Résolution de problème par une recherche heuristique dans un graphe

- La recherche heuristique est à la base de beaucoup d'approches en IA
- Une heuristique est utilisée pour guider la recherche :
  - les heuristiques exploitent les connaissances du domaine d'application
- Le graphe est défini récursivement (plutôt qu'explicitement)

Algorithme A*

- A* est une extension de l'algorithme de Dijkstra
  - utilisé pour trouver un chemin optimal dans un graphe via l'**ajout d'une heuristique**

- Une **heuristique** $h(n)$ est une fonction d'**estimation du coût restant entre un nœud *n* d'un graphe et le but** (le nœud à atteindre)

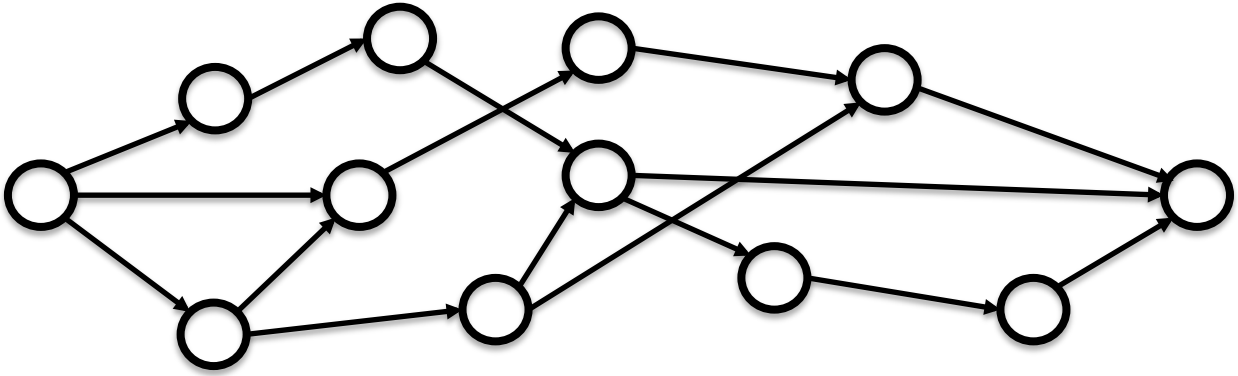
- Les heuristiques sont à la base de beaucoup de travaux en IA :
  - recherche de meilleures heuristiques
  - apprentissage automatique d'heuristiques

- Pour décrire A*, il est pratique de décrire un algorithme générique très simple, dont A* est un cas particulier

Variables importantes : *open* et *closed*

- ***Open*** contient les nœuds non encore traités, c'est à dire à la frontière de la partie du graphe explorée jusqu'à maintenant

- ***Closed*** contient les nœuds déjà traités, c'est à dire à l'intérieur de la frontière délimitée par *open*



Insertion des nœuds dans *open*

- Les nœuds *n* dans *open* sont triés selon l'estimé *f*(*n*) de leur « valeur »
  - on appelle *f*(*n*) une **fonction d'évaluation**

- Pour chaque nœud *n*, *f(n)* est un nombre réel positif ou nul, **estimant le coût du meilleur chemin partant du nœud initial, passant par *n*, et arrivant au but**

- Dans *open*, les nœuds se suivent en ordre croissant selon les valeurs *f(n)*.
  - le tri se fait par insertion : on s'assure que le nouveau nœud va au bon endroit
  - on explore donc les nœuds les plus « prometteurs » en premier

Définition de *f*

- La **fonction d'évaluation** *f*(*n*) tente d'estimer le coût du chemin optimal entre le nœud initial et le but, et qui passe par *n*
  - en pratique on ne connaît pas ce coût : c'est ce qu'on cherche !

- À tout moment, on connaît seulement le coût optimal **pour la partie explorée** entre le nœud initial et un nœud **déjà exploré**

- Dans A*, on sépare le calcul de *f(n)* en deux parties :
  - *g(n)* : coût du meilleur chemin ayant mené au noeud *n* depuis le nœud initial
    - » c'est le coût du meilleur chemin **trouvé jusqu'à maintenant** qui se rend à *n*
  - *h(n) :* coût **estimé** du reste du chemin optimal partant de *n* jusqu'au but. *h(n)* est la **fonction heuristique**
    - » on suppose que *h*(*n*) est non négative et *h*(*n*) = 0 si *n* est le noeud but

Exemples de fonctions heuristiques

- Chemin entre deux villes
  - distance **Euclidienne** (« à vol d'oiseau ») entre la ville *n* et la ville de destination

- *N-Puzzle*
  - nombre de tuiles mal placées
  - somme des distances des tuiles

- Qualité de la configuration d'un jeu par rapport à une configuration gagnante

Algorithme générique de recherche dans un graphe

Algorithme RECHERCHE-DANS-GRAPHE(*noeudInitial*)

1. déclarer deux nœuds : *n*, *n'*
2. déclarer deux listes : *open*, *closed* // *toutes les deux sont vides au départ*
3. insérer *noeudInitial* dans *open*
4. tant que (1) // *la condition de sortie (exit) est déterminée dans la boucle*
    5. si *open* est vide, sortir de la boucle avec échec
    6. *n* = nœud au début de open*;*
    7. enlever *n* de *open* et l'ajouter dans *closed*
    8. si *n* est le but (*goal*(*n*) est *true*), sortir de la boucle avec succès en retournant le chemin;
    9. pour chaque successeur *n*' de n (chaque *n*' appartenant à *transitions*(*n*))
        10. initialiser la valeur *g*(*n'*) à *g(n) + c(n,n')*
        11. mettre le parent de *n'* à *n*
        12. si *closed* ou *open* contient un nœud *n''* égal à *n*' avec *f(n') ≤f(n'')*
            13. enlever *n''* de *closed* ou *open* et insérer *n'* dans *open* (ordre croissant selon *f*(*n*))
        11. si *n*' n'est ni dans *open* ni dans *closed*
            15. insérer *n'* dans *open* (ordre croissant selon *f*(*n*))