--ENGLISH VERSION FOLLOWS--

Dans la plupart des cas, lorsque l'arbre de recherche est équilibré, il semble peu probable que la recherche en largeur soit supérieure en termes de performance. En fait, si toutes les feuilles se trouvent au même niveau, une recherche en largeurva toujours explorer l'arbre dans son entièreté avant de retourner une solution. Il s'agit alors d'un $O(b^m)$, où b est le nombre d'enfants que chaque noeud possède, et m est le nombre d'étages de l'arbre. Une recherche en largeurpeut toutefois être utile si l'arbre n'est pas équilibré, si aucune bone heuristique n'est trouvée (voir ci-dessous), or si la solution optimale se trouve sur l'une des branches les plus courtes de l'arbre. Par exemple, si on souhaite se rendre de A vers B sur une grille infinie où toutes les intersections de la grille sont des noeuds et la distance entre chaque noeud est constante, il est évident que le chemin le plus court serait celui passant par le plus petit nombre de noeuds; c'est d'ailleurs la solution qu'on obtiendrait avec le recherche en largeur. Toutefois, comme la recherche en largeur explorer la même distance dans toutes les directions, on pourrait alors trouver un moyen plus efficace de trouver une solution en obtenant une idée générale de la direction à prioriser.

L'algorithme A* utilise une heuristique qui permet ainsi de s'orienter dans la bonne direction, offrant ainsi une performance supérieure à la recherche en largeur lorsque l'heuristique est bonne. Une bonne heuristique en est une qui peut être calculée relativement rapidement et qui permet de pointer dans une bonne direction avec une forte probabilité; une mauvaise heuristique, par exemple, en serait une qui explore elle-même l'arbre de recherche dans son entièreté. La première heuristique implémentée dans le cadre du laboratoire est celle de Dijkstra. Elle a permis d'obtenir une solution beaucoup plus rapidement que celle implémentée avec la recherche ne largeur. Selon nous, ce résultat peut s'étendre à la majorité des cas semblables à celui étudié pour les raisons mentionnées ci-dessus. La deuxième heuristique implémentée, soit l'algorithme d'Edmond, a offert des résultats encore supérieurs à ceux avec Dijkstra. Si les deux heuristiques étaient des aussi bons indicateurs de la direction à prioriser, ce qui n'est pas la cas ici avec Edmond étant légèrement supérieur, alors l'élément différentiatif entre les deux serait la complexité de l'algorithme et le temps d'exécution. Dans cette situation, Dijkstra serait à favoriser, offrant un temps d'exécution de $O(E + VlogV)$, contrairement à $O(EV)$.

Enfin, la recherche par voisinage variable (VNS) est une alternative intéressante. Elle a l'avantage de permetre de trouver une solution très rapidement, puis d'essayer d'améliorer cette solution. Selon nous, si on cherche simplement une bonne solution, mais pas nécessairement la meilleure solution, globalement, le VNS est la méthode qui se généralise le plus aux problèmes à plus grande échelle. En fait, pour un graphe très grand, A* ou la recherche en largeur sont excessivement exigeants au niveau de l'utilisation de la mémoire et du processeur (CPU), alors que le VNS permet de trouver une solution généralement bonne en un temps décent.

Aussi, l'algorithme possède la propriété intéressante qu'il est possible de l'arrêter à tout moment afin d'obtenir la meilleure solution trouvée jusqu'à présent.

For most cases where the search tree is well balanced, it seems unlikely that the BFS algorithm might be superior in terms of performance. In fact, if all leaves are found at the same level, a breadth-first search will have to look through almost the entire tree before returning its solution. This would be $O(b^m)$, where b is the number of children each node has, and m is the number of levels the tree has. A BFS could prove useful in cases where the tree is seriously unbalanced, no good heuristics have been found, and especially if the best solutions tend to be leaves to the shortest branches. An example of this would be a case where something needs to go from A to B on an infinite grid where all the intersections on the grid are considered nodes, and the distance between each node is constant. Clearly, the best (shortest) path would be the one that requires passing through the smallest number of nodes (shortest branch of the search tree), and that is in fact the solution a BFS would return. Again however, that would only be acceptable if there were no good known heuristics to use, because until a BFS finds a solution, it will have looked at the same depth in all different directions, when there are normally simple ways to get a general idea of the direction to prioritize.

The A* star incorporates a heuristic which taps into the aforementioned idea of prioritizing a direction that is likely to lead to the solution faster. When that heuristic is good, this will generally prove far superior to a BFS for instance. Here we define a "good" heuristic as one that can be computed rather fast (an extreme counter-example would be a "heuristic" which itself searches the whole tree, ultimately leading to no gain whatsoever), and which has a very high probability of pointing in a direction that leads to a decent, if not optimal, solution. The first heuristic we implemented for our A* algorithm, the fastest path to endpoint (Dijkstra algorithm), has proved able to find a solution much faster than a BFS. We believe this finding would extend to most cases similar to the one studied for reasons explained in the paragraph on BFS. Evidently though, the fastest path to endpoint can be misleading, as it does not account for all the other nodes to visit before reaching the arrival. It is in that sense that the other heuristic we implemented for A*, the minimal spanning tree, is preferable, and does in fact prove faster in our tests. If both heuristics were about equally good indicators of the direction to prioritize, then the differentiating factor would be their own algorithmic complexity. That would normally crown Dijkstra instead, which typically runs in $O(E + V \log V)$ as opposed to $O(EV)$.[1][2]

Finally, the variable neighborhood search (VNS) is an interesting alternative. It has the advantage of allowing for a complete solution very quickly, before going on to improve it. We are inclined to believe that, granted we do not need the globally optimal solution but a general optimal solution, it is the most scalable of the algorithms we implemented in this lab. In fact, finding the fastest path or a minimal spanning tree at every iteration of an A* run (or worse, performing a BFS) on a massive graph seems quite dreadful in terms of CPU usage. However, a metaheuristic such as VNS can find an acceptable solution in a decent amount of time. Furthermore, the VNS has the interesting property that a time limit for

1 https://en.wikipedia.org/wiki/Edmonds%27_algorithm
2 https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

convergence can be set, since at any given time, it holds a complete best solution so far.