Conclusion : on peut trouver un chemin augmentant de G en cherchant un chemin de s à t dans G_C ce que l'on peut faire par exemple avec un parcours en largeur. On obtient alors un algorithme de recherche de couplage maximum de complexité $O(|A|.|S|)$.

3 Algorithme A*

Principe de l'algorithme :

1. Pour tout sommet x, on pose $dist(s,x) = +\infty$.
2. On pose $dist(s,s) = 0$.
3. On insère tous les sommets x dans la file de priorité avec $dist(s,x)$ comme priorité.
4. Tant que la file n'est pas vide :
   (a) on défile le sommet de priorité minimale noté x. Si $x = t$ alors on renvoie $dist(s,t)$.
   (b) Sinon, pour chacun de ses voisins y : si $dist(s,x) + p(x,y) \leq dist(s,y)$ alors $dist(s,y) \leftarrow dist(s,x) + p(x,y)$ et si y est dans la file : on met à jour la priorité de y dans la file par $dist(s,y) + h(y)$ sinon on insère y dans la file avec priorité $dist(s,y) + h(y)$.
5. Renvoyer t non accessible.

Notations : on note s le sommet d'origine de la recherche et t la destination. Pour tout sommet x, on pose $d(s,x)$ le poids d'un chemin de poids minimal de s à x dans le graphe et $dist(s,x)$ la valeur estimée par l'algorithme à un moment donné.
On note $h(x)$ la valeur de l'heuristique qui est une estimation à la baisse de $d(x,t)$.

Preuve de la correction de l'algorithme A* quand l'heuristique est admissible.

On commence par remarquer quelques propriétés immédiatement liées à l'algorithme. Les trois invariants suivants sont vérifiés :

1. $\forall x \in S, d(s,x) \leq dist(s,x)$.
2. Tous les sommets qui sont dans la file de priorité ont une priorité qui vaut $dist(s,x) + h(x)$.
3. Un sommet qui n'est plus dans la file de priorité a été défilé avec une priorité $dist(s,x) + h(x)$ car si sa distance est amenée à diminuer alors on le remet dans la file.

Montrons maintenant que l'algorithme A* calcule bien le poids d'un chemin de poids minimal s à t lorsque h est admissible.

Notons d la valeur renvoyée par l'algorithme. On a donc défilé t avec une priorité qui vaut $d + h(t) = d$ ($h(t) = 0$ car h est admissible et $d(t,t) = 0$). Supposons qu'il existe un chemin de poids minimal $c : s \to v_1 \to \ldots \to v_n \to t$ de poids $d' < d$. Nous allons montrer qu'avant que t ne soit défilé, chacun des v_i aura son estimation de distance correctement calculée et aura été inséré avec une valeur de priorité strictement inférieure à d ce qui garantit qu'il sera donc défilé avant t.

Pour cela, on montre par récurrence sur i que le sommet v_i est inséré avec une priorité égale à $d(s,v_i)+h(v_i) \leq d'$ avant que t ne soit défilé. Ainsi, lors de cette insertion (il peut être inséré plusieurs fois), on avait obtenu $d(s,v_i) = dist(s,v_i)$ et cette occurrence sera defilée avant t car a une priorité qui lui est inférieure strictement.

Init : Le début de l'algorithme consiste à défiler s et à mettre à jour la valeur de priorité de chacun de ses voisins et en particulier celle de v_1. v_1 est alors inséré avec une priorité qui vaut $p(s,v_1)+h(v_1) \leq p(s,v_1)+d(v_1,t) = d' < d$ car h est admissible. Comme c est de poids minimal alors la distance de s à v_1 est réalisée par l'arête sv_1 et on a bien $d(s,v_1) = dist(s,v_1) = p(s,v_1)$ lors de cette insertion. On a la garantit que v_1 sera défilé avant t.

Her : Supposons que v_i ait été inséré avec une priorité $d(s,v_i) + h(v_i)$ et que lors de cette insertion $d(s,v_i) = dist(s,v_i)$. On sait que v_i est défilé avant t car $d(s,v_i) + h(v_i) < d$ et lorsqu'on le défile on considère ses voisins et notamment v_{i+1}.
Soit v_{i+1} a déjà la bonne estimation de distance et dans ce cas il a effectivement déjà été inséré avec une priorité $d(s,v_{i+1}) + h(v_{i+1}) \leq d(s,v_{i+1}) + d(v_{i+1},t) = d' < d$. Il sera donc défilé avant t et dispose d'une bonne estimation.
Soit son estimation de distance est plus grande que sa distance réelle à s mais dans ce cas sa valeur de priorité va devenir à ce moment là $d(s,v_i) + p(v_i,v_{i+1}) + h(v_{i+1}) \leq d(s,v_i) + p(v_i,v_{i+1}) + d(v_{i+1},t) = d' < d$. De plus lors de cette insertion, on a bien $dist(s,v_{i+1}) = d(s,v_i) + p(v_i,v_{i+1}) = d(s,v_{i+1})$.

Finalement v_n est défilé avant t avec $dist(s,v_n) = d(s,v_n)$ et lors de son extraction on va comparer la distance estimée de t (qui est supérieure ou égale à d) et $d(s,v_n) + p(v_n,t) = d'$. On va donc insérer t avec une priorité d' ce qui contredit la valeur d obtenue par l'algorithme.