Grille de calcul distribué pour l'optimisation combinatoire avec applications en génomique

ESIEE-Paris - Projet de fin de troisième année - département d'informatique

René Natowicz - Mars 2009

Un problème d'optimisation combinatoire c'est :

« Trouver un *meilleur* élément, x^*, dans un ensemble fini, X.»

Plus précisément, une fonction réelle f est définie sur l'ensemble X ; l'élément x^* est un optimum de cette fonction (minimum ou maximum, selon le problème considéré) :

$$x^* = \arg\left(\text{optimum}_{x \in X} f(x)\right).$$

La fonction f est dite fonction « objectif ».

L'ensemble X étant de cardinal fini, ce problème est trivialement résolu par un parcours exhaustif de l'ensemble, avec calcul de la valeur $f(x)$ de chaque élément de l'ensemble. La difficulté vient de la taille de l'ensemble X : on s'intéresse ici à des ensembles dont le nombre d'éléments est « astronomique ». Cette caractéristique conduit à développer à des « stratégies de résolution » permettant de n'explorer qu'une toute petite partie de l'ensemble X.

Vous avez déjà rencontré des problèmes d'optimisation combinatoire dans votre cours d'algorithmique de deuxième année (ou de troisième année selon votre voie d'entrée à l'ESIEE), et dans votre cours de graphe de troisième année. Quelques-uns d'entre-eux, rares, peuvent être résolus par des algorithmes gloutons (calcul d'arbres de poids minimum par les algorithmes de Prim ou de Kruskal). Les autres problèmes que vous avez rencontrés ont été résolus par programmation dynamique : calcul d'un sac à dos de valeur maximum, calcul d'une répartition optimale des heures de travail pour la révision des contrôles (le célèbre problème de Juliette), calcul d'une d'une plus longue séquence commune à deux séquences, calculs de chemins de coût minimum, etc. Pour chacun de ces problèmes, vous avez mis en évidence une propriété d'optimalité récurrente : toute solution optimale s'obtient en « combinant » des solutions optimales calculées pour des sous-problèmes de tailles inférieures. Cette propriété d'optimalité récurrente permet de ne parcourir qu'une petite partie de l'ensemble X.

Pour chacun de ces problèmes que vous avez étudiés, l'approche de programmation dynamique retournait des solutions optimales en complexité polynômiale. Mais, comme chacun d'entre nous, la programmation dynamique a ses limites :

- il y a des problèmes pour lesquels le nombre de sous-problèmes à « combiner » augmente de façon non polynômiale, conduisant ainsi à des complexités elles-mêmes non polynômiales ;
- il y a des problèmes pour lesquels on ne sait pas mettre en évidence la propriété d'optimalité récurrente (et peut-être n'existe-t-elle pas...)

Pour ces problèmes, renonçant à l'optimalité des solutions, on a développé des méthodes qui n'explorent qu'une partie de l'ensemble et retournent des solutions proches de l'optimum, sans garantie d'atteindre cet optimum : exploration avec listes tabou, algorithmes évolutionnaires, algorithmes de colonies de fourmis, algorithmes d'essaims particulaires pour ne citer que les principales approches (*métaheuristiques* pour l'optimisation difficile).

Ces méthodes étant assez aisément parallélisables, on s'intéresse particulièrement à leurs implantations parallèles car elles permettent d'augmenter la taille du sous-ensemble exploré, donc de se rapprocher de l'optimum de la fonction objectif, tout en restant dans des temps de calcul « raisonnables. »

Les grilles de calcul sont une infrastructure matérielle et système pemettant de mettre en oeuvre avec un très faible coût, le parallélisme potentiel des méthodes d'exploration métaheuristiques. C'est l'objet de ce projet.

- Grille de calcul : Berkeley Open Infrastructure for Network Computing (BOINC).
- Langage de programmation : C.
- Applications : problèmes *benchmark* d'optimisation combinatoire ; calcul de sous-ensembles de gènes prédictifs de la réponse à la chimiothérapie ; calcul de sous-ensembles de gènes prédictifs de la mutation du gène TP53 (tumor protein P53).

Remarque : le projet « Calcul Distribué » (numéro 3 sur la liste des projets du département d'informatique) utilise également la plateforme de développement d'applications distribuées BOINC. L'interaction entre les deux groupes est vivement encouragée.

Quelques références rapidement accessibles :

- BOINC (en anglais) :
  `http://boinc.berkeley.edu/`
- BOINC (en français) :
  `http://www.boinc-af.org/`
- Optimisation combinatoire :
  `http://fr.wikipedia.org/wiki/Optimisation_combinatoire`
- Métaheuristiques :
  `http://fr.wikipedia.org/wiki/Métaheuristique`