Informatique - MPI

Quelques rappels

Un *problème d'optimisation* est un problème pour lequel à une *solution valide* est associée une *valeur optimale*. Résoudre un problème d'optimisation, c'est trouver une solution valide et sa valeur optimale associée. On distingue généralement les *problèmes de minimisation* et les *problèmes de maximisation*. Par exemple, le *problème du voyageur de commerce euclidien* est un problème de minimisation. Une solution valide est la donnée d'un cycle qui ne visite qu'une seule fois chaque sommet du graphe associé au problème. La valeur optimale est la longueur du cycle, plus petite longueur parmi toutes les longueurs des cycles.

En pratique, de nombreux problèmes d'optimisation ne peuvent être résolus de manière efficace, c'est-à-dire en temps polynomial en pratique. C'est pourquoi on abandonne l'idée même de trouver un algorithme efficace pour les résoudre, recherchant plutôt un ou plusieurs algorithmes susceptibles d'*approcher* la solution optimale. Un algorithme qui approche la solution optimale d'un problème en *garantissant* la qualité de l'approximation est appelé *algorithme d'approximation*. En l'absence de toute garantie sur la qualité de la solution, l'algorithme est appelé une *heuristique*.

Dans la suite du document, à toute instance I d'un problème d'optimisation, on associe :

- $\mathrm{OPT}(I)$: valeur d'une solution optimale au problème d'optimisation ;
- $\mathrm{ALG}(I)$: valeur d'une solution approchée au problème d'optimisation.

Dans la suite du document, on s'intéresse uniquement aux problèmes de minimisation, les résultats s'étendant sans difficulté aux problèmes de maximisation.

Définition 1 – ρ-approximation d'un problème de minimisation

Un *problème de minimisation* admet une ρ-approximation, avec $\rho > 1$, s'il existe un algorithme tel que, pour toute instance I du problème :

$$\mathrm{ALG}(I) \leqslant \rho \times \mathrm{OPT}(I)$$

Question 1. Montrer que tout problème de minimisation admettant une ρ-approximation admet également une ρ'-approximation, avec $\rho < \rho'$.

La réponse à la question précédente suggère que, pour tout problème de minimisation, il convient de déterminer la plus petite valeur de ρ pour laquelle un algorithme est une ρ-approximation de l'algorithme de minimisation.

Définition 2

Le rapport ρ est dit *strict* (*tight* en anglais) si :

$$\rho = \max_I \left(\frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)} \right)$$

En pratique, cette définition est de peu d'intérêt car on ne connaît généralement pas OPT ! C'est pourquoi on recherche une *borne inférieure*, notée LB de sorte que, pour toute instance I :

$$\mathrm{ALG}(I) \leqslant \mathrm{LB}(I)$$

sachant $\mathrm{LB}(I) \leqslant \mathrm{OPT}(I)$. En conséquence, trouver une borne inférieure constitue une étape cruciale de l'analyse d'un algorithme d'approximation, sa recherche pouvant parfois mener à la construction de l'algorithme d'approximation.

Load Balancing Problem

On considère n *tâches* à exécuter sur m *machines* identiques $M_1, \ldots, M_m$. L'exécution de la tâche j sur chaque machine prend une durée $t_j > 0$. Comment alors répartir les tâches sur l'ensemble des machines de manière à minimiser la durée totale de traitement de toutes les tâches par l'ensemble des machines ? Ce problème de répartition des tâches (ou problème de répartition de charge) est appelé *Load Balancing Problem* en anglais.

Considérons une instance I du problème, à savoir une répartition des tâches sur l'ensemble des machines. Notons $A_I(M_i)$ l'ensemble des tâches affectées à la machaine M_i par cette instance I. On définit la charge de M_i [1] par :

$$\mathrm{load}_I(M_i) = \sum_{j \in A_I(M_i)} t_j$$

1. Sa durée totale de travail.

À l'affectation de toutes les tâches possibles est associée une durée maximale d'exécution, qu'on peut noter :

$$T(I) = \max_{1 \leqslant i \leqslant n} \text{load}_I(M_i)$$

Le problème d'optimisation *Load Balancing* consiste à déterminer une répartition des tâches qui minimise cette durée. Ce problème étant NP-difficile, on s'oriente naturellement vers la recherche d'un algorithme d'approximation.

Question 2. Une première approche, de type *algorithme glouton*, répartit les tâches les unes après les autres en choisissant à chaque étape la machine dont la charge courant est la plus petite. L'algorithme suivant précise cette approche gloutonne.

Algorithme 1 : Algorithme d'approximation pour *Load Balancing*

```
fonction Greedy-Scheduling(t_1, ..., t_n, m)
    pour i de 1 à m faire
        load(M_i) ← 0
        A(M_i) ← ∅
    pour j de 1 à n faire
        affecter la tâche j à la machine M_k de charge minimale
        déterminer la machine M_k telle que load(M_k) = min_{1≤i≤m} load(M_i)
        A(M_k) ← A(M_k) ∪ {j}
        load(M_k) ← load(M_k) + t_j
```

□ **2.1.** Vérifier que cet algorithme effecte chaque tâche à l'une des m machines.
□ **2.2.** Montrer qu'il est possible d'exécuter une version de l'algorithme avec une complexité $O(n \log m)$.

Question 3. On va montrer que cet algorithme est une 2-approximation du problème d'optimisation initial. Dans la suite, I désigne une instance quelconque de *Load Balancing Problem* et G désigne l'instance du problème définie par la mise en œuvre de *Greedy-Scheduling*.
□ **3.1.** Montrer que :

$$\max\left(\frac{1}{m}\sum_{j=1}^{n} t_j, \max_{1 \leqslant j \leqslant n} t_j\right) \leqslant \text{OPT}(I)$$

En déduire une $\text{LB}(I)$.
□ **3.2.** On note M_{i^*} la dernière machine à recevoir une tâche de sorte que :

$$\text{load}_G(M_{i^*}) = \max_{1 \leqslant i \leqslant n} (\text{load}_G(M_i))$$

On note j^* la dernière tâche affectée à M_{i^*} et $\text{load}'_G(M_k)$ la charge de M_k juste avant l'affectation de la tâche j^*.
▷ **3.2.1.** Justifier que pour tout $1 \leqslant i \leqslant m$:

$$\text{load}'_G(M_{i^*}) \leqslant \text{load}'_G(M_i)$$

▷ **3.2.2.** En déduire :

$$\text{load}'_G(M_{i^*}) \leqslant \frac{1}{m}\sum_{i=1}^{m} \text{load}'_G(M_i)$$

Puis que :

$$\text{load}'_G(M_{i^*}) \leqslant \text{LB}(I)$$

▷ **3.2.3.** Montrer alors que :

$$\text{load}_G(M_{i^*}) \leqslant 2 \times \text{OPT}(I)$$

Question 4. L'approximation précédente peut être améliorée. Montrer que :

$$\text{load}'_G(M_{i^*}) \leqslant \text{LB}(I) - \frac{1}{m} t_{j^*}$$

En déduire :

$$\text{load}_G(M_{i^*}) \leqslant \left(2 - \frac{1}{m}\right) \times \text{OPT}(I)$$

Question 5. Supposons que toutes les tâches sauf une aient des durées d'exécution très faibles devant celle de cette dernière. Comment modifier l'algorithme glouton pour qu'il répartisse plus efficacement les tâches ?

Question 6. Supposons que les n tâches aient des durées $t_1, \dots, t_n$ qui vérifient $t_1 \geqslant \cdots \geqslant t_n$. Si $n < m$, montrer que :

$$t_m + t_{m+1} \leqslant \text{OPT}(I)$$

En déduire l'existence d'une $3/2$-approximation du *Load Balancing Problem*.