Chapitre 1

Concepts de base

1.1 Introduction

Un problème d'optimisation consiste à chercher une instanciation d'un ensemble de variables soumises à des contraintes, de façon à maximiser ou minimiser un critère défini par une fonction objectif. Lorsque les domaines associés aux variables sont discrets, on parle de problèmes d'optimisation combinatoire tels que le problème du voyageur de commerce, l'ordonnancement de tâches, la coloration de graphes, ect.

En pratique, la résolution des problèmes combinatoires est assez délicate du fait que le nombre de solutions réalisables croît généralement avec la taille du problème ainsi que sa complexité. Cela a poussé les chercheurs en intelligence artificielle à développer de nombreuses méthodes de résolution classées en deux catégories :

— Les méthodes exactes : elles permettent de trouver des solutions optimales pour des problèmes de taille raisonnable, mais elles sont inefficaces face aux applications de taille importante.

— Les méthodes approximatives : elles permettant de générer des solutions approchées, mais pas nécessairement optimales, aux problèmes d'optimisation difficiles en un temps acceptable.

Dans ce chapitre, nous introduisons quelques concepts de base relatifs à ces techniques de résolution.

1.2 Complexité algorithmique

En informatique, l'analyse de la complexité d'un algorithme consiste en l'étude formelle de la quantité de ressources nécessaires à son exécution. Le temps d'exécution constitue le facteur majeur. Il est quantifié indépendamment

des caractéristiques matérielles par une évaluation du nombre d'instructions élémentaires nécessaires en fonction du volume des données en calculant :

- La complexité au pire des cas : borne supérieure.
- La complexité au meilleur cas : borne inférieure.
- La complexité moyenne : le cas médian.

Pour évaluer la complexité d'un algorithme, on calcule de façon inductive la complexité au sein de chacun de ses blocs et on combine ces complexités partielles en suivant les règles suivantes :

La séquence :

$$\left.\begin{array}{l} Traitement_1 : \ Complexité_1(n) \\ Traitement_2 : \ Complexité_2(n) \end{array}\right\} \ Complexité_1(n) + Complexité_2(n)$$

La conditionnelle :

$$\left.\begin{array}{l} \underline{Si} \ (Condition) \ alors \\ \quad Traitement_1 : \ Complexité_1(n) \\ \underline{Sinon} \\ \quad Traitement_2 : \ Complexité_2(n) \\ \underline{Fsi} \end{array}\right\} \ Max\,(Complexité_1(n), Complexité_2(n))$$

La conditionnelle :

$$\left.\begin{array}{l} \underline{Tant\ que} \ (Condition) \ faire \\ \quad Traitement_i : \ Complexité_i(n) \\ \underline{FinTQ} \end{array}\right\} \ \sum Complexité_i(n)$$

Le plus souvent, on s'intéresse au comportement asymptotique des complexités quand la taille des entrées (notée n) devienne très grande. L'ordre de grandeur près est calculé via la notation de Landau O.

Notations de Landau : Soient f et g deux fonctions :

$$f, g : N \rightarrow R$$

On note $f(n) = O(g(n))$ lorsqu'il existe des entiers c et n_0 tel que pour tout $n >= n_0$:

$$f(n) <= cg(n)$$

Intuitivement, cela signifie que f est inférieur à g à une constante multiplicative près, pour les instances (données) de tailles suffisamment grandes. Dans la littérature, les complexités algorithmiques habituellement rencontrées peuvent être classés dans les catégories suivantes :

- Complexité constante : $O(1)$.
- Complexité logarithmique : $O\ (log(n))$.
- Complexité linéaire : $O\ (n)$.
- Complexité quasi-linéaire : $O\ (nlog(n))$.
- Complexité quadratique : $O\ (n^2)$.
- Complexité cubique : $O\ (n^3)$.
- Complexité polynomiale : $O\ (n^p)$.
- Complexité exponentielle : $O\ (2^n)$.

La figure suivante (Fig.1.1) montre l'allure des différentes fonctions asymptotiques :

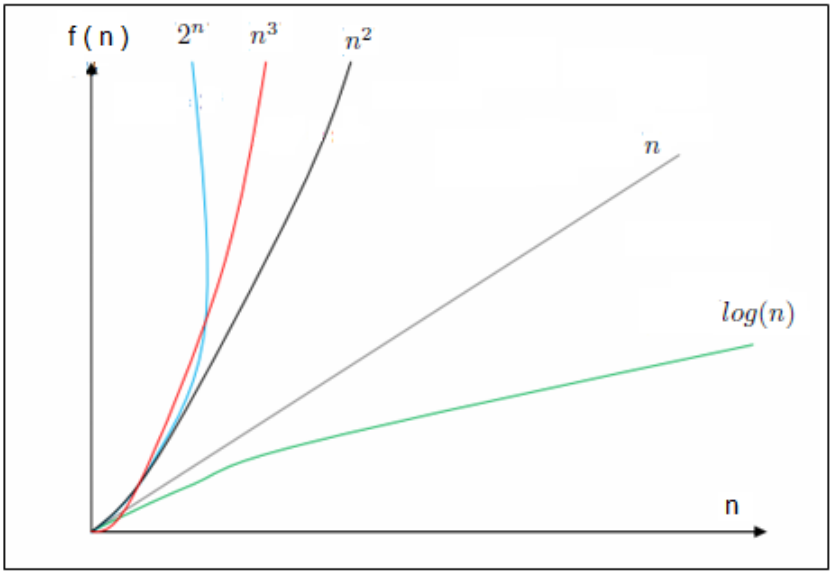

Figure 1.1 : Fonctions asymptotiques

1.3 Méthodes de résolution exactes

Face à un problème d'optimisation combinatoire donné, les méthodes exactes tentent de produire la solution la plus optimale par une recherche exhaustive. Les plus connues sont basées sur le principe de séparation et évaluation et les algorithmes proposés reposent sur :

— une exploration arborescente de l'espace de recherche.

— une énumération partielle de l'espace de solutions.

Dans le cadre de ce cours, on se limite à la présentation du BackTracking [1] comme un exemple type de ces méthodes. C'est un algorithme de recherche utilisé principalement dans les solveurs de contraintes. Formellement, un problème de satisfaction de contraintes est caractérisé par :

— un ensemble fini de variables X_1 , ..., X_n .

— chaque variable X_i a un domaine D_i de valeurs possibles.

— un ensemble fini de contraintes C_1 , ..., C_m sur les variables.

Un état d'un problème CSP est défini par une assignation de valeurs à certaines variables ou à toutes les variables : $X_i = v_i$.

- une assignation qui ne viole aucune contrainte est dite consistante ou légale.
- une assignation est complète si elle concerne toutes les variables.

Donc, les assignations complètes et consistantes constituent les solutions recherchées. Pour identifier ces configurations, l'algorithme BackTracking procède par un traitement itératif qui consiste à choisir une seule variable à la fois et d'envisager pour elle une valeur de son domaine en s'assurant que cette assignation est compatible avec l'affectation partielle courante. Si l'affectation actuelle viole certaines contraintes, une autre valeur sera testée. En cas où toutes les variables sont affectées, le problème est résolu. A n'importe quelle étape, si aucune valeur ne peut être affectée à une variable sans violer une contrainte, l'affectation de la dernière variable (juste avant l'échec) sera révisée en assignant une autre valeur à cette variable lorsqu'elle est disponible. Cela procède de suite jusqu'à ce qu'une solution soit trouvée ou que tous les tests des combinaisons de valeurs aient échoué. Le pseudo-code suivant (Fig.1.3) donne une implémentation permettant de générer toutes les solutions possibles tel que :

— A : l'affectation courante. Elle est initialisée avec $\emptyset$.

— V : L'ensemble des variables non instanciées. Il est initialisé avec X.

```
Fonction Backtracking ( A , V )
Début
    Si ( V = Ø ) alors
        Imprimer A  ( Retourner Vrai)
    Sinon
        Choisir x ∈V
        D ← Dx
        Consistance ← Faux
        Tant que ( D =/= Ø et non (Consistance)) Faire
            Choisir v dans D
            D ← D \ { v }
            Si existe pas c ∈C telle que c viole A union {x ← v } Alors
                Consistence ← Backtracking ( A union {x ← v } , V \ { x } )
            Fsi
        Fin TQ
        Retourner ( Consistance)
    Fsi
Fin
```

Figure 1.2 : Algorithme BackTracking

Malgré les améliorations apportées aux méthodes exactes et les variantes proposées, ces techniques restent limitées à la résolution des problèmes combinatoires de petites tailles et l'explosion combinatoire demeure l'inconvénient majeur. Pour remédier à ce problème, les méthodes approximatives ont été élaborées.

1.4 Méthodes de résolution approximatives

Les méthodes de résolution approximatives sont utilisées pour identifier des solutions approchées, mais pas nécessairement optimales, aux problèmes d'optimisation difficiles en un temps acceptable. Les techniques les plus connues sont appelées métaheuristiques. Elles génèrent des solutions de très bonne qualité pour des problèmes complexes et parfois même pour ceux dont on ne connait pas de méthodes de résolution exactes. La plupart des métaheuristiques sont basés sur des processus aléatoires et itératifs. Cela permet d'éviter les minima locaux en admettant une dégradation de la fonction objectif au cours de leur progression (Fig.1.3).

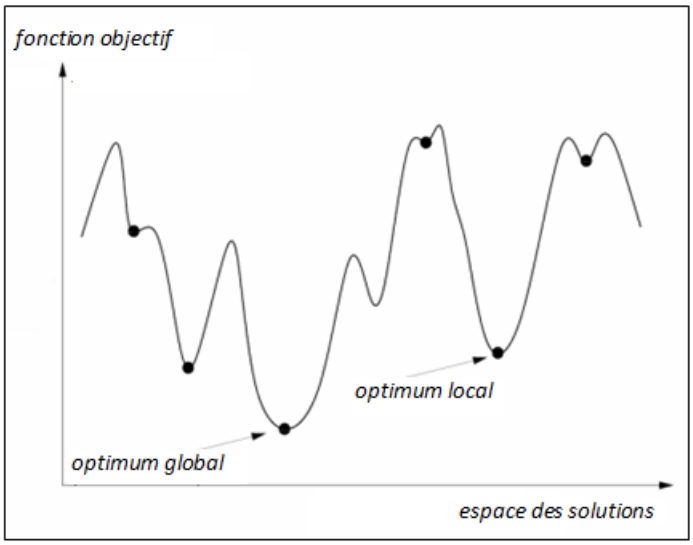

Figure 1.3 : Fonction objectif et optimums

Les métaheuristiques sont souvent inspirées des systèmes naturels dans de nombreux domaines tels que : la biologie (algorithmes évolutionnaires et génétiques) la physique (recuit simulé), et aussi l'éthologie (algorithmes de colonies de fourmis). En plus, ces méthodes utilisent un haut niveau d'abstraction, leur permettant d'être adaptées à une large gamme de problèmes différents. Les métaheuristiques sont classifiées généralement selon la manière avec laquelle les solutions potentielles du problème sont recherchées et exploitées durant le déroulement de l'optimisation. Ainsi, on distingue :

— Les métaheuristiques à trajectoire : dites aussi méthodes de recherche locale. L'idée de base est d'améliorer une seule solution à la fois d'une façon itérative. Ce type de techniques explorent l'espace de recherche en construisant des trajectoires qui dirigent la recherche d'une façon efficace vers les solutions optimales. Les métaheuristiques à trajectoire les plus connues sont le recuit simulé et la recherche avec tabous.

— Les méthodes à population : dites aussi méthodes globales. Elles conduisent la recherche vers les solutions optimales en exploitant les informations fournies par plusieurs points de l'espace de recherche. Les méthodes à population les plus connues sont les algorithmes génétiques, l'optimisation par colonies de fourmis et l'optimisation par essaims de particules.

La qualité des solutions trouvées par les métaheuristiques dépend de leur paramétrage et d'un l'équilibre à établir entre :

- un balayage de tout l'espace des solutions : la diversification.
- une exploration locale : l'intensification.

Ces dernières années, les métaheuristiques sont devenues un outil essentiel pour aborder des problèmes d'optimisation difficiles [2].