Table des matières

1) Introduction

Les organisations font face de nos jours à des problématiques de plus en plus complexes. Ceci est dû à la multiplicité des objectifs à atteindre et aux contraintes difficiles à prendre en considération et très souvent liée à la nature combinatoire des problématiques qu'elles traitent. Les approches existantes répondent partiellement et souvent pas du tout aux attentes des décideurs. Nous tenterons dans ce chapitre d'attirer le lecteur sur la difficulté de résolution de ces problèmes et de lui faire prendre conscience sur la nécessité d'aborder ces problématiques sous un angle qui lui permettra d'intégrer la dimension combinatoire afin de développer une approche raisonnable pour chercher une solution acceptable dans un temps raisonnable. Nous couvrirons aussi bien des applications modélisables par les graphes que celles par les modèles mathématiques d'une manière générale. Nous donnerons une classification des problèmes par rapport à la complexité de leur résolution.

L'optimisation combinatoire (OC) occupe une place très importante dans la résolution de problèmes en Informatique ou dans les autres disciplines de l'ingénierie aux technologies de pointes. Les problèmes d'optimisation combinatoire sont souvent faciles à définir mais, en général, ils sont difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les instances de ces problèmes. De nombreuses méthodes de résolution ont été développées en recherche opérationnelle (RO) et en intelligence artificielle (IA). Ces méthodes peuvent être classées sommairement en deux grandes catégories : les méthodes exactes (complètes) qui garantissent la complétude de la résolution et les méthodes approchées (incomplètes) qui perdent la complétude pour gagner en efficacité.

Les méthodes exactes tentent généralement d'énumérer, souvent de manière exhaustive, l'ensemble des solutions de l'espace de recherche auxquelles elles rajoutent des techniques pour détecter le plus tôt possible les échecs (calculs de bornes) et d'heuristiques spécifiques pour guider les différents choix. Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles les techniques de séparation et évaluation progressive (SEP) ou les algorithmes avec retour arrière. Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Malgré les progrès réalisés (notamment en matière de la programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème, les méthodes exactes rencontrent généralement des difficultés face aux applications de taille importante.

Les méthodes approchées constituent une alternative très intéressante pour la résolution de problèmes combinatoires de grande taille lorsque l'optimalité n'est pas une priorité. Parmi ces

méthodes, nous citons les méthodes gloutonnes dites d'insertion comme, par exemple, la méthode de Lin et Kernighan qui est restée longtemps l'approche préférée des praticiens pour la résolution du problème du voyageur de commerce. Des progrès importants ont été réalisés depuis ces dernières décennies où l'on constate l'apparition d'une nouvelle génération de méthodes approchées appelées *méta heuristiques* pour améliorer les heuristiques locales. Elles sont adaptables et applicables à une large classe de problèmes et offrent un processus de raffinement par l'hybridation des différentes approches. Grâce à ces méta heuristiques dont l'intérêt est certain en Informatique et dans d'autres disciplines comme la génétique, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation classiques de plus grande taille et pour de très nombreuses applications qu'il était impossible de traiter auparavant.

2) Méthodologie de résolution d'un problème d'OC

Pour traiter un problème de décision et plus particulièrement d'OC, nous devons adopter une méthodologie d'approche de façon à être guidé tout au long du processus de résolution. La figure 1.1 suivante illustre une méthodologie d'approche d'un problème d'OC.

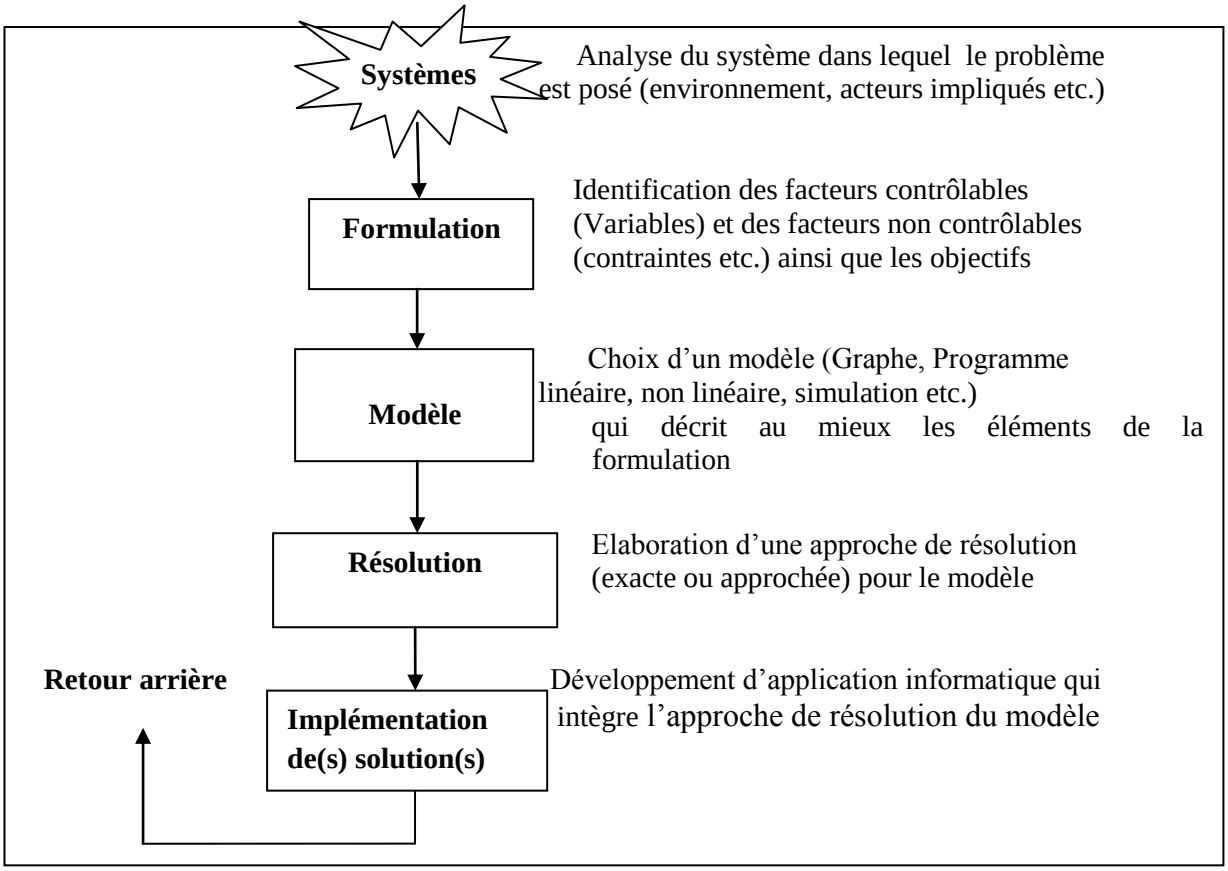

Figure 1.1 : Méthodologie d'approche d'un problème

2.1 Analyse du système

Une attention particulière doit être accordée à la phase d'analyse du système. Des réunions de « brain storming » doivent être organisées avec les décideurs pour bien cerner la problématique à traiter. Des approches d'analyse systématique comme CATWOE développée par Peter Checkland en 1981 permettent de guider dans le processus d'identification du problème, de réfléchir sur ce que l'on doit atteindre et de considérer l'impact de la solution sur les personnes impliquées. Cette approche simple est utilisée pour stimuler la réflexion sur le problème et l'implémentation de solution. consiste à dresser un tableau où les lignes sont les lettres de la méthode formulées sous forme de questions auxquelles il faut tenter de répondre objectivement. La tables CATWOE est présentée somme suit :

Code	Designation	Explication
C	Customers	Les clients ou les bénéficiaires finaux, les problèmes qu'ils rencontrent, comment vont-ils réagir aux propositions qui leurs seront soumises et qui sont les gagnants et les perdants.
A	Actors	Identifier les acteurs qui font le travail, l'impact sur eux et comment vont-ils réagir ?
T	Transformation process	Décrire le processus du système qui permet de transformer des entrées en sorties, quelles sont les entrées et d'où proviennent-elles? Quelles sont les sorties et où vont-elles? Quelles sont les étapes intermédiaires ?
W	Worldview	Quelle est la vue globale dans laquelle la situation s'adapte le mieux? Quelle est le problème réel sur lequel l'étude est faite? Quelle est le plus large impact de n'importe quelle solution?
O	Owners	Quels sont les propriétaires réels ou les propriétaires du système ou processus que vous souhaitez changer? Peuvent ils vous aider ou vous stopper ? Quels sont les conséquences si vont dans le même sens que vous? Pourquoi vont-ils suivre votre démarche?
E	Environmental constraints	Quelles sont les contraintes qui agissent sur la situation et vos idées ? Quelles sont les limites éthiquement, réglementairement, financièrement et ressources ? Les régulations en vigueur etc. ? Comment cela puisse contraindre vos solutions ? Comment les contourner ?

Table 1.1: L'approche CATWOE

Prenons à titre d'exemple le système ouvert comme Internet en considérant les points de vues des fournisseurs d'accès, des utilisateurs, des fournisseurs de services, des gouvernements (exécutifs) etc.

Code	Désignation	Explication
C	Customers	Les utilisateurs d'Internet et les problèmes de connexion et de débit qu'ils rencontrent quotidiennement.
A	Actors	Les fournisseurs d'accès, les utilisateurs, les fournisseurs de services et l'état qui fixe les régulations (ARTP) et qui contrôle le système
T	Transformation process	Prestation de service pour le Provideur et les fournisseurs de services sur Internet Outil incontournable pour le développement technologique et scientifique Publicité pour les entreprises Information et Moyens de lutte comme les réseaux sociaux (crise dans les pays arabes, etc.)
W	Worldview	Internet est un outil qui réduit les distances culturelles, ethniques, sociales et permet de promouvoir les valeurs universelles comme la tolérances, la lutte contre xénophobies, islamophobie et abolit les frontières
O	Owners	Algérie Télecom,
E	Environmental constraints	Les virus, les hackers etc. Nécessité de protéger les enfants Nécessité de protéger les libertés individuelles Lutte contre les nouveaux fléaux en cybercriminalité

Table1 : Exemple d'analyse d'un système par CATWOE

2.2 Formulation du problème

Dans cette phase, nous cherchons à identifier le(s) objectif(s) du problème et ses différentes contraintes. Pour cela, nous devons connaître les différents facteurs intervenants dans le système à savoir : Les facteurs contrôlables et les facteurs non contrôlables.

Les facteurs contrôlables sont ceux sur lesquels le(s) décideur (s) peu(ven)t directement agir. Généralement, ils représentent les variables de décisions.

Les facteurs non contrôlables représentent ceux qui sont imposés par l'environnement. Les contraintes sont un exemple de facteurs non contrôlables mais il en existe d'autres comme les coûts dans la fonction Objectif etc.

A titre d'exemple, considérons le problème du voyageur de commerce qui doit partir d'une ville donnée et visiter les (n-1) autres villes du réseau R=(X, U, l) une et une seule fois puis de revenir à sa ville de départ tout en minimisant la longueur totale parcourue.

2.3 Modélisation d'un problème

Un modèle est une représentation d'une réalité. Il peut prendre plusieurs formes : Mathématiques (Linéaire, non linéaire, entier, convexe etc.), graphe et ses différentes et ses topologies (arbre, arborescence, biparti etc.) incluant les réseaux sous toutes ses formes (Transport, Pétri, Automate à états finis etc.) et la simulation. Le processus de modélisation est long et fastidieux auquel nous devons accorder beaucoup d'attention. Cela est dû au fait que les parties impliqués dans ce processus ne parle souvent pas le même langage et des problèmes sémantiques peuvent surgir. De même, le modèle à construire doit être supporté par des données qui sont imprécises et comportent des aléas qui sont parfois liés au fait que les acteurs ne fournissent pas suffisamment d'effort à leur collecte. En général, la résolution du modèle permet de détecter ses insuffisances et de les corriger.

2.4 Recherche de solutions

Dans la littérature, les méthodes de résolution suivent quatre approches différentes pour la recherche d'une solution : l'approche de construction, l'approche de relaxation, l'approche de voisinage et l'approche d'évolution. La première classe regroupe les méthodes de résolution dites exactes et permettent de résoudre des problèmes faciles de la classe P et certaines instances de petites tailles des problèmes de la classe NP. Pour les méthodes exactes, il s'agit de rechercher le ou les solutions optimales. Par contre, pour les solutions approchées nous recherchons des solutions réalisables proches de l'optimum.

Les méthodes exactes fournissent une solution optimale exacte mais leur application pour certain problème n'est pas de tout rentable (en temps et espace mémoire) . Parmi les importantes méthodes, nous citerons les méthodes de séparation et d'évaluation SEP, A* et CSP. Le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre (dans le pire des cas). Pour améliorer l'efficacité de la recherche, on utilise des techniques variées pour calculer des bornes permettant d'élaguer le

plus tôt possible des branches conduisant à un échec. Parmi ces techniques, on peut citer les différentes relaxations : la relaxation de base en programmation linéaire, la relaxation lagrangienne, la relaxation agrégée (*surragote relaxation*) et la décomposition lagrangienne. De plus, on emploie des heuristiques pour guider les choix de variables et de valeurs durant l'exploration de l'arborescence. En programmation linéaire en nombres entiers, nous citons la méthode des coupes de Gomory ou l'algorithme de Branch and Bound pour des petites instances. Dans la première classe de méthode, nous cherchons à couper le polyèdre convexe de façon à rendre les points extrêmes des solutions entières. Par contre, dans la seconde approche l'approche tente de faire une exploration intelligente de l'espace des solutions en le divisant, à chaque itération, en deux sous espaces de façon à restreindre une des variables non entière à prendre des valeurs entières.

Les méthodes approchées tentent de rechercher de bonnes solutions réalisables. On dit qu'un algorithme est approximatif (heuristique) s'il conduit à une solution réalisable mais pas nécessairement optimal en temps raisonnable. Les méta heuristiques constituent une autre partie importante des méthodes approchées et ouvrent des voies très intéressantes en matière de conception de méthodes heuristiques pour l'optimisation combinatoire. On peut les classer en deux catégories: les méthodes de voisinage et les algorithmes évolutifs. Nous pouvons également combiner les différentes méthodes pour créer des méthodes hybrides.

2.5 Implémentation des solutions

Cette étape consiste à implémenter les solutions obtenues à l'étape précédente. Il s'agit de choisir les structures de données adéquates pour la modélisation des données du problème afin de développer une application permettant de répondre aux attentes du (des) décideur (s). La maitrise des méthodes de conception, de langage de programmation, des systèmes de gestion de base de données s'avère nécessaire.

Une fois les solutions implémentées, la validation opérationnelle avec les acteurs concernés est nécessaire. Un retour arrière aux étapes antérieures peut être nécessaire pour régler les conflits qui pourraient surgir de la validation du modèle.

3) Définition d'un problème d'O.C

Un ***Problème d'Optimisation Combinatoire*** (POC) consiste à chercher le minimum s* d'une application f le plus souvent à valeurs entières ou réelles sur un ensemble fini S

$$f(s^*) = \min_{s \in S} f(s) \quad \Leftrightarrow \begin{cases} \text{Trouver } s^* \in S \\ f(s^*) = \min_{s \in S} f(s) \end{cases}$$

Cette définition reste valable pour le cas d'une maximisation car il suffit de remarquer que :

$$\text{Max } f\text{ (s)} = \text{- Min (-}f\text{(s))}$$

Un problème d'optimisation combinatoire est défini par un ensemble d'instances. A chaque instance du problème est associé un ensemble discret de solutions *S*. Un sous ensemble *X* de *S* représentant les solutions admissibles (réalisables) et une fonction de coût *f* (ou fonction objectif) qui assigne à chaque solution $s \in X$ le nombre réel (ou entier) *f(s)*. Résoudre un tel problème (plus précisément une telle instance du problème) consiste à trouver une solution $s^* \in X$ optimisant la valeur de la fonction de coût *f*. Une telle solution *s** s'appelle une *solution optimale* ou un *optimum global*.

Une *instance I* d'un problème de minimisation est un couple *(X, f)* où $X \subseteq S$ est un ensemble fini de solutions admissibles, et *f* une fonction.

Le ***Problème d'Existence*** (PE) consiste à chercher dans un ensemble fini S s'il existe un élément s vérifiant une propriété P.

On peut formuler un (PE) comme un (POC) particulier. En effet, un problème d'existence peut être formulé par l'énoncé d'une question à réponses Oui / Non. On définit la fonction Objectif

$f : S \rightarrow [0, 1]$

$s \rightarrow f(s) = 0$ si et seulement si s vérifie la propriété P

Inversement, on peut ramener un POC à un PE car il existe aussi un problème d'existence associé à tout problème d'optimisation combinatoire où il suffit de rajouter à la donnée de S et *f* un entier K une propriété P tel que $f(s) \leq K$. Autrement dit, on ne cherche pas une solution optimale mais une solution de coût inférieur ou plus égale à K.

La résolution d'un POC peut se faire d'une manière intuitive par énumération complète de toutes les solutions vérifiant le système des contraintes et de prendre ensuite celle qui optimise f mais cette approche est impossible car lorsque S la recherche devient obsolète.

4) Exemples de problèmes d'OC

Cette section présente quelques exemples de problèmes d'Optimisation Combinatoire POC dans les graphes, en programmation linéaire et autre.

Soient R=(X, U, c) un réseaux et s et p deux sommets particuliers de ce réseaux tel que l'ensemble des prédécesseurs de s est vide noté $\Gamma^{-}(s) = \varnothing$ et l'ensemble des successeurs de p est

vide $\Gamma^{+}(p) = \varnothing$, le problème de la recherche d'un chemin de s à p dans R est un problème d'existence (P E)

Problème de la recherche d'un plus court chemin de s à p dans R est un problème d'optimisation combinatoire. En effet, soit S l'ensemble fini de tous les chemins de s à p dans R, à tout chemin $\mu \in S$ lui correspond un coût $f(\mu) = \sum_{u \in \mu} f(u)$. Il s'agit de trouver parmi tous les chemins de S celui de longueur minimale f(μ*), c'est-à-dire,

$$f(\mu^*) = \text{Min}_{\mu \in S} f(\mu) = \underset{\mu \in S}{\text{Min}} \sum_{u \in \mu} f(u)$$

En programmation linéaire, le problème de la recherche d'une solution de base réalisable (sommet) dans le polyèdre convexe délimitant l'espace de contraintes est un PE. Par contre, celui qui consiste à trouver parmi toutes les solutions de base réalisation, celui qui optimise une fonction Objectif est un POC. En effet, considérons le programme linéaire mis sous sa forme standard :

$$\begin{cases} Max\ F = cx \\ A\ x = b \\ x \geq 0 \end{cases}$$

Soit S = {x/ A x = b, x ≥ 0}. Ce problème est équivalent au problème suivant:

$$\begin{cases} Trouver\ x^* \in S \\ F(x^*) = Max_{x \in S}\ F(x) \end{cases}$$

5) Classification naïve des POC (Facile/Difficile)

Les problèmes d'optimisation combinatoires POC peuvent être classés d'une manière simple en problèmes faciles et problèmes difficiles. La première catégorie regroupe tous les problèmes dont on connait des algorithmes polynomiaux pour leurs résolutions. La seconde catégorie regroupe tous ceux dont on ne connait pas d'algorithmes pour leurs résolutions et que la seule façon de le faire est de trouver une solution approchée.

5.1 Problèmes faciles

Le problème de tri d'un tableau à n valeurs, les problèmes d'insertion et les problèmes de la résolution d'un programme linéaire dans le cas de modèle non dégénéré sont des exemples de problèmes faciles hors graphes.

Il existe beaucoup de problèmes faciles dans les graphes. On rappelle qu'un graphe G=(X, U) est la donnée d'un ensemble de sommets et U d'un ensemble d'arcs (cas orienté) ou arêtes (cas non orienté). Un arbre est un graphe connexe et sans cycle.

Le problème de la recherche d'un arbre recouvrant de poids minimum est un problème facile car l'algorithme de Kruskal permet de le résoudre d'une façon optimale en O(n2) où n=|X|.

Les problèmes de cheminement qui consiste à trouver un plus court chemin d'un sommet particulier de départ s et un sommet particulier d'arrivé p et celui de la recherche d'une arborescence des plus courts chemins issus d'une racine s se résolvent d'une façon polynomiale par les algorithmes de Belleman, Djikstra et Ford.

Le problème de la recherche d'un flot maximun dans un réseau (X, U, c) où c est la capacité d'un arc. Un flot f est un vecteur de R^m avec m=| U| qui assure la conservation de la matière en chaque sommet du graphe et où chacune de ses composantes ne dépasse pas la capacité de l'arc correspondant. Il s'agit dans ce cas de trouver parmi tous les flots existants celui qui maximise la valeur de l'arc retour u_r de capacité infini. Ce problème est résolu d'une façon polynomiale par l'algorithme de Ford et Fuklkerson en $O(n.m^2)$.

Un couplage C de G est un sous ensemble d'arrêtes tel que deux arrêtes u et v quelconque de C n'aient aucun sommet en commun. Le problème de la recherche d'un couplage C^* dans un graphe biparti G=(X, Y, U) se résout d'une façon polynomiale par la méthode Hongroise.

Le problème du parcours Eulerien: Une condition nécessaire et suffisante d'existence d'un tel parcours est que le graphe soit connexe et que les ses sommets sont de degrés pairs. Le problème d'existence est résolu en O(m).

5.2 Problèmes difficiles

Il existe dans la vie courante des problèmes de nature difficile car on ne connait pas d'algorithmes exactes pour leurs résolutions pour des instances de grandes tailles. Dans ce cas, nous recherchons des approches approchées (heuristiques, méta heuristiques etc.).

Dans les graphes, le problème de stable maximal est un POC. Un sous ensemble de sommets S $\subset$ X est un stable s'il n'existe pas d'arêtes entre chaque couple de sommets de S. En d'autres termes, le sous graphe G_S=(S, U) où U=$\varnothing$. Un stable S* est dit maximal si son cardinal est maximal.

Le problème de la recherche d'un transversal minimal est un POC. Un sous ensemble de sommets T est dit transversal si toute arête de G a au moins une extrémité dans T. En d'autres termes, T recouvre les arêtes de G. Il sera dit minimal si sa cardinalité est minimale.

Le problème de la recherche d'une clique maximale est un POC. Un sous ensemble de sommets C est dit clique de G si toute paire de sommets de C est reliée par une arête. Le problème est de trouver dans un graphe une clique de cardinalité maximale.

Un graphe G est dit k-coloriable si on peut colorier ses sommet avec k couleurs distincts sans que 2 sommets adjacents quelconques de G n'aient la même couleur. Le plus petit k tel que G est k-coloriable est appelé le nombre chromatique d'un graphe.

Le problème de la recherche d'un parcours Hamiltonien de moindre coût est un POC. Il s'agit de parcourir tous les sommets d'un graphe une et une seule fois et en revenant au point de départ tout en parcourant un minimum de distance.

Les problèmes difficiles hors des graphes sont nombreux. En voici quelque uns :

Le problème de sac-à-dos à valeurs entières. Il s'agit de déterminer le nombre d'exemplaire de n objets à prendre dans le sac à dos de façon à maximiser la valeur totale nutritive et ne pas dépasser son poids total p. Chaque objet i a un poids p_i et une valeur c_i et est disponible en quantité b_i

Le problème du Bin packing où il s'agit d'empaqueter n objets de poids pi dans un nombre limité de boite de capacité b. Le but est de répartir ces objets en un nombre minimal de boîtes.

Le problème de la sauvegarde de fichiers sur plusieurs disques en utilisant la commande COPY du MSDos. Cette commande, contrairement à BACKUP ne peut fragmenter un fichier sur deux supports différents. On connait la capacité de chaque support de stockage ainsi que les tailles des fichiers à sauvegarder, le problème consiste à sauvegarder les fichiers en un nombre minimal de supports.

Parmi les problèmes difficiles connus est celui du problème de satisfiabilité de n variables booléennes x_i, (i=1, …, n) et d'un ensemble de m clauses c_j (j=1,…, m) où il s'agit de déterminer les valeurs de chaque variable x_i qui rendent les m clauses vraies. On rappelle d'une clause est un littéral ou une disjonction de littéraux. Un littéral est une variable booléenne ou sa négation.

Exemple : Considérons la formule suivante composée d'une conjonction d'une 3-clause, 2-clause et 4-clause: $F= (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg y_1 \vee y_2) \wedge (z_1 \vee z_2 \vee \neg z_3 \vee z_4)$

Un 2-SAT est une formule à conjonction de 2-clauses, un 3-SAT à conjonction de 3-clauses etc. Cook a démontré que le problème SAT est un problème difficile.

6) Complexité algorithmique

Un problème est dit *polynomial* s'il existe un algorithme permettant de trouver une solution optimale pour toutes ses instances en un temps polynomial par rapport à la taille de l'instance. Un tel algorithme est dit *efficace* pour le problème en question. C'est notamment le cas de certains problèmes de plus court chemin dans un graphe pondéré, du recouvrement d'un graphe valué par un arbre de poids minimum, des problèmes classiques de flots, ainsi que pour les problèmes Horn-SAT et 2-SAT (notons cependant que MAX-2-SAT reste NP-difficile). Cependant, pour la majorité des problèmes d'optimisation combinatoire, aucun algorithme polynomial n'est connu actuellement.

La complexité a été développée en 1970 pour répondre à la question d'existence d'algorithme polynomiale pour une classe de problème dit difficile. Le principal résultat est la conjecture que P≠Np, c'est-à-dire que si on arrive à résoudre d'une façon polynomiale un de ces problèmes difficiles on arriverait alors tous les autres.

Le temps d'exécution d'un algorithme (complexité temporelle) dépend de la machine, du langage de programmation du compilateur (constructeur, version, option etc.) et des données. De même la complexité spatiale (espace mémoire nécessaire) dépend de la machine. Ces deux mesures dont insuffisantes pour tester l'efficacité d'un algorithme.

En pratique, la complexité d'un algorithme A est une fonction $C_A(n)$ donnant le nombre d'opérations caractéristiques exécutées par A dans le pire des cas en fonction de la taille n. On utilise la O notation pour donner une majoration de l'ordre de grandeur du nombre d'opérations qui est déterminée en fonction de la taille n des données en entrées du problème et du nombre d'opérations élémentaires qui interviennent dans l'algorithme dans le pire des cas.

Dans le cas des graphes, il est d'usage d'exprimer la taille des données en termes de nombre de sommets n et/ou du nombre d'ares m. En générale, la taille d'une donnée est la quantité mémoire nécessaire pour la stocker (mesurée en bits). On compte le nombre de mots mémoires qui peuvent être des entiers, réels etc.

La difficulté intrinsèque de ces problèmes est bien caractérisée par la théorie de la NP-complétude. De nombreux problèmes d'optimisation combinatoire (la plupart de ceux qui sont vraiment intéressants dans les applications !) ont été prouvés NP-difficiles. Cette difficulté n'est pas seulement théorique et se confirme hélas dans la pratique. Il arrive que des algorithmes exacts de complexité exponentielle se comportent efficacement face à de très grosses instances - pour certains problèmes et certaines classes d'instances. Mais c'est très souvent l'inverse qui se produit ; pour de nombreux problèmes, les meilleures méthodes exactes

peuvent être mises en échec par des instances de taille modeste, parfois à partir de quelques dizaines de variables seulement. Par exemple, on ne connaît aucune méthode exacte qui soit capable de colorier de façon optimale un graphe aléatoire de densité 1/2 lorsque le nombre de sommets dépasse 90. Or, pour le problème d'affectation de fréquences dans le domaine des réseaux radio-mobiles qui est une extension du problème de coloration, il faut traiter des instances comportant plus d'un millier de variables et on se contente de solutions approchées obtenues avec une méthode approchée.

Un problème est dit décidable si et seulement s'il existe un algorithme qui permet de le résoudre. A l'inverse, tout problème n'admettant pas d'algorithme pour sa résolution est qualifié d'indécidable. Par exemple,

Voici quelques exemples de complexités

Complexité constante : $C_A(n) = O(1)$; le nombre d'opérations est indépendant de la taille des données à traiter.

Complexité logarithmique : $C_A(n) = O(\log(n))$; on rencontre généralement une telle complexité lorsque l'algorithme morcèle un problème complexe en plusieurs sous problèmes de sorte que la résolution d'un seul de ces sous problèmes conduit à la solution du problème initial.

Complexité linéaire : $C_A(n) = O(n.\log(n))$; l'algorithme divise le problème en sous problèmes plus petits qui sont résolus de manières indépendantes. Dans ce cas, la résolution des sous problèmes mène à la solution du problème global.

Complexité quadratique : $C_A(n) = O(n^2)$; l'algorithme envisage toutes les paires de données parmi les n entrées (exemple, deux boucles imbriquées).

Complexité cubique : $C_A(n) = O(n^3)$ etc.

Complexité exponentielle : $C_A(n) = O(2^n)$. Dans ce cas, il n'existe aucune garanti que la solution s'obtienne en un temps raisonnable.

6.1) Définition :

Soient f et g deux fonctions de $R \rightarrow R$, on dit que f est d'ordre inferieure ou égale à g ou d'ordre au plus g, si on peut trouver un réel x_O et un réel positif c tel que $\forall\ x \geq x_O \quad f(x) \leq (g(x_O))$. On écrit $f = O(g)$ que l'on prononce Grand O de g .

Considérons deux algorithmes A_1 et A_2 pour un même problème de taille n avec : $C_{A1}(n) = 0{,}5\ n^2$ et $C_{A2}(n) = 5n$

A_2 fait plus d'opération et va moins vite que A_1 lorsque n = 5 , par contre pour n≥10, A2 fait moins d'opérations que A1 et donc il va plus vite car : $\forall n \geq 10$ on a: $5n \leq 0{,}5\ n^2$

6.2) Définition

Soient deux fonctions f et g deux fonctions de R→ R. Si la $\lim_{n\to+\infty} \frac{f}{g} = c$, avec c> 0 alors f et g sont de même ordre (on note f=O(g) et g=O(f) ⇔ f= Θ (g)). Si $\lim_{n\to+\infty} \frac{f}{g} = 0$ alors f=O(g). Si $\lim_{n\to+\infty} \frac{f}{g} = +\infty$ alors g=O(f).

Voici quelques exemples : $0\ (n^2) = \Theta\ (0{,}75\ n^2)$, $O(n^2 + 2n + 7 = 0\ (n^2)$, on néglige des termes d'ordre inférieur

Parfois, nous n'avons pas besoin d'être précis sur la notion d'opération caractéristique d'un algorithme. En effet, Il n'est pas besoin d'être très précis dans la notion d'opération caractéristique. En effet, soient $f(n) = 3\ n^2 - 4n + 2$ et $g\ (n) = \frac{1}{2}\ n^2$. Nous avons $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 6$ donc f=Θ (g).

Soient f (n) = $\log_2$ (n) et g(n) = n. En utilisant la règle de l'hôpital, nous trouvons $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0$ alors f = 0 (g)

Soient $f(n) = x^n$ et $g(n)= n^k$ où x et k sont arbitraires mais des constantes fixes supérieure à 1, nous avons $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = +\infty$ alors g = 0 (f), en d'autres termes g est plus vite que f.

On notera que l'exponentielle est d'ordre supérieur à n'importe quel polynôme en n.

6.3) Définition de la classe P

La classe P est constituée de tous les problèmes d'existence P E résolus par des algorithmes dont la complexité est polynomiale O(p(n)) où p(n) est un polynôme de degré n (n≥1). Elle inclut tous les problèmes faciles cités précédemment.

6.4) Définition de la classe NP :

La classe NP est constituée de problèmes PE dont on ne connait pas d'algorithme de complexité polynomiale et pour lequel une instance de ce problème est vérifiable polynomialement. Autrement dit, cette classe regroupe tous les problèmes d'existence dont une proposition de solution est « oui » est vérifiable polynomialement. En utilisant le principe du

superviseur qui stipule qu'ayant trouvé la solution à un PE par un algorithme polynomiale ou pas, peut on en convaincre facilement (polynomialement) une tierce personne (superviseur) qui n'a pas cherché la solution.
Pour montrer qu'un problème appartient à la classe NP complet il faut d'abord proposer un codage à la solution et de trouver un algorithme qui vérifie la solution par des données (certificat) puis de montrer que cet algorithme se résout d'une façon polynomiale

Remarque : Tous les problèmes d'existence PE sont des problèmes de type NP.

Exemple: Etant donné un ensemble S de n nombres entiers et un nombre entier b, existe-il un ensemble T⊂ S tel que : $\sum_i x_i$ $X_i = b \leq O$, on ne connaît pas d'algorithme polynomiale pour résoudre ce problème alors il n'est pas de classe NP.

- Un codage de la solution (certificat) peut être une liste de celles trouvées dans T.
- L'algorithme de vérification consiste à vérifier que les entiers de cette liste correspondent effectivement à des entiers de S et que leur somme vaut b.
- Cette vérification est possible.

6.5) CLASSE NP COMPLET

Il s'agit des problèmes les plus difficiles de NP et représentent le noyau de cette classe car si on trouvait un algorithme polynomiale pour un des problèmes alors on résoudrait de façon polynomiale tous ceux appartenant à cette classe.

La technique utilisée pour prouver qu'un problème appartient à la classe NP complet est de montrer qu'il est transformable polynomialement en un des problèmes déjà connu de cette classe.

Théorème de Cook : Le problème de satisfiabilité SAT est NP complet

Ce théorème est très important car il permet d'établir des équivalences pour certains problèmes de la classe NP complet.

Les étudiants sont priés de consulter les ouvrages et les pointeurs sur les sites recommandés en cours pour plus de détail sur cette partie.