

2016

UFR Sciences Angers
Master 1 Informatique



Placement de routes

Urbanisme

David Florian

Sous la direction de M. Da Mota Benoit et M.
Goëffon Adrien

Membres du jury

Soutenu publiquement le :
23 juin 2016



L'auteur du présent document vous autorise à le partager, reproduire, distribuer et communiquer selon les conditions suivantes :



- Vous devez le citer en l'attribuant de la manière indiquée par l'auteur (mais pas d'une manière qui suggérerait qu'il approuve votre utilisation de l'œuvre).
- Vous n'avez pas le droit d'utiliser ce document à des fins commerciales.
- Vous n'avez pas le droit de le modifier, de le transformer ou de l'adapter.

Consulter la licence creative commons complète en français :
<http://creativecommons.org/licences/by-nc-nd/2.0/fr/>

Ces conditions d'utilisation (attribution, pas d'utilisation commerciale, pas de modification) sont symbolisées par les icônes positionnées en pied de page.



REMERCIEMENTS

Je tiens à remercier mes enseignants référents, Adrien Goëffon et Benoit Da Mota, pour l'intérêt qu'ils portaient au sujet, leurs conseils, et leur accompagnement tout au long du projet.

Je me dois de remercier chaleureusement mes camarades de promotion, plus particulièrement ceux qui ont été présents à l'Université au cours de ces deux mois et demi de travail.

Je mentionne donc Ugo Rayet, pour les réflexions qu'il a pu apporter lors du développement d'idées et les pistes de résolutions de problèmes proposées, sans oublier d'évoquer la bonne ambiance de travail qu'il a su transmettre. Je remercie également les conseils donnés par Morgane Troysi, principalement concernant ce rapport, qui auront été appliqués tout au long de sa rédaction.

J'ajouterais que les présences fréquentes de Jérôme Fourmond, Morgane Troysi et Ugo Rayet, ainsi que celles de Pierre Granier—Richard, Thibaut Roperch et les autres étudiants en L3 présents en H002 pendant deux mois, ont été particulièrement appréciables et appréciées.

Table des matières

INTRODUCTION	5
1	Présentation du contexte.....5
2	Présentation du projet.....5
3	Introduction au sujet.....5
4	Contextualisation.....7
5	Etudes préliminaires.....7
6	Choix des outils.....7
DÉMARCHE	9
1	Organisation.....9
2	Le moteur.....10
2.1.	Représentation des données du problème.....10
2.2.	Evaluation des solutions.....11
2.2.1.	Nombre de parcelles exploitables.....11
2.2.2.	Calcul du ratio entre distance directe et distance par les routes.....11
3	La recherche de solutions améliorant les objectifs.....12
3.1.	Initialisation d'une solution.....12
3.2.	Maximisation des objectifs par recherche locale.....13
3.2.1.	Maximisation du nombre de parcelles exploitables.....13
3.2.2.	Maximisation de la circulabilité.....13
3.3.	Persistance des solutions non dominées.....14
4	L'interface graphique : Affichage et Interactions :.....15
4.1.	Affichage.....15
4.2.	Interactions.....16
4.3.	Affichage et utilisation du front Pareto.....17
DÉPLOIEMENT	19
CONCLUSION	21
SITOGRAFIE	22
TABLE DES ILLUSTRATIONS	23

Introduction

1 Présentation du contexte

Dans le cadre du Master Informatique, il nous est demandé de réaliser un projet de recherche, sur 10 semaines. Le but est de valider nos acquis, d'approfondir certaines connaissances et compétences et d'avoir une première approche d'un travail sur un projet de recherche, réalisé en quasi autonomie, puisque encadré par nos enseignants-chercheurs.

2 Présentation du projet

Ce projet a été proposé par deux de nos enseignants : Benoit Da Mota et Adrien Goëffon. Il utilise principalement des principes étudiés en classe cette année, dans le cadre de l'Optimisation Combinatoire et de l'option Résolution de Problèmes. L'alliance de ces deux domaines, et compte tenu de la complexité du sujet, nécessite un bon niveau en programmation, dans un langage, et des technologies, orientés vers la performance d'exécution. De plus, la visualisation d'une solution permettant de mieux rendre compte des résultats, la maîtrise de bibliothèques graphiques pouvait être un avantage non négligeable.

3 Introduction au sujet

Le but est de pouvoir fournir à un utilisateur des dispositions possibles de routes sur une surface. Elles doivent respecter plusieurs contraintes. La première stipule que *toutes les portions de routes doivent être connectées entre elles – par l'intermédiaire d'autres routes – et aux entrées-sorties*. La deuxième, que *la surface étudiée et les cellules la composant seront de forme rectangulaire*, dans un premier temps ; D'autres formes de cellules pourront être envisagés par la suite, une forme hexagonale ayant été évoquée lors du premier rendez-vous. La troisième définit ce qu'est une parcelle exploitable : *Est exploitable, toute parcelle ayant au moins une route dans son voisinage, à une distance maximale, appelé le paramètre de desserte, définie dans les paramètres du problème*.

A partir de ces règles, on souhaite offrir à l'utilisateur des solutions maximisant deux objectifs : le premier est un objectif de densité, c'est le nombre de parcelles exploitables de la surface, on cherche à maximiser cette valeur. Le second est l'accessibilité globale, ou circulabilité, entre les parcelles exploitables. On souhaite que les déplacements entre les parcelles, en passant par les routes, soient les plus courts possible, de n'importe quelle position, vers n'importe quelle autre. Pour ceci, on considère l'écart proportionnel entre la distance considérée « parfaite » – distance sans obstacles entre deux parcelles, que l'on appellera par la suite distance directe – et la distance par les routes. Cela nous donne un ratio, associé aux deux cellules concernées. Il suffit alors de faire la moyenne de tous les ratios de la surface, pour obtenir le ratio global. On cherche alors à minimiser cette valeur, ce qui maximise l'objectif d'accessibilité. On aurait également pu tenter de maximiser l'inverse de cette valeur.

La distance directe est le nombre de cellules d'écart entre deux parcelles. Pour le moment, c'est la distance Manhattan qui est utilisée. Elle est calculée en comptant le nombre de cellules se trouvant sur des lignes droites, deux maximum, reliant deux parcelles. On pourrait choisir d'utiliser la distance euclidienne, ou distance « à vol d'oiseau » lors des calculs, si cela nous semblait pertinent. La distance euclidienne peut alors être une valeur décimale, alors qu'elle est entière avec la distance Manhattan.

Si le premier objectif est plutôt simple à évaluer et maximiser, le second est bien plus complexe dans son approche, sa conception et son optimisation. Dans l'exemple ci-dessous, le ratio associé aux parcelles A et B est de 26 pour 1. Or, la moyenne des ratios pour toutes la surface est ici de 1.40137, ce qui est une valeur relativement correcte. En effet, du fait du placement des routes sur cette surface, deux cellules qui ne sont pas du même côté de la surface, auront toujours un ratio très proche de 1, et ces cas représentent environ la moitié des ratios calculés. On voit donc que la minimisation du ratio maximum, qui est élevé, et de la moyenne des ratios, qui est elle tout à fait correcte, sont deux choses distinctes.

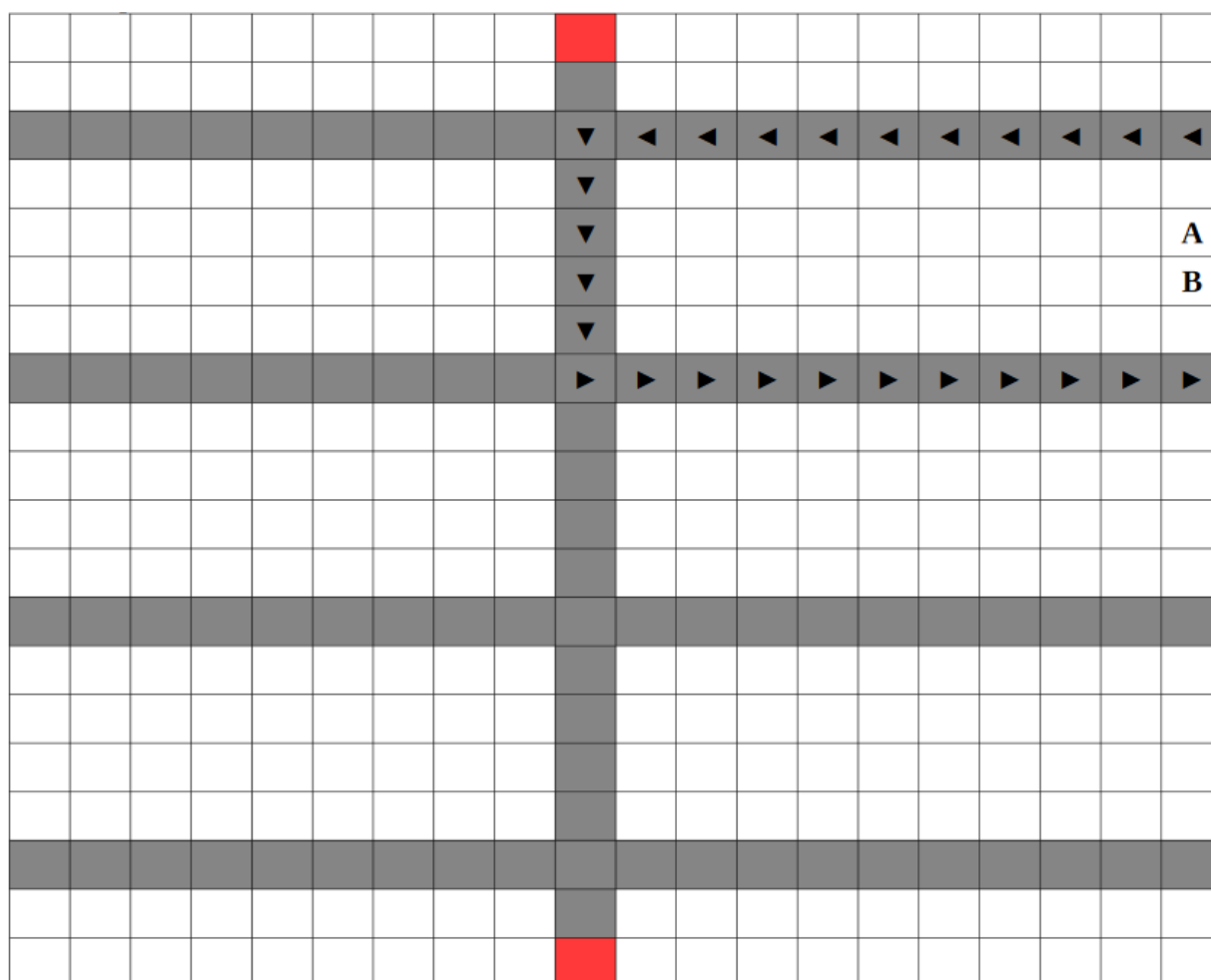


Illustration 1: Surface d'exemple.

4 Contextualisation

Le sujet peut être appliqué sur différents domaines. Une des ses applications est l'organisation et l'accès à des parcelles d'une exploitation agricole, selon le placement des routes et chemins desservant les parcelles. Des ajouts, tendant vers ce modèle, pourraient être l'ajout de la gestion des entrées et sorties entre les parcelles et les accès, ou des tailles de parcelles différentes, au lieu d'une unité fixe dans le cas présent (une route a la même dimension qu'une parcelle, elles occupent une cellule).

Une autre application serait la planification de l'urbanisation d'une ville, afin d'optimiser les emplacements pour habitations et bâtiments, tout en conservant une bonne circulation générale sur l'ensemble du réseau routier. Pour améliorer ce modèle, on pourrait ajouter des priorités sur certains bâtiments. Ou encore, travailler sur une ville déjà existante, que l'on pourrait modifier, mais avec comme contrainte de ne pas utiliser certaines zones, car contenant des infrastructures. On ne pourrait alors plus créer et planifier l'organisation en « partant de zéro », mais on serait contraint par l'environnement pré-existant.

On peut également imaginer appliquer cela au choix de lignes de bus. On ne pourrait alors placer des « routes de bus » seulement sur des routes au sens large, déjà fixées. Il faudrait alors définir des cellules pouvant devenir des « routes », dans la représentation existante du problème, et celles qui sont non modifiables, les habitations, parcs, magasins, etc.

D'autres cas d'utilisation peuvent être réfléchis, il suffit de modifier l'application afin d'ajouter les nouvelles contraintes du problèmes que l'on veut étudier.

5 Etudes préliminaires

Avant de commencer le développement « pur » du programme, il a fallu effectuer des premiers choix dans la manière de traiter le problèmes.

1. Réflexions sur les applications et enjeux du problème
2. Recherche de travaux similaires
3. Réflexion sur les langages, outils et approches du problème

6 Choix des outils

Développement orienté multiplate-formes, j'ai donc choisi des outils largement répandus, voire universels :

Développement avec en langage C++, avec l'IDE *Kdevelop*, *Cmake* et *Make*. Par la suite, le framework *Qt* pour la partie interface graphique a été intégré. *Qt Creator* a été utilisé, car adapté à ce framework. Des tests unitaires ont été réalisés avec la bibliothèque *CppUnit*. *Gnuplot* permet de créer des images de fronts Pareto, à partir de solutions et de leurs évaluations. Son utilisation a été facilitée grâce à l'interface C++ *gnuplot-cpp*, développé par Jeremy Conlin (jeremit0@gmail.com). J'ai cependant apporté quelques corrections, pour éviter les « warnings » de compilation et corriger la documentation, pour qu'elle soit reconnue valide par *Doxygen*, cette interface étant sous licence libre.

Les outils de débogage utilisés sont *Valgrind* et ceux intégrés à *Kdevelopp* et *QtCreator*. *Valgrind* a également été utilisé afin de supprimer les fuites mémoires. De plus, il comprend l'outil *Callgrind*, de génère des fichiers lisibles par *Kcachegrind*, pour l'analyse des performances de l'application. J'ai également utilisé l'outil *CppCheck* afin d'effectuer quelques micro-optimisations.

Le versionnage est géré à l'aide d'un dépôt Git, hébergé sur github.com. La documentation est donc créée grâce à *Doxygen*.

Mes plate-formes de développement et de test ont été Linux Mint 17.3 Rosa et Windows 10.

Démarche

1 Organisation

Une fois les choix d'outils faits , venait alors le moment de la réflexion sur la façon d'implémenter le problème, ses données et ses paramètres. Ayant choisi un langage orienté objet, il s'agissait donc de faire le choix des classes primordiales du programme, puis des structures de données utilisées, afin de représenter l'intégralité du problème.

J'ai décidé de séparer le programme et ses fichiers sources en plusieurs parties. Ceci a pour but de pouvoir compiler et exécuter le programme sans utiliser obligatoirement la partie interface graphique, d'améliorer sa portabilité, avec d'autres outils ou plate-formes de destination, et de changer facilement de solution pour la partie graphique. Cela améliore également la maintenabilité et la poursuite éventuelle du programme par d'autres personnes ou non.

Ainsi, le programme est divisé en trois parties, chacune représentant un niveau de fonctionnalités.

- Le cœur de l'application est appelé moteur (« Engine »). Il est chargé de représenter et de manipuler les données représentant les solutions ainsi que les paramètres d'une instance du problème. De plus, une classe « Coordinates » permet de pouvoir manipuler, comparer et stocker des coordonnées lors des opérations sur une surface.
- La partie algorithmique du projet est placée à la racine de celui-ci, car c'est elle qui est chargée de travailler sur les données du moteur et de les transmettre à l'interface. Elle résout les instances du problème, cherche, évalue et compare des solutions, puis les stocke. Nous avons donc une classe représentant une surface et son évaluation, « FieldEvaluation ». L'évaluation de la surface doit être à tout moment à jour, à partir du moment où la solution est réalisable. Il y a ensuite une classe permettant de trouver des voisins d'une solution, « LocalSearch » ; Il s'agit donc d'une recherche locale. Il serait envisageable de créer et d'utiliser d'autres méthodes de recherche de solutions, telles que des algorithmes génétiques ou de la programmation par contraintes. Enfin, puisque le problème est multi-objectifs, il nous faut stocker et mettre à jour les solutions ayant une évaluation non dominée. Ceci est pris en charge par la classe « Resolution », qui est chargée de gérer quelle est la solution sur laquelle on travaille et de maintenir à jour les solutions trouvées précédemment. C'est également elle qui doit exporter les résultats, sous la forme d'un front Pareto.
- La dernière partie est la partie interface graphique. Elle comprend l'affichage et les interactions pouvant être effectuées par l'utilisateur de l'application. L'application est pensée comme une aide à la prise de décision, il était donc essentiel que l'utilisateur ait la possibilité d'effectuer des actions avec celle-ci. L'interface doit donc être capable de communiquer de manière bilatérale avec les données du problème.

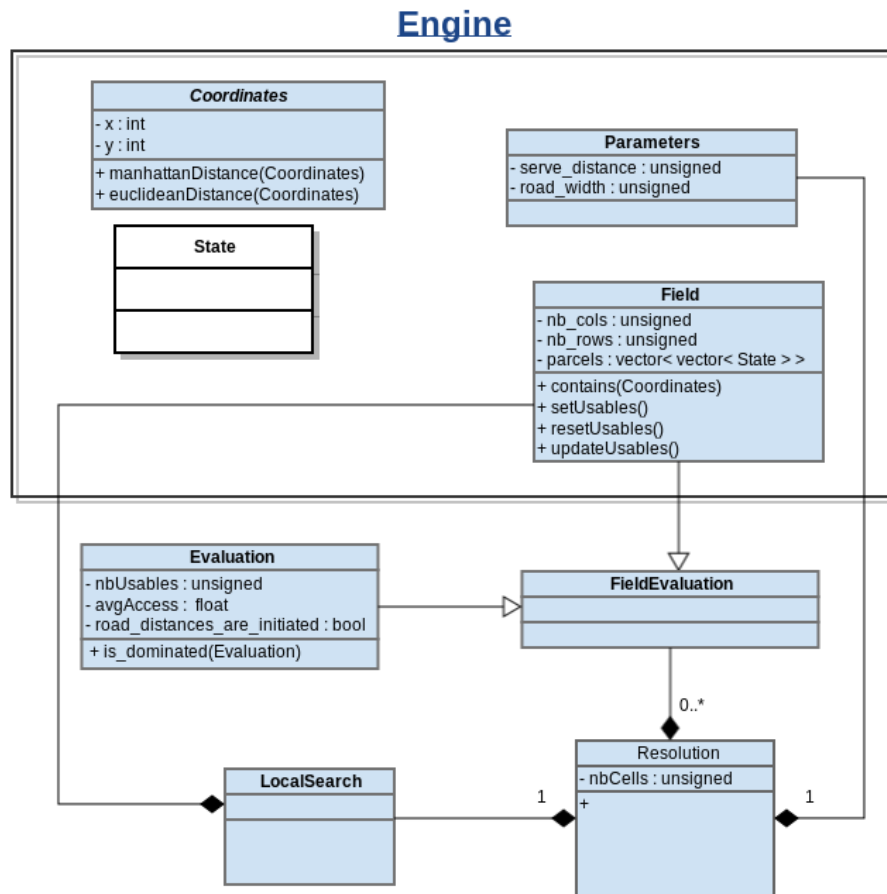


Illustration 2: Diagramme de classes du projet

2 Le moteur

2.1. Représentation des données du problème

La surface étudiée étant un rectangle, contenant des cases rectangulaires, la structure de donnée la plus évidente est de la stocker sous forme de matrice, à deux dimensions. C'est le choix qui a été fait, afin de faciliter la représentation du problème, donc sa compréhension et les réflexions sur ses spécificités et enjeux. Chaque cellule est donc représentée par un état dans la matrice. Une cellule est donc dans l'un de ces états : non définie, parcelle exploitable, parcelle non exploitable, route ou entrée-sortie.

Cependant, s'est posée la question de l'intérêt de l'utilisation d'une représentation à base de graphes, plutôt que matricielle. Cette solution paraît davantage indiquée dans le cas de l'utilisation d'un algorithme génétique, du fait du croisement possible de « branches » du graphe. De plus, cela aurait permis de plus facilement changer la forme des cellules, avec une forme hexagonale par exemple, par la suite. Malgré cela, cette représentation pouvait apporter une certaine confusion dans ce problème déjà complexe et vaste ; c'est pourquoi cette possibilité n'a pas été retenue.

2.2. Evaluation des solutions

2.2.1. Nombre de parcelles exploitables

Il s'agit de vérifier qu'elles sont les parcelles accessibles, donc exploitables. On applique cette règle : *Si une cellule, qui n'est pas une route, a une route dans son voisinage, à une distance inférieure ou égale à D cellules (D le paramètre de desserte définit au début, en distance Manhattan), elle est alors exploitable.* On compte ensuite le nombre de parcelles exploitables total.

2.2.2. Calcul du ratio entre distance directe et distance par les routes

Le ratio moyen permettant d'évaluer l'accessibilité est plus difficile et coûteux à calculer. En effet, il s'agit de trouver, pour chaque parcelle exploitable, la distance par les routes pour se rendre sur chaque autre parcelle exploitable de la surface. On divise ensuite la distance directe – Manhattan ou euclidienne – par la distance par les routes, pour obtenir le ratio associé à ces deux parcelles. Il faut ensuite faire la moyenne des tous les ratios ainsi obtenus. Il y a donc **$N \times (N-1)$** ratios à calculer, N étant le nombre de parcelles exploitables, pour obtenir le ratio total.

Or, pour calculer ce ratio, il nous faut donc, pour chaque parcelle, la distance vers chaque autre parcelle. Pour les obtenir, on a donc besoin de trouver le chemin le plus court pour aller d'une parcelle à une autre. Il s'agit d'un « pathfinding », une opération coûteuse, surtout lorsque l'on travaille avec les valeurs exactes et minimales, comme c'est le cas ici. De plus, on était amené à évaluer très fréquemment de nouvelles solutions, avec de légers changements. Une évaluation aussi peu performante aurait donc compromis la bonne progression du développement.

La solution a été d'utiliser le concept de **programmation dynamique**, vu cette année, en Optimisation Combinatoire. En effet, lors du calcul des plus courts chemins, il arrive fréquemment que l'on calcule plusieurs fois le chemin minimal entre une même route et une parcelle. Si deux routes sont adjacentes, il y a de fortes chances que le chemin le plus court vers une parcelle donnée soit le même, à une route près. Si la distance d'une route adjacente vers une parcelle donnée est déjà calculée, la distance du trajet vers cette parcelle passant par la route adjacente est donc égale à la valeur calculée, incrémentée de un. Pareillement, on peut utiliser le fait que le pathfinding est récursif. En effet, si un trajet précédemment calculé passait pour la route courante pour aller vers la parcelle voulue, la valeur de distance entre ces deux cellules est déjà calculée et on peut la réutiliser, sans nouveau calcul.

Cette amélioration a permis de grandement améliorer les performances de l'application. L'évaluation reste malgré tout une part importante des ressources utilisées par l'application, puisque refaite dans sa totalité fréquemment. La mise en place d'une éventuelle évaluation incrémentale, pour calculer l'accessibilité d'une solution, dérivée d'une autre dont on connaît l'accessibilité, s'est avérée bien plus restreinte et difficile à mettre en œuvre qu'escompté. Les tentatives sur ce sujet n'ont pas abouti. Cela vient du fait que l'ajout d'un chemin entre deux routes peut modifier la distance entre deux autres parcelles, qu'elles soient proches ou non du nouveau chemin. Il y faut donc une analyse d'un niveau élevé pour identifier les parties qui ont ou n'ont pas besoin d'être mises à jour.

3 La recherche de solutions améliorant les objectifs

3.1. Initialisation d'une solution

Si la solution initiale présente sur l'affichage de l'application n'est pas réalisable, on peut décider de relier les entrées-sorties (E/S) entre elles, ce qui va la rendre réalisable. Le placement des routes permettant de relier deux E/S est effectué par un algorithme. Il y a plusieurs configurations de placement des E/S à prendre en compte pour relier ces cellules en évitant de placer des routes près des bords, ce qui a rend moins de cellules exploitables.

On vérifie d'abord si les deux cellules sont sur des bords en face l'un de l'autre. Dans ce cas, une bonne méthode de placement des routes est de placer deux lignes de routes droites parallèles, reliées par une autre ligne perpendiculaire aux deux autres. Cela forme à l'intersection des deux routes parallèles. Le placement de la ligne perpendiculaire est aléatoirement sélectionné. Les deux objectifs ne sont pas pris en compte, on cherche à placer le moins de routes possible, lors de cette initialisation.

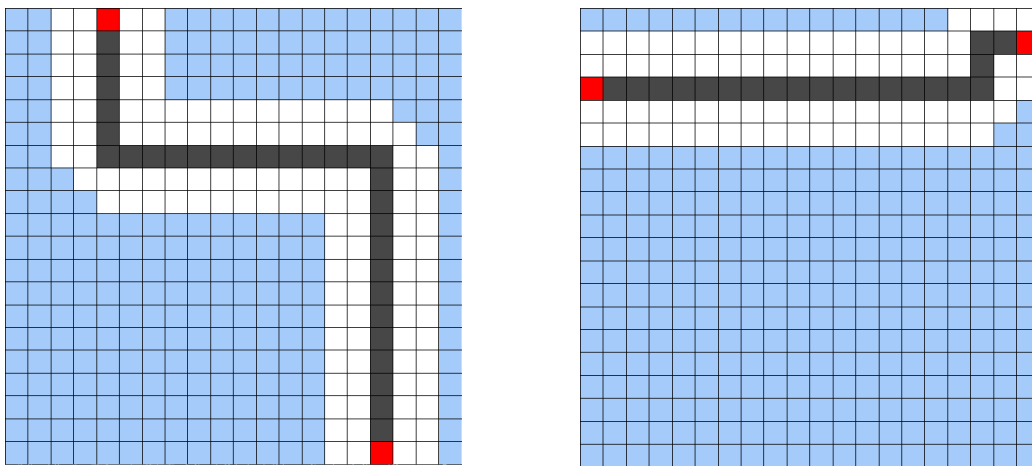


Illustration 3: Deux exemples d'initialisation avec coudes

Si les deux E/S se trouvent sur des côtés perpendiculaires, il suffit de faire un angle droit entre ces deux cellules. On préférera par contre faire cet angle à l'intérieur de la surface plutôt que le long de ses bords, afin d'avoir le plus de cellules exploitables possibles, avec le même nombre de routes placées. Il n'y a dans ce cas aucun aléatoire. Dans les cas où les deux E/S sont sur un même bord, il s'agit d'un cas particulier de l'angle droit, où seule une des deux lignes est placée.

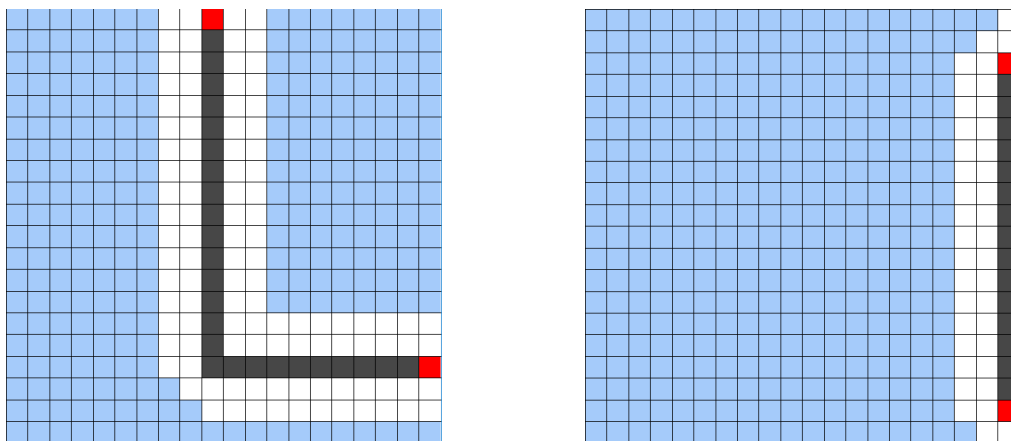


Illustration 4: Deux exemples d'initialisations avec angles droits

3.2. Maximisation des objectifs par recherche locale

Le choix du type d'algorithme, lors de la recherche de solutions améliorantes, s'est porté sur la recherche locale. Ce choix est justifié par le fait que la structure d'une surface simplifiait ce procédé et que j'étais à l'aise avec ce procédé. De plus, il ne restreindrait pas l'application future des évolutions évoquées.

3.2.1. Maximisation du nombre de parcelles exploitables

La première étape consiste à maximiser le nombre de parcelles exploitables. Pour ceci, un algorithme glouton a été utilisé. Pour chaque route ajoutée, on cherche celle qui augmentera le plus le nombre de routes exploitables en la plaçant. On parcourt donc toutes les parcelles adjacentes à au moins une route. Il s'agit de toutes nos parcelles pouvant éventuellement être « transformées » en routes. Pour chacune de ces parcelles, on compte combien a-t-elle de parcelles non exploitables dans son voisinage, à une distance (Manhattan) maximale correspondant au paramètre de desserte. On choisit donc la celle ayant le plus de parcelles non exploitables dans son voisinage, afin de les rendre exploitables lors de l'ajout de la route.

En revanche, on évite de sélectionner les routes qui ont le plus d'autres routes voisines. Il n'y a donc pas de stratégie de placement lorsqu'on place plusieurs routes. Ainsi, l'ajout d'une route peut diminuer l'intérêt d'une route précédemment placée. Cette méthode n'est donc pas optimale, mais puisqu'on est sur un problème multi-objectifs, il ne serait pas judicieux de se concentrer sur un seul des ces objectifs.

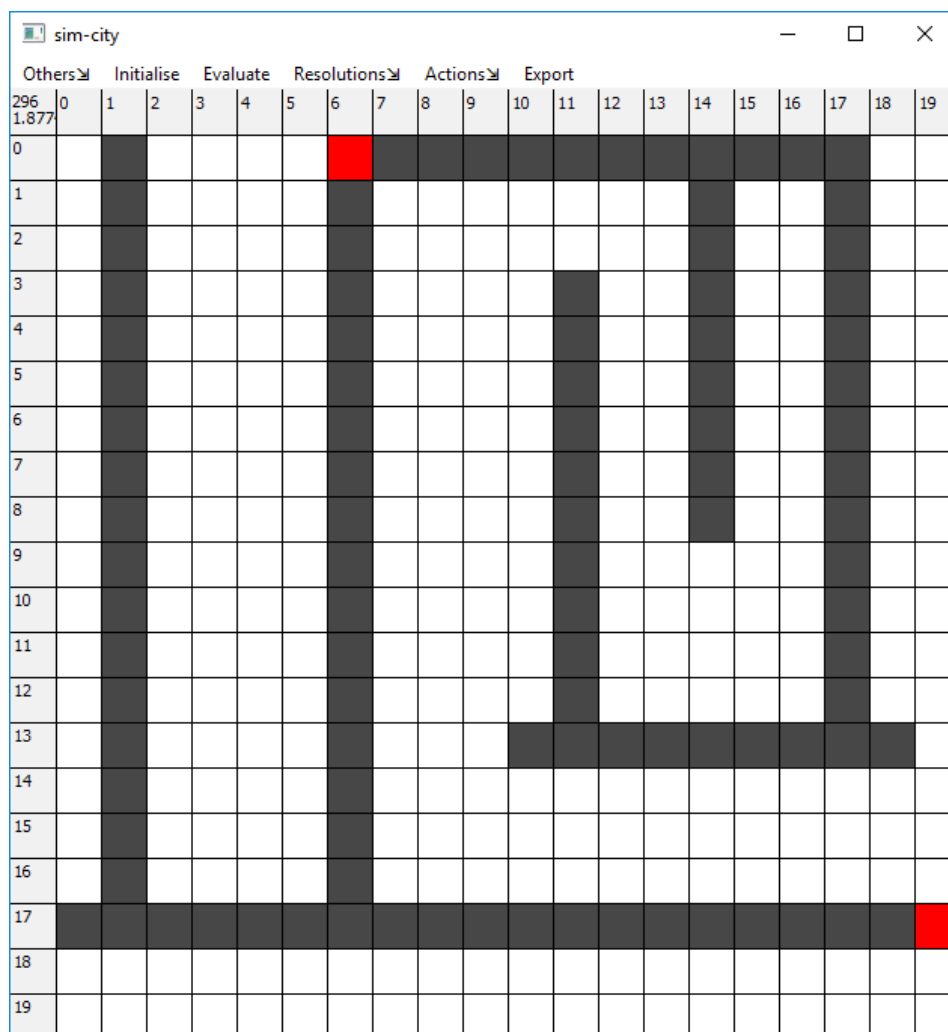


Illustration 5: Parcelle après maximisation complète du nombre d'exploitables

3.2.2. Maximisation de la circulabilité

La seconde étape va donc chercher à maximiser le second objectif : l'accessibilité. Dans ce cas, l'ajout de routes les unes après les autres ne permettait pas de maximiser l'objectif correctement. En effet, l'ajout d'une route seule peut soit supprimer une parcelle exploitable et : Créer un passage entre deux routes séparées par seulement une cellule, ce qui va créer de nouveaux itinéraires de déplacement, mais ce cas est assez rare si la distance de desserte est supérieure à un. Ou ne pas créer de possibilités de chemins améliorants. Dans ce cas, si de nouvelles parcelles sont rendues exploitables, le changement d'accessibilité globale dépendra seulement du fait que l'accessibilité de la parcelle supprimée soit inférieure ou supérieure à la moyenne. Sinon, il dépendra également de l'accessibilité des nouvelles parcelles exploitables. Il était donc nécessaire de procéder de manière différente.

Pour pallier cette difficulté, il a donc été envisagé de chercher à ajouter des « chemins » de routes entre deux routes existantes, afin de créer des passages améliorant globalement la circulation, à

l'aide d'un changement local. Pour cela, un algorithme du type glouton est une fois encore utilisé. On va chercher à essayer tous les ajouts de chemins qui peuvent être considérés pertinents, et choisir celui qui améliore le plus la circulabilité.

Pour effectuer cela, on parcourt toutes les routes présentes sur la surface. Pour chacune, on cherche toutes les routes accessibles dans son voisinage, à partir d'une distance donnée au lancement de la maximisation. Dans le cas présent, la distance est fixée à deux fois la distance de desserte, cela permet de créer des chemins entre deux routes parallèles, contenant entre elles un nombre de parcelles égal à deux fois la distance de desserte, un cas courant dans les solutions créées suite à une maximisation du nombre de parcelles exploitables. Mais cette valeur pourrait facilement être modifiée ou définie par l'utilisateur.

Pour chaque parcelle accessible, on teste chaque chemin possible – y allant de manière directe, sans faire de détour, avec deux lignes droites maximum –, les autres sont exclus. On calcule donc le gain de chaque chemin. Celui-ci est obtenu en calculant l'évaluation avant l'ajout du chemin, en plaçant les routes sur la surface, puis en évaluant le voisin ainsi obtenu. Le gain est le résultat de la soustraction de la première valeur d'accessibilité à la seconde. Ensuite, les routes sont retirées de la surface. Parmi tous les gains calculés, le plus élevé est conservé, avec le chemin associé. Il faut donc ajouter à nouveau les routes du chemin, puis réévaluer la surface afin d'obtenir les nouvelles valeurs d'accessibilité et de nombre de cellules exploitables.

Il y avait la possibilité d'ajouter plusieurs chemins à chaque passage, soit en plaçant autant de chemins qu'il est demandé d'en ajouter, soit en ajoutant seulement ceux qui ont un gain proche du gain maximal. Mais cela n'a pas été fait, car l'ajout d'un chemin peut modifier l'intérêt de l'ajout d'un autre chemin et on risquerait d'ajouter plusieurs chemins côte à côte. Il faudrait en conséquence apporter plusieurs modifications, pour un gain de temps par rapport à une perte en précision des résultats qui n'a pas été évalué, et n'est donc pas forcément substantielle.

On peut choisir d'alterner les étapes, en fournissant un nombre de routes à ajouter pour le nombre d'exploitables – sachant que la valeur 0 permet d'ajouter des routes tant qu'on améliore l'objectif – et le nombre de chemins pour l'accessibilité. Lorsque la solution contient trop de routes déjà placées, on peut alors réinitialiser la surface avec les mêmes E/S, puis utiliser une stratégie différente d'ajout de routes, tout en conservant les solutions non dominées déjà trouvées. Il aurait été pertinent, d'ajouter une option pour pouvoir supprimer des routes présentes, afin de maximiser l'un des deux objectifs. Lorsqu'on supprime une route, on peut potentiellement « gagner » une route exploitable. Si les cellules qui sont transformées en routes ne sont pas essentielles lors du pathfinding, et que les nouvelles parcelles exploitables créées.

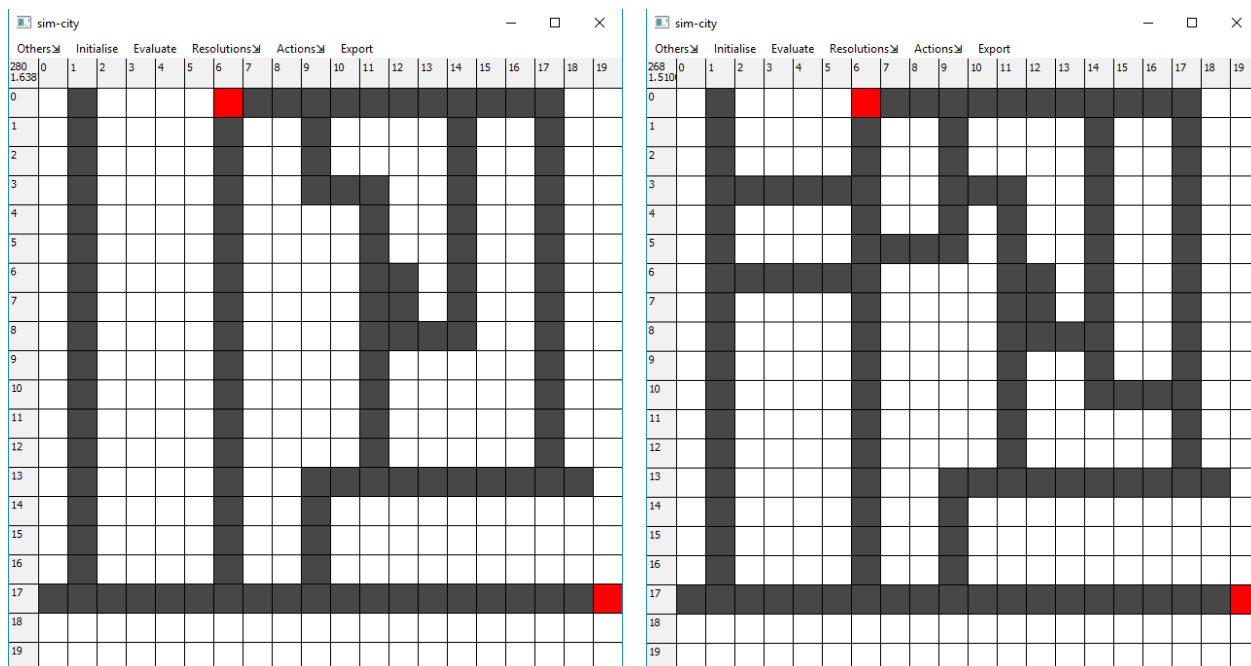


Illustration 6: La même parcelle, après ajout de 3 puis 7 chemins

3.3. Persistance des solutions non dominées

Afin de conserver les résultats intéressants trouvés au cours d'une résolution, il stocker les solutions dont l'évaluation des objectifs n'est pas dominées par une autre solution. Ici, la méthode du **générer et tester** est utilisée ; quand une solution est évaluée, on vérifie immédiatement si elle est dominée par une solution précédente. Dans le cas contraire, on la propage, afin de supprimer les autres solutions qu'elle pourrait dominer, puis on la stocke dans l'ensemble des solutions non dominées. Si une solution est dominée, ses valeurs d'évaluation sont stockées dans un « buffer », prête à être dessinée, si l'on veut garder une trace des solutions écartées. Une solution stockée contient la surface et son évaluation, avec toutes ses données calculées, on peut donc retravailler dessus par la suite, si on le souhaite.

Lors de l'évaluation d'importantes quantités de données sont générées lors de la recherche de chemins. Les valeurs de distance entre toutes les routes et toutes les autres routes sont stockées dans une matrice à quatre dimensions (2 pour les coordonnées de départ et deux pour les coordonnées d'arrivée). Puisqu'on utilise la programmation dynamique, il faut stocker toutes les valeurs calculées afin de les réutiliser pour calculer les suivantes. On pensait également les utiliser lors d'une mise à jour incrémentale des données lors de l'ajout de routes, pour ne pas les recalculer entièrement lors de la nouvelle évaluation, mais il s'est avéré qu'elles ne pouvaient pas être utilisées. On peut alors se poser la question de l'intérêt de les conserver lors du stockage d'une solution en vue de la réutilisation future.

Une solution serait de stocker tout le chemin en plus de la valeur de distance entre chaque case, afin de mettre à jour seulement les chemins qui peuvent être affectés par l'ajout de routes. Mais cela

entraîne une quantité bien plus importante de données, et la faisabilité et la pertinence de cette solution n'ayant pas été démontrées, cette possibilité n'a pas été mise en place.

4 L'interface graphique : Affichage et Interactions :

4.1. Affichage

L'affichage est donc géré avec le framework Qt. Une surface est affichable à l'aide d'un widget, représenté par une classe. Cette classe nécessite la surface à afficher et son évaluation, afin de pouvoir dessiner une « Hotmap ». Les E/S sont rouges, les routes grises, les parcelles non exploitables bleu clair et les exploitables blanches. Cet affichage doit être mis à jour lors d'une interaction par l'utilisateur et à la suite de l'exécution d'un algorithme de résolution. La solution affichée par l'application est la dernière solution trouvée.

Un affichage alternatif est possible. En effet, une fois qu'une solution est réalisable, l'utilisateur peut avoir la possibilité d'afficher une « Hotmap » de celle-ci, afin de le guider dans ses prises de décisions. Ce terme désigne l'affichage de couleurs, allant de vert à rouge, selon l'écart avec la moyenne de l'accessibilité de la parcelle. Cet affichage permet de repérer les zones qui sont les moins accessibles, et donc d'essayer de placer des routes afin d'améliorer leur accessibilité, dans le but d'améliorer la circulabilité. Plus une région contient de parcelles d'une couleur proche du rouge, moins son accessibilité est bonne. On peut donc concentrer nos efforts à l'améliorer manuellement.

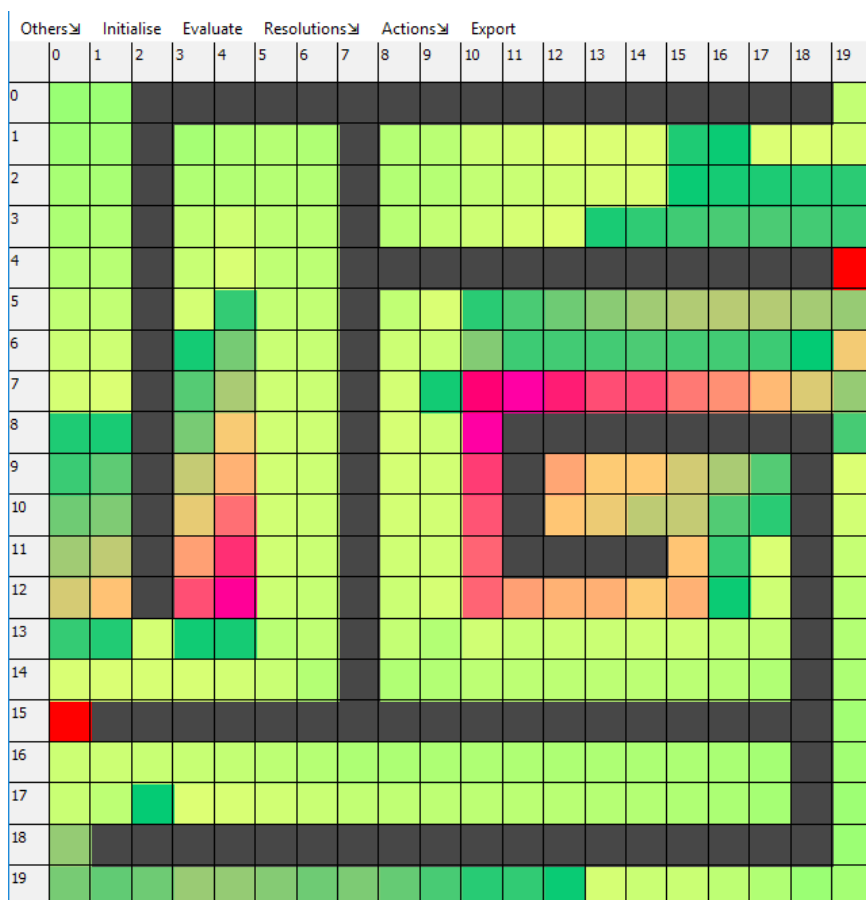


Illustration 7: Exemple de « Hotmap »

4.2. Interactions

Le widget d’affichage d’une surface permet non seulement de l’afficher, mais également, grâce aux fonctionnalités du framework Qt, de recevoir et de traiter des entrées de la part de l’utilisateur. L’application étant conçue comme une aide à la décision, elle gère différentes interactions avec celui-ci.

Au lancement du logiciel, l’utilisateur peut ajouter ou supprimer des E/S et des routes, en cliquant sur les zones concernées, avec le pointeur de la souris. Un clic gauche va alors placer une route, tandis qu’un clic droit ajoutera une E/S. Il peut choisir de créer une solution réalisable manuellement, puis de l’évaluer. Sinon, il peut choisir de placer uniquement des E/S et demander à l’application de les relier automatiquement, afin de créer une solution réalisable automatiquement. Il a ensuite la possibilité de lancer une résolution sur une solution initiale complétée automatiquement ou manuellement. Pour chaque action du bouton « Initialise », une E/S et la suivante sont reliées. Ainsi, si le nombre N d’E/S est supérieur à deux, on est sûr de toutes les relier avec N clics sur le bouton « Initialise ».

Comme indiqué plus haut, on peut choisir de d’abord maximiser le nombre d’exploitables, avant d’essayer de « corriger » la solution trouvée. Une autre possibilité est de procéder par petites itérations, en indiquant un nombre de routes à ajouter à chaque étape de maximisation du nombre d’exploitables et un nombre de chemins à chaque étape de maximisation de la circulabilité. Pour exécuter une résolution, on clique sur « Resolution » dans la barre des menus, puis on choisit « Usable » ou « Access », pour augmenter respectivement le nombre d’exploitables ou la circulabilité. On pourrait ajouter une option, qui permettrait de maximiser les deux objectifs en une seule étape. L’utilisateur devrait alors indiquer une valuation pour chaque objectif, ce qui transformerait temporairement le problème en un problème mono-objectif. En effet, la fonction objectif, donnant l’évaluation d’une solution, serait alors de la forme :

$$\max f(x) = \alpha \times \frac{nbexploitables}{nbtotal} + \beta \times \frac{1}{accessibilite}$$

D’autres menus sont disponibles, afin d’avoir des informations sur l’application, ou pour remettre vider la matrice, en repartant d’une solution vierge, ou contenant la configuration initiale de la surface.

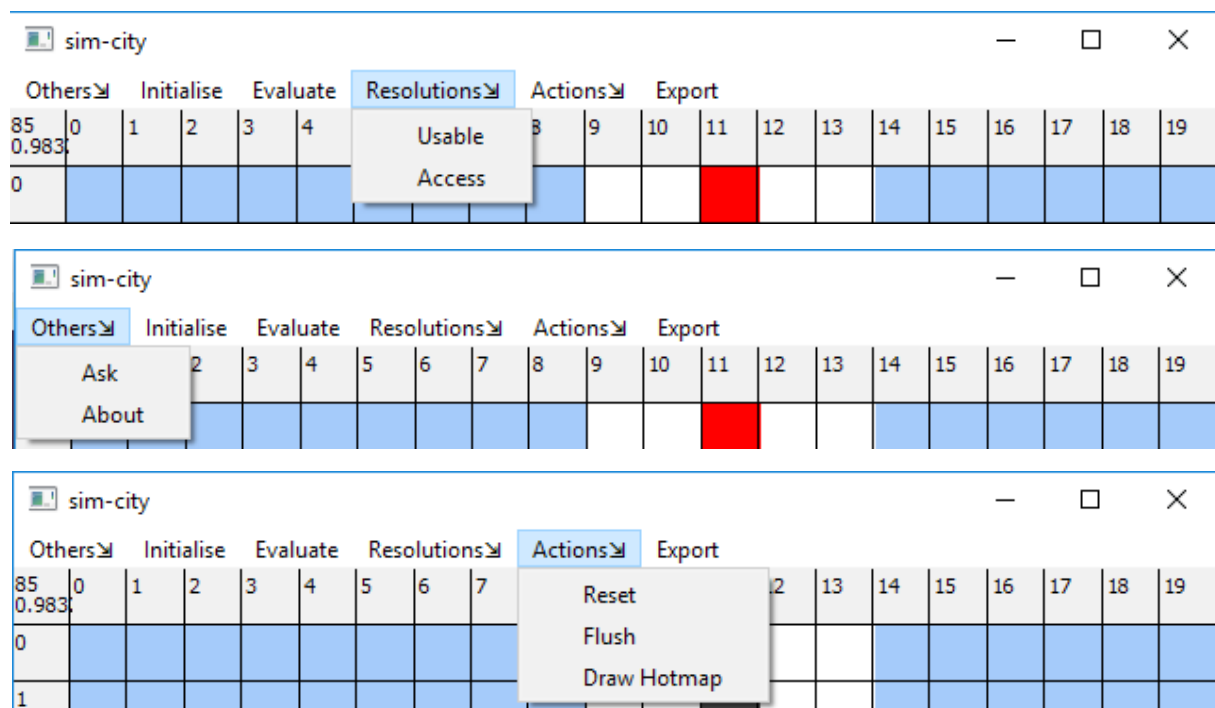


Illustration 8: Menus de l'application

4.3. Affichage et utilisation du front Pareto

La dernière étape pour l'utilisateur est de choisir une des solutions qui lui convient, afin de pouvoir la mettre en place s'il le désire. Pour effectuer cela, il peut voir les évaluations de toutes les solutions trouvées non dominées, qui sont stockées en mémoire, placées sur un front Pareto. Ainsi, il lui suffit de choisir le compromis entre les deux objectifs qui lui convient le mieux. Pour afficher les évaluations des solutions dans un Gnuplot a été préféré à la création d'un repère orthonormé et aux dessins des points dans celui-ci. Le principal avantage est la puissance et la rapidité de mise en place de Gnuplot. Mais si on souhaite continuer l'application, ce choix pourrait être remis en cause et on pourrait passer à une solution « maison », permettant une interaction simplifiée avec les solutions et une meilleure lisibilité.

Dans la seconde illustration, au lieu de croix, ce sont des numéros qui sont affichés. L'avantage est que l'utilisateur peut alors choisir d'afficher une des solutions précédentes en donnant son numéro. Le défaut est que c'est moins lisible, ici, les numéros 45 à 47 sont à peine lisibles.

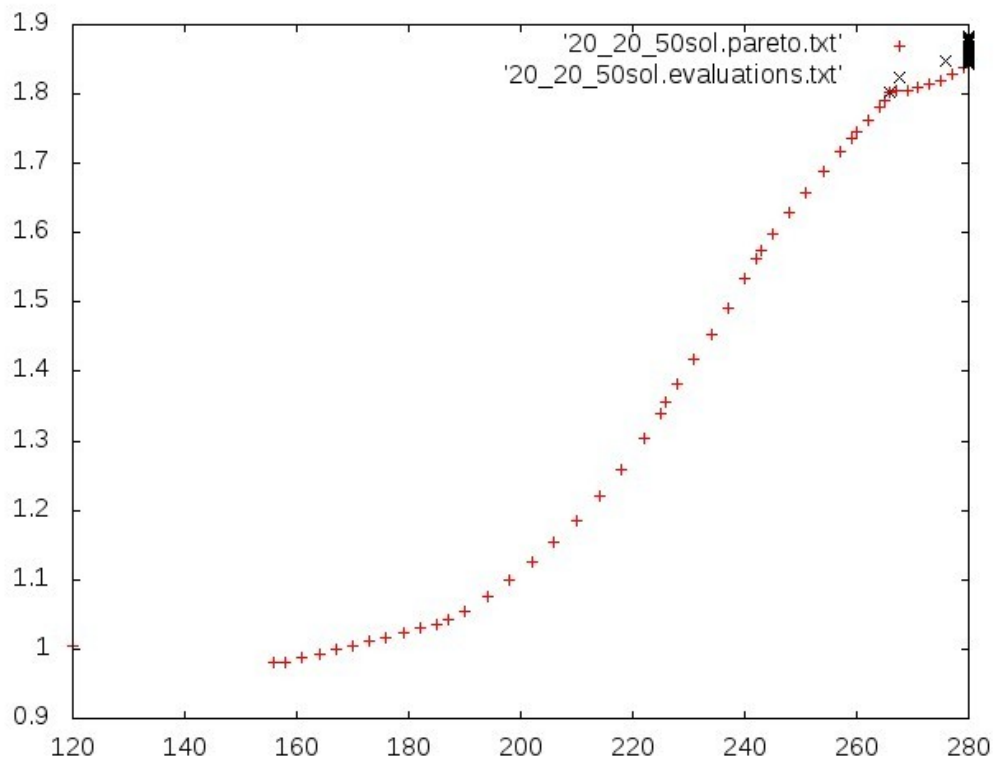


Illustration 9: Front Pareto, sans numérotation

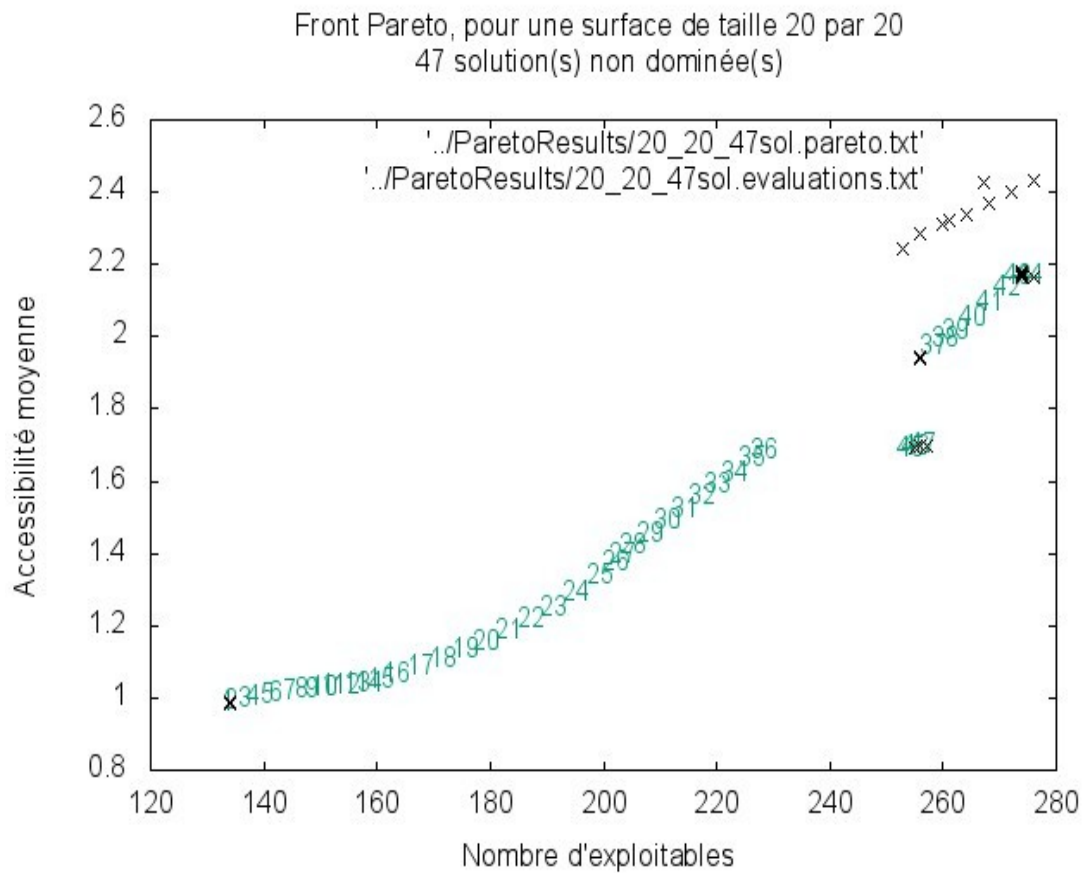


Illustration 10: Front Pareto, avec numérotation

Déploiement

J'ai décidé de donner un caractère professionnel à l'application. Se trouvait donc sur le site Github de celle-ci, un « readme » au format markdown, décrivant son but et détaillant les fonctionnalités à venir, celles envisagées, et celles déjà réalisées, mis à jour selon l'avancement du développement. Les menus de l'application sont en anglais. Le code est en anglais, en dehors des commentaires qui sont intégralement en français.

Une balise indiquant si la dernière version du projet compile correctement est également présente, grâce à l'intégration du service Travis. Ensuite, un léger site web, dédié à l'application a été mis au point, à partir du contenu du readme. Une documentation a été créée, à l'aide de *Doxygen*, et ajoutée au site web. Toutes les classes, fonctions et attributs publics sont donc documentés, en plus d'une description succincte du but du programme. Un système ressemblant à des logs a été mis en place, à partir de « flags » préprocesseurs, réunis dans un fichier. Il suffit de passer la valeur des flags que l'on souhaite à 1, au lieu de 0. Il faut alors recompiler le programme avant de lancer l'exécution, pour obtenir les logs concernant les flags activés. Afin de protéger mon travail, une licence GPLv3, dernière version en date de la licence GPL, a été appliquée au projet, et l'ajout d'une balise dans le readme permet d'indiquer clairement cette licence.


Une fois le développement bien avancé, des outils de profilage, de vérification de code et de fuites mémoires ont été utilisés. Ainsi, *valgrind* a permis de « chasser » les fuites mémoires, même si certaines persistent, principalement à cause de l'utilisation de Qt et de ses éléments graphiques. *Callgrind* est un outil intégré à *valgrind*, a permis de générer des fichiers lisibles par *Kcachegrind*, un utilitaire graphique permettant de visualiser quelles sont les fonctions les plus appelées et les plus coûteuses. J'ai tenté d'utiliser *gprof*, mais, n'étant pas un outil graphique, il était plus difficile de repérer les « points chauds » du programme qu'en utilisant *Kcachegrind*, il lui a donc été préféré. L'outil *Cppcheck*, utilisé sous Windows, a lui permit d'effectuer quelques micro-optimisations du code, principalement des portées de variables qui pouvaient être réduites.

Tout au long du développement, les tests ont été exécutés, afin de vérifier que les fonctionnalités du moteur qui étaient correctes le soient toujours. De plus, dans l'éventualité de reprise du programme, afin de continuer son développement ou sa maintenance, par une autre personne ou non, cela permet de s'assurer que les modifications apportées n'empêchent pas le fonctionnement des anciennes fonctions.

View on GitHub

Urbanisme

TER visant à l'optimisation du placement de routes sur une surface donnée, maximisant les zones exploitables et leur accessibilité.



Documentation du projet (Doxygen)

Liste des classes

Index des classes

Hiérarchie des classes

Membres de classe

Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

C Coordinates	Représentation et opérations simple sur des coordonnées dans un plan
C CoordinatesTestFixture	Tests sur les méthodes de la classe Coordinates
C Evaluation	Classe chargée de travailler sur les données en entrée du problème pour trouver des solution réalisables, optimisant les objectifs suivants :
C Field	Classe représentant la surface -le terrain- et contenant les opérations que l'on effectue sur celui-ci. L'état d'une instance de cette classe à la fin de l'algorithme représente une solution trouvée après la recherche. On peut donc avoir plusieurs Field lors d'une exécution
C FieldEvaluation	La classe permettant de stocker une surface et de calculer et mettre à jour son évaluation
C FieldTestFixture	Tests sur la classe Field
C FieldWidget	Widget d'affichage de l'automate cellulaire. Ce composant permet de suivre graphiquement l'évolution de l'automate. Il utilise un système de buffering et s'appuie sur la classe Foret du moteur
C Gnuplot	The Gnuplot class, can receive parameters, then transmit them to Gnuplot via pipes Pipes are closed when class is destroyed
C GnuplotException	A C++ interface to gnuplot
C LoadProgress	Classe abstraite, pour suivre l'évolution des chargements (en %)
C LoadWindow	Permet d'afficher une fenêtre avec une barre de progression pour les chargements Utilise la classe abstraite LoadProgress
C LocalSearch	La classe chargée d'effectuer des recherches locales sur les différentes solutions TODO ajouter les entrées et sorties
C MainWindow	Fenêtre principale de l'application. Chargée d'afficher la surface, les menus, les résultats et gérer le lancement des opérations
C MinPathRoadTestFixture	Tests sur la méthode <code>calcRoadDistance()</code> de la classe Evaluation
C NextCoordinatesTestFixture	Tests sur la méthode <code>nextCoordinates()</code> de la classe Field
C Parameters	Représente les paramètres du problèmes
C Resolution	La classe effectuant l'ensemble de la résolution et le front Pareto Elle contient l'intégralité des données du problème et est chargée de lancer les algos, de stocker les solutions et de construire le front pareto résultant

Projet TER : Urbanisme

Compilation Travis

Licence

build passing

License GPLv3

Organisation du travail et du développement

I. Etudes préliminaires :

Illustration 11: Deux pages du site, dont une page de la documentation.

Conclusion

Le sujet demandait des compétences dans divers domaines. Ceux-ci sont l'optimisation combinatoire, la résolution de problèmes, la bonne utilisation – organisation et optimisation – du développement dans un langage adapté et performant, et la création d'une interface graphique complète.

Le développement de l'application a fait appel à plusieurs notions. Il a été nécessaire de mettre en place, et de respecter, certaines méthodologies, d'essayer différents types d'algorithmes et approches, étudiées durant l'année du master 1 :

- ◆ La programmation dynamique
- ◆ Mettre en place une évaluation de solutions, puis une recherche de voisins grâce à une recherche locale, à partir de cette évaluation.
- ◆ La gestion du multi-objectifs et la création d'un front Pareto, en générer et tester, et des contraintes engendrées
- ◆ Une utilisation importante de structures de données complexes : matrices à 4 dimensions, listes d'éléments contenant d'importantes données calculées.
- ◆ La gestion de ces données : utilisation d'une ancienne solution, sans devoir recalculer ses données, gestion de la mémoire, mettre à jour les données lorsque c'est pertinent et seulement lorsque c'est pertinent pour éviter d'effectuer des longs calculs s'ils ne sont pas utiles dans certains cas.
- ◆ Gestion de la mémoire et du partage des données, contenues dans les structures, entre les différents niveaux de l'application : affichage, interactions, résolution, export et sauvegarde ...
- ◆ Des concepts du génie logiciel, comme la STL du C++ et une organisation du travail
- ◆ La mise en place de tests et devoir maintenir un code clair et commenté était indispensable pour le bon déroulement du projet sur une période de temps plus étendue qu'à l'accoutumée.

Il aurait été vain de tenter de traiter l'ensemble de l'étendue du problème. Il a donc fallu analyser et définir les éléments primordiaux à mettre en place : déterminer l'intérêt des algorithmes génétiques dans ce cas précis, utiliser ou non des graphes, améliorer le pathfinding et prendre en compte la largeur des routes dans ce dernier, ... ; On sélectionne alors les aspects à traiter lors de la résolution. Ainsi, certaines possibilités n'ont pas été traitées. Il est donc possible de continuer le projet, afin de comparer différents algorithmes et implémentations, d'ajouter des interactions et éléments d'information fournies à l'utilisateur.

Ce sujet était très intéressant et instructif, car très large dans les compétences développées et les domaines d'application. Il a donc tout à fait rempli son rôle de validation d'approfondissement des acquis de cette année, tout offrant une première approche concrète d'un travail de recherche. Je remercie donc une nouvelle fois Benoit Da Mota et Adrien Goëffon pour leur proposition de ce sujet.

Table des illustrations

Illustration 1: Surface d'exemple.....	6
Illustration 2: Diagramme de classes du projet.....	10
Illustration 3: Deux exemples d'initialisation avec coudes.....	12
Illustration 4: Deux exemples d'initialisations avec angles droits.....	13
Illustration 5: Parcelle après maximisation complète du nombre d'exploitables.....	14
Illustration 6: La même parcelle, après ajout de 3 puis 7 chemins.....	16
Illustration 7: Exemple de « Hotmap ».....	17
Illustration 8: Menus de l'application.....	19
Illustration 9: Front Pareto, sans numérotation.....	20
Illustration 10: Front Pareto, avec numérotation.....	20
Illustration 11: Deux pages du site, dont une page de la documentation.....	22

ENGAGEMENT DE NON PLAGIAT

Je, soussigné(e)
déclare être pleinement conscient(e) que le plagiat de documents ou d'une
partie d'un document publiée sur toutes formes de support, y compris l'internet,
constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.
En conséquence, je m'engage à citer toutes les sources que j'ai utilisées
pour écrire ce rapport ou mémoire.

signé par l'étudiant(e) le

**Cet engagement de non plagiat doit être signé et joint
à tous les rapports, dossiers, mémoires.**

Présidence de l'université
40 rue de rennes – BP 73532
49035 Angers cedex
Tél. 02 41 96 23 23 | Fax 02 41 96 23 00

