وزارةالتعليم العالي و البحث العلمي

BADJI MOKHTAR – ANNABA UNIVERSITY

UNIVERSITE BADJI MOKHTAR –ANNABA

جامعة باجي مختار- عنابة-

Faculté des Sciences de l'Ingéniorat

Département d'informatique

Année **:** 2014

MEMOIRE

Présenté en vue de l'obtention

du diplôme de MAGISTER en Informatique

Algorithmes évolutionnaires pour la génération distribuée des espaces d'états dans un environnement dynamiques

Option :

Informatique Embarquée (INEM)

Par

DJEHAICHIA Brahim

DIRECTEUR DE MEMOIRE : SAIDOUNI Djamel Eddine, Pr., Univ Constantine 2

DEVANT LE JURY

PRESIDENT : BOUDOUR Rachid, Pr., Univ Badji Mokhtar Annaba

EXAMINATEURS : KIMOUR Mohamed Tahar, Pr., Univ Badji Mokhtar Annaba
GHANEMI Salim, Pr., Univ Badji Mokhtar Annaba

A MES PARENTS

A TOUTE MA FAMILLE

A TOUS MES AMIS

REMERCIEMENTS

Tout d'abord je remercie DIEU pour m'avoir donné la force et le courage de finaliser ce travail, pour tout ce que j'avais reçu jusqu'à présent.

Je tiens à exprimer toute ma reconnaissance et ma gratitude à mon directeur de mémoire, Mr SAIDOUNI Djamel Eddine, professeur à l'université de Constantine 2. Je le remercie pour ses conseils et ses encouragements. Merci de m'avoir proposé ce sujet et accueilli dans votre laboratoire, de m'avoir encadré, orienté et aidé.

Je ne saurais jamais comment remercier tous les enseignants qui ont assuré notre encadrement durant l'année théorique de magister. Merci pour cette année magnifique.

Mes respects et mes remerciements aux membres du jury Mr BOUDOUR Rachid, professeur à l'université Badji Mokhtar Annaba pour m'avoir fait l'immense honneur de présider le jury, Je tiens à remercier Mr KIMOUR Mohamed Tahar, professeur à l'université Badji Mokhtar Annaba et Mr GHANEMI Salim, professeur à l'université Badji Mokhtar Annaba pour avoir accepté d'évaluer mon travail.

Je voudrais remercier aussi mes collègues avec qui j'ai partagé des moments inoubliables. Merci à tous les membres du laboratoire MISC particulièrement, Meriem Bensouyad, Nousseiba Guidoum, Souad Guellati pour leur précieuse aide.

Enfin, je remercie ma famille qui a toujours encouragé et soutenu toutes mes idées et mes projets.

ملخص

التحقق الشكلي من الأنظمة هو حاجة هامة، و خصوصا عندما يتعلق الأمر بالأنظمة الحساسة. واحدة من المشاكل المواجهة خلال التحقق من هذه الأنظمة هي مشكلة انفجار تألفي لفضاء الحالة، الحل لهذه المشكلة هو استعمال الحوسبة الموزعة. هذا يفترض أن الحساب يتم تنفيذه على الشبكة. توجد حلول كثيرة في الأدب ولكن هذه الحلول لا تأخذ بعين الاعتبار ديناميكية النظم الموزعة لذلك اقترحنا خوارزميات تطورية لتوزيع فضاء الحالة في بيئة ديناميكية. أن البحث عن التوزيع الأمثل يأخذ بعين الاعتبار هدفين متناقضين: موازنة الحمولة و الحد من الأقواس بين المواقع. وهذا لا يزال يمثل مشكلة صعبة. ومن هنا كانت الحاجة إلى البحث عن حلول مستوحاة من الطبيعة.

كلمات مرشدة: تحمل الأخطاء، الخوارزميات التطورية، تلوين قوي صارم للمخطط، توزيع المخططات.

RESUME

La vérification formelle des systèmes est un besoin important, en particulier lorsqu'il s'agit de systèmes critiques. L'un des problèmes rencontré lors de la vérification de ces systèmes est le problème de l'explosion combinatoire de l'espace d'états. Une solution pour pallier ce problème consiste en calcul distribué de cet espace. Ceci suppose que le calcul s'effectue sur un réseau. Des solutions existent dans la littérature mais qui ne prennent pas en considération la dynamicité du système distribué. Pour cela nous envisageons la proposition d'algorithmes évolutionnaires pour la génération d'espace d'états dans un environnement dynamique. En effet la recherche d'une distribution optimale prenant en compte deux objectifs contradictoires, à savoir l'équilibrage de charge et la réduction des arcs intersites, reste un problème difficile. D'où la nécessité de rechercher des solutions inspirées de la nature.

Mots clés : Tolérance aux fautes, Algorithmes évolutionnaires, coloration forte stricte, Distribution des graphes.

ABSTRACT

Formal verification of systems is an important need, especially in the case of critical systems. One of the problems encountered during the verification of these systems is the problem of combinatorial explosion of the state space. A solution to solve this problem consists in using the distributed computing. This assumes that the calculation is performed on a network. A number of solutions exist in the literature, but these solutions do not take into account the dynamicity of the distributed system. For that we proposed an evolutionary algorithm for the generation of state space in a dynamic environment. In fact, the search for an optimal distribution taking into accounts two contradictory objectives: load balancing and reduction of arcs inter-site. Remains a difficult problem. Hence the need to seek solutions inspired by nature.

Keywords: Fault tolerance, Evolutionary algorithms, Strict Strong Coloring, graph distribution.

TABLE DES MATIERES

Remerciements 2

ملخص 3

Résumé 4

Abstract 5

Table des Matières 6

Table des Illustrations 9

Listes Des Tableaux 11

Introduction Générale 12

Chapitre 1 Tolérance aux fautes 15

1.1 Introduction 15

1.2 Sûreté de fonctionnement 16

1.2.1 Attributs de la sûreté de fonctionnement 16

1.2.2 Difficultés de la sûreté de fonctionnement 17

1.2.3 Moyens d'assurer la sûreté de fonctionnement 18

1.3 Tolérance aux fautes 19

1.3.1 Détection d'erreur 20

1.3.2 Rétablissement du système 20

1.3.2.1 Traitement d'erreur 21

1.3.2.2 Traitement de faute 21

1.4 Tolérance aux fautes dans les systèmes répartis 22

1.4.1 Redondance spatiale et temporelle : réplication 22

1.4.2 Redondance informationnelle : mémoire stable 23

1.4.2.1 Reprise par sauvegarde 24

1.4.2.2 Reprise par journalisation 25

1.5 Conclusion 26

Chapitre 2 algorithmes évolutionnaires 27

2.1 Introduction 27

2.2 Algorithmes évolutionnaires 27

2.2.1 Terminologie et notations 28

2.2.2 Principe général : 28
2.3 Algorithmes génétiques 30
2.3.1 Représentation 31
2.3.2 Evaluation : fitness 32
2.3.3 Sélection 32
2.3.4 Croisement 33
2.3.5 Mutation 34
2.3.6 Opérateur de remplacement 35
2.3.7 Critère d'arrêt 35
2.4 Stratégies d'évolution 35
2.4.1 Représentation 36
2.4.2 Evolution 36
2.4.3 Sélection 36
2.4.4 Croisement 37
2.4.5 Mutation 37
2.4.6 Auto-adaptation 37
2.5 Programmation génétique 38
2.6 Programmation Evolutive 39
2.7 Conclusion 40
Chapitre 3 Graphes et coloration forte strict 41
3.1 Introduction 41
3.2 Initiation aux graphes 42
3.2.1 Définitions 42
3.2.2 Différents types de graphes 43
3.2.3 Les prédécesseurs, successeurs d'un sommet: 45
3.3 Coloration de graphe 47
3.3.1 Définition: 48
3.4 Coloration forte stricte 49
3.4.1 Coloration forte 49
3.4.2 Coloration forte stricte 49
3.4.3 Quelques propriétés 50
3.5 Algorithme de coloration forte stricte généralisé pour les graphes 51
3.5.1 L'algorithme 52
3.5.2 L'explication 53

3.6 Algorithme de distribution du graphe basée sur la coloration forte stricte 55
3.6.1 L'initialisation 56
3.6.2 L'optimisation 57
3.7 Algorithme évolutionnaire basée sur la coloration forte stricte 60
3.7.1 Représentation de coloration forte stricte 61
3.7.2 Opérateur d'initialisation 62
3.7.3 Opérateur de croisement 64
3.7.4 Opérateur de Correction 65
3.8 Conclusion 67
Chapitre 4 Contributions 68
4.1 Introduction 68
4 .2 Distribution de l'algorithme évolutionnaire basée sur la coloration forte stricte 68
4.2.1 Présentation de l'approche 69
4.3 Positionnement du problème 71
4.4 La distribution à la volée 71
4.4.1 La Génération. 72
4.4.2 La répartition. 73
4.4.3 Envoyer et Recevoir des parties. 73
4.5 Mécanisme tolérant aux pannes 74
4.6 Intégration de nouvelles machines 75
4.7 Détection de la terminaison 77
4.7.1 Les canaux vides. 77
4.7.2 La collection des informations. 78
4.8 Mise en œuvre 79
4.8.1 Langage Matlab. 79
4.8.2 Matlab pour les algorithmes évolutionnaires. 80
4.8.3 Matlab Distributed Computing Server. 81
4.9 Conclusion 83
conclusion générale 84
Références 85

TABLE DES ILLUSTRATIONS

Figure 1.1: L'arbre de la sûreté de fonctionnement [01] _____ 15
Figure 1.2 : La chaine fondamentale des entraves _____ 17
Figure 1.3: Techniques de tolérance aux fautes _____ 19
Figure 1.4 : Les techniques de tolérance aux fautes dans les systèmes répartis [09] _____ 25
Figure 2.1Le schéma général d'un algorithme évolutionnaire en pseudocode_____ 29
Figure 2.2 Le schéma général d'un algorithme évolutionnaire en diagramme_____ 29
Figure 2.3 branches des algorithmes évolutionnaires (EA). _____ 30
Figure 2.4 codage binaire d'un chromosome _____ 31
Figure 2.5 codage des variables réelles_____ 31
Figure 2.6 Méthode de sélection par roulette _____ 32
Figure 2.7 Méthode de sélection par tournoi _____ 33
Figure 2.8 Croisement en 1-point de deux chromosomes _____ 34
Figure 2.9 Croisement en 2-point de deux chromosomes _____ 34
Figure 2.10 Opération de mutation. _____ 34
Figure 2.11 Opération de mutation [19]. _____ 38
Figure 2.12 Exemple de croisement en programmation génétique _____ 39
Figure 3.1 les ponts de Königsberg et le graphe associé _____ 42
Figure 3.2 Exemple d'un graphe _____ 43
Figure 3.3 Exemple de multi-graphe _____ 44
Figure 3.4 Exemple de simple _____ 44
Figure 3.5 le sous-graphe du graphe de la figure3.2 induit par l'ensemble {b, c, d, g, f}. _____ 45
Figure 3.6 le graphe partiel du graphe de la figure3.2 défini par l'ensemble d'arêtes {(a,d), (b,d), (d,e), (e,f), (f,d)}. _____ 45
Figure 3.7 La matrice d'adjacence de G_____ 46
Figure 3.8 exemple de graphe G et sa liste d'adjacence associée _____ 47
Figure 3.9 Exemple d'un graphe coloré_____ 48
Figure 3.10 (a) un graphe G, (b) une coloration forte de graphe G, (c), (d) deux colorations forte stricte de graphe G [25]. _____ 50
Figure 3.11 coloration forte stricte d'un graphe G avec DCS = {1,2} et NDCS = {3} [25]. _____ 51

Figure 3.12 le résultat de la deuxième étape. 54
Figure 3.13 deux graphes non distribué [27]. 57
Figure 3.14 les deux graphes après l'initialisation [27]. 57
Figure 3.15 Le résultat du processus de regroupement [27]. 58
Figure 3.16 Le résultat du processus de division [27]. 59
Figure 3.17 La coloration propres et fortes stricte du graphe G [37]. 62
Figure 4.1 modèle client/serveur 70
Figure 4.2 la vérification de chaque canal [31]. 78

LISTES DES TABLEAUX

Tableau 3.1 Représentation des colorations propres et fortes stricte [37]. _____ 61

Tableau 3.2 Exemple illustratif de l'opérateur de croisement appliqué sur le graphe de la figure 4.7 [37]._____ 66

Table 4.1 les options d'algorithme génétique standard proposées par Global Optimization Toolbox [35]._____ 80

INTRODUCTION GENERALE

A l'origine, les Systèmes informatiques étaient utilisés uniquement pour des besoins spécifiques. Le progrès technique important ainsi que l'utilisation combinée de moyens informatiques, électroniques et de procédés de télécommunication ont entraîné une évolution de cette exploitation dans des nombreux domaines. De tels systèmes embarqués dans les transports aériens, dans le domaine de l'énergie de télécommunications ou encore le paiement électronique, la moindre erreur d'exécution dans de tels systèmes, dits critiques, peut engendrer des conséquences dramatiques humain et matériel. Plusieurs dégâts liés à des défaillances de logiciels supposes corrects sont déjà arrivés :

- le problème du Therac-25 qui a causé la mort de deux personnes et quatre autres ont été irradiés.
- l'explosion du premier vol d'Ariane 5, le 4 Juin 1996. 500 millions de dollars de pertes matérielles.

Le besoin de démontrer la validité des programmes informatiques ou du matériel électronique en utilisant des techniques de raisonnement rigoureux, à l'aide de logique mathématique au lieu d'utiliser les tests traditionnels (c'est le domaine de la vérification formelle) n'est plus à reconnaître lorsque l'on parle des systèmes critiques.

La vérification formelle consiste alors à prouver que l'ensemble des fonctionnements d'un modèle mathématique du système développé satisfait toutes les spécifications en utilisant des méthodes formelles mathématiques. La conception du modèle mathématique représentant le système, se fait principalement à l'aide d'un langage formel. L'une des principales techniques de vérification est : Les vérificateurs par modèles. Elle est réalisée par une exploration de l'ensemble des états accessibles (espace d'états).

Des branches comme la physique, la biologie ou la météorologie réalisent des simulations numériques de plus en plus complexes, ce qui rend L'exploration d'espace d'états plus difficile vu la taille du modèle du système qui augmenté de manière exponentielle. Il s'agit du problème de l'explosion d'états, principale limitation de vérification basée sur les modèles.

La tendance actuelle pour lutter contre le problème de l'explosion de l'espace d'états consiste à utiliser les systèmes distribués. Un système distribué est un ensemble de machines autonomes géographiquement éloignés et s'interconnectant par un réseau de communication, Chaque ordinateur exécute des séquences de calculs, par exemple exploré une partie d'espace d'états. L'un des principaux avantages de systèmes distribués est l'augmentation de la puissance de calcul et de stockage.

Les performances des systèmes distribués de vérification basées sur l'espace d'états dépendent fortement de l'équilibre de charge des différents processus. Il faut donc choisirune fonction de répartition qui permet de garantir un équilibre de charge. Les trois aspects principaux de l'équilibre de charge sont :

- **Equilibrage spatial :** L'espace mémoire pour stocker le sous ensemble d'états doit être environ le même par chaque processus
- **Localité :** Le nombre d'arcs reliant des états appartenant à deux propriétaires différents, appelés arcs traversants (cross arcs), doit être aussi réduit que possible.
- **Equilibrage temporel :** il faut aussi s'assurer que les processus soient les moins possibles inactifs.

Un autre défi de l'utilisation des systèmes distribués et qui revient à la nature même de ces systèmes est qu'ils sont à la fois hétérogènes et dynamiques, c'est-à-dire que les machines utilisés n'ont pas les même capacités de calcul et de stockage et leur nombres peut varier à tout moment durant l'exécution d'une application.

Notre travail avait un objectif à deux volets :

1. Equilibrer la charge : nous proposons pour cela, un algorithme évolutionnaire pour la génération distribuée d'espace, nous nous focalisons dans cet algorithme sur le principe de la localité pour équilibrer la charge et pour trouver une bonne répartition de l'espace d'états. La minimisation du nombre d'arcs traversants s'effectue grâce à la coloration forte stricte, en mettant les états les plus connectés dans le même site.

2. Lutter contre le problème causé par la dynamicité des systèmes distribués : une machine qui quitte le système, produit des défaillances dans le fonctionnement du système global. Pour résoudre ce problème, nous avons proposé un mécanisme tolérant aux pannes basé sur le repris qui assure le bon fonctionnement du système même lorsque l'une de ces machines ne fonctionne plus.

La suite de ce document est organisée comme suite :

- Le chapitre 1 est une introduction relative à la sûreté de fonctionnement. Il présente la tolérance aux pannes comme un moyen de sûreté de fonctionnement et il se termine par une présentation de la tolérance aux fautes dans les systèmes répartis.
- Les algorithmes évolutionnaires, leur principe et leurs familles sont présentés dans le chapitre 2.
- Le chapitre 3 inclus quatre parties : une initiation à la théorie des graphes, coloration des graphes, la coloration forte stricte et les algorithmes de coloration forte stricte, de distribution du graphe et un algorithme évolutionnaire séquentiel de distribution basée sur cette coloration.
- Au début du chapitre 4 nous proposons une idée pour distribuer l'algorithme évolutionnaire de distribution basée sur la coloration forte stricte. La distribution à la volée d'un espace d'états en utilisant cet algorithme évolutionnaire ansai que le mécanisme tolérant aux pannes sont présentés dans le chapitre 4. La mis en œuvre est exposé à la fin de ce chapitre.
- Enfin, une conclusion résume les apports essentiels de ce travail et donne des perspectives à moyen et long terme.

CHAPITRE 1
TOLERANCE AUX FAUTES

1.1 Introduction

La tolérance aux fautes est l'aptitude d'un système informatique à accomplir sa fonction, éventuellement de manière réduite ou dégradée, et d'éviter une panne complète en dépit de l'occurrence de fautes [01], que sa soit une détérioration du matériel, de défaut logiciel ou erreurs dues à la négligence et la mauvaise utilisation de la machine. La tolérance aux fautes garantit la sûreté de fonctionnement.

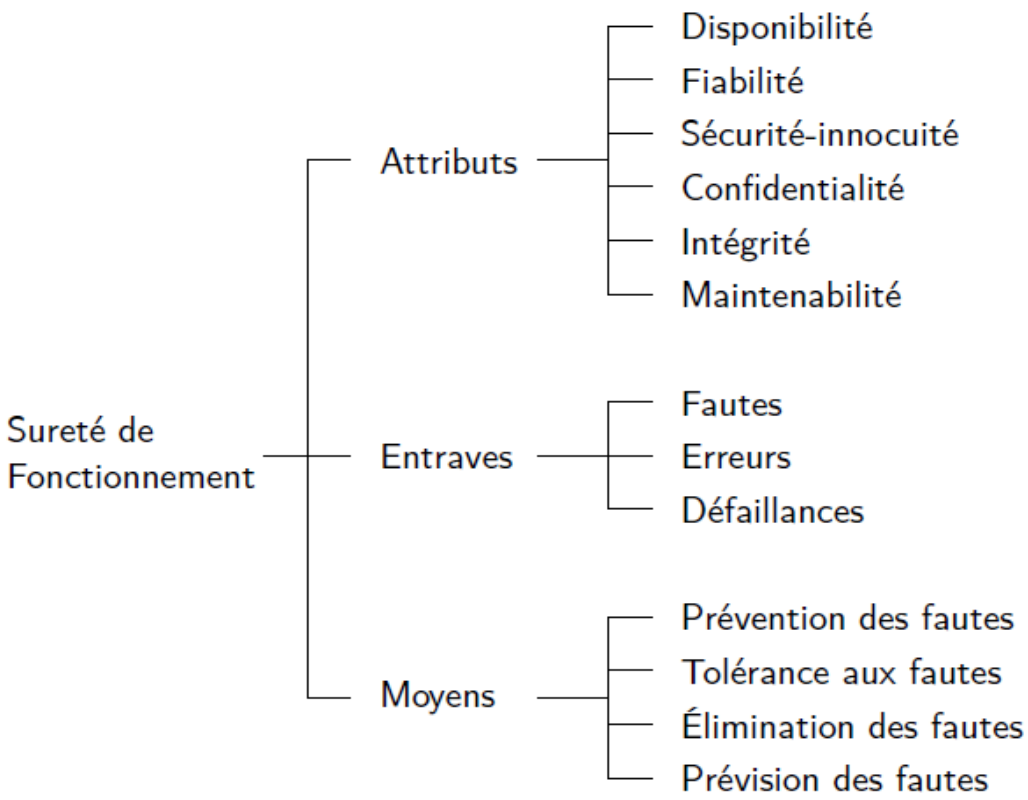


Figure 1.1: L'arbre de la sûreté de fonctionnement [01]

1.2 Sûreté de fonctionnement

La sûreté de fonctionnement d'un système informatique est la caractéristique qui permet d'assurer à ses utilisateurs une confiance avérée dans le service qui leur est délivré [02]. Ce dernier est aperçu par ses utilisateurs à travers son comportement. L'utilisateur est un autre système (humain ou physique) qui agit réciproquement avec le système considéré [03]. Quant à la sûreté de fonctionnement, elle peut être présentée autour de trois notions telles que décrites dans la figure 1.1 extraite de [01] à savoir : ses attributs, ses entraves et ses moyens.

1.2.1 Attributs de la sûreté de fonctionnement

Les attributs de la sûreté de fonctionnement se définissent à travers les différentes propriétés que doit vérifier le système.

Ils permettent d'évaluer la qualité du service délivré par le système. Ces attributs sont au nombre de six et sont définis dans [02] comme suit :

- **disponibilité** : c'est la qualité exigée par la majorité des systèmes sûrs de fonctionnement, ça consiste en la période pendant laquelle le système est disponible pour son utilisation. (en éloignant les temps de défaillance et de réparation);
- **fiabilité**: cet attribut sert à évaluer la continuité du service c'est-à-dire le taux en temps de fonctionnement durant lequel le système ne subit aucune défaillance ou aucune faute;
- **sécurité-innocuité** : semblables à la fiabilité mais par rapport aux conséquences graves engendrées par les fautes ;
- **confidentialité** : cet attribut évalue l'aptitude du système à fonctionner malgré les fautes volontaires et illégalement introduites;
- **intégrité** : l'intégrité d'un système détermine sa capacité à assurer des modifications approuvées des données;
- **maintenabilité** : cette qualité décrit la flexibilité et la souplesse du système face à des modifications apportées en vue de sa maintenance.

L'importance des attributs diffère selon l'application et l'environnement auquel est destiné le système informatique en question. La disponibilité, l'intégrité sont d'une façon générale exigées bien évidement a des degrés variables selon l'application .Par contre, la fiabilité, la sécurité-innocuité, la confidentialité peuvent être ou non requis. Dans le pilotage de fusée par exemple les applications sont critiques c'est pourquoi une grande importance doit être donnée à tous les attributs de la sûreté de fonctionnement alors qu'une importance est donnée à la maintenance et la fiabilité concernant les applications parallèles à longue durée d'exécution.

1.2.2 Difficultés de la sûreté de fonctionnement

Dans les difficultés de la sûreté de fonctionnement, on différencie trois types : les fautes, les erreurs et les défaillances.

- Fautes : une faute ou panne est caractérisée par sa nature, son origine et son étendue temporelle [04]. La nature d'une faute précise la manière dont elle a été provoquée : intentionnellement ou accidentellement. La cause de l'apparition d'une faute est révélée par son origine. L'étendue temporelle caractérise la persistance ou la durée d'une faute (temporaire ou permanente).
- Erreurs : une erreur est la conséquence d'une faute. L'emploi d'un état erroné par le système, engendre tout de suite la survenance d'une défaillance [05]. [06], les erreurs ont été classées selon leur type comme suit :

  –Premièrement : le service ne convient pas, en valeur, à celui spécifié.

  –Deuxièmement : le service n'est pas délivré dans l'intervalle de temps spécifié.

  De multiples catégories ont été définies dans [06]. Le service rendu est toujours en avance ou toujours en retard ou encore arbitrairement en avance ou en retard.

  On considère aussi comme un type d'erreur si jamais le système néglige ou oublie de rendre le service. On appelle un arrêt (crash) le fait que le système ne délivre plus de services..
- Défaillances : appelé défaillance : l'inaptitude d'un élément du système à garantir le service spécifié par l'utilisateur (son comportement n'est pas conforme à sa spécification).

Cette dernière est nettement marquée par son domaine, sa compréhension et sa perception, par les utilisateurs et ses répercussions sur l'environnement [04].

Fautes, erreurs et défaillances sont liées par des relations de causalité illustrées sur la figure 2.1.

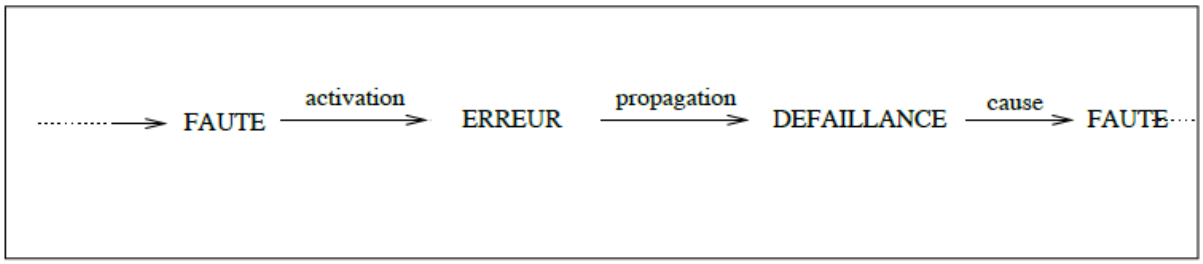


Figure1.2: La chaîne fondamentale des entraves

Une faute activée cause une erreur, qui peut se répandre dans un composant ou d'un composant à un autre jusqu'à produire une défaillance. Il est très important de noter que la défaillance d'un composant peut entraîner une faute permanente ou temporaire dans le système qui le contient, par contre la défaillance d'un système cause une faute permanente ou temporaire pour les systèmes avec lesquels il interagit.

1.2.3 Moyens d'assurer la sûreté de fonctionnement

Au cours des 50 dernières années, de nombreux moyens ont été mis au point pour la sûreté de fonctionnement, Ces moyens peuvent être regroupés en quatre grandes catégories [2]:

- **La prévention des fautes :** elle interdit la présence des fautes dans le système, en se basant sur des règles de développement (modélisation, utilisation de langage fortement typé, preuve formelle, etc.) ;
- **L'élimination des fautes :** tient énormément à atténuer la présence des fautes que ça soit en nombre ou degré de gravité ou de sévérité de la faute. Ce moyen ne peut être réalisé que pendant la phase de développement d'un système par vérification, diagnostic et correction, ou bien pendant sa phase opérationnelle par le biais de la maintenance;
- **La prévision des fautes** cherche à estimer de façon qualitative et quantitative la survenance et les conséquences des fautes. C'est par le biais de la détection d'erreurs et le rétablissement du système qu'on peut la mettre en œuvre;
- **La tolérance aux fautes** vise à préserver le service malgré l'occurrence de fautes (essaie de cacher ou d'occulter la présence des fautes et permet de continuer à fournir le service demandé malgré leur apparition).

La section suivante va présenter les différentes approches pour réaliser la tolérance aux fautes.

Ces méthodes peuvent être groupées de différentes façons.

1. Prévention des fautes et éliminations des fautes peuvent êtres considérées comme constituant « d'évitement des fautes. On cherche à concevoir un système sujet a moins de fautes possible. Il est de même pour la prévention des fautes.

2. Prévention des fautes et tolérance aux fautes peuvent se regrouper sous le concept « d'acceptation des fautes ; partant du principe qu'il ya toujours des fautes inévitables. Seulement il est préférable d'évaluer leur influence sur le système dans le but de diminuer le degré de gravité des défaillances qu'elles peuvent causer ,(si possible jusqu'à suppression totale des défaillances) , on remarque que ses deux méthodes sont complémentaires pour comprendre ou concevoir un système sur de fonctionnement [38].

1.3 Tolérance aux fautes

Dans le présent mémoire, nous visons spécialement et plus particulièrement la tolérance aux fautes qu'on considère comme un moyen de la sûreté de fonctionnement et qui a pour but de maintenir un service correct délivré par le système en dépit de l'occurrence ou présence de fautes. La tolérance aux fautes ne peut être mise en œuvre que s'il y a détection d'erreur et rétablissement du système. La figure 1.3 liste les techniques de tolérance aux fautes [38].

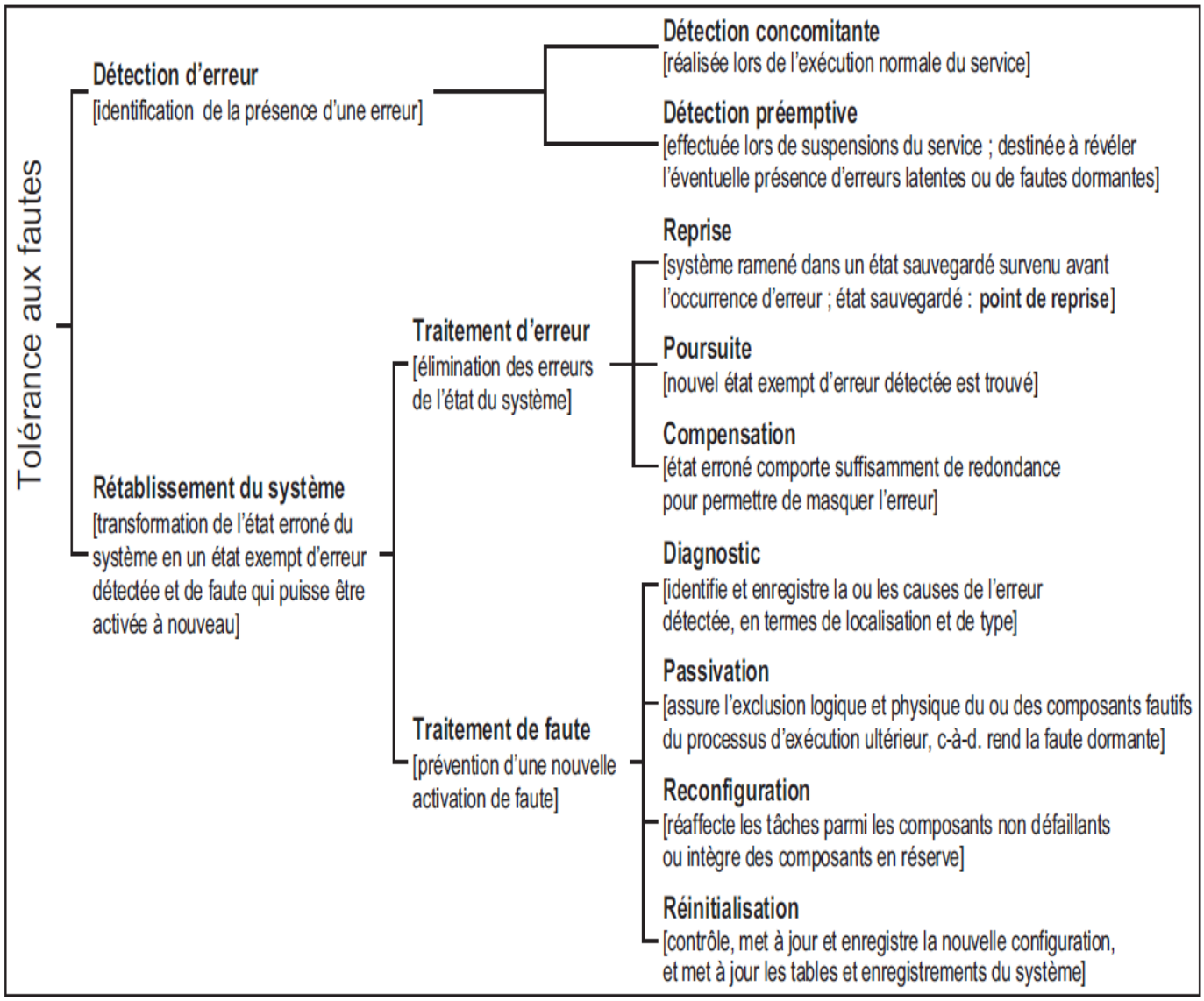


Figure 1.3: Techniques de tolérance aux fautes

1.3.1 Détection d'erreur

La détection d'erreur est basée sur une redondance qui peut prendre plusieurs formes : redondance au niveau information ou composant, redondance temporelle ou algorithmique. Les formes de détection d'erreur les plus couramment utilisées sont les suivantes :

- **Les codes détecteurs d'erreur** ils visent en particulier les erreurs induites par les fautes physiques. La détection est basée sur une redondance dans la représentation de l'information [07].

- **La méthode de duplication et comparaison** le cout matériel de cette méthode est très important mais malgré ca, elle reste un moyen de détection très utilisé vu sa simplicité de mise en œuvre. Au minimum, deux unités redondantes et obligatoirement indépendantes face aux fautes que l'on souhaite tolérées sont utilisées dans cette méthode.

  Typiquement, il existe une redondance des composants matériels concernant les fautes physiques, et diversification pour des fautes de conception [4].

- **Le contrôle temporel et d'exécution** par "chien de garde" (watch dog) est l'outil le plus couramment utilisé pour la détection d'erreur en ligne, il est particulièrement utilisé pour déceler la défaillance d'un périphérique en assurant que son temps de réponse n'excède pas une valeur seuil ou bien pour contrôler régulièrement l'activité d'une unité centrale. Aussi, se contrôle ce définit par faible cout économique [4].

- **Le contrôle de vraisemblance** il cherche à détecter des erreurs en valeur aberrantes pour le système. Sa mise en œuvre peut se faire par du matériel de détection, par exemple des adresses mémoires erronées, ou bien par un logiciel de vérification de la conformité des entrées, des sorties ou des variables internes du système par rapport à des invariants [04].

1.3.2 Rétablissement du système

Le rétablissement du système peut être assuré par deux méthodes complémentaires :

Le traitement d'erreur qui vise l'élimination des erreurs, si possible avant la survenance d'une défaillance.

Le traitement de faute qui sert à éviter l'activation à nouveau de la faute à l'origine de l'erreur.

1.3.2.1 Traitement d'erreur

Trois techniques sont utilisées pour le traitement d'erreur, il s'agit de : la reprise, la poursuite et la compensation [1].

- **La reprise :** est la technique la plus couramment utilisée. Elle consiste à sauvegarder périodiquement l'état du système pour le ramener dans un état antérieur à l'occurrence de l'erreur. Cet état sauvegardé est appelé *point de reprise*. Les principaux inconvénients de la reprise sont la taille des sauvegardes, la difficulté d'effectuer des sauvegardes cohérentes, et le surcoût temporel nécessaire à leur établissement.
- **La poursuite :** cette technique consiste à mettre en place un nouvel état ne contenant pas d'erreur qui permet au système de continuer à fonctionner de façon admissible éventuellement en mode dégradé. La mise du système dans un état sûr, par exemple en l'arrêtant est une forme limite de celle technique.
- **La compensation :** Cette technique exige que l'état du système contienne suffisamment de redondance, pour permettre, malgré les erreurs qui pourraient l'affecter, sa transformation en un état dépourvu d'erreur . Cette compensation peut intervenir immédiatement suite à une détection d'erreur ou exécutée systématiquement (masquage d'erreur).

1.3.2.2 Traitement de faute

Ce traitement comprend quatre phases successives : Le diagnostic, la passivation, la reconfiguration et la réinitialisation [10].

- La 1ere phase : le diagnostic de faute : qui sert à identifier la localisation et le type de faute responsable de l'état erroné du système [10].
- La 2eme phase : la passivation : elle est destinée à remplir l'objectif principal de traitement de faute en empêchant une nouvelle activation des fautes. Elle s'opère en excluant la participation du composant erroné de la délivrance du service par des moyens physiques ou logiciels.
- La 3eme phase : la reconfiguration [38] : intervient lorsque le système est incapable de délivrer le même service qu'auparavant. Cette phase vise à compenser l'isolement du composant défaillant, soit par basculement des composants – redondant, soit en réattribuant ses tâches à d'autres composants.
- La 4eme phase : la réinitialisation : cette phase consiste à vérifier et mettre a jour la nouvelle configuration du système [10]. .

1.4 Tolérance aux fautes dans les systèmes répartis

Un système distribué est connu par le nombre important de composants. Une faute d'un seul des composants peut mener à la défaillance de tout le système. Ainsi, la défaillance du système est inévitable même si chaque composant présente une probabilité de faute très faible. La fiabilité des systèmes distribués est tributaire de l'occurrence des fautes et l'aptitude à leur faire face [08].

La tolérance aux fautes apparaît comme un élément indispensable aux systèmes répartis. Elle est impérativement actionnée par l'utilisateur d'un mécanisme de redondance [02] .Cette redondance peut être spatiale, (duplication de composants), temporelle, (traitement multiple) ou bien informationnelle, (c.à.d. redondance des données, codes, signatures) [09] .Pour répondre à ce besoin, plusieurs techniques ont été conçues. Ces techniques peuvent être séparées en deux classes : les techniques basées sur la duplication et les techniques basées sur une mémoire stable.

1.4.1 Redondance spatiale et temporelle : réplication

La tolérance aux fautes par réplication consiste à utiliser des copies multiples d'un même composant ou processus sur des processeurs différents. Cette approche par duplication rend possible le traitement des pannes en les masquant. C'est-à-dire le groupe de processus doit être géré de façon à donner l'illusion d'un seul processus logique qui continue à fournir un service correct, malgré la défaillance d'un sous ensemble des membres du groupe. On distingue généralement trois stratégies : la réplication passive, active et semi-active [01].

- **Duplication active :** une seule copie (la copie primaire) reçoit les requêtes et effectue toutes les opérations. Pour assurer la cohérence, la copie primaire diffuse son état interne régulièrement aux copies secondaires. Cet état sert de point de reprise en cas de défaillance En cas de défaillance de la copie primaire, une des copies secondaires est élue pour prendre sa place.

- **Duplication passive :** cette réplication se caractérise par la correspondance (symétrie) du comportement des copies d'un composant duplique ou toutes les copies jouent un rôle identique : traitement de tous les messages réceptionnés, mise a jour de son état interne de façon indépendante et production des messages de sortie. Cette stratégie écourte l'utilisation des points de reprise onéreux .Par conséquent, elle nécessite un système de diffusion atomique et exige, pour garantir la cohérence [08], que l'exécution des requêtes soit déterministe.

- **Duplication semi-active :** c'est une solution hybride entre la réplication active et la réplication passive. Cette stratégie ressemble à la réplication active dans le sens où toutes les copies réceptionnent les messages d'entrée et peuvent aussi les traiter. Cependant, elle se rapproche de la réplication passive son traitement asymétrique ou une copie privilégiée assure la responsabilité de certaines dispositions (par exemple ,l'acceptation des messages ,ou la préemption du traitement en cours). Appelée la meneuse, cette copie privilégiée peut dicter ses décisions aux autres copies dénommées par le terme des suiveuses sans les consulter. Ainsi, la meneuse peut endosser seule la responsabilité de transmettre les messages de sortie [1].

La tolérance aux fautes par duplication est alors réalisée par masquage d'erreur. La défaillance d'une copie est masquée par le comportement des copies non défaillantes. Le principal désavantage de cette méthode par réplication est qu'elle nécessite de nombreuses ressources : pour tolérer p défaillances, il est nécessaire d'avoir p + 1 composants identiques. Cette méthode n'est donc pas adaptée aux calculs parallèles où la performance (temps de calcul) est souvent le critère prépondérant : les ressources doivent être exploitées en priorité pour le calcul.

1.4.2 Redondance informationnelle : mémoire stable

La mémoire stable n'est qu'une abstraction [11]. Il s'agit d'un support constant de stockage qui a pour rôle d'assurer l'accessibilité aux données et leur protection contre les pannes pouvant affectées le système. En cas de panne, un état correct stocké antérieurement sur la mémoire stable reste accessible et permet au système de revenir à un état antérieur. [09].

Un support de stockage est vu comme une mémoire stable si et seulement si les trois conditions suivantes sont vraies :

1. **Accessibilité :** il existe à tout moment de l'exécution un chemin permettant d'accéder aux données sauvegardées même en présence des pannes.
2. **Protection :** les pannes affectant le système ou l'application n'altèrent pas les données.
3. **Atomicité :** les mises à jour des données sur le support de stockage se font de manière atomique.

Le principe de cette technique est de remplacer l'état d'erreur par un état cohérent en utilisant les informations stockées sur la mémoire stable. Cette approche utilise la redondance d'informations. Ces informations correspondent à la sauvegarde de l'état des processus ou bien à la journalisation d'évènements, et sont stockées sur la mémoire stable.

Le point critique pour réaliser la tolérance aux fautes par mémoire stable est la constitution d'un état correct du système. La constitution d'un tel état introduit un surcoût qui va dépendre des contraintes imposées au système : nombre et types de défaillances à tolérer, reprise globale du système ou uniquement des processus défaillants, etc. [08].

Deux approches ont été proposées dans la littérature pour construire un état global cohérent d'un système réparti [11] :

– **A priori :** les sauvegardes des différents états des processus coopérants sont coordonnées pour constituer un état global cohérent ;

– **A posteriori :** les sauvegardes des états des processus sont indépendantes; la reconstitution d'un état global cohérent se fait lors de la reprise.

L'état global d'une application parallèle est composé :

• de l'état local de tous les processus participant au calcul,

• et de l'état de tous les canaux de communication entre les processus.

Les deux sous-sections suivantes présentes les protocoles de reprise basés sur la sauvegarde et les protocoles basés sur la journalisation [8].

1.4.2.1 Reprise par sauvegarde

Les règles de reprise par sauvegarder assurent la concrétisation des sauvegardes régulières de l'état des processus. L'action de redémarrage nécessite l'utilisation du dernier ensemble de sauvegarde composant un état global cohérent [11].

On distingue généralement trois approches pour la reprise par sauvegarde selon le mode de construction de l'état global cohérent de la reprise :

- **Sauvegarde non coordonnée :** l'analyse des différences entre les points de sauvegarde au moment de la reprise est assurée par un algorithme pour essayer de délimiter l'ensemble des sauvegardes les plus actuelles composant un état global cohérent. Cette demande génère un surcoût important et une absence de coordination pouvant provoquer un effet domino (réaction en chaîne) .Durant l'élaboration de l'état global cohérent, les dépendances entre les messages peuvent entraîner un retour à l'état initial [01].

- **Sauvegarde coordonnée :** permet d'assurer que seules des lignes de reprise cohérentes sont créées. Il existe différentes manières de coordonner les processus. *La sauvegarde coordonnée bloquante* : chaque processus ne peut sauvegarde son état local qu'après que tous les processus sont arrêtés et que les

canaux de communication sont vidés. *La sauvegarde coordonnée minimale*: ne synchronise un processus qu'avec les processus dont il dépend réellement.

- **Sauvegarde induite par les communications :** c'est un compromis entre les deux approches précédentes Chaque processus peut créer indépendamment des points de reprise locale qui sont complétés par des points de reprise forcés (une sauvegarde qui doit être effectuée pour empêcher l'effet domino).

1.4.2.2 Reprise par journalisation

Le principe de la tolérance aux fautes par journalisation est de sauvegarder l'histoire de l'application. Les protocoles par journalisation utilisent à la fois la sauvegarde locale de l'état des processus et la journalisation des évènements déterministes pour permettre la reprise de l'application. Il est alors possible de reprendre l'exécution des processus défaillants (et uniquement des processus défaillants) à partir de leur dernière sauvegarde en rejouant les évènements non déterministes sauvegardés [08].

Deux méthodes principales permettent de réaliser la tolérance aux fautes dans les systèmes répartis (figure 1.4) : la duplication et l'abstraction d'une mémoire stable. Dans les deux cas, la tolérance aux fautes est réalisée par l'emploi d'un mécanisme de redondance.

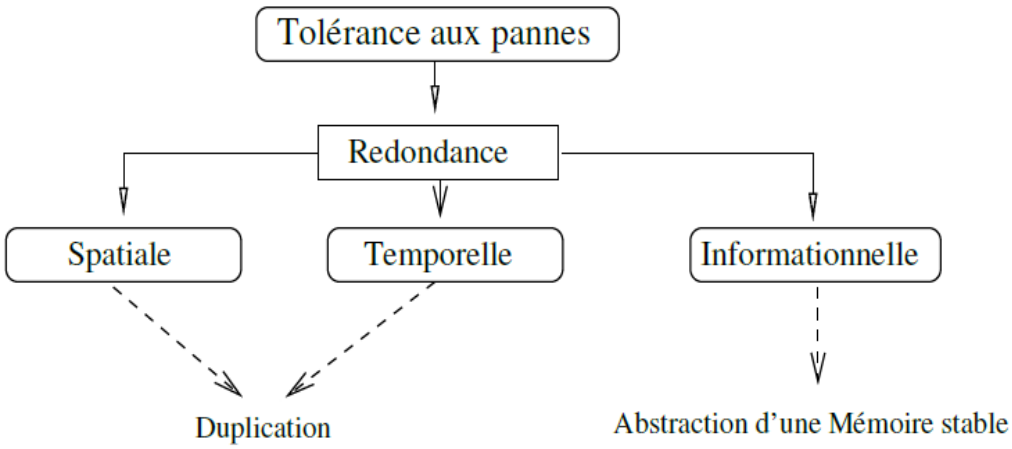


Figure 1.4 : Les techniques de tolérance aux fautes dans les systèmes répartis [09]

1.5 Conclusion

Dans ce chapitre, nous avons présenté en premier lieu d'un point de vue très général, les grands principes de la sûreté de fonctionnement des systèmes informatiques, en détaillant les principales méthodes. Nous avons expliqué aussi que la tolérance aux fautes est un moyen pour assurer la sûreté de fonctionnement d'un système et qu'on utilise toujours la redondance pour l'obtenir.

La dernière partie de ce chapitre a présentée un aperçu de la tolérance aux fautes pour les systèmes distribués. Les techniques de tolérance aux fautes spécifiques aux systèmes distribués sont basées sur la réplication ou sur l'utilisation d'une mémoire stable.

CHAPITRE 2
ALGORITHMES EVOLUTIONNAIRES

2.1 Introduction

Les algorithmes évolutionnaires, comme de nombreux algorithmes (les réseaux de neurones artificiels, l'algorithme de recuit simulé de la thermodynamique…) qui ont trouvé leur inspiration dans les phénomènes physiques ou biologiques, se sont inspirés des processus évolutionnaires qui se produisent dans la nature. Cette inspiration n'est pas surprenante et elle est tout à fait justifiée vu la puissance et l'évidence de l'évolution naturelle. Cette dernière s'opère en deux étapes : la sélection et la reproduction [13].

- On appellle sélection naturelle, ce mécanisme central qui a lieu au niveau des populations en choisissant, et par sélection les individus les mieux adaptés à leur environnement.
- La reproduction implique ou met en cause une mémoire : l'hérédité, sous la forme de gènes. Au niveau moléculaire, ce matériel héréditaire subit un changement permanant par le biais de mutation et recombinaisons qui engendrent une grande pluralité ou diversité.

Ces principes, présentés pour la première fois par Darwin en 1860 dans son livre *L'origine des espèces au moyen de la sélection naturelle ou la lutte pour l'existence dans la nature*, et les travaux de Mendel en 1866 retraçant dix années d'expériences d'hybridation chez les végétaux (recombinaison des gênes) ont inspiré bien plus tard les chercheurs en informatique. Ils ont donné naissance à une classe d'algorithmes regroupés sous le nom d'Algorithmes Evolutionnaires (ou Evolutionary Algorithms (EA)).

2.2 Algorithmes évolutionnaires

Parmi l'ensemble des problèmes il existe une catégorie pour lesquels il est difficile, voire impossible, de trouver une solution en un temps limité. Il est alors utile de trouver une technique permettant la localisation rapide de solutions sous-optimales [13].Pour cela, un système capable d'améliorant sa performance au cours du temps, semble ouvrir la voie à une recherche intéressante.

Les algorithmes évolutionnaires constituent une approche originale : ils sont avant tout des méthodes stochastiques d'optimisation globale. Ils font partie du champ de la vie artificielle. La vie artificielle est l'étude des systèmes conçus par l'homme, qui présentent des comportements similaires aux systèmes vivants naturels. Leur souplesse d'utilisation pour des fonctions objectives non régulières sur des espaces de recherche non-standard permet leur utilisation pour des problèmes qui sont pour le moment hors d'atteinte des méthodes déterministes plus classiques [14].C'est pourquoi, dans de nombreux domaines, les chercheurs ont été amenés à s'y intéresser. Il est surprenant de savoir que cette idée d'appliquer ces principes remonte aux années quarante, bien avant l'apparition des ordinateurs. Dès 1948, Turing a proposé "genetical or evolutionary search" [12], depuis, les EA ont été appliqués à différents problèmes : la recherche opérationnelle (la coloration de graphes, le problème du voyageur de commerce…etc), en robotique (la recherche de trajectoire ou l'évitement d'obstacles), en vision, en recalaged'images médicales, etc. [13].

2.2.1 Terminologie et notations

Le vocabulaire utilisé dans les AE est directement inspiré de celui de la théorie de l évolution et de la génétique:

- **Gène** : un gène correspond à un seul symbole (0 ou 1 dans le cas binaire).
- **Chromosome** : est une chaîne de gènes.
- **Un Individu**: n'est autre qu'une solution possible du problème à résoudre.
- **Une Population**: est un ensemble d'individus.
- **Une Génération**: c'est l'ensemble de la population à un moment donné du processus.
- **La Fonction d'évaluation (fitness en anglais)** : est la pièce maîtresse dans le processus d'optimisation. C'est l'élément qui permet de mesurer la qualité des individus ou son poids; cette fonction d'adaptation peut représenter la fonction objective à optimiser.

2.2.2 Principe général :

Simuler l'évolution d'une population d'individus divers à laquelle on applique différents opérateurs d'évolution comme les recombinaisons, les mutations, la sélection, etc. La figure 2.1représente Le schéma général d'un algorithme évolutionnaire en pseudo code et la figure 2 illustre le même schéma en diagramme.

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Figure 2.1Le schéma général d'un algorithme évolutionnaire en pseudo code

L'algorithme commence par l'initialisation de la population en choisissant des individus (généralement par tirage aléatoire), après il évalue les individus (calcul des valeurs de la fonction d'évaluation pour tous les individus), ensuite il génère à chaque fois, une nouvelle population en suivant les étapes suivantes:

- sélection des individus les plus performants(les plus adaptés se reproduisent) ;
- Application des opérateurs de variation aux parents sélectionnés, ce qui génère de nouveaux individus, on parlera de mutation et de croisement
- Evaluation des enfants ;
- Remplacement de la population par une nouvelle population (les plus adaptés survivent).

L''evolution stoppe quand le niveau de performance souhaité est atteint, ou qu'un nombre fixé de générations s'est écoulé [14].

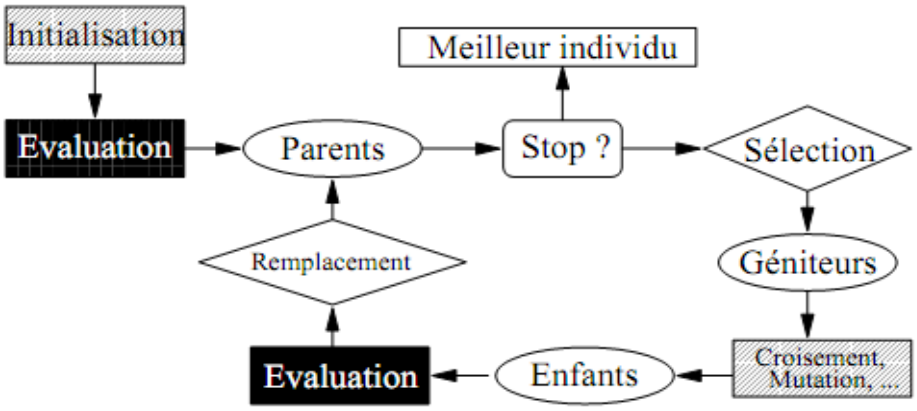


Figure 2.2 Le schéma général d'un algorithme évolutionnaire en diagramme

Plusieurs types d'évolution ont été développés, donnant naissance à quatre grandes tendances (figure 2.3): les Algorithmes Génétiques (ou Genetic Algorithms (GA)), les Stratégies d'Evolution (ou Evolution Strategies (ES)), la Programmation Evolutive (ou Evolutionary Programing (EP)), et la Programmation Génétique (ou Genetic Programing (GP)) [12].

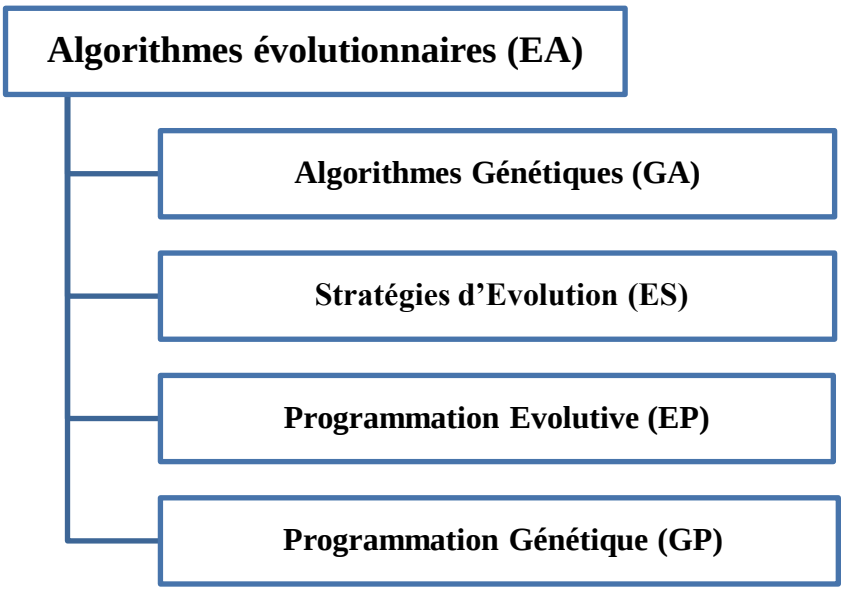


Figure 2.3 Branches des algorithmes évolutionnaires (EA).

Dans la suite de ce chapitre, nous présentons les différents types d'algorithmes évolutionnaires.

2.3 Algorithmes génétiques

C'est en 1975 que Holland donne la première description du processus des algorithmes génétiques puis Goldberg (1989) les a utilisés pour résoudre des problèmes concrets d'optimisation. Les GA sont certainement la branche des EA la plus connue et la plus utilisée. La particularité de ces algorithmes est le fait qu'ils font évoluer des populations d'individus codés par une chaîne binaire [13]. De manière générale, un algorithme génétique a comme but d'optimiser la fonction d'adaptation, il simule l'évolution d'une population d'individus en utilisant des opérateurs évolutionnaires jusqu'à la satisfaction d'un critère d'arrêt.

Pour mettre en œuvre un algorithme génétique, il faut respecter un certain nombre d'étapes: une représentation génétique du problème, une évaluation d'adaptation des individus, Ces individus seront manipulés par un opérateur de croisement et de

mutation qui les choisit selon un mode de sélection, Les individus issus de la phase croisement et de mutation seront insérés dans la nouvelle population par une méthode d'insertion. A chaque fois et avant la sélection des nouveaux individus, un test d'arrêt sera effectué pout décider quand arrêter l'algorithme[15].

Les différentes étapes de ce dernier sont présentées, en détail, dans les sous-sections suivantes.

2.3.1 Représentation

La représentation des individus est basée sur le codage binaire de l'information, chaque individu est représenté sous forme de chaînes de bits. Dans la littérature, il y a plusieurs types de codages. Parmi les codages les plus utilisés on peut citer :

- **Codage binaire :** ce type de codage est le plus utilisé. son principe est de coder la solution selon une chaîne de bits (les valeurs 0 ou 1).

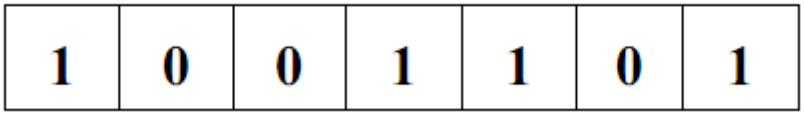


Figure 2.4 codage binaire d'un chromosome

- **Codage réel :** son principe est de coder les solutions en tant que suites de nombres entiers ou de nombres réels, il peut-être utile dans le cas où l'on recherche le maximum d'une fonction réelle.

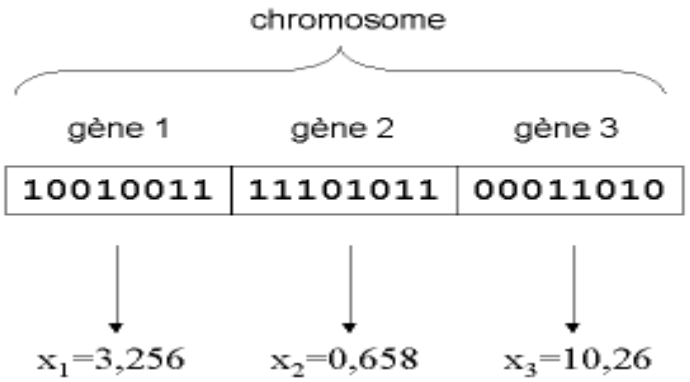


Figure 2.5 codage des variables réelles

- **Codage de Gray :** Dans le cas d'un codage binaire on utilise souvent la « distance de Hamming », On peut parler des limites de cette mesure, le passage du nombre 7 (0111) au nombre 8 (1000) en 4 mutations. Le codage de Gray proposant (0100) pour 7 et (1100) pour 8, il a comme propriété qu'entre un

élément n et un élément n + 1, donc voisin dans l'espace de recherche, un seul bit diffère.

2.3.2 Evaluation : fitness

C'est la fonction à optimiser, elle est utilisée pour mesurer les performances et les qualités de chaque individu afin que les plus forts soient retenus dans la phase de sélection, puis modifiés dans la phase de croisement et mutation.

Les GA favorisent le croisement plutôt que la mutation, en donnant de forts taux de croisement et de faibles taux de mutation. (Goldberg 89) propose dans son livre 80% de croisement et 3% de mutation [13].

2.3.3 Sélection

La sélection a pour objectif d'identifier les individus qui doivent se reproduire. Elle consiste à sélectionner les individus proportionnellement à leur performance. Les individus les mieux adaptés ont plus de chances d'être sélectionnés. Il existe différentes méthodes de sélection :

- **La sélection par roulette :** cette méthode, la plus ancienne des méthodes, où chaque individu de la population occupe une section proportionnellement à sa fonction d'adaptation. A chaque fois qu'un individu doit être sélectionné, un tirage à la loterie s'effectue et propose un candidat. Les bons individus seront plus souvent sélectionnés que les mauvais, et un même individu pourra avec cette méthode être sélectionné plusieurs fois.

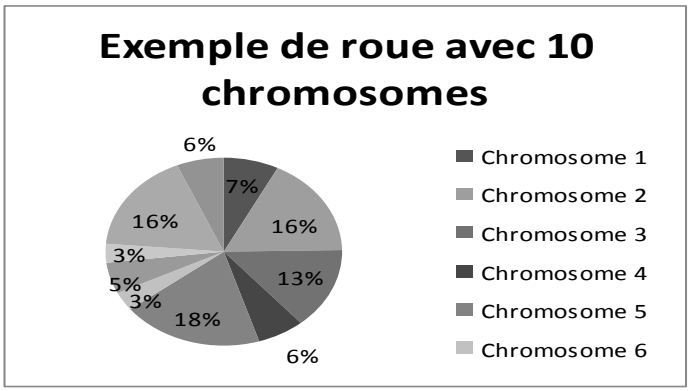


Figure 2.6 Méthode de sélection par roulette

- **La sélection par le rang :** dans cette approche de sélection, tous les chromosomes ont l'opportunité d'être sélectionnés. Seulement le grand inconvénient de cette méthode est la convergence lente vers la bonne solution. On tient à noter que la sélection par rang est semblable à la sélection par roulette qui consiste à faire une sélection en utilisant une roulette sauf pour les proportions dont les secteurs sont proportionnels aux rangs des individus (P pour le meilleur, 1 pour le moins bon, pour une population de taille P).

- **La sélection par tournoi :** est celle avec laquelle on obtient les résultats les plus satisfaisants. Elle consiste à choisir aléatoirement k (taille du tournoi) individus de la population. On en tire k uniformément dans la population, et on sélectionne le meilleur de ces k individus. Le nombre d'individus sélectionnés a une influence sur la pression de sélection, lorsque k = 2, la sélection est dite par « tournoi binaire ».

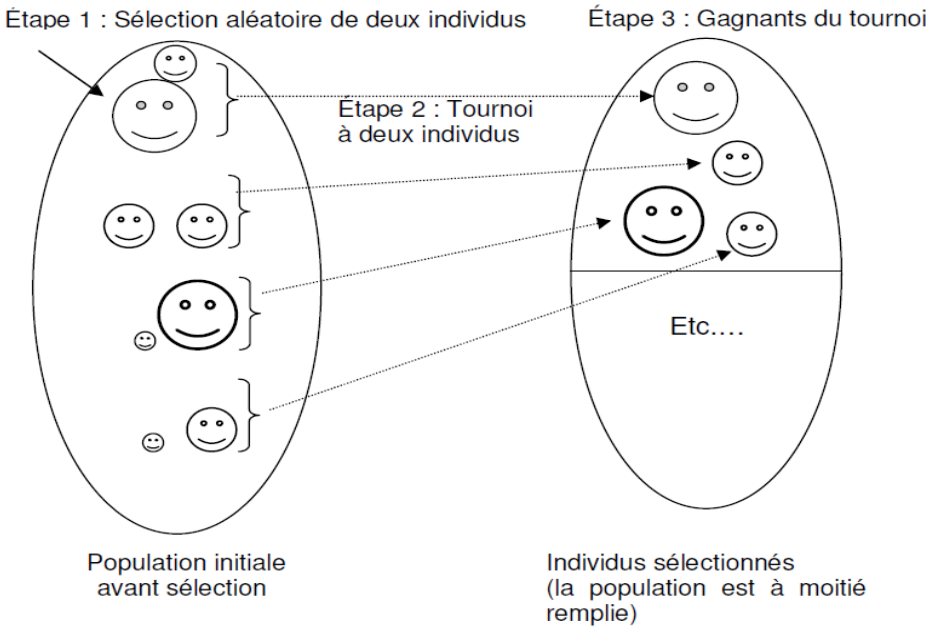


Figure 2.7 Méthode de sélection par tournoi

2.3.4 Croisement

Le Croisement est le processus par lequel une nouvelle solution d'individu est créée à partir de l'information contenue dans les deux (ou plusieurs) solutions parents sélectionnés par l'opérateur de sélection, il est considéré par beaucoup comme l'une des caractéristiques les plus importantes dans les algorithmes génétiques [12]. Selon la littérature, plusieurs opérateurs de croisement sont proposés, les plus utilisés sont :

- **croisement en 1-point:** il consiste à sélectionner aléatoirement une position de césure et de permuter ensuite les deux parties droites de chacun des chromosomes des deux parents ce qui produit deux nouveaux enfants.

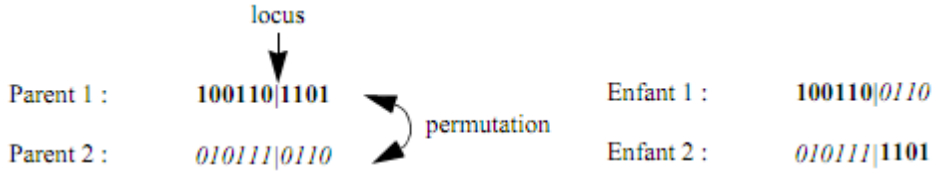


Figure 2.8 Croisement en 1-point de deux chromosomes

- **Croisement en plusieurs points :** ce type de croisement est utilisé en choisissant aléatoirement plusieurs points de coupure pour dissocier chaque parent et permuter les différentes parties de la même manière que le croisement en 1-point.

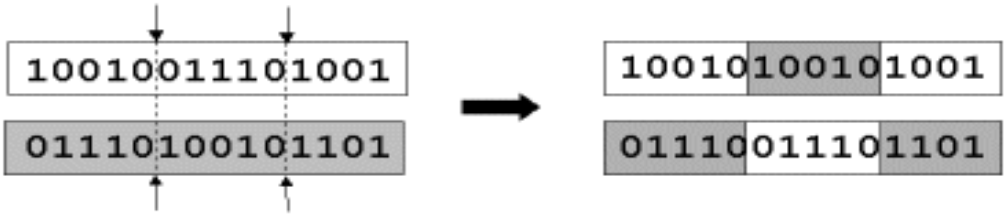


Figure 2.9 Croisement en 2-point de deux chromosomes

2.3.5 Mutation

La mutation est le nom générique donné au changement de l'information génétique dans le chromosome. Elle est exécutée seulement sur un seul parent et crée un seul enfant en appliquant une sorte de changement aléatoire sur le chromosome [12]. Pour un codage binaire, cela revient à changer un 1 en 0 et vice versa (figure 2.10).



Figure 2.10 Opération de mutation.

Cet opérateur joue un rôle primordial dans le processus d'optimisation, elle introduit de la diversité dans le processus de recherche des solutions et peut aider l'AG à ne pas stagner dans un optimum local.

2.3.6 Opérateur de remplacement

Le travail de cet opérateur consiste à réintroduire les descendants obtenus par application de croisement et de mutation dans la nouvelle population. On trouve essentiellement deux méthodes de remplacement différentes:

- **Le remplacement stationnaire :** dans ce cas, les parents sont remplacés par les enfants mutés sans tenir compte de leurs performances. Le nombre d'individus de la population est constant tout au long du cycle d'évolution.
- **Le remplacement élitiste :** dans ce cas, on copie quelques meilleurs individus dans la nouvelle population, donc les enfants d'une génération ne remplaceront pas nécessairement leurs parents. L'objectif est d'éviter que les meilleurs individus soient perdus après les opérateurs de croisement et de mutation. Cette méthode permet de conserver, à une itération donnée, le meilleur individu trouvé dans toutes les populations [15].

2.3.7 Critère d'arrêt

Le test d'arrêt joue un rôle très important dans le jugement de la qualité des individus. Le critère d'arrêt indique que la solution est suffisamment approchée de l'optimum. Plusieurs critères d'arrêt sont possibles : On peut arrêter l'algorithme après un nombre fixé a priori de générations; l'algorithme peut aussi être arrêté lorsque la population cesse d'évoluer ou n'évolue plus suffisamment; on peut aussi arrêter l'algorithme lorsque le traitement des individus devient coûteux en temps [16].

2.4 Stratégies d'évolution

Les stratégies d'évolution (evolution strategies, en anglais, abrégé ES) sont apparues dans les années 60 avec les travaux de Ingo Rechenberg à l'université technique de Berlin [12].Ils ont été ensuite développée durant les années 1970, principalement par les travaux Hans-Paul Schwefel.

Les ES sont donc des paradigmes de recherche qui appartiennent à la famille des algorithmes évolutionnaires ils furent utilisé le plus souvent sur des problèmes d'optimisation continus, discrets, contraints, multi-objectifs, etc. Les particularités de ces méthodes sont [13]:

- coder les paramètres du problème à résoudre en nombres réels.
- effectuer une sélection déterministe des individus en ne choisissant que les n (n>1) individus classés selon leur performance.
- encoder les paramètres d'évolution directement dans le génotype afin de les faire évoluer au même titre que les valeurs des paramètres solutions du problème.

2.4.1 Représentation

Les ES sont pratiquement des algorithmes pour l'optimisation continue, donc la représentation des individus d'une manière générale est basée sur La représentation continue, ou représentation réelle, Dans cette approche, la recherche s'effectue dans R^n ou une partie de R^n[16]. les solutions dans cet espace de recherche sont des vecteurs n-dimensionnels noté $\vec{x}$ $(x_1,\ldots,x_n)$, chaque composant x_i du vecteur $\vec{x}$ est représente par un nombre réel . Les individus ont un certain nombre de paramètres en particulier les paramètres de mutation. Ces paramètres peuvent être divisés en deux ensembles [12]: Un vecteur d'écarts-types $\vec{\sigma}$ est associé au vecteur$\vec{x}$. Si tous les éléments de $\vec{x}$ sont mutés selon la même loi, $\vec{\sigma}$ ne comporte qu'un seul élément, sinon $\vec{\sigma}$ est de même taille que$\vec{x}$, et un vecteur de rotation $\vec{\alpha}$. Il est possible, grâce à ce vecteur, de favoriser une direction de mutation des individus. Un individu est représenté alors par :

$$\langle \underbrace{x_1,\ldots,x_n}_{\vec{x}}, \underbrace{\sigma_1,\ldots,\sigma_{n_\sigma}}_{\vec{\sigma}}, \underbrace{\alpha_1,\ldots,\alpha_{n_\alpha}}_{\vec{\alpha}} \rangle$$

2.4.2 Evolution

Les ES favorisent la mutation plutôt que la recombinaison. Travaillant sur des réels, la mutation suit une loi généralement gaussienne avec des écarts-types généralement codés dans le génotype [13].

2.4.3 Sélection

La sélection des parents dans les stratégies d'évolution n'est pas basée sur les valeurs de la Fonction d'évaluation. Chaque fois qu'un opérateur de recombinaison nécessite un parent, il est tiré au hasard avec une distribution uniforme de la population des individus [12].

La sélection des individus est déterministe [13],Deux types de sélection existent, qui sont les sélections (μ,λ) et (μ+λ), et signifient que λ descendants sont générés à partir d'une population de μ individus. La première sélection « , » consiste à gérer l'élitisme par le biais de la différence $\mu - \lambda$ (on garde les $\mu - \lambda$ meilleurs individus de la population courante et on complète par les λ descendants), tandis que la seconde « +» est une version plus adaptative dans sa gestion de l'élitisme : à partir d'un ensemble intermédiaire de taille $\mu + \lambda$ constitué des μ individus de la population courante et des λ descendants, on sélectionne les μ meilleurs individus de la génération suivante. Cette dernière méthode permet de ne pas perdre les meilleurs individus d'une génération à une autre mais accroît les possibilités que la population converge prématurément vers une solution qui n'est peut être pas optimale mais qui représente un minimum local [13].

2.4.4 Croisement

Le croisement opère ici rarement sur le génotype contenant les variables du problème. Cependant, elle semble très utile pour l'évolution des paramètres de mutation [13].Il implique les deux parents qui créent un enfant. Il existe deux variantes du croisement qui se distinguent parla manière de combiner les deux parents. On utilisant le croisement discret un des deux parents est choisi de manière aléatoire, avec des chances égales soit pour les parents. Croisement intermédiaire discret

$$z_i \begin{cases} (x_i + y_i)/2 & croisement\ intermédiaire \\ x_i ou y_i & croisement\ discret \end{cases}$$

2.4.5 Mutation

Dans le cadre de stratégies d'évolution, la mutation la plus employée est la mutation gaussienne qui consiste à rajouter un bruit gaussien au vecteur des variables. La distribution gaussienne la plus générale en dimension n est la distribution multi variée N (m, C), de moyenne m et de matrice de covariance C(une matrice n × n symétrique définie positive). La forme la plus générale est alors

$$\mathbf{X} := \mathbf{X} + \sigma N(0, C)$$

Ajuster les paramètres d'une ES revient alors à rechercher les meilleures valeurs du pas σ et de la matrice de covariance au cours de l''evolution. Les premiers essais ont concerné le cas isotrope dans lequel le seul degré de liberté est le pas σ.

2.4.6 Auto-adaptation

Les paramètres de la mutation comme : le pas σ, matrice de covariance C elle même, ses derniers sont attachés à chaque individu, et sont soumis eux même à des mutations. Suivant les paramètres de mutation utilisés on perçoit nettement trois variantes:

1-la mutation isotrope qui n'utilise qu'un pas σ par individu, la matrice de covariance étant alors l'identité ;

2- la mutation non-isotrope attache à chaque individu un vecteur de pas (σ1, . . ., σn) et la matrice de covariance est alors la matrice diagonale ayant σi2 en i-éme position sur la diagonale. Les différentes variables sont encore mutées de façon indépendante, mais avec des variances différentes ; enfin une matrice de covariance complète est attachée à chaque individu par la mutation corrélée.

Pour muter un individu il faut alors muter d'abord les paramètres de mutation, puis enfin muter les variables du problème (l'individu lui-même) avec les nouveaux paramètres de mutation. Les détails peuvent être trouves dans [17].

2.5Programmation génétique

La programmation génétique (Genetic Programming en anglais, abrégé GP)est la plus jeune des membres de la famille des algorithmes évolutionnistes. En plus de sa représentation particulière (ces programmes étant représentés sous forme d'arbres). Elle diffère des autres branches des algorithmes évolutionnaires dans son domaine d'application. Alors que les EA sont généralement appliquées à des problèmes d'optimisation. La GP pourrait plutôt être utilisée à l'apprentissage automatique [12]. Elle a été imaginée à l'origine pour optimiser des programmes écrits en langage LISP.

La représentation utilisée dans la PG est l'arborescente. Chaque programme étant codé sous la forme d'un arbre [18]. Les branches dans l'arbre représentent les fonctions qui prennent des arguments, les feuilles représentent des fonctions à zéro-argument, des variables ou des constantes. La fitness dans la PG est mesurée en testant le programme qu'elle représente. La figure suivante illustre la représentation arborescente de la fonction min(x,y*2)+(x+5).

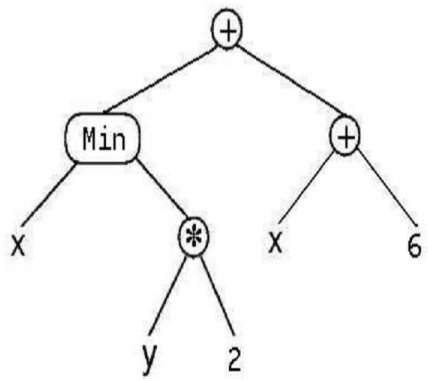


Figure 2.11 Opération de mutation[19].

- **Le croisement :** est appliqué sur un individu par un simple changement de son nœud avec un autre nœud d'un autre individu de la population comme illustré sur la figure 2.11.

▪ **LA Mutation :** La mutation est une autre caractéristique importante de la programmation génétique. Elle peut remplacer un sous-arbre un nœud ou simplement remplacer l'information d'un nœud.

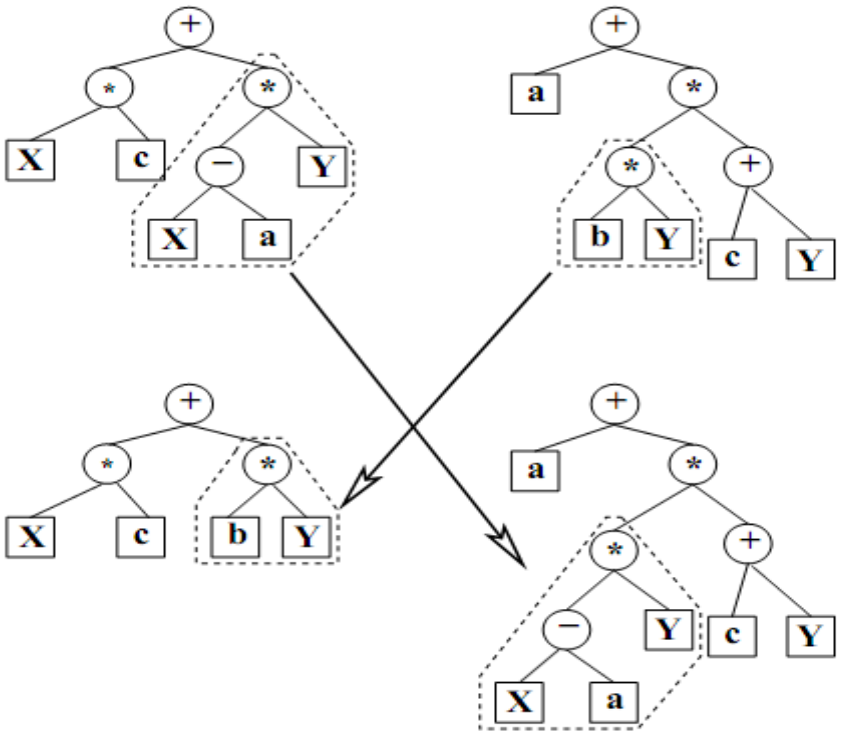


Figure 2.12 Exemple de croisement en programmation génétique

2.6Programmation Evolutive

La programmation évolutive (Evolutionary Programming en anglais, abrégé EP) a été introduite par Fogel en 1966 essentiellement pour faire évoluer les machines à états finis et a, par la suite, été étendue aux problèmes d'optimisation de paramètres [11]. A la différence des autres branches de la famille des algorithmes évolutionnaires, la PE n'utilise pas une représentation spécifique. Elle se focalise sur l'opérateur de mutation approprié à un problème spécifique. Pour résoudre un problème, on génère aléatoirement une population de taille μ, et chaque individu i va générer λ descendants suite à l'application de l'opérateur de mutation. Une opération de sélection naturelle permet par la suite de former une nouvelle génération de taille μ à partir des parents et des descendants [11].

2.7 Conclusion

Ce chapitre a présenté les algorithmes évolutionnaires en toute généralité, Il faut retenir d'une part leur robustesse ils apportent rapidement une solution acceptable et d'autre part leur souplesse d'utilisation, ils Permettent de traiter des espaces de recherche important. Et malgré leurs points forts, les algorithmes évolutionnaires présentent des inconvénients comme le problème des optimums locaux, la délicatesse de choix de la fonction d'évaluation. Ils nécessitent plus de calculs que les autres algorithmes méta heuristiques.

CHAPITRE 3
GRAPHES ET COLORATION FORTE STRICT

3.1 Introduction

La réduction du nombre de messages échangés dans un système distribué est un objectif très important que beaucoup de travaux essayent d'atteindre. La solution la plus évidente est de minimiser le nombre d'arcs reliant des machines différents.

La coloration forte stricte est une idée innovante pour répendre à ce problème du fait qu'elle repose sur la propriété de dominance. Cette propriété est utilisé pour regroupe le sommet dominant et sa classe de couleur dominée dans un seul site afin de minimiser le nombre d'arcs traversant engendrant un gang considérable du temps globale d'exécution.

Dans notre travail nous nous basons sur la coloration forte stricte pour obtenir un bon équilibre spatial d'espace d'états.

Plusieurs algorithmes basant sur cette coloration sont développés par Bouzenada, Bensouyad, Guidoum, Reghioua, Saïdouni [27], [28] et [37] dans le but de réaliser un bon partitionnement. Nous expliquons dans ce chapitre les algorithmes proposé par cette équipe.. Mais avant ça nous allons en premier lieu, rappeler quelques éléments de la théorie des graphes. Ensuite nous allons essayer de faire une simple et brève initiation à la coloration des graphes.

3.2 Initiation aux graphes

La théorie des graphes fait partie du domaine de la recherche opérationnelle. Elle s'intéresse à l'étude des structures appelées graphes. L'histoire de la théorie des graphes débute peut-être avec les travaux de Leonhard Eulerau 18ème siècle et trouve son origine dans l'étude du célèbre problème des ponts de Königsberg (Le problème consistait à trouver une promenade en partant d'un point donné de la ville, de traverser tous les ponts sans passer deux fois par le même et de revenir au point de départ). Euler représenta cette situation à l'aide d'un "dessin" où les sommets représentent les terres et les arêtes, les ponts comme le montre la figure 3.1.



Figure 3.1 les ponts de Königsberg et le graphe associé

3.2.1 Définitions

Un **graphe** permet de décrire un ensemble d'objets et leurs relations, un graphe G est un couple (V, E) où

- **V** : est un ensemble (fini) d'objets. Les éléments de V sont appelés les **sommets** du graphe.
- **E** : est un sous-ensemble de VxV. Les éléments de E sont appelés les **arêtes** ou **arcs** du graphe.

Une arête e du graphe est une paire e=(x,y) de sommets. Les sommets x et y sont les extrémités de l'arête.

La figure 3.2 montre l'exemple d'un graphe G avec un ensemble de sommets nommés de a à f, comportant 9 arêtes.

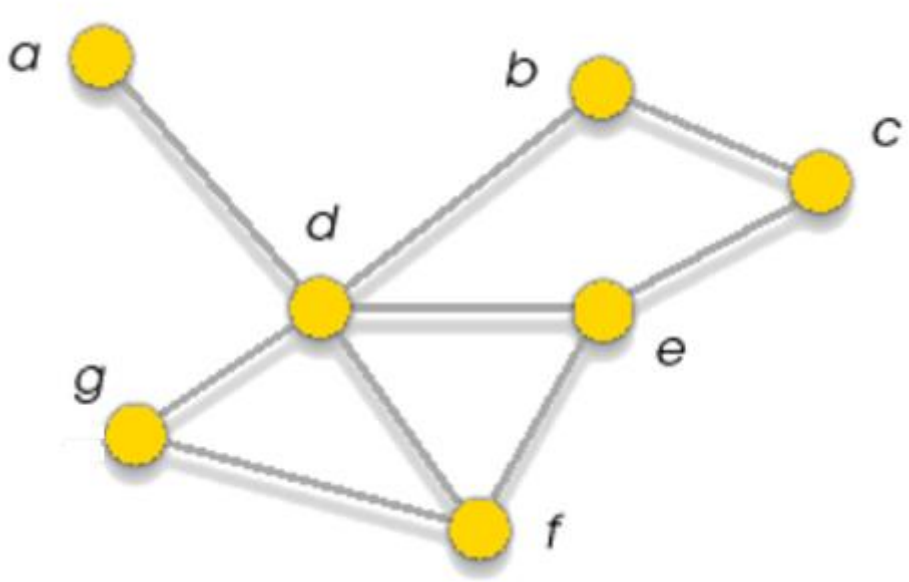


Figure 3.2 Exemple de graphe

- **G= (V, E).**
- **V= {a, b, c, d,e, f, g}.**
- **E={ (a,d), (b,c), (b,d), (d,e), (e,c), (e,f), (f,d), (d,g), (g,f) }.**

3.2.2 Différents types de graphes

Un **graphe orienté** G= (V, E) est un graphe dont l'ordre au sein des couples appartenant à E est intrinsèquement présent [21].Si $V = \{v_i \mid i \in I\}$ et si a = (v_i, v_j) et b = (v_j, vi), $i, j \in I$, alors a ≠ b. Autrement dit, un graphe oriente est un ensemble de sommets qui ne sont pas reliés entre eux par des arcs dans un ordre bien défini. Les sommets v_i,v_j sont appelés respectivement l'*origine (extrémité initiale)* et la *destination (extrémité terminale)*de l'arc a.

Un **graphe non orienté** G= (V, E) est un graph dont l'ordre au sein des couples appartenant à E n'est pas importante. Si E est une relation symétrique sur V, on dira que G est un graphe non oriente. Autrement dit, G est non orienté si

$$\forall v_1, v_2 \in V : (v_1, v_2) \in E \Rightarrow (v_2, v_1) \in E \text{ [21].}$$

On simplifie ainsi la représentation de la relation entre sommets par un arc non orienté appelé arête.

Un **multi-graphe** G = (V,E) est un graphe pour lequel il peut exister plus d'un arc reliant deux sommets donnés. Un exemple de représentation d'un multi-graphe est donné à la figure 3.3 [21].

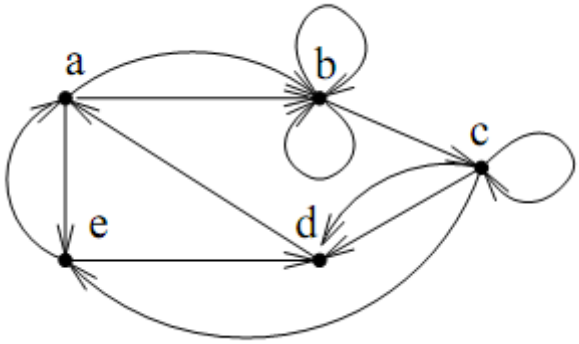


Figure 3.3 Exemple de multi-graphe

Un graphe G = (V, E) est dit **simple (ou strict)** s'il ne s'agit pas d'un multi-graphe et si G ne contient pas de boucle, c'est-à-dire que quel que soit $v \in V$, (v, v) n'appartient pas à E. Un exemple de graphe simple est donné à la figure 3.4 [21].

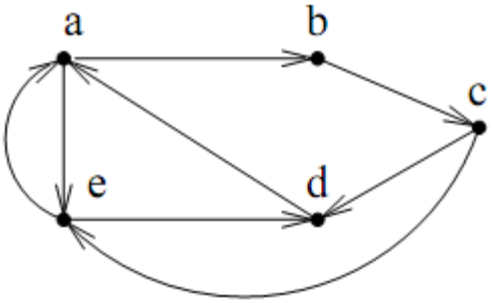


Figure3.4Exemple de graphe simple

Un **sous-graphe** G′= (V′,E′) de graphe G = (V,E) est un graphe tel que :

- V′ est un sous-ensemble de V ($V' \subseteq V$).
- E′ sont les arêtes induites par E sur V′, c'est à dire que les arêtes de E dont les 2 extrémités sont des sommets de E′ ($E' \subseteq E \cap (V' \times V')$).

Ainsi, si G′ est obtenu en enlevant à G certains sommets et/ou certains arcs ou arêtes alors G′ est un sous-graphe de G. Il est nécessaire d'enlever tous les arcs (ou arêtes) incidents à v Quand on enlève le sommet v de G, Par contre, on peut très bien enlever un arc (ou une arête) de G sans pour autant enlever le moindre sommet de G [21].

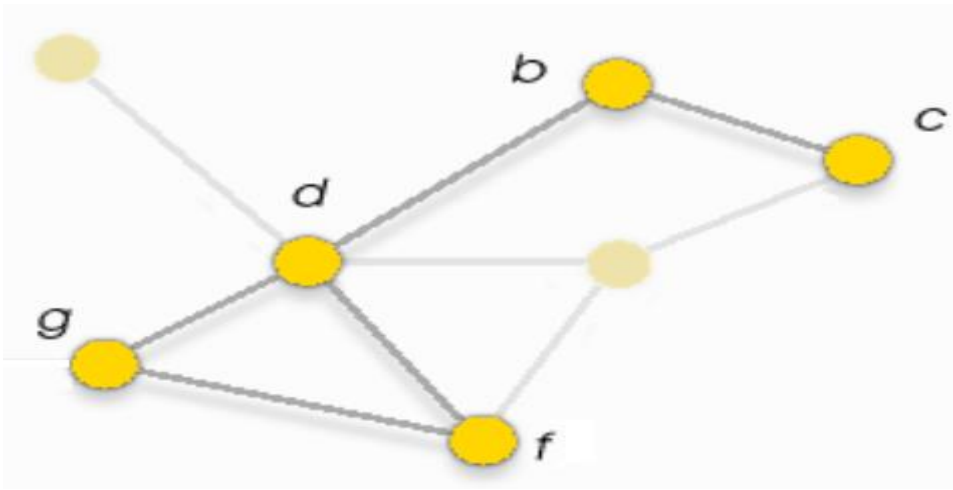


Figure 3.5 Le sous-graphe du graphe de la figure 3.2 induit par l'ensemble {b, c, d, g, f}.

Un **graphe partiel** G′= (V′,E′) de graphe G = (V,E) est un graphe tel que E′ est un sous-ensemble de E (E′ ⊆E).

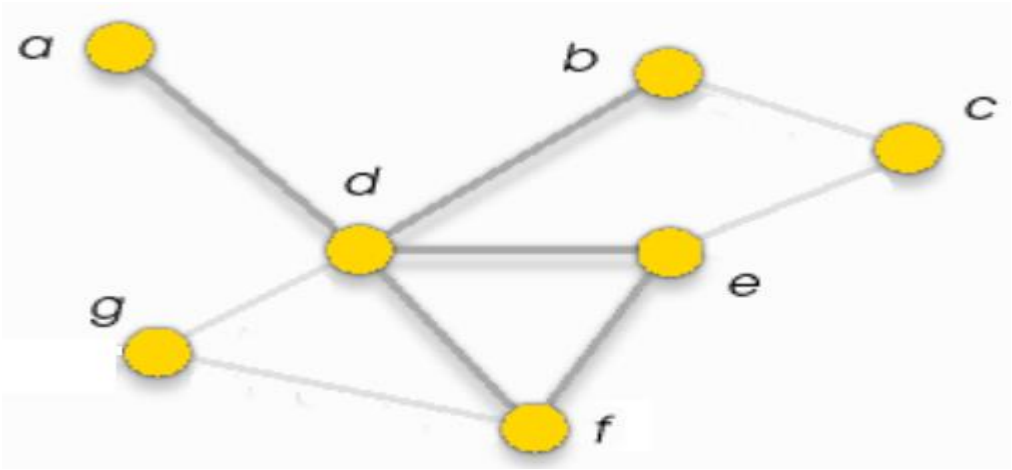


Figure 3.6 Le graphe partiel du graphe de la figure 3.2 défini par l'ensemble d'arêtes {(a,d), (b,d), (d,e), (e,f), (f,d)}.

3.2.3 Les prédécesseurs, successeurs d'un sommet:

Pour un graphe orienté:

Les prédécesseurs d'un sommet X sont l'ensemble des sommets à partir des quels on peut atteindre le sommet courant (X), qu'on note Γ-(X).

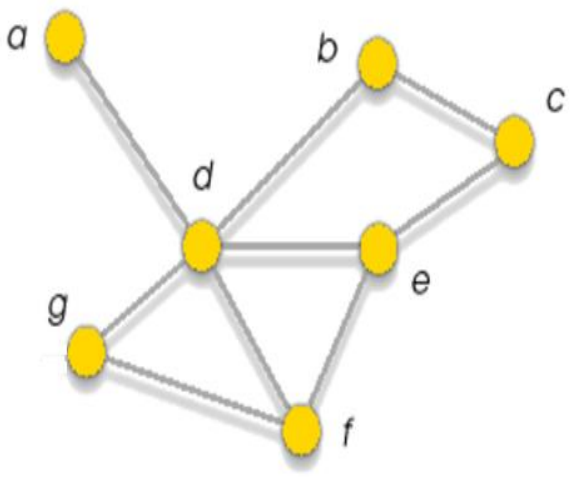
Les successeurs du sommet X sont les sommets auxquels on peut accéder à partir du sommet courant, qu'on note Γ+(X).

3.2.4 Représentation d'un graphe:

Un graphe peut être représenté par deux façons classiques: par une matrice d'adjacence ou par un ensemble de listes d'adjacence.

Représentation par matrice d'adjacence: Soit le graphe G= (V, E) contenant n sommets. La représentation par matrice d'adjacence de G consiste en une matrice M de taille n*n, ses éléments prennent deux valeurs 1 ou 0. Les lignes et les colonnes sont les sommets du graphe. Ainsi chaque élément de la matrice M [i, j] est égal à 1 si (i, j) $\in$ E. et M[i, j] = 0 sinon [22].

Si le graphe est valué (des valeurs sont associées aux arcs), on peut utiliser une matrice d'entiers, de telle sorte que M [i, j] soit égal à la valuation de l'arc (i, j) si (i, j) $\in$ E



	a	b	c	d	e	f	g	
	0	0	0	1	0	0	0	*a*
	0	0	1	1	0	0	0	*b*
	0	1	0	0	1	0	0	*c*
	1	1	0	0	1	1	1	*d*
	0	0	1	1	0	1	0	*e*
	0	0	0	1	1	0	1	*f*
	0	0	0	1	0	1	0	*g*

Figure 3.7 La matrice d'adjacence de G

Représentation par listes d'adjacence: Soit le graphe G= (V, E) contenant n sommets. Une autre idée pour représenter le graphe est les listes d'adjacences elle consiste en un tableau T de liste de sommets. Le tableau comporte n cases, Chacune des cases pointe vers une liste de sommets. La liste d'adjacence T[v_i] n'est rien d'autre qu'une liste chainée contenant la liste de tous les sommets successeurs de v_i [22].

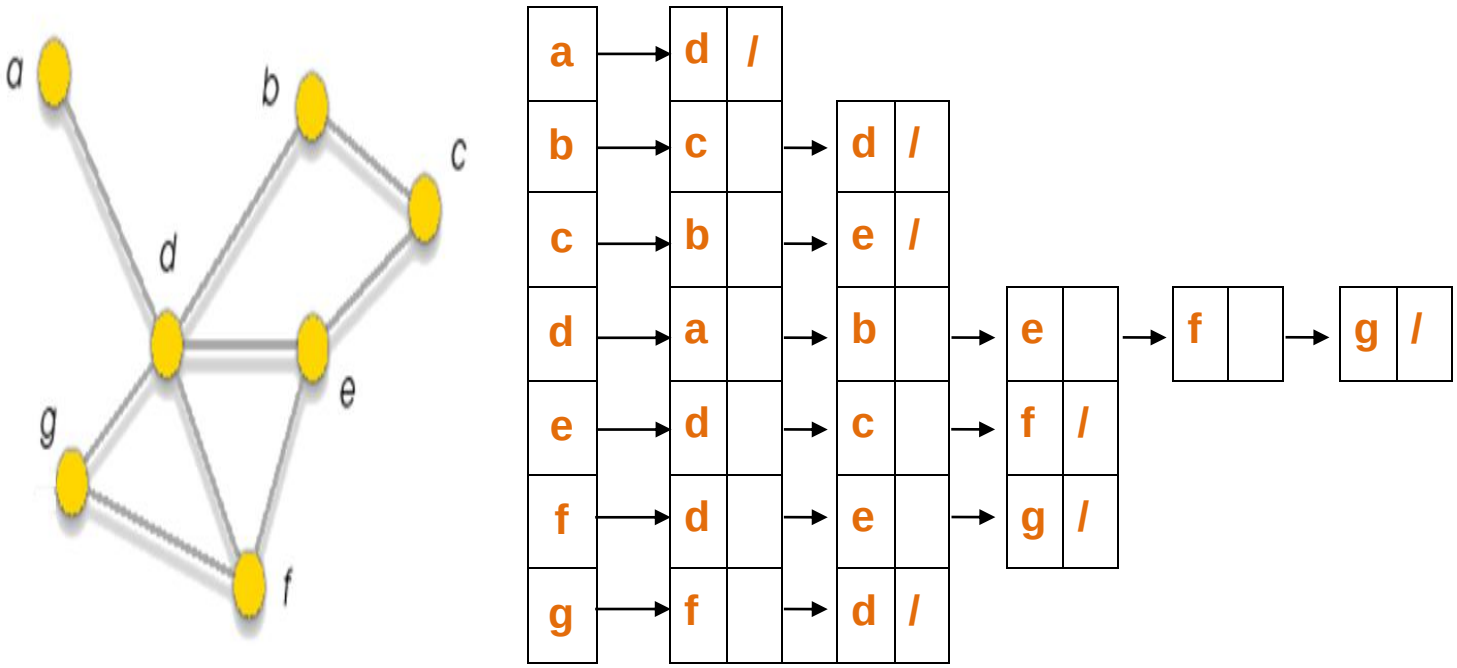


Figure 3.8 Exemple de graphe G et sa liste d'adjacence associée

3.3Coloration de graphe

La coloration de graphes remonte à l'année 1852. En observant la carte d'Angleterre Francis Guthrie remarqua qu'il peut colorer les comtés avec seulement quatre couléurs pour que deux comtés voisins soient de couleurs différentes. Le frère de Guthrie Frederick transmit la question à son professeur de mathématiques, Auguste de Morgan. A son tour Morgan mentionna ce problème dans une lettre adressée à Hamilton le 23 octobre 1852 [21]. Depuis De nombreux travaux ont été publiés lors du siècle suivant dans ce domaine.

3.3.1 Définition:

Il y a plusieurs façon de colorier les graphes On peut déjà vouloir colorier différents éléments d'un graphe (les sommets, les arêtes, ...). La contrainte la plus courante est celle de la propreté [23]. La coloration propre de sommets est une fonction qui attribue une couleur à chacun de ses sommets de telle sorte qu'une même couleur ne soit pas attribuée à deux sommets reliés par une arête [22].

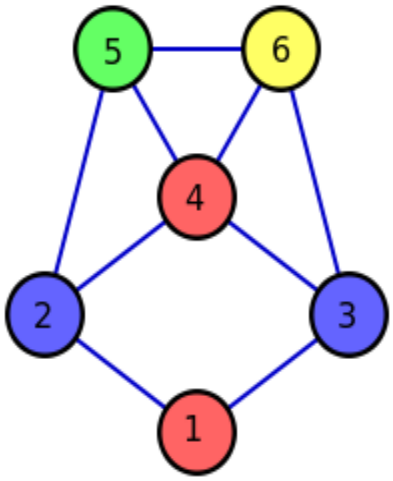


Figure 3.9 Exemple d'un graphe coloré

D'une autre manière la coloration propre d un graphe G = (V,E) est une fonction $f : V \longrightarrow C$ ou C est un ensemble de k couleurs, telle que pour toute arête (v_i, v_j)∈E : $f(v_i) \neq f(v_j)$. On dira que G est k-colorable [23].

Une k-coloration est aussi une partition des sommets en K ensembles indépendants.

3.3.1 Nombre chromatique d'un graphe:

Le nombre chromatique d'un graphe G = (V,E) est un minimum de couleurs nécessaires pour colorier un graphe G, il est noté χ(G). Le nombre chromatique est toujours compris entre 1 et le nombre de sommets. Autrement dit χ(G) est le plus petit entier k tel que G soit k-coloriable.

Le problème de coloration consiste à chercher à colorier un graphe en utilisant le moins possible de couleurs. Le problème de K-coloration est NP- difficile pour un nombre de couleurs K>3 [24].

3.4 Coloration forte stricte

Dans ce sous chapitre nous avons donc pour but de présenter la Coloration forte stricte (Strict Strong Coloring) proposé en 2009 par Haddad et Kheddouci, ils ont développé aussi un algorithme pour colorer les arbres. Bouzenada et al ont généralisé cette méthode de coloration pour qu'elle fonctionne sur tout type de graphe [26].

3.4.1 Coloration forte

Soit le graphe G= (V, E) k-coloration propre C1, C2,..., Ck, le graphe G est dit si pour chaque sommet u ∈ V il existe un indice i ∈ {1, 2,...,K} de telle sorte que u est adjacent àtous les sommets de Ci [25].

Le concept dela coloration forte a été introduit par I.Zverovich dans son article « A new kind of graph coloring » en 2006 Il a prouvé aussi la NP-complétude du problème de coloration forte pour k ≥4 [25].

3.4.2 Coloration forte stricte

La coloration forte stricte d'un graphe est une coloration forte telle qu'il n'y a pas de classe de couleur vide.

Plus formellement la K-coloration forte stricte (k-SSColoring) du graphe G est une k-coloration propreC_1, C_2,..., C_k telle que pour chaque sommet $u \in V$ il existe un indice $i \in \{1, 2,...,K\}$ de telle sorte que u est adjacent à tous les sommets de C_i et $C_i \neq \emptyset$ [25].

Le nombre chromatique fort strict (strict strong chromatic number) notée $\chi_{ss}(G)$ le nombre minimal de couleurs parmi les colorations fortes strictes [25].

On peut facilement voir, que la coloration forte stricte offre à chaque sommet une propriété de dominance (dominance property), c'est à dire pour chaque sommet *v*, il existe une classe de couleur non vide C_i tel que *v* est adjacent à tous les sommets de C_i [25].Autrement dit chaque sommet *v* domine la classe de couleur non vide C_i. Ce type de coloration peut être très utile pour de nombreuses applications comme la broadcasting (diffusion), Par exemple, dans une application de diffusion, un algorithme intuitif est que chaque sommet v ∈ V transmet le message reçu à la classe de couleur qu'il domine.

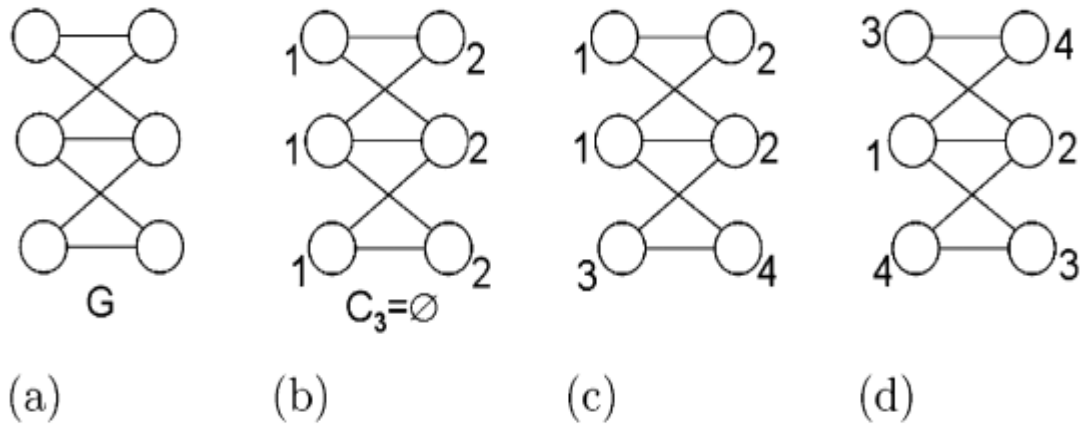


Figure3.10 (a) un graphe G, (b) une coloration forte de graphe G, (c), (d) deux colorations forte stricte de graphe G[25].

3.4.3 Quelques propriétés

Dans la figure 3.10 (b) la propriété forte est obtenue parle fait que chaque sommet domine une classe de couleur vide. Malheureusement, la domination d'une classe vide n'a pas de sens dans la pratique. La coloration forte stricte de graphe n'admet pas la classe de couleur vide, en ajoutant cette contrainte, la propriété de dominance induite par cette coloration devient plus efficace.

Soit G= (V,E) un graphe. L'ensemble des voisins du sommet u estnoté N (u). Soit Y⊆V et x∈V\Y. On note u~Y lorsque le sommet u est adjacent à tous les sommets de Y

Soit C une coloration propre de G. Un sommet u de G est dite forte stricte s'il existe un C_j non-vide tels que u est adjacent à tous les sommets de C_j c'est-à-dire $u \sim C_j$. Alors la coloration forte stricte de G est une coloration propre de G tel que tout sommet de G est forte stricte. On dit que la couleur j est une couleur dominée par un sommet u (ou u dominent i) si et seulement si $u \sim C_j$ [25].

Soit la K-coloration forte stricte C_1, C_2,..., C_k du graphe G, on définit la fonction D dominance comme suit:

$$D: \begin{cases} V \rightarrow \{1,2,...,k\}, \\ v \rightarrow D(v) = \min\{i: v \sim C\}. \end{cases}$$

Pour toute coloration forte stricte d'un graphe, l'ensemble de couleurs peut être divisé en deux sous-ensembles: l'ensemble de couleur dominé (DCS) et l'ensemble de couleur non dominé(NDCS) [25].

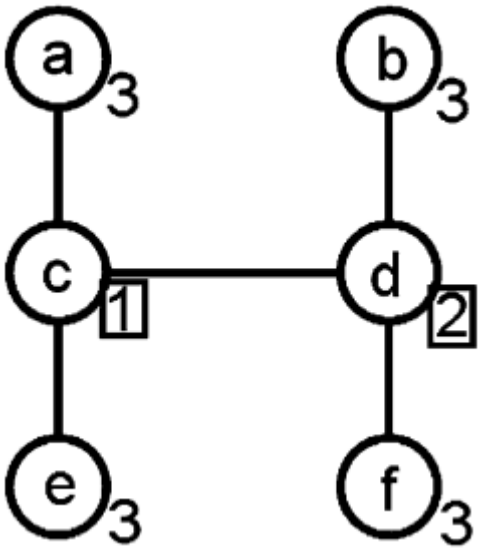


Figure3.11 coloration forte stricte d'un graphe G avec DCS = {1,2} et NDCS= {3}[25].

La figure 3.11 illustre la fonction de dominance, le graphe G est un graphe forte stricte colore on a C1={c}, C2={d}etC3={a,b, e, f}Il est facile de voir que tous les sommets sont forts stricts. En effet, D(a) =1, D (b) =2, D(c) =2, D(d)=1, D(e) =1etD(f) =2,La couleur 3 n'est pas alors une couleur dominé [25].

Haddad and Kheddoui ont démontré que trouver le nombre minimum exact de couleurs qui vérifient la propriété de dominance pour chaque sommet est un problème NP- difficile.

3.5 Algorithme de coloration forte stricte généralisé pour les graphes

Nous avons dit au chapitre précédent que Haddad and Kheddouci ont développé un algorithme de coloration forte stricte pour les arbres uniquement, dans ce sous chapitre, nous détaillons la généralisation de cette méthode de coloration pour tout type de graphe qu'elle ait été proposée par Bouzenada et al dans [26].

3.5.1 L'algorithme

Input: Un graph G = (V, E)
Output: A number of colors k
Initialization:
• *Satisfied:*Une variable qui permet de mémoriser l'ensemble des sommets ayant la propriété de dominance,initialisée à vide.
• *Colored:*Une variable qui permet de mémoriser l'ensemble des sommets colorés,initialisée à vide.
• *k*: Un entier initialisé à 0.
• $D \subseteq V$ *x* entier: un ensemblecontenant undegré desommetà un moment donné de calcul.
• *C*: Une variable qui permet de mémoriser l'ensemble des classes de couleur indexé par *k*.

Stepl: // Assurer la propriété de dominance
Repeat

$$\mathbf{D:} = \bigcup_{u \in (V - Colored)} \{(u, \deg(u) - |N(u) \cap Satisfied|\}$$ **……..…… (α)**

Select (u, n) such that u is a vertex with a maximal number **n** in the set **D…...….. (β)**
k:=k+1;
C [k]: = {u};// Attribuer une nouvelle couleur au sommet *u*……**(ϒ)**
Colored : = Colored U {u};
Satisfied : = Satisfied u N (u);
Until (Satisfied = V)

Step2: // coloration propre
Y :=(V - colored); // sommets non colorés.
While (Y≠ 0) **do**
k:=k+ 1;
B := 0; // ensemble de sommets.
Choisir un sommet v ∈Y de degré maximum ;
C [k]:= C [k] ∪ {u};
B:= B ∪ N (v);
Y:= Y- ({v} ∪ N (v));
While (Y 0) **do**
Choisir un sommet v ∈Y de degré maximum ;
C [k] := C [k] ∪ {u};
B := B ∪ N (v);
Y := Y- ({v} ∪ N (v));
End While

3.5.2 L’explication

Cet algorithme passe par deux étapes, il permet d'abord de vérifier la propriété de dominance puis il colore de façon optimale les sommets non encore colorés en utilisantun algorithme de coloration propre.

La première étape assure la propriété de dominance à la fin de cette étape, chaque sommet doit être adjacent à au moins une classe non vide de couleur [26].

La ligne **(α)** de l'algorithme calcule le degré de chaque sommet non colorés sans prendre en considération ses voisins appartenant à l'ensemble *Satisfied* (les sommets qui vérifient déjà la propriété de dominance on dit que ses sommets sont satisfaits). Après Le sommet *u* ayant le plus grand degré est alors sélectionné et utilisé dans la suite pour créer une nouvelle classe de couleur C_k. La couleur du sommet *u* est *k*(*u*est ajoutée à l'ensemble *Colored*). Les voisins de *u* N (*u*) dominent la couleur *k* donc ils vérifient la propriété de dominance et ils sont considérées comme ayant satisfaits à ce faitN (*u*) est ajoutée à l'ensemble *Satisfied*. Le processus est répété jusqu'à ce que tous les sommets deviennent des éléments de l'ensemble *Satisfied* [26].

À la fin de la première étape chaque sommet non coloré est adjacent à au moins un sommet coloré. Lors de la deuxième étape et pour compléter le processus de coloration, il ne reste qu’à colore l'ensemble des sommets qui ne sont pas encore colorés en utilisant un algorithme de coloration propre. Bouzenada et al ont appliqué l’Algorithme RLF (Recursive Largest First Leighton 1979).L'algorithme RLF est l'un des plus populaires parmi les algorithmes de coloration, il est notamment connu pour son efficacité.

La seconde étape de l'algorithme commence par la création de deux ensembles, Y l’ensemble des sommets non encore colorés et B un ensemble initialisé au début à l'ensemble vide. Après elle choisit et elle colore le sommet *u* de degré maximum, ensuite on déplace les voisins du sommet *u* Y vers B et le sommet *u* est retiré de l’ensemble Y. De la même manière on cherche un autre sommet de Y qui a le degré le plus grand, il sera colore de la même couleur de *u*, ses voisins aussi sont déplacés de Y vers B et on répète cette opération jusqu'à ce que l’ensemble Y soit vide. Enfin le contenu de l'ensemble B sera déplacé vers Y. Ce cycle est répété jusqu'à ce que l'ensemble Y soit vide [26].

Pour mieux comprendre le déroulement d'un algorithme on applique cet algorithme pas à pas sur un graphe de 8 sommets. Les résultats de chaque étape sont présentés dans les deux figures ci-dessous.

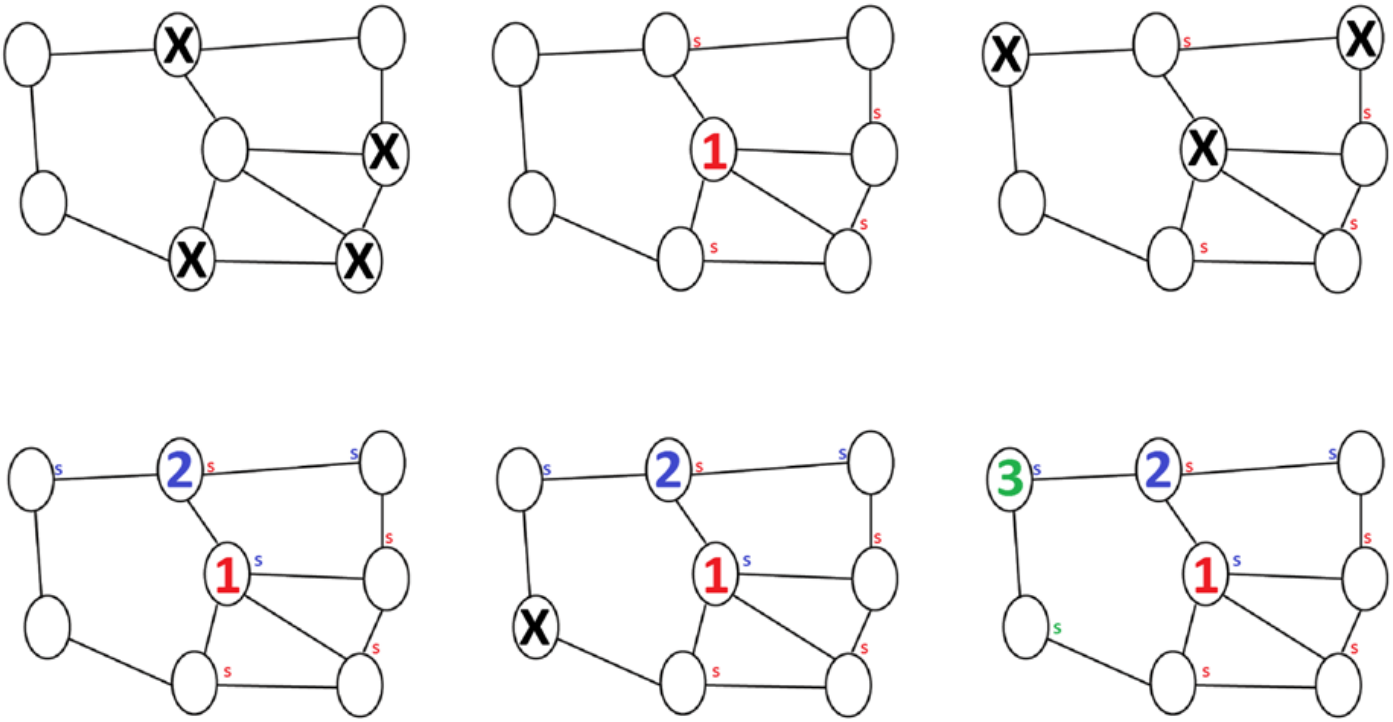


Figure 3.12 Le résultat de la première étape.

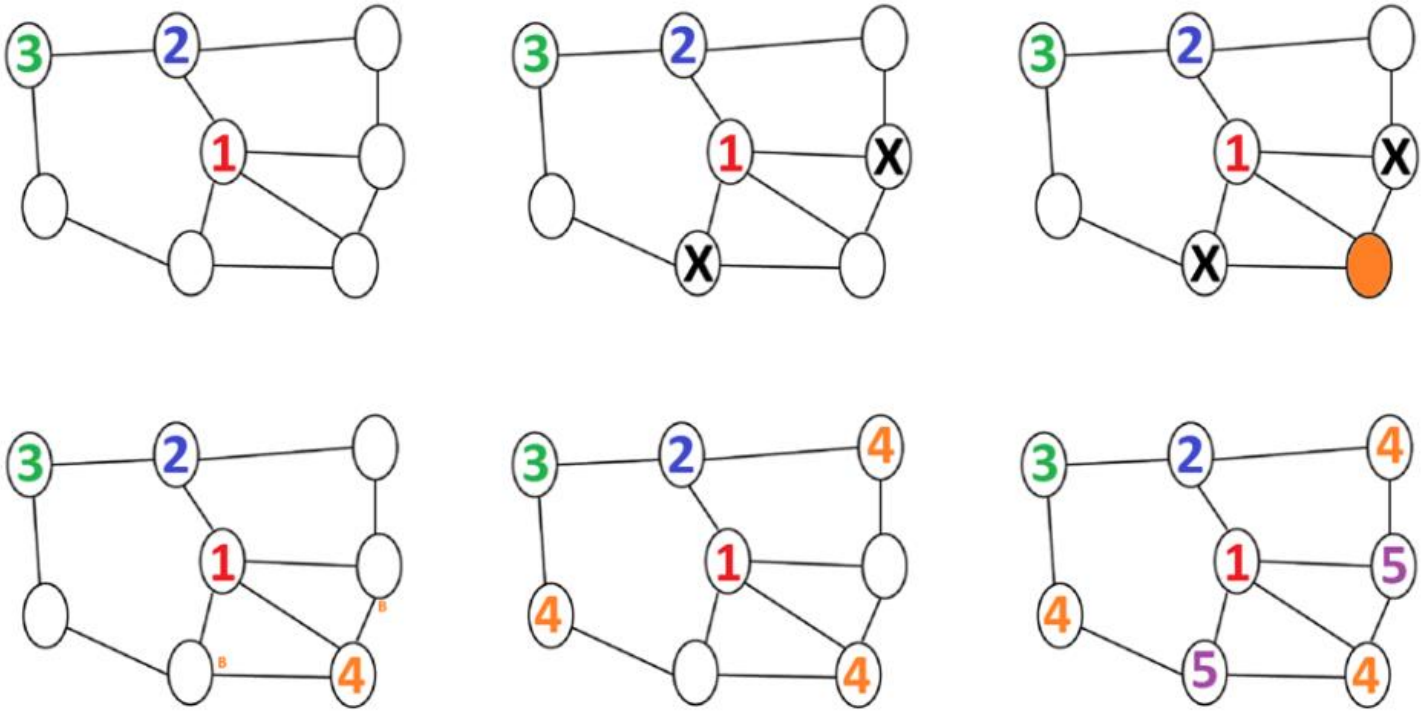


Figure 3.12 Le résultat de la deuxième étape.

3.6 Algorithme de distribution du graphe basée sur la coloration forte stricte

Après avoir proposé GGSSCA (Generalized Graph Strict Strong Coloring Algorithm) - qui à été présenté dans le paragraphe précédent - la même équipe a adopté la coloration forte stricte pour le problème de la distribution de graphe après constatation que la cette coloration est a un comportement adapté à ce genre de problème [27].

L'idée de cet algorithme consiste à utiliser la première étape de l'algorithme GGSSCA qui donne le nombre initial de partitions de sorte que chaque couleur présente une de ses parties. Le sommet coloré est considéré comme le centre de la partition et ses voisins sont placés avec lui dans la même partition. Ensuite un processus de regroupement (**resp.**division) est utilisé pour fusionner (**resp.**diviser) les parties initiales si le nombre de couleurs est supérieur (**resp.** inférieur**)**au nombre d'unités de travail (workers) [27].

L'algorithme global :

Input: Graph G= (V, E)
W: the number of workers.
Output: π_w distribution (partition) de graph G en W sous-graphs (partition).
Initialization:
DC: Le nombre de couleurs obtenues après exécution de la première étape de GSSCA.
Centers: est l'ensemble dessommets, obtenus parGGSSCA.
Begin
Step1: Initialization process
Centers: = GGSSCA.step1 (G);
DC:=|Centers|;
Initial distribution construction process; // Crée la distribution initiale π_{DC} de G;……**(α)**
Step2: Optimization process
If (DC> W) **Then**
Grouping process ();…….....**(β)**
Else
If (DC < W) **Then**
Splitting process ();……..……..**(γ)**
End if
End if
End

Comme on peut le voir l'algorithme se déroule en deux étapes : initialisation et optimisation.

3.6.1 L'initialisation

En appliquant la première étape de l'algorithme GGSSCA, on obtient un nombre initial de partitions et un centre pour chacune de ses partitions. Chaque partition (V_i) contient initialement un centre qui est un sommet coloré ($u \in$ *Centers)*. Puis les voisins N (u) seront ajoutés à V_i avec élimination de chaque voisin qui est déjà associé à une autre partition. Ce processus garantit que toutes les partitions obtenues sont indépendantes [27].

Processus de construction de la distribution initiale :

Input: Graph G= (V, E).
DC.
Centers.
Output: Partition π_{DC} of G / | π_{DC} |=DC
Initialization:
π_{DC} = {V_1, V_2,….V_{DC}}: telle que pour tout $V_i \in \pi_{DC}$, V_i est une partition indépendante. Au départ V_i = {u / $u \in$ Centers}.
Belong (v): est une fonction booléenne égale à vrai lorsque le sommet v appartient déjà à une partition. Belong (v) = true if $v \in V_i$ / i ∈{1..DC}.
Conflicting _vertices u: l'ensemble des voisins de u qui appartiennent déjà à une partition.
i: entier initialisé à 0;
Begin
While (i<DC)
Conflicting _vertices $_u$ = {$v \in$ N (u)/ u ∈ V_i et Belong (v)};
$V_i = V_i \cup$ (N (u) – Conflicting vertices $_u$);
End while
End

Pour expliquer le déroulement de l'algorithme, nous allons reprendre le même exemple proposé par Guidoum et al dans [27]. Le résultat de la première étape de GGSSCA sur les graphes de la figure 3.14 est montré dans la figure3.15.

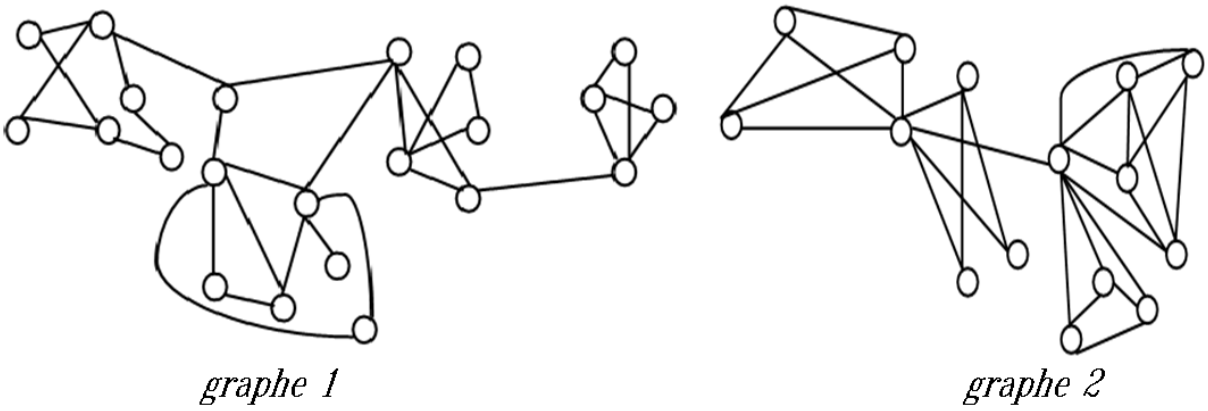


Figure 3.13 deux graphes non distribué[27].

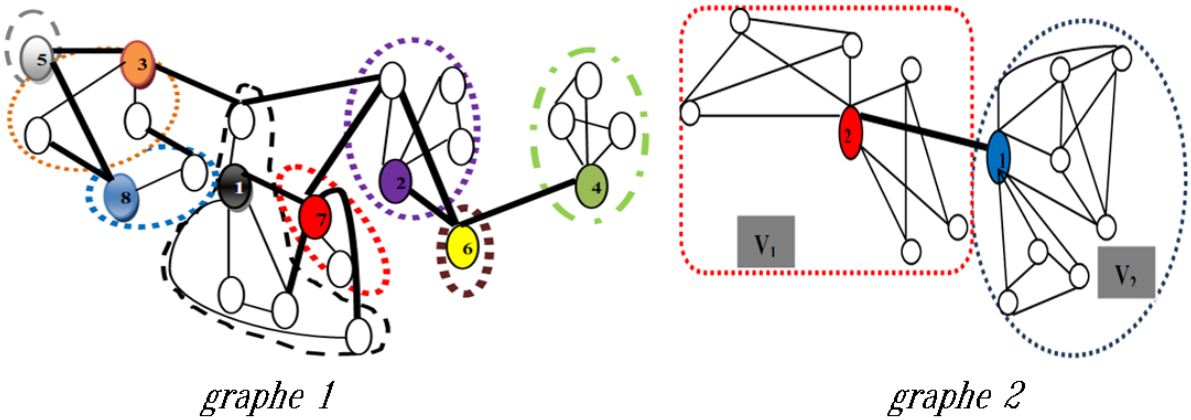


Figure 3.14 les deux graphes après l'initialisation [27].

3.6.2 L'optimisation

Dans cette étape, le choix entre le processus de regroupement et de division se fait selon le nombre de couleurs dominées DC. Si le nombre d'unités de travail W est inférieure à DC alors il y a un regroupement sinon il ya un processus de division [27].

Processus de regroupement

L'algorithme a une boucle principale qui se termine lorsque le nombre de partition est égal à W.

A chaque itération, l'algorithme sélectionne deux parties V_i et V_j à fusionner. Premièrement, il sélectionne la partition (V_i)la moins connectée c'est-à-dire la partition qui a le plus petit nombre des arcs locaux (un arc local relie deux états locaux). Ensuite,

il recherche la partition voisine la plus connectée (V_j), ses deux partitions seront regroupées pour construire une seul partition équilibrée [27].

Processus de regroupement :

Input: π_{DC} initial distribution of G with DC parts
W number of workers
Output: π_w distribution de G en W workers.
Begin
$\pi = \pi_{DC}$;
Repeat
Select V_i which has the minimum number of local edges |E*ii*|;
Select V_j which has the maximum number of cross (external) edges |E*ij*|;
Merge $V_i$and V_j into one part;
$|\pi| = |\pi| - 1$;
Until ($|\pi| = W$)
$\pi_w = \pi$;
End

Le résultat de l'exécution complète du processus de regroupement sur le graphe 1 de la figure 3.13 est montré dans la figure 3.14.

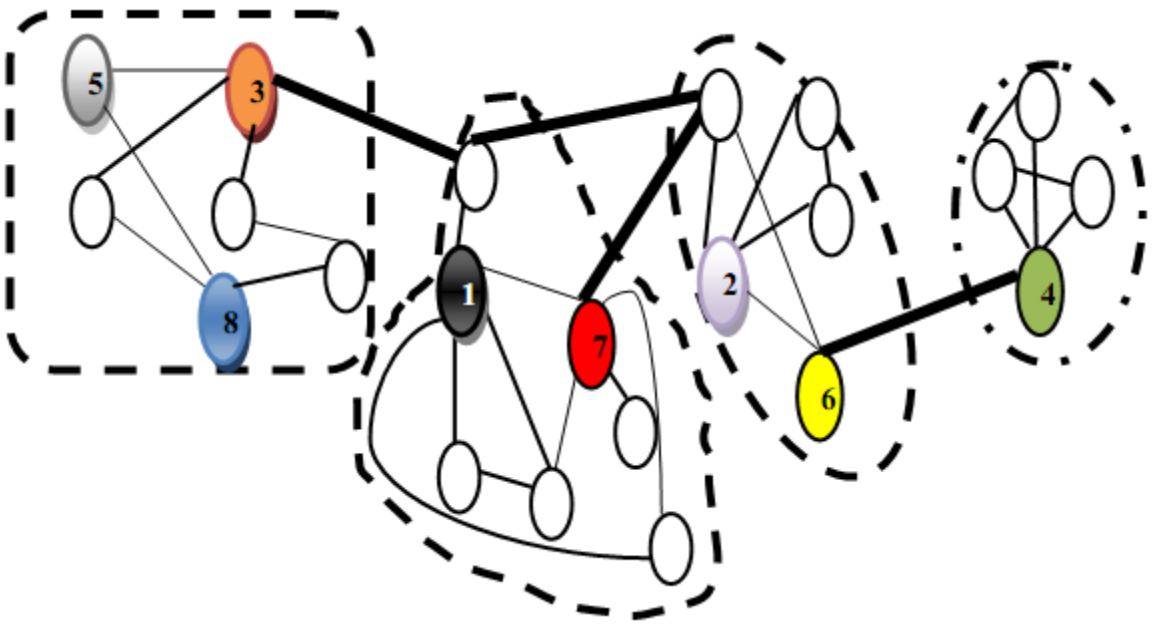


Figure 3.15 Le résultat du processus de regroupement [27].

Processus division

Ce processus a pour but de sélectionner la partition (V_i) qui sera divisée en deux. Cela augmente le nombre de partition de 1. Ce processus sera répété jusqu'à ce qu'on obtienne un nombre de partition égal à W. A chaque itération, l'algorithme sélectionne la partition (V_i) ayant le nombre maximum de sommets, et il recherche les sommets qui ont une forte connexion afin de les transférer dans la nouvelle partition. La distribution sera mise à jour en ajoutant la nouvelle partition (V_{i+1}) et en supprimant tous les sommets existants dans la partition principale [27].

Processus division:

Input: π_{DC} initial distribution of G with DC parts
W number of workers
Output: π_w distribution de G en W workers.
Begin
$\pi = \pi_{DC}$;
Repeat
Select V_i which has the maximum number of nodes;
Compute the connectivity between nodes in V_i;
Put the vertices, strongly connected, in the same partition $V_{|\pi|+1}$;
Remove all vertices existing in $V_{|\pi|+1}$ from V_i;
$|\pi| = |\pi| + 1$;
Until ($|\pi| = W$)
$\pi_w = \pi$;
End

Figure 3.16 Montre le résultat après exécution du processus de division.

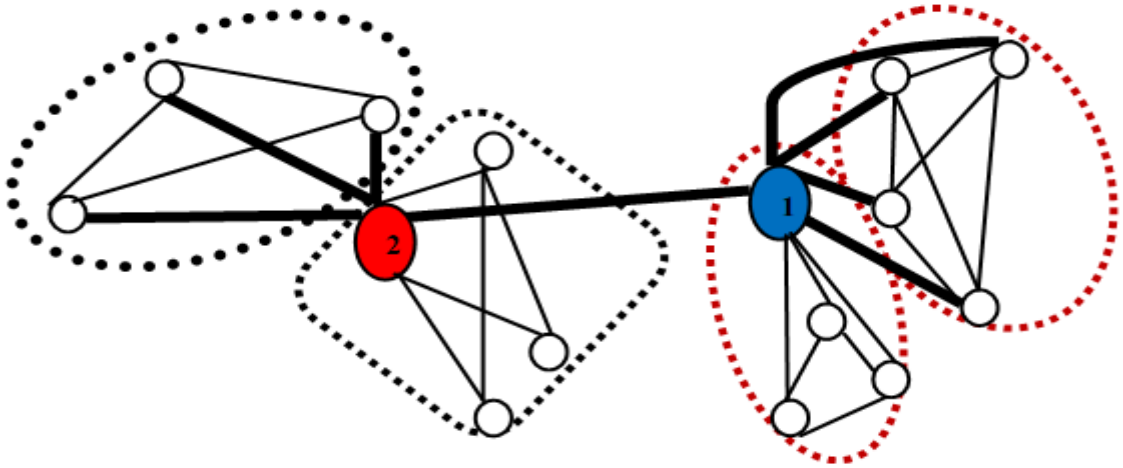


Figure 3.16 Le résultat du processus de division[27].

3.7 Algorithme évolutionnaire basé sur la coloration forte stricte

Comme tout autre algorithme évolutionnaire, l'idée de cet algorithme est de faire évoluer un ensemble de configurations considérées comme la population initiale (construit en utilisant *InitPopulation*) pour créer de nouvelles et de meilleures configurations[37].

Input: Graph $G = (V, E)$.
Output: Best configuration found with minimal number of colors k.
Initialization
S^*: Best current individual ;
k^*: Best current value ;
Best (*Pop*): returns the best individual of the population *Pop* which respects SSColoring constraints with minimal colors;
Pop: Current population ;
Pct_sel: Selection proportion ;
P_{cs}: Crossing probability ;
P_{ct}: Correction probability ;
Begin
Pop = InitPopulation (|Pop|);
S^*= *Best* (*Pop*);
k^*= $|S^*|$;
While not_stop_condition () **Do**
Sub_p = Random_Selection (*Pop*, *Pct_sel*);
Cr_p = Crossing (*Sub_p*, P_{cs});
Correct_p = Correction (*Sub_p* ∪ *Cr_p*, P_{ct});
If |*Best* (*Cr_p* ∪ *Correct_p*)| <k^* **Then**
S^*= *Best* (*Cr_p* ∪ *Correct_p*)
k^*= $|S^*|$;
End If
Pop = Update (*Pop*,*Sub_p* ∪ *Cr_p* ∪ *Correct_p*);
End While
End

Une fois que le processus est terminé *InitPopulation*, l'algorithme entre dans une boucle itérative jusqu'à ce que la condition d'arrêt soit vérifiée (quand un nombre de générations prédéfini MaxG en est atteint) [37].

A chaque génération, un certain nombre de configurations est sélectionné au hasard pour être des parents et retiré de la population actuelle. Après cela, l'opérateur de croisement est appliqué sur les parents sélectionnés pour générer de nouvelles configurations (descendants) avec une probabilité de croisement P_{cs} , Ensuite, l'opérateur de correction est utilisé, pour un nombre fixe d'itérations *MaxIter* et avec une probabilité P_{ct} de correction, afin d'améliorer la qualité des parents sélectionnés ainsi que leurs descendants produits. L'évaluation de ces configurations se fait selon la fonction de fitness qui calcule le nombre de sommets contradictoires. Enfin, la population est mise à jour par l'insertion des éléments dans la population actuelle. Ces éléments sont choisis parmi les parents enlevés, descendants et configurations corrigées pour faire partie de la population qui va évoluer à la prochaine génération [37].

3.7.1 Représentation de coloration forte stricte

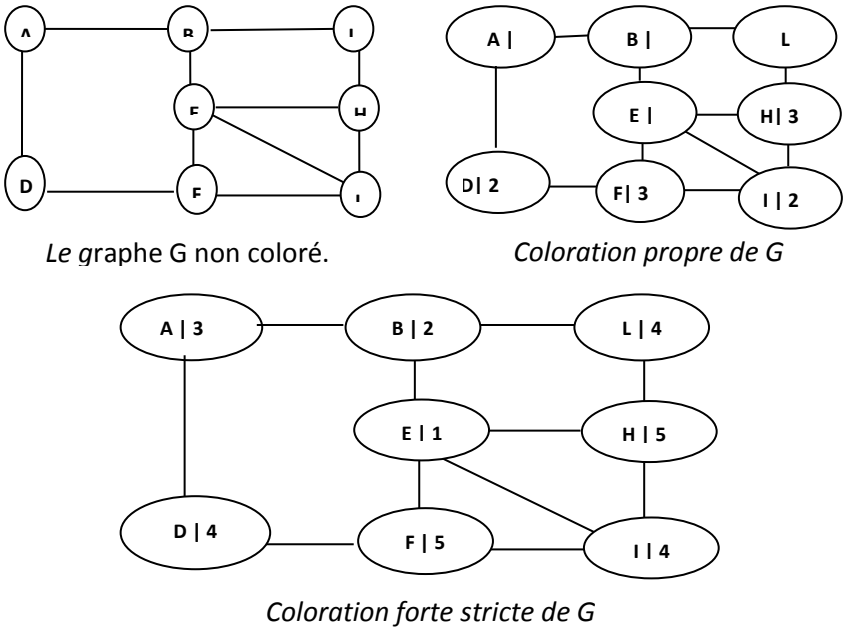
En plus d'une coloration propre, la SSColoring doit vérifier la propriété de dominance pour tous les sommets du graphe. Pour cette raison, les informations concernant cette propriété doivent être représentées.

A cet effet, une représentation spéciale a été proposée dans [37]. Une configuration k-coloration forte stricte sera une matrice à deux lignes et k colonnes où la première ligne indique que chaque couleur a une classe satisfaite spécifique Sat_i(c'est-à-dire : chaque sommet, appartenant à cette classe satisfaite Sat_i domine la classe de couleur C_i) et la seconde ligne est une partition de sommets en classes de couleurs.

Pour plus d'illustrations, soit le graphe de la figure 4.7 caractérisé par V = {A, B, L, D, E, F, I, H}.Ses colorations propres et fortes strictes sont présentées sur la même figure. Leurs représentations correspondantes sont indiquées dans le tableau 1.

<table>
<tr><th>colorations</th><th>représentation des configurations</th><th>k</th></tr>
<tr><td>colorations propre</td><td>
<table>
<tr><td>A, E,L</td><td>B, D, I</td><td>F, H</td></tr>
</table>
</td><td>3</td></tr>
<tr><td>colorations forte stricte</td><td>
<table>
<tr><td>i / $class$</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr>
<tr><td>Sat_i</td><td>B, F, I, H</td><td>A, E, L</td><td>D</td><td>/</td><td>/</td></tr>
<tr><td>C_i</td><td>E</td><td>B</td><td>A</td><td>L, D, I</td><td>F, H</td></tr>
</table>
</td><td>5</td></tr>
</table>

Tableau 3.1 Représentation des colorations propres et fortes stricte [37].



*Le g*raphe G non coloré. *Coloration propre de G*

Coloration forte stricte de G

Figure 3.17 La coloration propres et fortes stricte du graphe G [37].

3.7.2 Opérateur d'initialisation

Dans l'algorithme proposé, la population *Pop* a une taille constante fixe| *Pop*|, c'est un ensemble de configurations ayant la propriété de dominance. Cette propriété devrait être préservée tout au long du processus de calcul [37].

L'opérateur *InitPopulation* (| *Pop* |) initie la population avec | *Pop* | configurations utilisant "*Randomized configuration construction heuristic*" pour créer chaque configuration. Cette heuristique commence avec χ (nombre chromatique) classes de couleur vides(C_1 = ... $C_\chi = \emptyset$) et χ classes satisfaits initialement égales à l'ensemble vide (Sat_1 = ... $Sat_\chi = \emptyset$). Autrement dit tous les sommets du graphe appartiennent à l'ensemble des sommets non satisfait *Not_sat,* et ils ne sont pas aussi colorées, ils appartiennent donc à l'ensemble des sommets non colorés *Not_col* [37].

Tout d'abord, on choisit au hasard un sommet *v* non satisfait (v ∈ *Not_sat)*, on le met dans Sat_i et on le retire de *Not_sat*. Puis, tous ses voisins non colorés (N (*v*) ∩ *not_col*) sont mis dans C_i et retirés de *Not_col*. Ce processus est répété jusqu'à ce que χ sommets sont affectés à Sat_i [37].

Ensuite, un autre procédé est appliqué pour satisfaire le reste de sommets. Pour chaque sommet *v* non satisfait [37]:

- Mettez *v* dans Sat_j si il existe *j* tel que v domine C_j (v ~ C_j).
- Sinon créez i où *v* est inséré dans Sat_i et ses voisins non colorés dans C_i.

Enfin, les sommets non encore colorés sont insérés dans une nouvelle classe de couleurs C_i(Sat_i est vide) Cette étape peut également générer des conflits dans la configuration du produit final[37].

Input: Graph $G = (V, E)$.
Chromatic number χ.
Output: Configuration S.
Initialization
S: Initially, For l=1 to χ Do $Sat_l = \emptyset$ and $C_l = \emptyset$;
Not_sat: The set of vertices that have not the dominance property. Initially, *Not_sat*= *V*;
Not_col: The set ofuncolored vertices. Initially, *Not_col*= *V*;
i: Integer initialized to 0;
Begin
While (i< χ) **Do**
v = rand (Not_sat); % Take randomly a not satisfied vertex v.
i= i+1;
Satisfying_vertex (v, i); % Satisfying_vertexis a procedure which aims to satisfy the vertex vby attributing the color ito its neighbors.
End While
While (*Not_sat* $\neq \emptyset$) **Do**
If ($\exists$ j|*v* $\in$*Not_sat* and **v** ~ C_j)**Then**
$Sat_j = Sat_j \cup \{v\}$; % Put *v* in Sat_j.
Not_sat= *Not_sat*-{*v*}; % Remove *v* from *Not_sat*.
ELSE
i= i+1;
Satisfying_vertex (v, i);
End If
End While
If (Not_col $\neq \emptyset$) **Then**
Put all (*v* $\in$ *MNot_col*) in $C_i$such that the corresponding$Sat_i = \emptyset$;
End If
End
Procedure:Satisfying_vertex (*v, i*)
Begin
$Sat_i = \{v\}$; % Put *v* in Sat_i.
If ($N(v) \cap Not_col \neq \emptyset$) **Then**
$C_i = (N(v) \cap Not_col)$; % Put not colored neighbors of *v* in C_i.
Not_col= Not_col – C_i; % Remove not colored neighbors of *v* from *Not_col*.
ELSE
u = rand (N (v)); % Choose randomly *u* one of neighbors of *v*.
$C_{C(u)} = C_{C(u)} - \{u\}$; % Remove *u* from its old color class $C_{C(u)}$.
$C_i = \{u\}$; % Put *u* in C_i.
End If
Not_sat= Not_sat-{*v*}; % Remove *v* from *Not_sat*.
End

Randomized configuration construction heuristic

La population produit comprend des configurations qui vérifient la propriété de dominance sans assurer de coloration propre. Cette heuristique sera appliquée pour construire les configurations de la population initiale[37].

3.7.3 Opérateur de croisement

L'opérateur de croisement est très important dans l'évolution d'une population. Il combine alors entre deux individus qui peuvent donnes naissance à des nouveaux individus qui profitent des avantages des deux parents. En général, le croisement nécessite d'abord l'identification de quelques propriétés importantes pour résolution du problème et ensuite le développement d'un mécanisme de croisement de façon à favoriser à la transmission de ses propriétés de père en fils [37].

Dans notre cas, une configuration est considérée comme une partition de sommets en classes de couleur et classes satisfaits [37].

Soit les deux configurations S^1= $\{(Sat_1^1,C_1^1),\ldots(Sat_k^1, C_k^1)\}$ et S^2=$\{(Sat_1^2,C_1^2),\ldots(Sat_{k'}^2,C_{k'}^2)\}$qui sont sélectionnées au hasard parmi la population utilisant l'opérateur de sélection. Notant queS1 et S2 peuvent avoir différentes tailles. L algorithme de croisement génère le nouvel individu S= $\{(Sat_1, C_1),\ldots,(Sat_n, C_n)\}$ comme suit[37].

Input: Configurations: S^1= $\{(Sat_1^1,C_1^1),\ldots(Sat_k^1, C_k^1)\}$ and S^2= $\{(Sat_1^2,C_1^2),\ldots(Sat_{k'}^2,C_{k'}^2)\}$.
Output: S= $\{(Sat_1, C_1),\ldots(Sat_n, C_n)\}$.
Begin
l = 1;
Do
Sat_l=Sat_i^j; % Sat_i^j has a maximum cardinality in $S^j|j\in\{1,2\}$.
C_l =C_i^j ;
Remove all *elements* of Sat_i^j and C_i^j from S^1 and S^2;
l = l+1;
Until ($\cup_{f=1}^{l-1} Sat_f$ = = V)
While($Not_col \neq \emptyset$)**Do**
If($\exists j|v\in Not_col$ and $v \sim Satj$**Then**
Cj= $Cj\cup\{v\}$; % Put v in Cj.
ELSE
C_l= $C_l\cup\{v\}$; % Put v in C_l such that correspondingSat_l= Ø.
End If
End While
End

L'algorithme de croisement construit étape par étape, la classe de couleur et la classe satisfait de l'enfant. A l'étape *f*, il construit les classes Sat_f et C_f : il choisit la classe satisfait Sat_i^j, qui a le nombre maximal de sommets dans l'un des parents S^j où j ∈ {1,2}, pour devenir la classe satisfait Sat_f de l'enfant *S*. Selon le choix de *i* et *j*, la classe de couleur C_f est la classe de couleurC_i^j. Les sommets appartenant à Sat_i^j etC_i^j sont retirés de leurs parents S^1 et S^2 [37].

À la fin du processus, l'enfant *S* s'assure que tous les sommets sont colorés en ajoutant les sommets non encore colorés à C_i (si *i* existe) tel que v ~ Sat_i, sinon ils seront ajoutés a une nouvelle classe de couleur C_i. *S* vérifie la propriété de dominance parce que tous les sommets sont satisfaits ($\bigcup_{i=1}^{k} Sat_i = V$)[37].

En appliquant l'opérateur de croisement sur deux configurations S^1 et S^2du graphique de la figure 4.7 (Le résultat est montré dans le tableau 4.2), cinq classes de couleur sont obtenues lors de la création de l'enfant *S*. A la première étape, la colonne ({A, E, L}, {B}) du parent S^1est choisi et affectée à la première colonne de l'enfant *S*.car il a le nombre maximal de sommets satisfaits dans S^1 et S^2.Cette colonne est composée deSat_1^1= {A, E, L} and C_1^1= {B}. Puis les sommets appartiennent àSat_1etC_1 sont retirés à la fois de S^1 et S^2comme suit : en S^1, on enlève la colonne complète. en S^2, on enlève les sommets de leurs classes respectives [37].

De la même manière, les trois autres classes sont construites, A la fin de quatrième étapes, tous les sommets sont affectés à des classes satisfaits.

Pour les sommets A et H les seuls qui restent non affectés à une classe de couleur, ils sont attribués à une nouvelle classe de couleur C_5 tels que Sat_5= Ø. En effet, leur attribution à une classe de couleur existante conduit à perdre la propriété de domination de certains sommets [37].

3.7.4 Opérateur de Correction

Le but de l'opérateur de correction est d'améliorer la qualité des individus de la population. Cet opérateur est appliqué pour un maximum d'itérations *MaxIter* avec une probabilité de P_{ct}. Les configurations concernées par l'amélioration sont celles appartenant à l'ensemble des parents sélectionnés et des autres qui sont produits par le croisement [37].

A chaque itération, la configuration générée remplace la configuration actuelle. Cette configuration est obtenue en déplaçant certains sommets d'une classe de couleur à une autre. Plus précisément, à chaque itération, une classe de couleur en conflit C_i est choisie au hasard pour traiter et éliminer tous les sommets en conflit. Après avoir choisi C_i, le sommet *x*, appartenant à C_i ($x \in C_i$)ayant le conflit maximale avec d'autres sommets dans la même classe de couleur (C_i), est déplacé à une autre classe de couleur C_j [37]tel que:

- *x* n'est en conflit avec aucun sommet de la classe de couleur C_j. Cette condition permettra d'assurer la propriété de coloration propre.

- x domine la classe satisfait correspondante Sat_j ($x \sim Sat_j$). Cette condition est requise pour préserver la propriété de dominance.

Si une telle classe d'opérateur ne présente pas entre les classes de couleurs existantes, une nouvelle colonne (Sat_k, C_k) est créée avec une classe de couleur C_k contenant le sommet x ($C_k = \{x\}$) et une classe satisfait vide ($Sat_k = \emptyset$)[37].

Le processus relatif aux sommets en conflit dans la même classe de couleur C_i sera exécuté d'une manière itérative jusqu'à ce qu'aucun sommet en conflit ne soit trouvé dans C_i. Après cela, de nouvelles itérations seront exécutées pour d'autres classes de couleurs en conflit jusqu'à ce que les classes de couleur en conflit soient terminées ou qu'un maximum d'itérations soit atteint [37].

1ere iteration

Parent S^1	A, E, L	D, I	B	H	F	/
	B	F	A	E, L	I	D, H
Parent S^2	B, H	E, I	D	A, L	F	
	E, L	F, H	A	B	D, I	
EnfantS	A, E, L					
	B					

2eme iteration

Parent S^1		D, I	B	H	F	/
		F	A	E, L	I	D, H
Parent S^2	B, H	I	D		F	
	E, L	F, H	A		D, I	
Enfant S	A, E, L	B, H				
	B	E, L				

.
.

5eme (dernière) itération

Parent S^1						/
			A			H
Parent S^2						
		H	A			
Enfant	A, E, L	B, H	D, I	F	/	
	B	E, L	F	D, I	A, H	

Tableau 3.2 Exemple illustratif de l'opérateur de croisement appliqué sur le graphe de la figure 4.7 [37].

3.8 Conclusion

La dernière partie de ce chapitre a présenté la coloration forte stricte qui peut être très utile pour de nombreuses applications le broadcasting, la dissémination de données, …etc. En se basant sur la coloration forte stricte Guidoum et al ont proposé plusieurs algorithmes, nous nous focalisons sur l'algorithme évolutionnaire séquentiel de distribution basée sur la coloration forte stricte et nous allons proposer une approche pour rendre cet algorithme parallèle. Nous allons parler de cet algorithme dans le détail dans le chapitre suivant.

CHAPITRE 4
ALGORITHMES EVOLUTIONNAIRE TOLERANT AUX FAUTES

4.1 Introduction

Les systèmes distribués modernes sont dans la plupart des cas dynamiques dans le sens où l'environnement d'exécution change au cours du temps. Le nombre de stations de travail dans les infrastructures de calcul n'est pas fixé (les sites peuvent rejoindre ou quitter le système à un moment donné) et cela même durant l'exécution. De plus, la charge des processeurs et de la communication varie aussi selon l'activité d'autres utilisateurs. Cette dynamicité peut également causer des défaillances, Lorsqu'un site quitte le système brusquement soit en se déconnectant volontairement ou lors d'une panne au moment de la génération distribuée de l'espace d'état, il engendre deux problèmes principaux :

- une partie du graphe disparait avec les machines défaillantes.
- L'incohérence du système l'empêche de poursuivre sa génération correctement.

Notre contribution est constituée de trois grandes étapes : nous proposons au début une idée pour paralléliser l'algorithme évolutionnaire de distribution basée sur la coloration forte stricte présenté dans le chapitre précédent ensuit, nous distribuons à la volée cet algorithme et à la fin nous présentons un mécanisme tolérant aux pannes.

4 .2 Distribution de l'algorithme évolutionnaire basée sur la coloration forte stricte

Même si Les algorithmes évolutionnaires prouvent toujours leur efficacité quand il s'agit de résolution de problèmes d'optimisation, les temps de calcul peuvent être très importants. Une façon de minimiser ce temps est l'utilisation du parallélisme.

Les algorithmes évolutionnaires parallèles peuvent trouveur des solutions plus appropriés et plus robustes par rapport a leurs équivalents séquentiels [29]. Ils ont aussi

la possibilité de profiter naturellement de la puissance de calcul des environnements distribués.

En plus de ces avantages, le haut degré de parallélisme présent dans Les algorithmes évolutionnaires (le comportement d'un individu au cours d'une itération est totalement indépendant) nous a encouragés à distribuer l'algorithme évolutionnaire de coloration forte stricte présenté dans la sous section précédente.

4.2.1 Présentation de l'approche

Il ya plusieurs niveaux possibles où un algorithme évolutionnaire peut être exécuté en parallèle: au niveau de population, au niveau de l'individuel ou au niveau de la fonction d'évaluation (fitness) [28].

La parallelisation au niveau la fonction d'évaluation est l'une des plus puissantes et des plus populaires de la littérature parce qu'elle est facile à implémenter.

La distribution de l'algorithme évolutionnaire ne nécessite aucune modification dans l'algorithme standard puisque le comportement d'un individu au cours d'une itération est totalement indépendant du comportement des autres pendant la même itération. En outre, dans de nombreux problèmes du monde réel, l'évaluation des individus est de loin l'étape la plus consommatrice en temps. C'est également une condition nécessaire pour que le temps de communication soit petit par rapport au temps de calcul [28].

Un modèle classique adoptée très souvent dans les environnements distribués et parallèle est le modèle client/serveur.

Dans notre cas, l'approche que nous avons utilisée consiste à donner le contrôle à un processus serveur (master) qui distribuera les tâches aux autres processus client (slaves).C'est le serveur qui gère la population et c'est lui qui distribue et communique les individus aux clients. L'évaluation des individus se fait simultanément sur différents processus client. Le serveur collecte les résultats et applique les opérateurs génétiques pour produire les prochaines générations [28]. Cette architecture est représentée graphiquement comme dans la figure 4.1.

S'il ya plus d'individus que le nombre des postes de travail, ce qui est souvent le cas, alors les individus à évaluer sont divisés aussi régulièrement que possible entre les postes de travail disponibles [28].

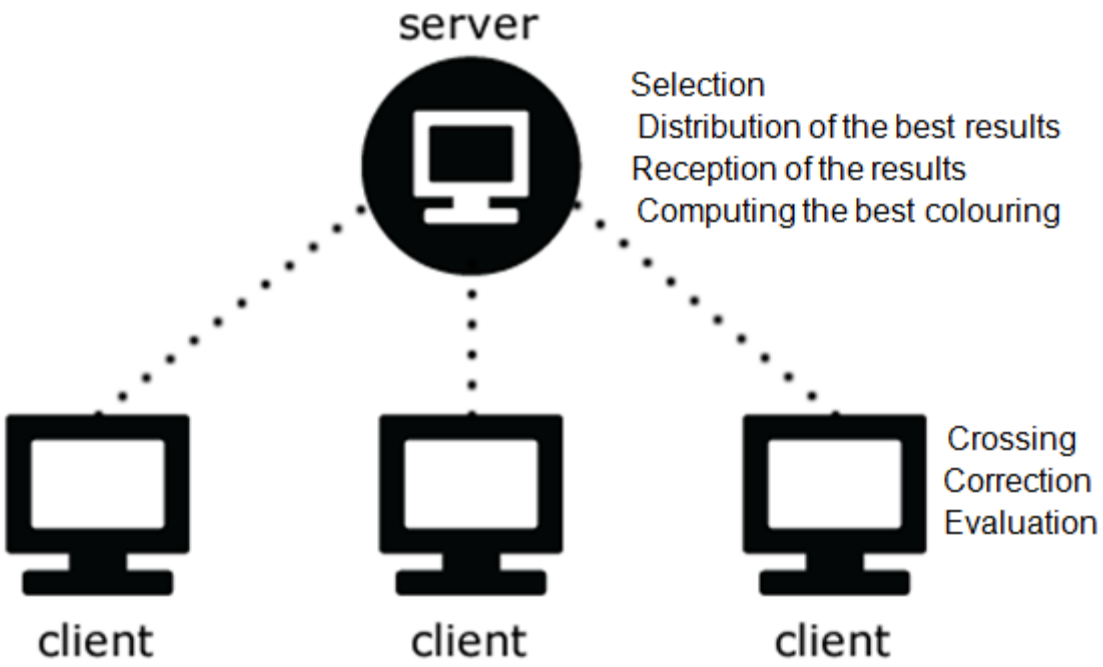


Figure 4.1modèle client/serveur

Ce qui suit est une description informelle de la distribution de l'algorithme évolutionnaire

Begin

Pop = InitPopulation (|Pop|);

S^*= *Best* (*Pop*);

k^*= $|S^*|$;

While not_stop_condition () **Do**

Sub_p = Random_Selection (*Pop*, *Pct_sel*);

For eachSelection subpopulation**Do In Parallel**

Cr_p = Crossing (*Sub_p*, P_{cs});

Correct_p = Correction (*Sub_p* ∪ *Cr_**p***, P_{ct});

If |*Best* (*Cr_p* ∪ *Correct_p*)| <k^***Then**

S^*= *Best* (*Cr_p* ∪ *Correct_p*)

k^*= $|S^*|$;

End If

End Parallel For

Pop = Update (*Pop*,*Sub_p* ∪ *Cr_p* ∪ *Correct_p*);

End While

End

4.3Positionnement du problème

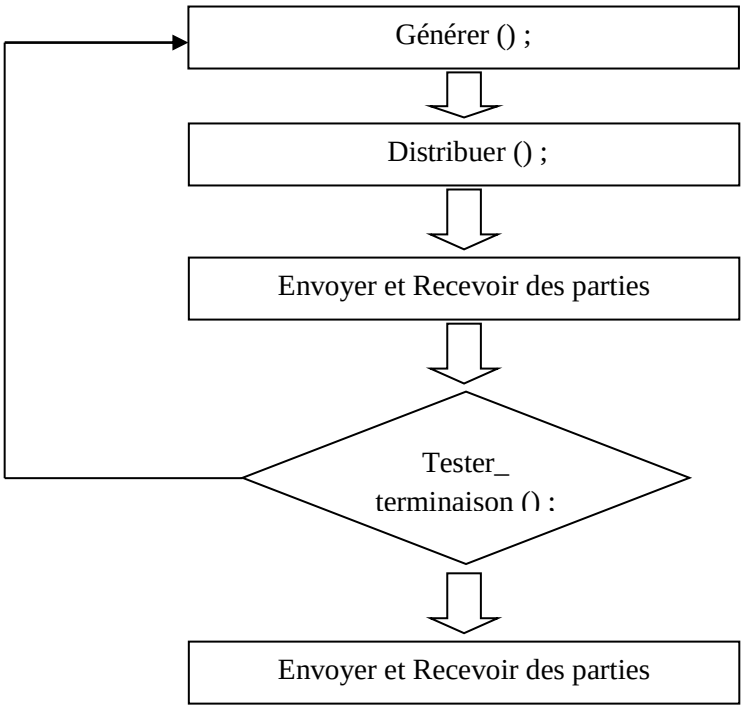
Quand un site tombe en panne lors de la génération, non seulement la partie du graphe stocké dans ce site sera égarée, mais aussi les parties des graphes engendrés par ce site ne seront ni générés ni distribué.

Pour résoudre le problème de déconnexion d'un site, nous proposons un mécanisme de tolérance aux pannes qui repose sur la reprise (recouvrement) où tous les sites dont les successeurs ou les prédécesseurs de leurs états sont stockés dans le site sujet de panne participeront au recouvrement. Après la déconnexion d'un site tous les autres sites seront immédiatement informés, de cette manière, on évitera d'envoyer les états aux sites déconnectés.

Avant d'expliquer notre mécanisme tolérant aux pannes, nous exposons en premier lieu l'algorithme de distribution à la volée des états basés sur l algorithme évolutionnaire présenté dans le chapitre présidant

4.4La distribution à la volée

Dans cette section nous nous intéressons à la génération distribuée de l'espace d'états en se basant sur l'algorithme évolutionnaire de distribution.



L'algorithme est exécuté en boucle par tous les sites y compris le site initiateur S0, il procède en trois étapes, chaque site doit générer des états, répartir les états générés en utilisant l'algorithme évolutionnaire basé sur la coloration forte stricte et finalement envoyer et recevoir les parties résultant aux sites appropriés.

Dans notre algorithme la génération et la répartition des états sont deux processus différents.

4.4.1 La Génération.

La génération d'espace d'états consiste essentiellement à explorer en largeur un grand graphique pour qu'on puisse atteindre chaque nœud à partir de l'état initial au moins une fois. Afin de rendre la génération d'espace d'états plus rapide, nous distribuons la tâche d'exploration sur plusieurs sites qui effectuent cette exploration en même temps. C'est-à-dire qu'elle est réalisée par plusieurs sites où chacun génère une partie de l'ensemble des états du graphe en calculant les successeurs (les fils) des états qu'elle possède [30].

Initialization:
- *Nmax:*Nombre max d'états à générer dans une génération (génération par lot).
- *new_gen :*Le nombre de nouveaux états générés dans cette génération.
- *U* : Ensembledes états non exploré.
- *S* :Ensemble des états exploré.

Begin
If (ce site est l'initiateur) **then** $U := \{u_{inst}\}$;//
Else $U := \emptyset$;
End If;
$S := \emptyset$;
While (pas de détection de la terminaison) **do**
While($\exists u \in U$) && (*new_gen* < *Nmax*)**do**
For Chaque Transition *e* **do**
u_{new}:=NewState (*u*, *e*);
$U := U \cup \{u\}$;
new_gen = *new_gen* +1;
End For
$U := U \setminus \{u\}$;
$S := S \cup \{u\}$;
End While
$U := U \cup$ ReceiveStates \ *S*;
End While
End

L’approche que nous avons utilisée pour distribuer la génération consiste à permettre à chaque site de générer un nombre fini d'états Nmax. Le partitionnement de ces états s’effectue grâce à une fonction de répartition basée sur l'algorithme évolutionnaire de distribution.

Au début seul le site initiateur a un travail à faire (un état à explorer). Tout site peut être initiateur.

4.4.2La répartition.

Après la génération d’espaces d’états, il est important de spécifier la nature de la fonction utilisée pour mapper les Etats à des sites. Elle associe à chaque état un numéro correspondant au processus propriétaire.

Les performances du système global dépendent directement du choix de la fonction de répartition. Un bon choix doit être efficace pour coder et calculer, et devrait également atteindre les objectifs de performance suivants:

- **Equilibrage spatial :**L’espace mémoire pour stocker le sous ensemble d’états Si doit être environ le même par chaque processus
- **Localité :** Le nombre d’arcs reliant des états appartenant à deux propriétaires différents, appelés arcs traversants (cross arcs), doit être aussi réduit que possible.
- **Equilibrage temporel :** il faut aussi assurer que les processus soient les moins possibles inactifs.

Notre choix du site qui devrait stocker un état donné (devenir propriétaire de cet état) se porte sur l'algorithme évolutionnaire basé sur la coloration forte stricte, Cette méthode de répartition essaie surtout de minimiser le nombre d’arcs traversants en mettant les sommets les plus connectés dans le même site.

La répartition des états est un répétitif, à chaque fois qu’un site génère Nmax états ce processus sera déclenché.

4.4.3 Envoyer et recevoir des parties.

Après qu’un site finit la répartition des parties du graphe, il commence à envoyer les parties aux sites appropriés. Chaque site reçoit une ou plusieurs parties de différents sites.

Notre mapping est base sur l'algorithme évolutionnaire qui donne comme résulta à chaque fois qu’il est exécuté des parties du graphe (plusieurs états relier entre eux) ou chaque partie sera envoyée ensuite à son site. Ceci Signifie que l'envoi se fait

par lot (plusieurs états dans un seul message destiné à un site donné) ce qui peut être très avantageux en matière du temps du fait qu'on a réduit de façon considérable le nombre de messages envoyés.

Au début seul le site initiateur génère des états. Ensuit les Nmax premiers états seront répartis et envoyés aux autres sites. C'est à partir de ce moment que les autres sites commencent la génération puis la répartition des états et enfin l'envoi des parties.

4.5 Mécanisme tolérant aux fautes

Nous expliquons dans ce paragraphe notre mécanisme de tolérance aux fautes. Ce mécanisme consiste à non seulement à récupérer tous les états qui seront perdu lors de la déconnexion d un site mais rétablir aussi leurs liens de connexion.

Au cours de notre recherche pour trouver une méthode de recouvrement des états perdus, nous avons remarqué que les états traite par un site *i* peuvent être classés en fonction de leurs liens de connexion en trois catégories :

- Des états enregistrés dans un site*i*, et leurs prédécesseurs sont enregistrés dans les autres sites.
- Des états enregistrés dans un site *i*, et leurs successeurs sont enregistrés dans les autres sites.
- Des états et leurs prédécesseurs ou successeurs sont enregistres dans le même site.

La solution que nous proposons utilise les prédécesseurs des états perdus stockes sur les autres sites pour permettre la repris de ces états. A chaque déconnexion d'un site tous les autres sites reçoivent une notification de déconnexion, à la réception de cette notification, tout site met à jour ces informations réseaux (le nombre des sites les liens avec les sites en pannes).

Chaque site qui contient un ou plusieurs prédécesseurs des états stockés dans le site sujet de panne commence la génération des états successeurs a partie de ses états connectés avec les états de site en panne. Ces nouveaux états générés représentent la première catégorie des états perdus (Les états enregistrés dans le site en panne et qui ont été génères par d'autres sites), ils seront ensuite sauvegardés dans un vecteur spécial de l'enregistrement de nouveaux états.

Si un état perdu a plusieurs prédécesseurs dans différents sites, il sera génère plus d'une fois, dans ce cas, seul le site qui a le plus de connexion avec cet état garde cet état.

Pour la deuxième catégorie des états c'est-à-dire les états dont les prédécesseurs sont enregistrés dans le site en panne, il suffirait simplement de rétablir les liens de connexion entre ces états et les états nouvellement générés. Si à chaque génération d'un état nous essayons de rétablir les liens avec ces successeurs en testant l'existence de ces successeurs dans chaque site du système cela perturbe le processus globale de génération a cause de la perte colossale de temps. Pour éviter tout ça, Tout site dont les prédécesseurs de leurs états sont stockés dans le site déconnecté envoie une liste de ces états à tous les autres sites. Au fur et à mesure de la génération des nouveaux états chaque site vérifie si ces états appartiennent déjà aux listes envoyées par les autres sites, si c'est le cas un lien est crée entre le prédécesseur du nouvel état et le site émetteur de la liste qui contient l'état. Le nouvel état ne sera pas enregistré dans le vecteur de nouveaux états.

La génération à partie de successeurs des états perdus se poursuit jusqu'à ce que le nombre de nouveaux états atteint Nmax (nombre max d'états à générer dans une génération), à ce moment tout les états du vecteur de nouveaux états seront ajoutés à la liste de la génération principale de ce site pour qu'ils soient ensuite répartis.

Si le site déconnecté est l'initiateur, un autre site est choisi pour être le nouvel initiateur, ce dernier ajoute l'état initial à son vecteur de nouveaux états, et le recouvrement ce fait comme pour tous les autres sites.

Afin d'améliorer la performance du système et d'augmenter le niveau de tolérance aux fautes l'état initial du graphe sera stocké sur tous les sites. Cela permet d'assurer le recouvrement même si il ne reste qu'un seul site en marche.

4.6 Intégration de nouvelles machines

La seconde propriété des systèmes distribués dynamiques est la possibilité d'intégration de nouvelles ressources au moment de l'exécution. La manipulation de ces

nouvelles ressources et malgré l'amélioration qu'elles peuvent apporter au système reste très difficile et compliquée.

Selon les auteurs de [31] la plus part des travaux sur les environnements dynamiques n'ont pas proposé des solutions pour l'intégration des nouveaux sites. L'ajout d'un site selon ces travaux est considéré comme un surcoût qui dépasse le bénéfice attendu par l'intégration. Cependant Benmounah et Boucheham dans [31] ont proposé une solution inspiré de La Mine de TauTona qui est la mine d'or la plus profonde du monde Avec 3,9 kilomètres de profondeur, le problème de cette mine est le trajet de l'ascenseur jusqu'au front qui prend plus d'une heure. Si chaque mineur prend l'ascenseur tout seul, l'avancement de creusement sera plus lent. Par contre l'utilisation de l'ascenseur par groupe augmentera l'avancement du creusement. Benmounah et Bouchehamont adopté ce même mécanisme l'intégration par vague.

Dans notre cas, nous avons adopté à notre tour, ce mécanisme qui représente un bon compromis entre les performances globales et le coût (temps d'intégration). L'inconvénient de l'utilisation de ce mécanisme est que si le nombre de sites voulant rejoindre le système n'atteint pas le seuil (le nombre minimal pour l'intégration), ils resteront en attente jusqu'à la fin de l'exécution.

Pour améliorer cette idée et éviter ce problème nous proposons :

Tout nouvel site qui veut rejoindre le système, est inséré dans une file d'attente directement après leur arrivé.

Pour rajouter tous les sites de la file d'attente au système, il suffit que l'une de ces deux choses arrive :

1. le seuil d'intégration est atteint.
2. le délai d'attente est dépassé.

Le seuil d'intégration et le délai d'attente sont des paramètres réglables par l'utilisateur selon la nature du réseau ou les besoins de l'application.

Dane [31] le plus grand problème rencontré était la perturbation de la fonction de répartition après l'intégration de nouveaux sites, le hachage d'un nœud avant et après l'intégration n'est pas le même, ceci est dû au fait que le hachage repose principalement sur le modulo qui est en fonction du nombre de sites.

Notre répartition n'est pas en fonction de nombre de sites, ce qui nous éviterait tous ces problèmes. À l'intégration de nouveaux sites l'initiateur envoie les informations de ces nouveaux sites à tous les autres sites. À la réception de ce message chaque site met à jour le nombre des sites et construit les liens avec les nouveaux sites.

4.7 Détection de la terminaison

La Détection de la terminaison ne se produit pas tout simplement lorsque l'ensemble des états inexplorés pour un site donné est vide, car plusieurs États pourraient être envoyés plus tard par les autres processus. L'algorithme de détection de terminaison doit être lancé, plutôt, quand tous les processus ont épuisé leur ensemble d'état inexploré [30]. Donc la terminaison globale du système est atteinte lorsque :

1. Tous les calculs locaux sont finis, c'est- à-dire que chaque machine n'a aucun état à explorer.
2. Tous les canaux de communication sont vides c'est- à-dire qu'il n'y a plus de message qui transite à travers le réseau ce qui signifie que toutes les transitions émises ont été reçues par leur destinataire.

Depuis les travaux de Dijkstra et Scholten ansi que Francez dans l'année quatre vingt qui ont décrit le problème de la détection de la terminaison (*distributed termination detection* DTD), plusieurs algorithmes ontété proposé pour résoudre le problème [31]. Dans notre étude, nous avons utilisé l'algorithme développé par Benmounah et Boucheham [31]. Cet algorithme est adopté spécialement pour les environnements dynamiques.

4.7.1Les canaux vides.

Dans les systèmes statiques la propriété des canaux vides peut être assurée par la comparaison entre le nombre de messages envoyés avec le nombre des messages à l'aide de deux compteurs $sent_i$ et $received_i$ pour chaque processus P_i, la somme de toutes les $sent_i$ doit être égale à la somme de toutes les $received_i$ dans le système. Malheureusement ce problème est plus compliqué pour les systèmes dynamiques du fait que le nombre des messages envoyé et reçu par les sites en pannes ne doit pas être pris en compte.

Pour résoudre le problèmeBenmounah et Boucheham [31] ont modifié les compteurs $sent_i$ et $received_i$ de telle manière qu'ils deviennent indexés par les identifiants des voisins de processus P_i, plus précisément, $sent_{i,j}$ est le nombre demessages envoyés parP_i àP_j, $received_{i,j}$est le nombre de messages reçus par Pi de Pj. De cette manière tous les messages envoyé $sent_{x,i}$ et reçu $received_{X,i}$ par un site en pannes X seront ignorés. La figure suivante illustre cette approche.

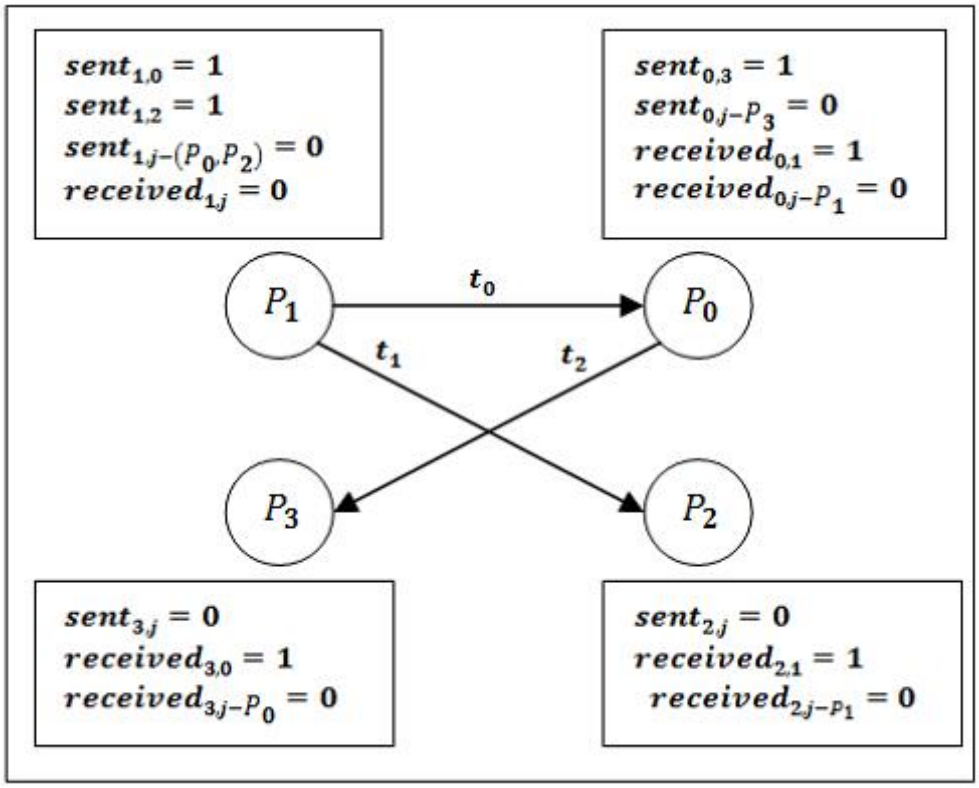


Figure 4.2 La vérification de chaque canal [31].

4.7.2La collection des informations.

La détection de la terminaison ne sera terminée que lorsque tous les processus seront visités pour collecter les informations concernant l'inactivité/activité des machines ainsi que le nombre de messages envoyés et reçus $sent_{i,j}$, $received_{i,j}$ pour chaque site. L'objectif est de minimiser le nombre de passages par site; le meilleur résultat qu'on puisse espérer obtenir est de visiter chaque processus une seule fois. Malheureusement ce résultat ne peut être obtenu que dans les graphes de Hamiltoniens.

La part la plus importante des travaux de la littéraire a utilisé pour la résolution de ce problème les structures virtuelles telle que : anneaux, arbres…etc. Dans [31] l'arbre dynamique virtuel est la topologie choisie, l'initiateur est la racine, c'est lui qui lance le protocole de la détection de la terminaison. Il envoie une demande de terminaison à tous ses voisins qui sont considérés comme des fils. Les fils transmettent à leurs voisins qui n'ont pas encore reçu une demande de terminaison. Les fils d'un site sont tous les voisins qui ont reçu la demande de leur part. Les feuilles de l'arbre sont les processus qui n'ont aucun fils.

La collecte des informations commence à partir des feuilles. Les feuilles envoient ses valeurs à leurs parents qui à leurs tour envoient les valeurs à leurs parents de la même manière et dans un processus récursif chaque site collecte les informations de ses fils puis les transfère a son père.

Au moment où l'initiateur récupère les informations collectées, si tous les processeurs sont inactifs et le nombre de messages envoyés égal au nombre de messages reçus, l'initiateur conclut que le processus de génération distribué est terminé.

4.8 Mise en œuvre

Le modèle d'architecture sur lequel nous aurons appliqué la distribution est compose de plusieurs ordinateurs faiblement couplés, où chaque ordinateur (site) a son propre processeur et sa propre mémoire. La communication entre les sites est réalisée par un réseau dont on ignore la topologie. Les sites sont à la fois dynamiques et hétérogènes (c'est à dire qu'ils n'ont pas la même puissance de calcul, de stockage. Ils utilisent ainsi des systèmes d'exploitation variés). En cas de perte d'un site tous les autres sites seront prévenus et ils seront aussi prévenus en cas d'ajout d'un nouveau site.

Le langage de programmation qui a été utilisé dans le développement de notre application est **Matlab R2013a** sous **Windows 7**sur un réseau d'ordinateurs**.** Matlab est reconnu pour son dynamisme et son excellence pour l'implémentation des algorithmes évolutionnaires et génétiques. Matlab supporte aussi la programmation parallèle et distribuée. Matlab Distributed Computing Server est conçu spécialement pour gérer la programmation distribuée et parallèle.

4.8.1 Langage Matlab.

Matlab (**mat**rix **lab**oratory) est un environnement de développement destiné pour le calcul numérique (traitement du signal et de l'image, les communications…etc.), un langage de programmation de quatrième génération et multi-paradigme. Développé par la société The Math Works, Matlab offre la possibilité de manipuler des matrices, de tracer la représentation graphique des fonctions et des données, d'implémenter des algorithmes, de créer des interfaces utilisateurs et communiquer avec des programmes écrits dans d'autres langues comme JAVA, C, C++ [32].

La modélisation, la simulation et l'analyse de systèmes dynamiques est assurée par le paquet supplémentaire intégré à Matlab **Simulink**[32]**.**

Pour permettre à ses étudiants d'utiliser les bibliothèques Fortran LINPACK (pour l'algèbre linéaire) et EISPACK (pour le calcul numérique des valeurs) sans apprendre le Fortran, Cleve Moler chef de département informatique à l'Université du Nouveau-Mexique, a commencé à développer Matlab dans la fin des années 1970. Il s'est rapidement étendu à d'autres universités et a trouvé une forte audience au sein de

la communauté de mathématiques appliquées. Jack Little et Steve Bangert, des ingénieurs de l'Université du Stanford ont réécrit Matlab en C, ils rejoignent Cleve Moler en 1984 et ensemble, ils créèrent The Math Works pour commercialiser la version 1.0 de MATLAB [32].

Matlab atteint un million d'utilisateurs en 2004, depuis le nombre d'ingénieurs, de scientifiques et d'économistes utilisant Matlab ne cesse d'augmenter. Matlab est utilisé dans le domaine industriel ainsi qu'académique [32].

4.8.2Matlab pour les algorithmes évolutionnaires.

Durant les dernières années, l'intérêt pour les algorithmes évolutionnaires est soulevé régulièrement, par rapport aux procédures d'optimisation traditionnelles. Matlab est devenu un standard dans une large gamme d'applications techniques. Il possède une architecture ouverte et extensible qui permet aux utilisateurs de développer des nouvelles routines pour leurs propres applications. Ces qualités offrent un environnement uniforme sur lequel sont construits des outils d'algorithmes évolutionnaires [33].

Matlab est doté d'un nombre important de toolbox (barre d'outils). Au nombre de ces toolbox, il y a Global Optimization toolbox qui fournit des méthodes de recherche de solutions globales pour les problèmes contenant plusieurs minimums ou maximums. Il inclut le solveur d'algorithme génétique qui offre plusieurs options. Ces options sont présentées sur le tableau suivant [34].

Etape	Option d'algorithme génétique
Création	Uniforme, faisable
Etalonnement d'adéquation	Basé sur le rang, proportionnel, de premier niveau (tronquage), linéaire shift
Sélection	Roulette, sélection uniforme stochastique (SUS), par tournoi, uniforme, par reste
Enjambement	Arithmétique, heuristique, intermédiaire, diffusé, à point unique, à deux points
Mutation	Faisable adaptatif, gaussien, uniforme
Tracés	Meilleure adaptation meilleur individu, distance entre les individus, diversité de population, attente des individus, contrainte max., plage, index de sélection, conditions d'arrêt

Table 4.1 les options d'algorithme génétique standard proposées par Global Optimization Toolbox[35].

Global Optimization toolbox permet aussi de spécifier [34]:

- La taille de la population.
- Le nombre d'élites parmi les enfants.
- La fraction d'enjambement.
- La migration parmi les sous-populations.
- Les limites, contraintes linéaires et non linéaires pour un problème d'optimisation.

4.8.3 Matlab Distributed Computing Server.

Matlab Distributed Computing Server permet d'exécuter des programmes Matlab et des modèles Simulink qui ont besoin des calculs considérables sur un réseau d'ordinateurs, clusters et clouds à l'aide de Parallel Computing Toolbox. Le serveur supporte les travaux par lots, les calculs parallèles et les grandes données distribuées [36].

Principales fonctionnalités [36]

- Exécution des fonctions compatibles GPU sur des ressources de calcul distribué.
- Exécution des calculs parallèles à partir des applications et des composants logiciels générés en utilisant Matlab compilateur sur des ressources informatiques distribuées.
    - Supporter toutes les plates-formes matérielles et d'exploitation prise en charge par Matlab et Simulink.
    - Ordonnancement des applications en utilisant l'ordonnanceur de tâches intégré ou à l'aide des ordonnanceurs tiers comme la Platform LSF, Microsoft Windows HPC Server 2008, Altair PBS Pro, et TORQUE.

Utilisation de Matlab Distributed Computing Server

Matlab Distributed Computing Server s'exécute sur des ressources de calcul distribué, tels que des ordinateurs dans un cluster où les machines virtuelles dans un service de cloud. Le server permet à plusieurs workers (moteurs de calcul Matlab qui fonctionnent indépendamment des sessions client) de recevoir et d'exécuter des codes Matlab et des modèles Simulink. Les utilisateurs peuvent facilement exécuter simultanément leurs applications sur le serveur [36].

C'est à travers Parallel Computing Toolbox que les utilisateurs de Matlab communiquent avec Matlab Distributed Computing Server. Les utilisateurs réalisent des applications parallèles utilisant la boîte à outils sur leurs stations de travail. Pour exécuter des programmes sur le serveur, ils initient soit une session interactive soit soumettent des tâches pour les exécuter en lot [36].

Avec Matlab Compiler, les utilisateurs de Matlab peuvent créer des exécutables autonomes ou des bibliothèques partagées de programmes parallèles Matlab pour une distribution libre de redevances dans les applications Web ou de bureau. Ces exécutables et bibliothèques partagées peuvent distribuer les calculs Matlab aux workers de Matlab Distributed Computing Server [36].

4.9 Conclusion

Pour minimiser le nombre des arcs traversants et pour lutter contre le problème de déconnexions brusque des sites, nous avons présenté dans ce chapitre une approche de distribution d'espace d'états tolérante aux pannes.

CONCLUSION GENERALE

Les travaux que nous avons présentés dans ce mémoire concernent l'étude d'un mécanisme de tolérance aux pannes pour un algorithme évolutionnaire la génération distribuée à la volée des espaces d'états dans un environnement dynamiques.

Notre méthode de génération distribuée utilisé trois types de processus qui collaborent pour partitionner l'espaces d'état : un processus générateur qui génère les états et les transmet à un processus de répartition. A la fin un processus envoie chaque partie à son processeur propriétaire.

Notre méthode consomme dans le choix de l'emplacement d'un état un temps plus importants par rapport aux autres fonctions de hachage du fait que la plupart de ces fonctions décident dans une seule instruction qui est le propriétaire d'un état(exemple en utilisant la fonction modulo), cependant notre choix de l'emplacement d'un état se fait a l'aide d'un algorithme évolutionnaire basé sur la coloration forte stricte dans le but de regroupe les états les plus connecté dans un seul site.

L'utilisation de cette méthode offre deux grands avantages :

1. Contrairement aux autres fonctions de hachage l'envoi des états aux processeurs propriétaires se fait par vague ce qui minimise considérablement l'accès au réseau cela nous fait gagner du temps.

2. Le résultat final de la génération nous donne une partition dont le nombre d'arcs traversants réduit. Ce qui peut être très bénéfique pendant la vérification, surtout si le nombre des propriétés à vérifier est important.

Des nouveaux opérateurs de l'algorithme évolutionnaire (la mutation par exemple) pouvant être utilisé pour de futurs travaux afin d'améliorer les résultats obtenus.

REFERENCES

[01] Jean Arlat, Yves Crouzet, Yves Deswarte, Jean-Charles Fabre, JeanClaudeLaprie et David Powell : Encyclopédie de l'Informatique et des Systèmes d'Information, chapitre Tolérance aux fautes, pages 241–270. Vuibert, 2006.

[02] AlgirdasAvizienis,Jean-Claude Laprie, Brian Randell et Carl E. Landwehr:Basic concepts and taxonomy of dependable and securecomputing. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, 2004.

[03] J.-C. Laprie, J. Arlat, J.-P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.-C.Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac et P.Thévenod, Guide de la sûreté de fonctionnement, 369p., 1995.

[04] J-C. Laprie. ARAGO 15 : Concepts de base de la tolérance aux fautes. MASSON, Paris, 1994.

[05] D. Conan. Tolérance aux fautes par recouvrement arrière dans les systèmes informatiques répartis. Phd thèse, informatique, Université PARIS 6, 1996.

[06] D. Powell. Failure mode assumptioncoverage. In 22th IEEE Sumposium on FaulttolerantComputing, 1992.

[07] John Wakerly : Errordetecting codes, self-checking circuits and applications. Computer Design and Architecture Series. Elsevier, 1978.

[08] Xavier Besseron : Tolérance aux fautes et reconfiguration dynamique pour les applications distribuées à grande échelle, Thèse de doctorat, l'Université de Grenoble, avril 2010.

[09] Samir Jafar : Programmation des systèmes parallèles distribués : tolérance aux pannes, résilience et adaptabilité, Thèse de doctorat, Institut National Polytechnique De Grenoble, Juin 2006.

[10] Benjamin Lussier : TOLÉRANCE AUX FAUTES DANS LES SYSTÈMES AUTONOMES, INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE, avril 2007.

[11] Elmootazbellah N. Elnozahy, Lorenzo Alvisi, Yi-MinWang et David B. Johnson : A survey of rollback-recovery protocols in message-passing systems.

ACMComputing Surveys, 34(3):375–408, 2002.

[12] Eiben A.E., and Smith J.E. Introduction to Evolutionary Computing, Springer, 2003.

[13] Anne Spalanzani, Algorithmes évolutionnaires pour l'étude de la robustesse des systèmes de reconnaissance automatique de la parole, Phd thèse, Informatique, Université Joseph Fourier - Grenoble I, 1999.

[14] Marc Schoenauer, Optimisation évolutionnaire, Springer,2007.

[15] Tarek Chaari. Un algorithme génétique pour l'ordonnancement robuste : application au problème du flow shop hybride. Phd thèse, Automatique, Génie Informatique, Université de Valenciennes et du Hainaut-Cambrésis, 2010.

[16] Nikolaus Hansen, Dirk V. Arnold and Anne Auger, Evolution Strategies , 2013.

[17] Rodolphe Le Riche, Marc Schoenauer, Michèle Sebag. Un état des lieux de l'optimisation évolutionnaire et de ses implications en sciences pour l'ingénieur.

[18] Evelyne Lutton, Darwinisme artificiel : une vue d'ensemble, INRIA - Rocquencourt - Equipe COMPLEX - Projet FRACTALES, 2004.

[19] Oussama EL GERARI, Contributions à l'Amélioration des Techniques de la Programmation Génétique et de la Programmation Évolutionnaire, Thèse de doctorat, l'Université du Littoral Côte d'Opale, 2011.

[20] Wikipédia, l'encyclopédie libre, Théorie des graphes.

[21] Michel Rigo, Théorie des graphes,Université Liège, 2009/2010.

[22] Christine Solnon,Théorie des graphes et optimisation dans les graphes,INSA de Lyon.

[23] Olivier Togni, Coloration de graphes : quelques contributions, Université de Bourgogne, 2006.

[24] benoit.robillard, Complexité de la coloration avec préférences dans les graphes de conit bipartis, Université Paris-Sud 11.

[25] Haddad and Kheddouci, A strict strong coloring of trees, Université de Lyon, 2009.

[26] Bouzenada, Bensouyad, Guidoum, Reghioua, Saïdouni, A Generalized Graph Strict Strong Coloring Algorithm, université de constantine 2, 2012.

[27] Guidoum, Bensouyad, Saïdouni, The strict strong coloring based graph distribution algorithm, université de constantine 2, 2012.

[28] Marco Tomassini, Parallel and Distributed Evolutionary Algorithms, University of Lausanne, 1999.

[29] Calégari, P, Parallelization of Population-Based Evolutionary Algorithms for Combinatorial Optimization Problems. Doctoral thesis, Institut Fédéral Suisse de Technologie, Lausanne, 1999.

[30] Gianfranco Ciardo, Distributed and Structured Analysis Approachesto Study Large and Complex Systems, College of William and MaryWilliamsburg.

[31] Benmounah, Boucheham, Adaptation de la Génération Distribuée à la Volée des Espaces d'États aux Environnements Dynamiques.

[32] Wikipedia, the free encyclopedia, MATLAB.

[33] Hartmut Pohlheim, GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLABDocumentation, Research and Technology, Berlin, Germany.

[34] Global Optimization ToolboxSolve multiple maxima, multiple minima, and nonsmooth optimization problems.

[35] MATLAB site official.

[36] MATLAB Distributed Computing ServerPerform MATLAB and Simulink computations on clusters, clouds, and grids.

[37] Bensouyad, Guidoum, Saïdouni, An Efficient Evolutionary Algorithm for Strict Strong Graph Coloring Problem, université de constantine 2, 2014.

[38] J.C. Laprie, Dependable computing and fault tolerance: concepts and terminology, Springer verlag wien gmbh.