1 Assemblage de voitures sur une chaîne de montage

Description

On considère le problème d'ordonnancement de voitures suivant (*car sequencing*).

On doit assembler N voitures de différentes catégories sur une unique chaîne de montage. Chaque catégorie de voitures a différentes options (par exemple : toit ouvrant (TO), ABS, ...). Pour chaque option i, il existe une contrainte de capacité indiquant qu'on peut traiter au plus C_i véhicules dans une séquence de longueur L_i.

On modélise le problème par une séquence indiquant les catégories des voitures se suivant sur la chaîne de montage. Par exemple, soit le problème suivant :

- 3 catégories : A, B, C
- un nombre de véhicules à "monter" par catégorie
- 2 options : TO, ABS
- options par catégorie :
  - A : TO
  - B : ABS
  - C : TO et ABS
- Contraintes de capacité par option :
  - TO : 1 voiture sur 2 ($C_1 = 1, L_1 = 2$)
  - ABS : 2 voitures sur 3 ($C_2 = 2, L_2 = 3$)

On cherchera à résoudre le problème par de la recherche locale en minimisant la somme des coûts des contraintes non satisfaites par option. Pour une option i, on comptera une pénalité P_i pour chaque séquence de longueur L_i ne satisfaisant pas la contrainte de capacité.

Par exemple une séquence A C comptera 1 pour l'option TO ; une séquence B C B comptera 1 pour l'option ABS.

La configuration A C B C A B A aura donc un coût total de 3 :

- 2 pour la contrainte TO : 1 sous-séquence A C et 1 sous-séquence C A
- 1 pour la contrainte ABS : 1 sous-séquence C B C

Ordonnancement avec contraintes de capacité seules

1. Citer 3 voisinages simples que l'on peut envisager pour modifier localement une séquence. Pour chaque voisinage, indiquer sa taille (nombre de voisins d'une configuration) et le nombre de sous-séquences à vérifier quand on évalue un mouvement de manière incrémentale.
2. On part de la configuration suivante : A B C B A C B
   Calculer le coût de cette séquence en donnant les sous-séquences qui ne satisfont pas les contraintes de capacité.
3. On considère le voisinage qui consiste à échanger 2 voitures consécutives et l'algorithme de descente stricte qui n'accepte qu'un mouvement améliorant l'évaluation d'au moins 1. Quel type de configuration retourne cet algorithme (en général) ?
4. Avec le même voisinage, on accepte les plateaux (mouvements gardant le coût constant). Montrer qu'on peut obtenir une suite de mouvements conduisant à une configuration satisfaisant toutes les contraintes.

   Comment peut-on faire pour s'assurer que l'algorithme ne cyclera pas et aboutira à cette solution ?

2 Codage d'une métaheuristique dédiée à SAT

Coder dans le langage de votre choix l'algorithme GSAT (et/ou WalkSAT) vu en cours et le tester sur des instances de la série `uf20-91` de la "SATLIB" (maintenue par Holger H. Hoos).

Travail à effectuer

1. Lire (*parser*) une formule booléenne au format CNF (*conjunctive normal form*) et générer la formule en mémoire.
   Difficulté : choix des structures de données (comment représenter les variables, les clauses, etc) pour rendre les algorithmes efficaces (ex : évaluer le nombre de clauses violées).
2. Programmer GSAT et/ou WalkSAT.
3. Expérimenter sur les formules de la série `uf20-91`... que vous devriez pouvoir résoudre avec GSAT en choisissant la valeur du paramètre max_Tries à 10000 et la valeur de max_Moves valant 50.
   Pour les séries comportant plus de variables, nous conseillons d'améliorer l'implémentation de GSAT ou surtout de coder l'algorithme WalkSAT.

Conseils

Sources : Lire le cours (url : `http://www.lirmm.fr/~trombetton/cours/local.pdf`). Vous pouvez aussi lire la page wikipédia sur les algorithmes GSAT et WalkSAT et la page suivante donnant un pseudo-code de GSAT : `http://cs.stackexchange.com/questions/219/implementing-the-gsat-algorithm-how-to-select-which-literal-to-flip`.

Format d'une clause (dans les fichiers de formules booléennes) : la clause $\neg x_2 \vee \neg x_4 \vee x_{10} \vee \neg x_{15}$ est représentée dans le fichier par une ligne $-2\ -4\ 10\ -15\ 0$.

3 Codage d une metaheuristique dediee au coloriage de graphe - MAX-CSP

Coder dans le langage de votre choix l'algorithme Metropolis - recuit simulé -, la méthode tabou ou l'algorithme ID Walk afin de résoudre des instances de coloriage de graphe.

On considèrera les définitions de voisinage vus en cours pour les problèmes MAX_CSP. Par exemple, un voisin défini par le changement de valeur d'une seule variable, ou une version plus *intensifiante* qui choisit une variable en conflit, c'est-à-dire dont la valeur courante viole au moins une contrainte...

Travail à effectuer

1. Ecrire un (*parseur*) capable de lire une instance dans le format simple des fichiers *.col trouvés à l'URL : `https://mat.tepper.cmu.edu/COLOR/instances/` ; par exemple, l'instance décrite dans le fichier `le450_15b.col`. Le parseur génère des structures de données permettant d'évaluer le coût d'une configuration de manière incrémentale, c'est-à-dire rapide, après chaque mouvement. Lesquelles ? *That is the question.*
2. Programmer une métaheuristique permettant de traiter une instance de coloriage (Metropolis - recuit simulé -, la méthode tabou ou l'algorithme ID Walk).
   En plus des paramètres de la métaheuristique (ex : longueur de la liste tabou), le programme accepte un paramètre correspondant au nombre de couleurs recherché : $maxCoul$. Le problème de coloriage de graphe en $maxCoul$ couleurs est donc encodé comme un problème maxCSP, les domaines des variables (sommets) comprenant $maxCoul$ valeurs correspondant aux couleurs possibles. Si la recherche se termine avec 0 contrainte violée, alors la configuration renvoyée est un coloriage en $maxCoul$ couleurs.