1 L'heuristique

L'heuristique proposée ordonne les taches selon un poid. Plusieur méthode pour calculer le poid sont proposées :

- $weight1(agent, tache) = gain_{agent,tache}$
  Ce poid permet d'assigner une tache à un agent en fesant en sorte que cela nous rapporte le plus possible.
- $weight2(agent, tache) = ocupation_{agent,tache}$
  Ce poid permet d'assigner une tache à un agent en fesant en sorte que l'occupation soit minimale.
- $weight3(agent, tache) = \frac{ocupation_{agent,tache}}{capacity\_max_{agent}}$
  Ce poid permet d'assigner une tache à un agent en fesant en sorte que l'occupation soit minimale, en considérant l'occupation absolue.
- $weight4(agent, tache) = \frac{-gain_{agent,tache}}{ocupation_{agent,tache}}$
  Ce poid permet d'assigner une tache à un agent en considérant le ration gain/occupation.

Le principe de l'heuristique est de considérer la différence entre la plus petite et la seconde plus petite valeure de poid pour chaque tache. Les taches sont assignées suivant l'ordre décroissant de cette différence. Pour cela on calcule la désirabilité de chaque tache si on l'assigne à l'agent le plus intéressant.

$desirability_{task} = \max_{agent} \min_{autre\_agent} (weight(autre\_agent, task) - weight(agent, task))$

Ou encore :

$desirability_{task} = \min_{autre\_agent} weight(autre\_agent, task) - weight(agent_{task}, task)$

Avec :

$agent_{task} = \arg\min_{agent} weight(agent, task)$

Seul les agents pour lesquelles rajouter la tache ne fait pas dépasser leur capacité maximale sont considérés.

Data: $Task$ L'ensemble des taches.
begin
  while $Task \neq \varnothing$ **do**
    for $task \in Task$ **do**
      $Agent_{task} = \varnothing$ **for** $agent \in Agent$ **do**
        if $ocupation_{agent,task} <= capacity\_max_{agent}$ **then**
          $Agent_{task} = Agent_{task} \cup \{agent\}$
        end
      end
      $agent_{task} = \underset{agent \in Agent_{task}}{\arg\min} \; weight(agent, task)$
      $desirability_{task} =$
      $\underset{autre\_agent \in Agent_{task}}{\min} \; weight(autre\_agent, task) -$
      $weight(agent_{task}, task)$
    end
    $max\_desirability\_task = \underset{task \in Task}{\arg\max} \; desirability_{task}$
    $assign(max\_desirability\_task)$ $capacity\_max_{agent_{task}} =$
    $capacity\_max_{agent_{task}} - ocupation_{agent_{task}, max\_desirability\_task}$
    $J = J \setminus \{max\_desirability\_task\}$
  end
end

La fonction assign permet de remplir les variables d'affecation $x_{i,j}$ en la mettant à 1 pour la tache ayant la désirabilité maximum et l'agent qui va avec cette tache. Elle met cette variable à 0 pour toute les autres taches associées avec ce même agent. L'algorithme s'arrête quand toute les taches ont été affectées. Si il n'y a plus d'agent ayant une capacité suffisante pour les taches restantes, l'algorithme ne trouve pas de solution admissible.

2 Les voisinnages

Etant donné un solution $x_{i,j} \in \{0, 1\}$, on peut considérer les solutions x' correspondant à assigner une tache à un autre agent que celui auquel elle est assigné. On a $(nb\_agent - 1) * nb\_tache$ solution voisine. Toute les solutions obtenuent ne seront pas admissible, car un agent ayant déja atteint sa capacité maximal d'occupation se verra assigner une nouvelle tache.

Si on considère X la matrice des $x_{i,j}$, les X' obtenuent par permution des lignes de X forme un voisinage. Cela reviens à échanger toutes les taches effectuées par deux agents. Toute les solutions obtenuent ne seront pas réalisable mais sauf dans des cas très particuliés, c'est a dire des instances dont il n'existe que très peu de solution réalisable de base, il devrait y avoir des solutions réalisables dans ce voisinage.

Une autre idée de voisinage est de choisir une tache j_1, puis de chercher l'agent i_2 pour lequel le poid $weight(i_2, j_1)$ est minimum. Si l'agent i_2 est différent de celui auquel la tache est affectée i_1, chercher la tache j_2 parmis les taches

effectuées par l'agent i_2 pour lequel le poid $weight(i_1, j_2)$ est minimum. Puis echanger j_1 et j_2. Cela revient à échanger deux taches en prenant en compte la désirabilité pour choisir l'agent qui effectura la première tache, et pour choisir la seconde tache avec laquelle la première tache sera échangée.

3 Exemple de voisinage

Voici l'instance utiliser pour les exemples de voisinage :

Nombre d'agent : 3 Nombre de tache : 6 Agent #1 Capacité d'occupation maximal : 12 Gain par tache : 5 7 7 6 8 5 Occupation par tache : 2 5 5 7 2 5

Agent #2 Capacité d'occupation maximal : 11 Gain par tache : 7 5 7 5 5 5 Occupation par tache : 5 2 7 7 3 3

Agent #3 Capacité d'occupation maximal : 10 Gain par tache : 5 6 5 7 7 5 Occupation par tache : 7 6 2 7 8 3

Solution réalisable de départ :

$$x = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$