INF4215 – TP2

# Explication de l’heuristique

Notre approche par rapport au problème comporte deux composantes majeures afin de limiter le temps d’exécution de l’algorithme par rapport à une recherche en profondeur : la limitation des états durant l’expansion et l’utilisation d’une heuristique la plus précise possible. La limitation des états permet d’optimiser énormément l’algorithme si on élimine certains états pour lesquels on peut déduire que les choisir serait une augmentation nette du coût sans approcher du but. Dans le cas de notre algorithme plusieurs cas sont considérés, par exemple les action « take » sur les paquets qui ne pourraient pas être chargés ne sont pas considérées. Aussi, les « moveAndDrop » sur les nœuds on ne peut ni prendre un paquet ni en déposer un sont ignorés. Ce genre de tests demande beaucoup d’instructions supplémentaires, cependant nous avons pu voir que cela n’avait pas vraiment d’impact sur le temps d’exécution. Selon le profiling effectué, nous avons trouvé que plus de 60% du temps d’exécution est passé dans la copie des états pour créer un nouveau nœud. Ainsi, il est avantageux d’éviter autant que possible de créer des nœuds superflus, surtout qu’au final ces nœuds risquent d’être étendus et de créer des nœuds enfants de façon exponentielle.

Pour ce qui est de l’heuristique, celle que nous avons trouvée semble être relativement précise et approxime dans la mesure du possible le coût total sans utiliser trop de temps d’exécution. Selon nos tests le temps d’exécution avec notre heuristique est environ 6 fois plus petit qu’avec une recherche classique en profondeur (h=0). La première partie de l’heuristique consiste au coût de chaque action « take ». Cette valeur est exacte pour tout problème car les take ne dépendent que du poids des paquets et ces valeurs sont accessibles en tout temps. Ainsi, pour p un paquet à l’un des bureaux et pp son poids, ce coût est de . Ensuite vient le coût des actions « moveAndDrop », beaucoup plus complexe à évaluer. Nous avons séparé ce coût en deux parties  de la façon suivante : devient où spl est le chemin le plus court entre deux nœuds, pc le poids chargé et pa le poids de l’agent. Cela permet d’approximer l’impact du poids de l’agent à l’aide d’une estimation du nombre de mouvements restants.

La première partie de l’approximation du moveAndDrop se fait en calculant la multiplication du poids de chaque paquet par la distance la plus courte entre leur position et leur destination (cela est fait pour les paquets chargés ainsi que pour les paquets aux bureaux). Ensuite, la deuxième partie estime le nombre de déplacements totaux à effectuer en calculant le poids total dans chaque couple (position, destination) de paquets. Ensuite, pour chacun de ces couples, on calcule le nombre de déplacements minimaux (ceil(totalWeight/maxLoad)) pour ce couple. On additionne le tout et on multiplie par le poids de l’agent pour obtenir une très bonne approximation du coût de déplacement total à partir du nœud. Finalement, on multiplie l’heuristique par k afin de s’assurer qu’elle est à la même échelle que le problème.

# Question 1

Pour ce problème cela est très dépendant de ce que l’on recherche. Si l’on veut simplement atteindre le but, la recherche locale est très clairement la meilleure option. En effet, avec une heuristique le moindrement bonne il est clair qu’après un certain temps l’algorithme convergera vers une solution. Si l’on cherche un coût minime, l’algorithme A\* est la solution. Cependant pour ce problème il est loin d’être garanti qu’une solution pourrait être trouvée à une situation complexe. En effet, le problème présente un nombre d’états très élevés et comme chaque nœud est unique (n’a qu’un seul parent) A\* finit par avoir du mal à converger sans heuristique efficace. De plus il est loin d’être évident de trouver une heuristique assez efficace pour gérer des situations complexes car le coût total jusqu’au but n’est pas très évident à calculer. Nous aurions donc tendance à dire que la recherche locale présente de meilleurs résultats pour ce problème étant donné qu’elle est beaucoup plus fiable pour trouver un résultat. Utiliser A\* expose au risque de ne pas trouver de solution du tout et encore moins une solution optimale car l’algorithme ne converge pas rapidement pour des situations trop complexes.