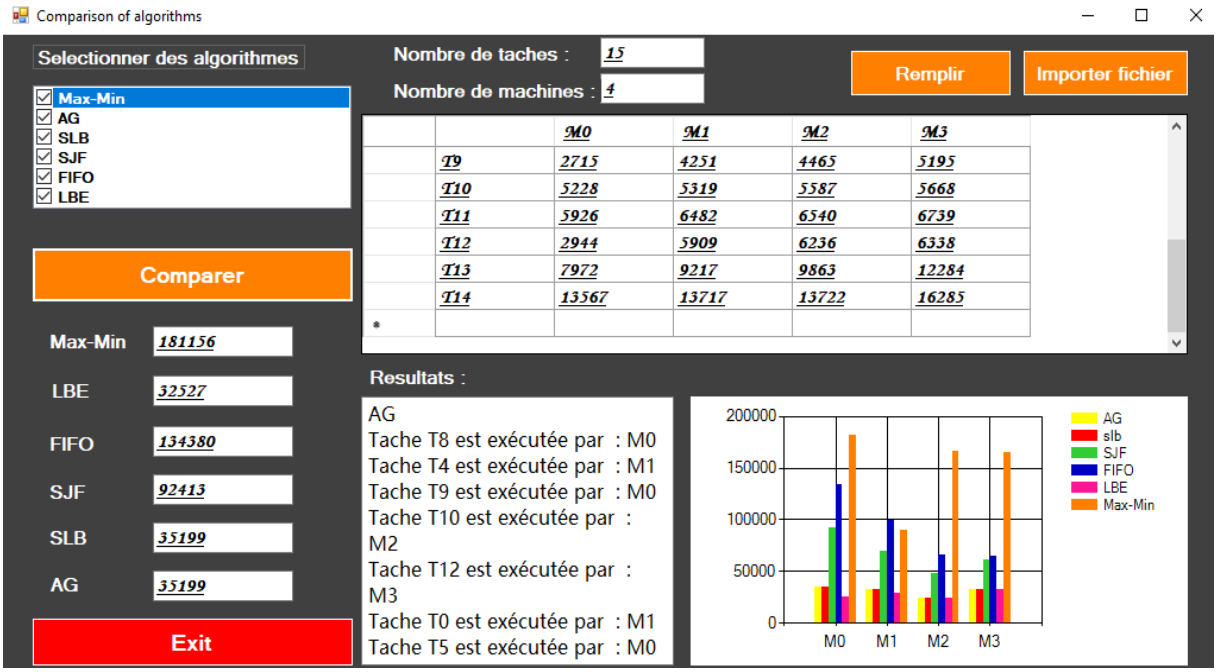


Figure 18 - Comparaison entre les algorithme.

8- Evaluation des algorithmes proposés

Afin de valider notre stratégie d'ordonnancement de tâches sur environnements de Simulation, nous avons opté pour une évaluation et une comparaison avec les approches proposées. Pour se faire, des séries d'expériences ont été réalisées.

Les algorithmes proposés ont été appliqués à des matrices avec 12 différents types, jusqu'à 16 machines hétérogènes et jusqu'à 512 tâches hétérogènes Ces matrices ont une des propriétés suivantes :

Hétérogénéité des tâches : L'hétérogénéité des tâches est définie comme : lo : low et hi : high.

Hétérogénéité des machines : L'hétérogénéité machine est définie comme : lo et hi.L'unité de temps pour tous ces matrices est second.

Les algorithmes sont exécutés sur 03 base de données (THMH / THML / TLML) ; en cliquant sur bouton **Exécute**

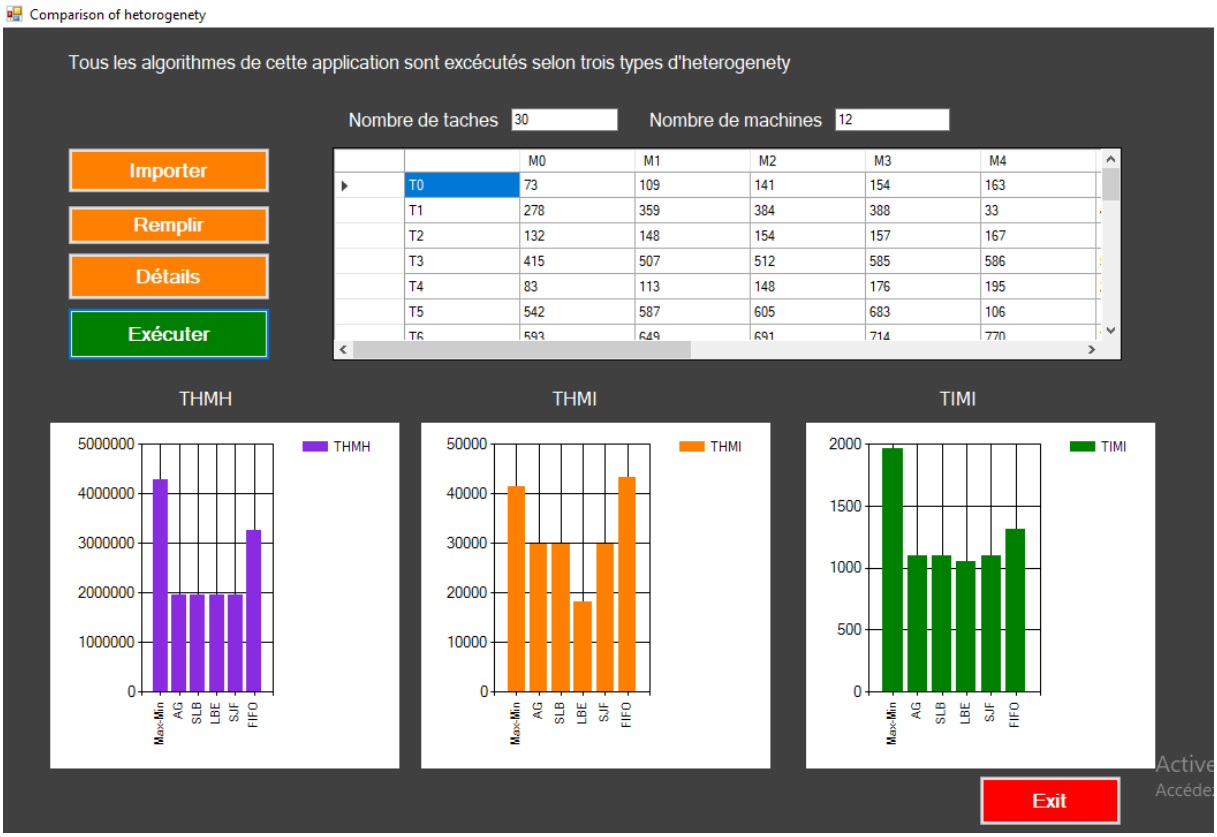


Figure 19: comparaison de l'hétérogénéité.

Botton **details** pour afficher l'exécution et l'affectation des taches de chaque algorithme sur les 03 bases (THML, THMH, TLML), ***la figure 20*** montre un exemple d''exécution pour l'algorithme Génétique.

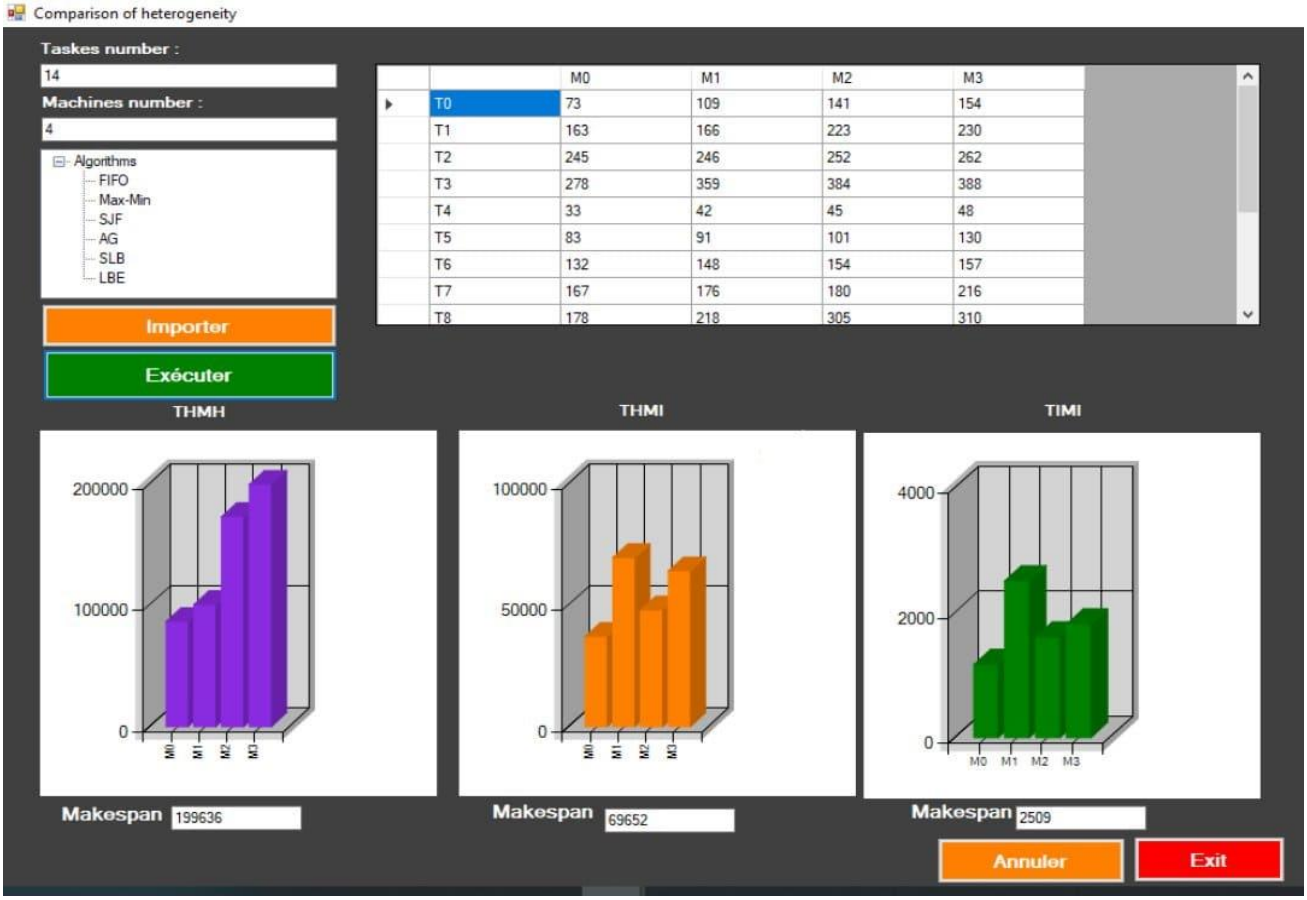


Figure 20: détaille de l'exécution de l'algorithme AG.

Conclusion Génerale :

Dans ce présent mémoire, nous intéressons au problème d'ordonnancement des tâches indépendantes sur environnements hétérogènes de calcul. Le problème d'attribuer de façon optimale (ordonnancer) les tâches de calcul aux machines dans un environnement hétérogène de calcul distribué s'est considéré comme un problème NP-complet qui nécessite le développement des techniques heuristiques.

Notre travail fait appel à une démarche des simulations et de test afin d'évaluer les performances des algorithmes appliqués à un système d'ordonnancement des tâches indépendantes dans un environnement hétérogène, pour la résolution du problème étudié.

Dans ce travail, on a testé plusieurs algorithmes afin d'optimiser le temps d'exécution totale des taches 'Makespan'. Le défi c'était d'attribuer chaque tache à sa machine candidate favorable en conservant la notion d'équilibrage.

La simplicité des méthodes utilisées soutient l'hypothèse selon laquelle les algorithmes génétiques peuvent fournir une solution quasi optimale très flexible et conviviale au problème général de l'ordonnancement des tâches.

Les algorithmes génétiques sont performants pour résoudre les problèmes d'optimisation. Les résultats de l'expérience dans ce travail montrent clairement que l'approche proposée est capable de trouver une solution optimisée. La simulation montre que l'AG donne de bons résultats pour un nombre limité de tache et de machine, on peut utiliser les AGs comme des algorithmes d'initialisation pour le méta heuristiques d'équilibrage de charge SLB et LBE.

Généralement l'efficacité d'un AG dépend souvent de la nature du problème d'optimisation. Selon les cas, le choix des paramètres et des opérateurs sera souvent critique, mais aucune théorie générale ne permet de connaitre avec certitude la bonne para-métrisation. Il faudra faire plusieurs expériences pour s'en approcher.

Références des bibliographiques

[01] Z. BOUAFIA, Algorithmes d'ordonnancement des taches dans un environnement Cloud, Revue Méditerranéenne des Télécommunications Vol. 5, N° 2, June 2015 .

[02] L. Baccouche. « Un Mécanisme d'Ordonnancement Distribué de Tâches Temps Réel. Thèse de doctorat, l'institut national polytechnique de Grenoble ». In: (1995) (cf. p. 14, 17).

[03] Ahmed Gara-Ali. Ordonnancement de tâches et de périodes d'indisponibilité de durée variable. Autre. Université Grenoble Alpes, 2016. Français. ffNNT : 2016GREAI027ff. fftel-01492812f.

[04] Xiaojun Ye. Modélisation et simulation des systèmes de production : une approche orientée-objets. Modélisation et simulation. INSA de Lyon, 1994. Français. ffNNT : 1994ISAL0049ff. fftel-00821121.

[05]https://waytolearnx.com/2018/07/difference-entre-ordonnancement-preemptif-etnon-preemptif.html consulte le : 15-05-202

[**06]** Richard WOLSKI Francine BERMAN. « Adaptive Computing on the Grid Using AppLeS. IEEE Transactions on Parallel and Distributed System, » in : (2002) (cf. p. 15).

[07] J. Carlier, A. Moukrim ; *Problèmes d'ordonnancement à contraintes de ressources et applications* ; HeuDiaSyC UMR CNRS 6599 UTC, Compiègne.

[08] A. BEN HMIDA SAKLY *; Méthodes arborescentes pour la résolution de problèmes d'ordonnancement flexible ;* thèsedoctorat *; 2009 ;* Université De Toulouse.

[09] Raphael Yende. COURS DE METHODES DE CONDUITE DES PROJETS INFORMATIQUES. Licence. Congo-Kinshasa. 2019, 90p. ffcel-02004689f

[10] Jean-Noel Martin. No Free Lunch et recherche de solutions structurantes en coloration. Modélisation et simulation. Université de Technologie de Belfort-Montbeliard, 2010. Français. ffNNT : 2010BELF0147ff. fftel-00607481f

[11] Jean-Charles Boisson. Modélisation et résolution par métaheuristiques coopératives : de l'atome à la séquence protéique. Recherche opérationnelle [cs.RO]. Université Lille 1, 2008. Français. ffNNT : 2008LIL10154ff. fftel-00842054f

[12] https://www.gerad.ca/~alainh/Metaheuristiques.pdf

[13] Hélène RENARD, Équilibrage de charge et redistribution de données sur plates-formes hétérogènes, ÉCOLE NORMALE SUPÉRIEURE DE LYON,2005

[14] BElabbas yagoubi équilibrage de charge dans les grilles de calcul, université d'oran,2007

[15] BElabbas yagoubi équilibrage de charge dans les grilles de calcul, université d'oran,2007

[16] Nicolas Durand. Algorithmes Génétiques et autres méthodes d'optimisation appliqués à la gestion de trafic aérien. Optimisation et contrôle [math.OC]. INPT, 2004. fftel-01293722f

[17] Elham Mohamed Jeanne Rachelle Ngo Ntep Ebanda, La méthode kangourou dans la prévention et le traitement de la douleur chez le prématuré en néonatologie, 09 juillet 2015

[18] Holland, Adaptation in Natural and Artificiel Systems. University of Michigan Press: Ann Arbor, 1975.

[19] BENGHANI A, les algorithme génétique, Université ibn khaldoune Tiaret ,2020

[20] Said Bourazza. Variantes d'algorithmes génétiques appliquéees aux problèmes d'ordonnancement. Mathématiques [math]. Université du Havre, 2006. Français. fftel-00126292v2

[21]Christelle Reynès. Etude des Algorithmes génétiques et application aux données de protéomique. Sciences du Vivant [q-bio]. Université Montpellier I, 2007. Français. fftel-00268927f

[22] Nicolas Durand. Algorithmes Génétiques et autres méthodes d'optimisation appliqués à la gestion de trafic aérien. Optimisation et contrôle [math.OC]. INPT, 2004. fftel-01293722f.

[23] http://igm.univ-mlv.fr/~dr/XPOSE2013/tleroux_genetic_algorithm/fonctionnement.html

[24] http://umoncton.ca/umcm-cormier_gabriel/SystemesIntelligents/GIND5439_ Chapitre7.pdf

[25] Said Bourazza. Variantes d'algorithmes génétiques appliquéees aux problèmes d'ordonnancement. Mathématiques [math]. Université du Havre, 2006. Français. fftel-00126292v2