UNIVERSITÉ D'ANGERS

TER

Problème d'exploitabilité et d'accessibilité dans un graphe

Auteur :
Vincent HENAUX

Tuteurs :
Dr. Benoit DA MOTA
Dr. Adrien GOËFFON

Rapport du travail encadré de recherche
de fin de Maîtrise d'informatique

dans le

département d'informatique
UFR Sciences

2017

Table des matières

Chapitre 1

Introduction

Dans le cadre du Master 1 d'informatique de l'UFR Sciences d'Angers, je dois effectuer un travail encadré de recherche. C'est un travail de fin d'année qui a pour objectif soit de nous initier, soit, au contraire, nous spécialiser dans un domaine de recherche.

Pour réaliser ce travail je me suis intéressé à un problème d'optimisation appliqué dans la théorie des graphes. Derrière l'apparente simplicité du problème des enjeux industriels, notamment dans l'électronique, en dépendent. Le problème sera présenté en détail dans le prochain chapitre. Mais pour décrire très simplement le problème, l'objectif est de rendre une carte très accessible avec des routes tout en limitant au maximum leur nombre.

Au delà de la résolution du problème, nous nous intéresserons surtout au comment le résoudre :
-Une méthode exhaustive est-elle suffisante à la résolution des problèmes de petite taille ? de taille raisonnable ? de taille industrielle ?
-Si la méthode exhaustive n'est pas satisfaisante, quelles méthodes de résolution tenterons nous d'implémenter et pourquoi ?

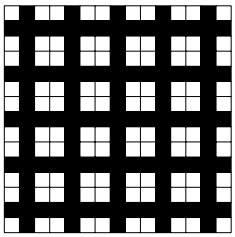
Dans une première partie nous étudierons une méthode de résolution complète et nous en tirerons les conclusions qui semblent s'imposer pour cette approche.

Puis chaque partie qui suivra sera consacrée à une méthode de résolution précise, avec en fin de chapitre un bilan et une direction à suivre.

Chapitre 2

Présentation et formalisation du problème

Le problème d'accessibilité et d'exploitabilité est un problème d'organisation des villes sur un terrain quelconque. En somme on cherche à savoir laquelle des configurations des routes pour circuler entre les différents points d'intérets d'une ville est la meilleure. C'est ainsi que l'on peut commencer par une résolution de ce problème sur une carte divisée en plusieurs cases qui, au choix du solveur, sera soit une route, soit une parcelle exploitable. Une configuration qui semble évidente est celle des villes américaines. Une grille de route qui donne accès à tout le reste de la ville à partir de n'importe quel point et ce dans un temps raisonnable.



Le but de ce T.E.R. est de généraliser ce problème non seulement du point de vue de la représentation des informations mais aussi du point de vue de la méthode de résolution du problème. Nous choisissons de représenter la carte à "résoudre" sous forme de graphes, ainsi une grille ne sera plus une grille pour le solveur mais un graphe grille. Une case deviendra un noeud et l'adjacence de deux cases sera une transition entre les deux noeuds correspondants aux cases.

Soit V un ensemble de noeuds.
Soit N le nombre de noeuds de V.
Soit E un ensemble d'arêtes sur V tel que E $\subseteq$ VxV.
Soit γ une fonction de pondération des arêtes telle que $\gamma : E \to \mathbb{R}$.
Soit π une fonction de pondération des noeuds telle que $\pi : V \to \mathbb{R}$.
Soit G un graphe pondéré tel que G = (V,E,γ,π).

Une configuration des routes du graphe est une solution du problème. Cependant ces solutions sont soumises à des contraintes, si elles les suivent alors elles sont faisables, si non elles sont invalides.

Une solution du problème est une suite de N bits. Chaque noeud défini comme une route sera un 1 et chaque parcelle exploitable un 0.
L'ensemble S des solutions est défini par S = $\{0,1\}^N$

On fixe la solution triviale Λ qui ne demande aucun calcul, néanmoins de piètre qualité, comme étant la solution qui définie tout les noeuds comme des routes.

Le problème étant un problème d'accessibilité entre noeuds nous avons besoin de l'ensemble des chemins.

Soit Ψ_λ, $\lambda \in$ S, l'ensemble des suites d'arêtes qui forment un chemin, dont les noeuds intermédiaires sont définis commes des routes par λ, i.e. $\forall \psi \in \Psi_\lambda$ on a une liste de la forme $((n_0, n_1), (n_1, n_2), ..., (n_{k-1}, n_k))$ telle que $\forall$j$\in [\![$1,k-1$]\!]$, $\lambda_{n_j} = 1$.
Et soit $\overset{b}{\underset{a}{\Psi}}_\lambda \subseteq \Psi_\lambda$ l'ensemble des chemins commençants par le noeud a et finissants par le noeud b.
ex : {((a,b)) , ((a,n_1),(n_1,b)) , ... , ((a,n_1),(n_1,n_2),...,(n_k,b))}

Une solution faisable est une solution qui respecte deux contraintes du problème :
-Une solution λ respecte la contrainte de l'accessibilité des noeuds par une route si et seulement si $\forall$ v $\in$ V, $\exists$ w $\in$ V t.q $\lambda_w = 1$ et (v,w) $\in$ E.
-Une solution λ respecte la contrainte de la connexité des routes si et seulement si $\overset{b}{\underset{a}{\Psi}}_\lambda \neq \emptyset, \forall$ a,b $\in$ V t.q. $\lambda_a = \lambda_b = 1$.

Une solution valide peut être évaluée, au cours de cette opération un score lui sera attribué. Cependant, même si l'on décrit cette évaluation ici, elle reste arbitraire. L'évaluation n'est donc pas unique mais bien multiple, celle présentée ici est celle qui semble être la plus pertinente.
Le second objectif de ce T.E.R. est l'optimisation de ce score. En vérité le score est double. Tout d'abord pour évaluer l'exploitabilité du solution, à savoir si une solution a trop de routes par rapport au nombre de parcelles exploitables. Puis pour évaluer l'accessibilité d'une solution, c'est à dire comparer la solution à évaluer avec la solution Λ où tous les noeuds sont des routes, qui est elle très accessible. Ce problème d'optimisation est donc bi-objectif.

L'exploitabilité est définie comme le coût d'opportunité lié à la mise en oeuvre de la solution. C'est donc la somme des coûts des noeuds qui sont définis comme des routes par la solution. i.e. $\sum_{i=1}^{N} \pi(V_i) \times \lambda_i$.
L'accessibilité est quant à elle évaluée comme le ratio maximal des plus courts chemins entre parcelles exploitables en passant par les voies d'accès de la solution à évaluer divisé par leur correspondant de la solution Λ.
i.e. $max\left(\frac{min((\sum_{i=1}^{|\psi|} \lambda(\psi_i))\forall\psi\in\overset{b}{\underset{a}{\Psi}}_\lambda\subseteq\Psi_\lambda)}{min((\sum_{i=1}^{|\psi|} \lambda(\psi_i))\forall\psi\in\overset{b}{\underset{a}{\Psi}}_\lambda\subseteq\Psi_\Lambda)} \forall a, b \in V \ t.q. \ \lambda_a = \lambda_b = 0\right)$

L'objectif final du problème d'optimisation est non seulement de minimiser les deux objectifs, mais de faire une liste de solutions non comparables entre elles regroupées dans un front Pareto. On peut le spécifier à l'aide de l'opérateur "<" tel que $\lambda_1 < \lambda_2 \iff$ exp(λ_1) < exp(λ_2) et acc(λ_1) < acc(λ_2), $\lambda_1, \lambda_2 \in$ S.
Soit le front Pareto $\subseteq$ S tel que $\forall \lambda_1 \in$ FP (front Pareto), $\nexists \lambda_2 \neq \lambda_1 \in$ FP tel que $\lambda_2 < \lambda_1$.

En partant d'ici l'idée la plus simple de méthode, pour résoudre ce problème, est un algorithme exhaustif des solutions. C'est donc un tel procédé que l'on étudiera en premier.

Chapitre 3

Algorithme complet

3.1 Algorithme

Pour chaque solution s :

    Si s correspond aux contraintes :

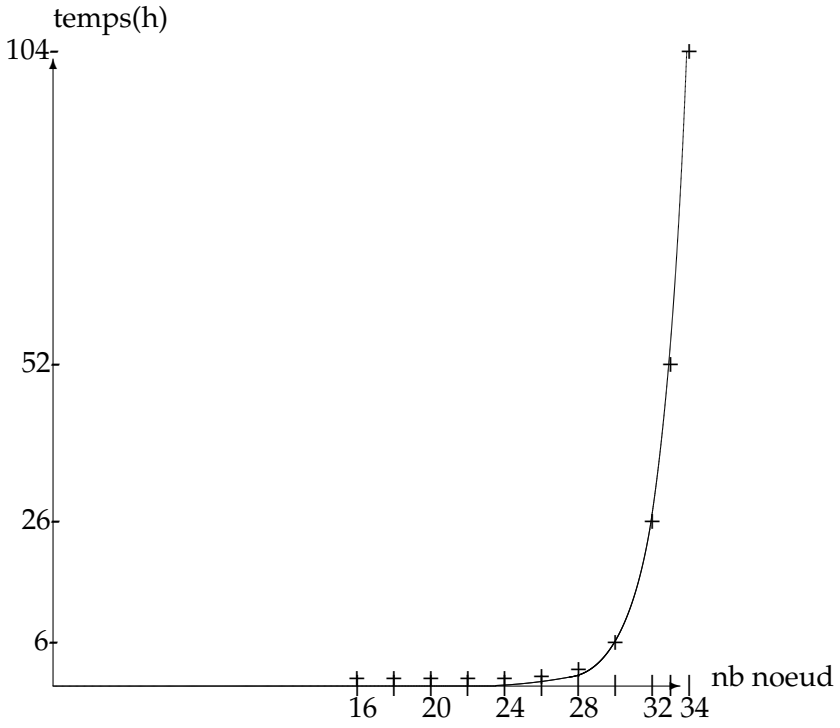
        On met à jour le front Pareto avec s

    On retourne le front Pareto

Cet algorithme est extrèmement simple a écrire, puisque basiquement il ne fait qu'évaluer toutes les solutions possibles. Pour au final, ne garder que les meilleures.

3.2 Complexité

Cependant, en contre partie de cette facilité d'écriture et la garantie d'avoir l'optimum Pareto, cet algorithme a une complexité exponentielle de $\mathcal{O}(2^{nb\_noeud})$. Ceci implique qu'un algorithme exhaustif ne passera pas à l'echelle.
Voici l'évolution du temps d'execution en fonction du nombre de noeud :

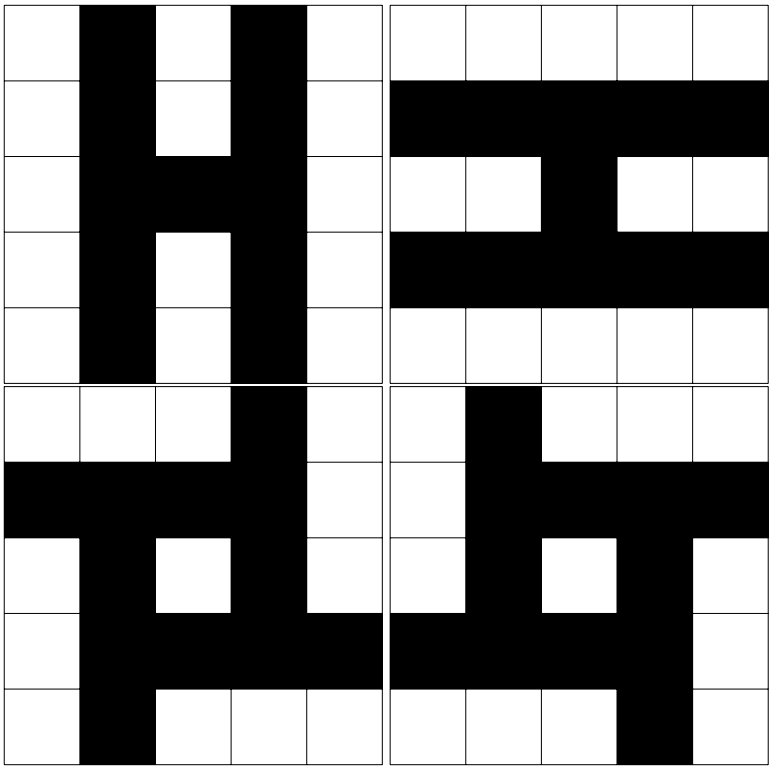


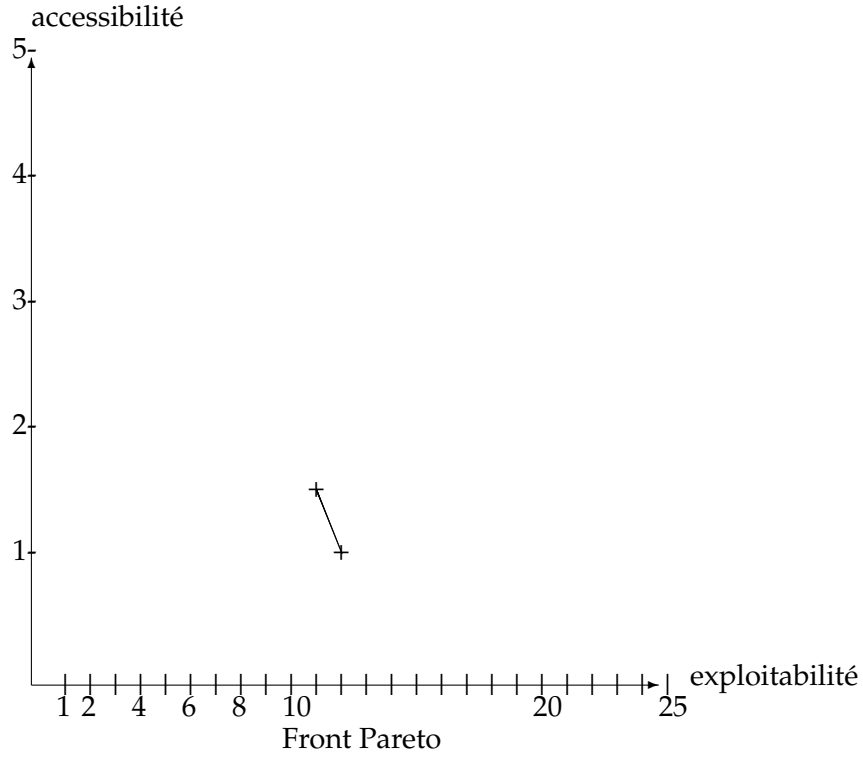
3.3 Améliorations possibles

Étant donné que l'on ne peut pas savoir à l'avance si une solution fera partie ou non de l'optimum de Pareto on est incapable d'avoir un critère objectif pour s'intéresser ou non à une solution précise. C'est ainsi que l'on peut, au lieu d'évaluer chaque solution possible, n'en évaluer qu'une partie. Si cette partie est de taille fixe et non une proportion de l'ensemble des solutions alors on aura un temps d'execution qui ne sera plus lié à la grandeur de l'espace de recherche mais seulement au temps de valuation d'une solution. La complexité d'un tel algorithme est donc la même que celle de l'évaluation d'une solution soit $\mathcal{O}(nb\_noeud^3)$. Même si l'apparition d'un cube n'est pas souhaitée elle reste souhaitable par rapport à une exponentielle.

Que l'implementation d'un tel algorithme soit déterministe en évaluant une solution toutes les N solutions ou non déterministe en évaluant ou non une solution avec une probabilité de 1/N, il n'y a aucune garantie quant à la qualité des solutions générées. Cette méthode reste donc assez peu satifaisante.

3.4 Résultats sur un graphe grille 5x5




accessibilité
5
4
3
2
1
exploitabilité
1 2 4 6 8 10 20 25
Front Pareto

Chapitre 4

Évaluation des solutions

L'évaluation des objectifs d'une solution est un énorme enjeu. En particulier c'est l'objectif de l'accessiblité qui pose problème. En effet pour pouvoir le calculer nous avons besoin de tout les plus courts chemins entre chaque couple de noeuds du graphe. Cette opération étant coûteuse nous avons essayé de mettre en place deux algorithmes.

4.1 Calcul des plus courts chemins

4.1.1 Floyd-Warshall

Floyd-Warshall est un algorithme de programmation dynamique de complexité $\mathcal{O}(nb\_noeud^3)$. Son fonctionnement repose sur trois boucles imbriquées qui parcourent les noeuds du graphe. Les deux boucles internes passent par tous les couples de noeuds possibles pour chaque itération de la boucle externe. La boucle externe réprésente un noeud intermédiaire. C'est à dire, pour chaque noeud on vérifie que chaque chemin est plus court que si l'on passait par ce noeud. Si ça n'est pas le cas on change la valeur du chemin.

Pour évaluer l'accessiblité d'une solution on a besoin de tous les plus courts chemins entre tous les noeuds indépendamment du fait qu'ils soient exploitables ou non. Pour cette partie aucun choix nous est permis, on fait tout simplement tourner Floyd-Warshall sur le graphe.
Ensuite il nous faut les plus courts chemins en passant seulement par les noeuds définis par la solution comme des routes. Pour utiliser Floyd-Warshall tout en le limitant dans ses mouvements pour obtenir le résultat voulu on applique des masques sur la matrice des distances sur les lignes de chaque noeud qui ne sont pas des routes. Cette méthode requiert cependant que la matrice et donc que le graphe soient symétriques.

4.1.2 Floyd-Warshall appliqué aux routes

Pour obtenir le bon résultat, la méthode précédente devait supprimer des informations des données de base. Ceci veut tout simplement dire que tout n'est pas utile à l'obtention du résultat. Aucune méthode ne pourra utiliser l'ensemble des données puisque le résultat attendu ne s'en sert tout simplement pas. Cependant au lieu d'effectuer des calculs pour ces données non utiles on peut, à la place, les supprimer totalement dès le début. Ainsi au lieu d'executer un Floyd-Warshall sur l'ensemble des noeuds on l'effectue sur l'ensemble des routes. Puis on calcule l'accessibilité de la solution à partir de ce résultat intermédiaire.

Chapitre 5

Recherche locale bi-objectif pure, deterministe par minimums locaux depuis solution triviale

5.1 Algorithme

Soit c la solution triviale
Tant que c n'est pas NULL :
    Soit max intialisé à NULL
    Pour chaque voisin v de c :
        Si v domine max :
            max = v
    On met à jour le front Pareto avec max
    c = max
    Si max n'est pas dans le front Pareto :
        c = NULL
On retourne le front Pareto

Cet algorithme cherche tous les minimums locaux en partant de la solution triviale et en n'autorisant que la visite de voisins faisables sur les contraintes du problème. La visite des voisins se fait toujours dans le même ordre arbitraire, celui des données. Tout ceci implique que cet algorithme est deterministe.

5.2 Complexité

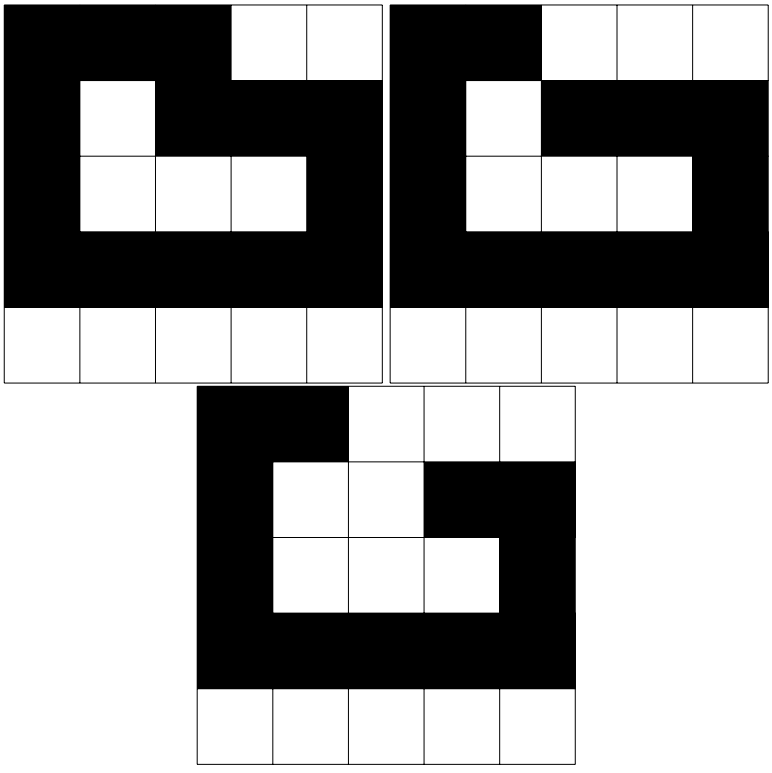
Le calcul de la complexité de cet algorithme n'est pas évident car la boucle externe "Tant que" n'est reliée explicitement à aucune donnée du problème. Cependant au vu des tests effectués le nombre d'itérations reste dans le même ordre de grandeur que le nombre de noeuds du graphe.
De plus la taille du voisinage est le nombre de noeuds, on a donc une complexité totale de $\mathcal{O}(nb\_noeud^5)$.

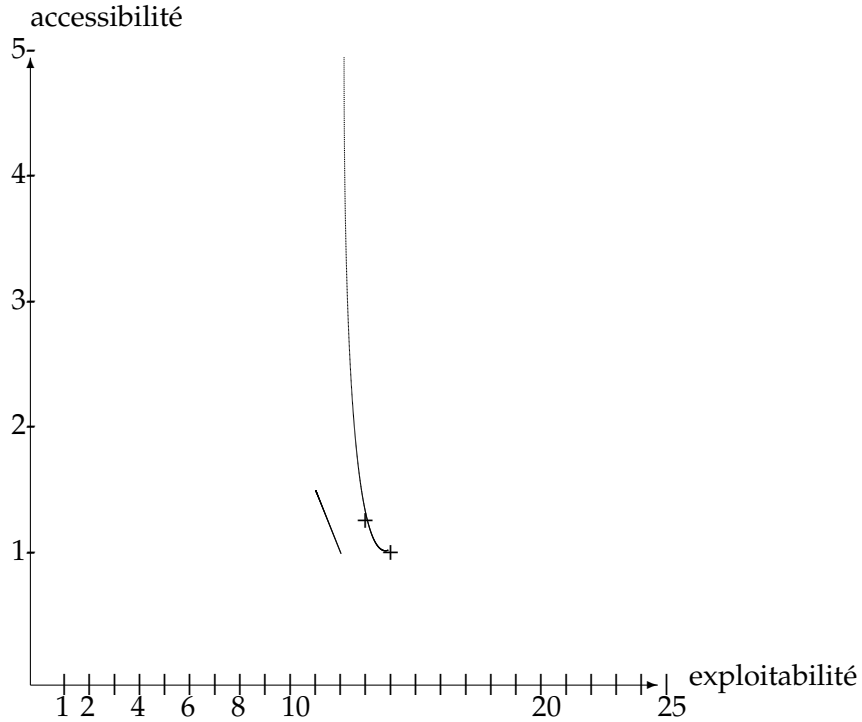
On peut comparer la complexité de cet algorithme avec l'amélioration possible de l'algorithme complet. En terme de complexité pure cet algorithme est bien pire, cependant nous sommes ici garanti d'une certaine qualité de solution, ce qui n'était pas le cas précedemment. Ceci en fait un algorithme plus intéressant sur des graphes de petite taille.

5.3 Améliorations possibles

Cette recherche locale étant de complexité polynomiale son temps d'éxecution reste raisonable quand la taille du problème grandie, de plus il est déterministe et le résultat est garanti non vide. Cependant pour arriver à de tels avantages on doit s'encombrer de beaucoup de tests sur les solutions et d'une limitation non négligeable sur l'exploration de l'espace de recherche.
C'est donc en espérant briser ces limitations que l'on a tenté d'implémenter des recherches locales plus évoluées. C'est en ceci que les prochains chapitres consisteront.

5.4 Résultats sur un graphe grille 5x5





Front Pareto avec comme référence l'algorithme complet

Chapitre 6

Recherche locale bi-objectif traduite en mono-objectif

6.1 Traduction du bi-objectif en mono

L'un des principaux problèmes de l'algorithme précédent était la méthode de parcours de l'espace de recherche, notamment le fait que l'on recherche le meilleur méliorant sans avoir d'outil qui nous permettrait de choisir entre deux solutions qui se dominent l'une l'autre sur un objectif et inversement sur le deuxième.
L'objectif qui résultera de la traduction ne sera qu'une simple combinaison linéaire de la normalisation des deux objectifs du problème. Ainsi une solution qui fait gagner un peu sur chaque objectif pourra être avantagée sur une solution qui améliore considérablement un objectif en sacrifiant trop l'autre.

6.1.1 Problème de la continuité des objectifs

Le problème que soulève cette traduction est la nécessité de la compensation d'une perte sur un objectif par un gain sur l'autre, et ce pour n'importe quel score.

L'objectif de l'exploitabilité étant discret, même si la normalisation de cet objectif ne sera pas continue, un gain sur cet objectif aura toujours un gain identique sur l'objectif normalisé et ceci est suffisant.

Cependant l'objectif de l'accessibilité s'exprime quant à lui sur un intervalle $[1\,;+\infty] \subset \mathbb{R}$ pour le normaliser la fonction inverse est intéressante, mais pas pleinement satisfaisante car il suffit que l'accessibilité passe de 1 à 2 pour passer de 1 à 0.5 après normalisation. Cet écart étant trop grand pour la perte qu'il représente on décide de normaliser la racine carré de l'objectif plutôt que l'objectif lui même. Ceci permettra, à la recherche locale, de détériorer l'accessibilité au profit de l'exploitabilité plus facilement.

6.1.2 Contradiction des objectifs

Dans chaque problème d'optimisation multi-objectif, les différents objectifs se contredisent, ou au minimum se limitent dans en terme de grandeur les uns les autres. Ici la contradiction vient du fait que pour obtenir la plus faible accessibilité possible il nous faut un grand nombre de routes alors que l'on veut aussi minimiser ce chiffre. De plus le nombre de route est un objectif discret alors que l'accessibilité est continue. Ceci couplé au fait que l'on veut qu'une perte en exploitabilité puisse être totalement compensée par un gain en accessibilité, peu importe laquelle

dans une certaine mesure. Notre besoin revient à vouloir rendre discret un intervalle [1;+∞] dans un intervalle borné et où tous les pas entre deux valeurs sont les mêmes. Cet objectif n'étant pas réalisable, on en déduit que les objectifs d'exploitabilité et d'accessibilité sont contradictoires.

6.1.3 Évaluation des solutions non faisables

Pour finir de nous libérer des limitations de l'algorithme du chapitre 4 il nous faut pouvoir supprimer tous les tests, sur les solutions, nécessaires pour un algorithme qui veut garantir un ensemble de solutions non vide.
Afin d'arriver à ce résultat on a besoin d'être capable d'évaluer les solutions non faisables. Le mono-objectif ne sera alors pas seulement une combinaison linéaire des objectifs d'exploitabilité et d'accessibilité mais aussi de la faisabilité de la solution.
Ainsi nous serons capable de générer une solution aléatoire sans se soucier de la faisabilité de celle-ci et donc en temps linéaire, de plus on pourra explorer des solutions jusqu'à présent impossibles à atteindre, car elles ne respectent pas les contraintes, ce qui rend le paysage de recherche moins rugueux et donc plus simple à parcourir pour une méthode de recherche locale.

6.2 Semi-deterministe : solutions de départs aléatoires

6.2.1 Algorithme

Soit c une solution aléatoire
Soit it le nombre d'iterations alloué
Pour i de 1 à it :
    Pour chaque voisin v de c :
        Si v domine c :
            On met à jour le front Pareto avec c
            c = v
            break;
    Si c n'a pas changé :
        c = une solution aléatoire
On retourne le front Pareto

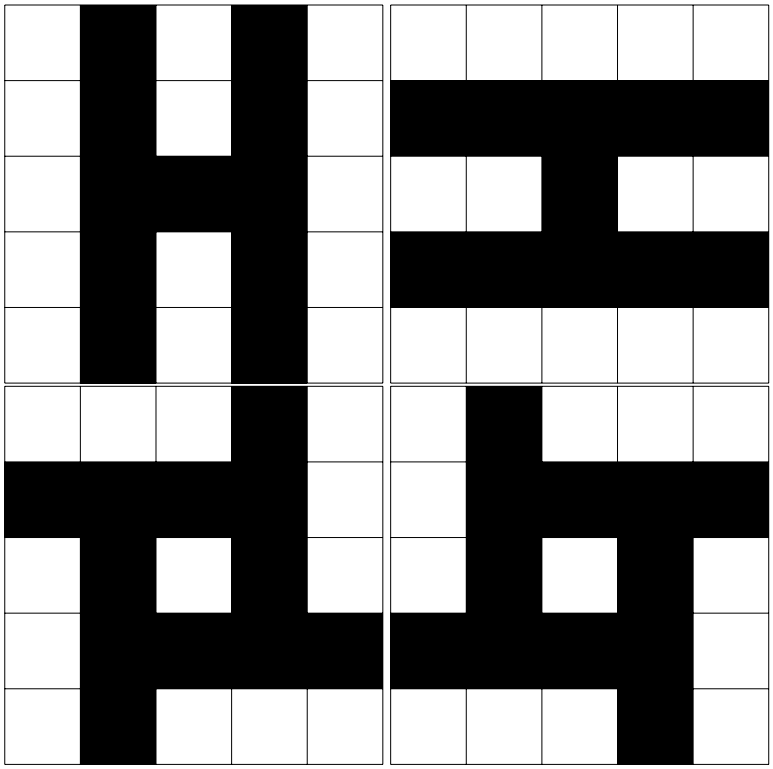
Cet algorithme est très simple dans son fonctionnement, on choisit une solution aléatoire sans aucun prérequis puis on effectue une simple descente par premier méliorant sur celle-ci. Une fois que l'on atteint un minimum local on répète l'opération jusqu'à tant que le nombre d'itérations alloué à l'éxecution tombe à zéro.

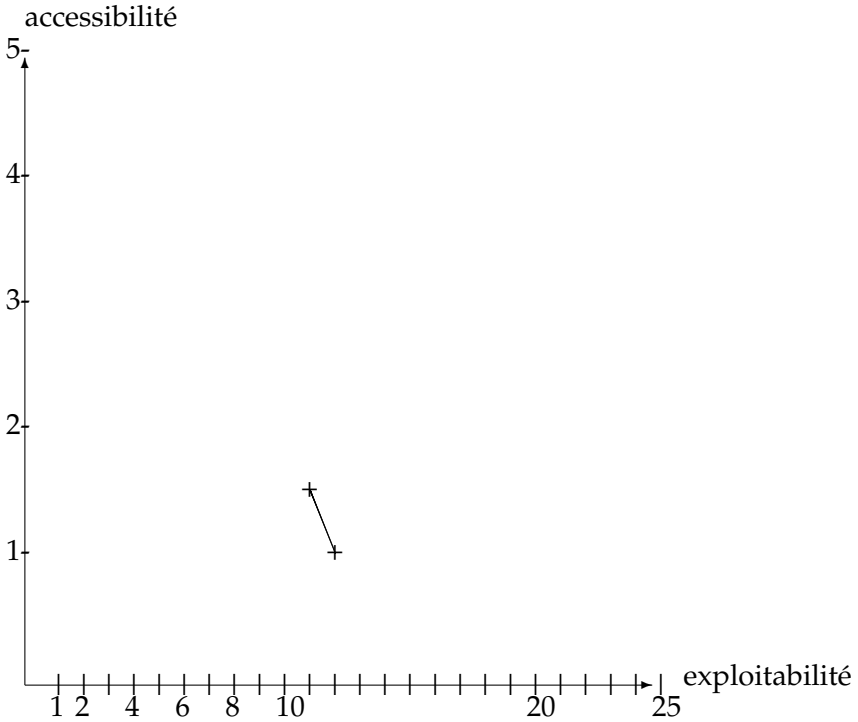
6.2.2 Complexité

Dans la théorie de la complexité on ne s'intéresse qu'aux pires cas, pour cet algorithme le pire cas a une complexité de $\mathcal{O}(it \times nb\_noeud^4)$ ce qui reste une amélioration par rapport à l'algorithme précédent surtout si l'on considère it comme une constante. Cependant dans la pratique nous serons beaucoup plus proche d'une

complexité $\mathcal{O}(it \times nb\_noeud^3)$ de par le fait que l'on s'intéresse au premier méliorant et non le meilleur. Ainsi en terme de complexité cet algorithme peut même rivaliser avec la possible amélioration de l'algorithme exhaustif tout en ayant les mêmes garanties de qualité des solutions de l'algorithme du chapitre 3.

6.2.3 Résultats sur un graphe grille 5x5





Front Pareto (Optimum Pareto ici)

6.3 Semi-deterministe : exploration aléatoire du paysage de recherche

6.3.1 Algorithme

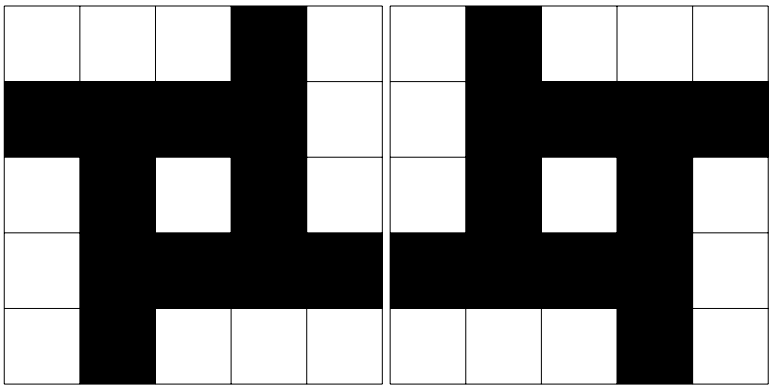
Soit c la solution triviale
Soit it le nombre d'iterations alloué
Pour i de 1 à it :
    On mélange les voisins de c
    Pour chaque voisin v de c :
        Si v domine c :
            On met à jour le front Pareto avec c
            c = v
            break ;
    Si c n'a pas changé :
        c = la solution triviale
On retourne le front Pareto

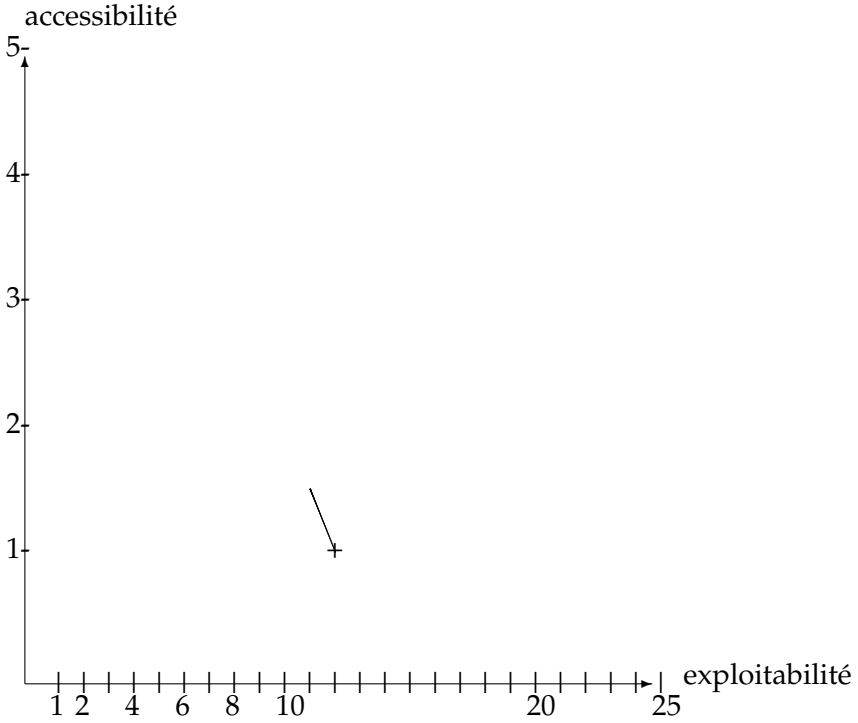
Cet algorithme est, dans son fonctionnement, opposé au précédent. En effet au lieu de choisir une solution aléatoire pour ensuite parcourir son voisinage de façon arbitraire, on garde toujours la solution triviale en départ de recherche puis on parcourt son voisinage dans un ordre aléatoire. Une fois que l'on atteint un minimum local on répète simplement l'opération jusqu'à tant que le nombre d'itérations alloué à l'éxecution tombe à zéro.
L'intéret de cette méthode par rapport à l'algorithme 4.2 est qu'ici nous sommes garanti d'un ensemble de solutions non vide faisable d'une certaine qualité. En effet la méthode 4.2 pouvait tomber en boucle sur des minimums locaux non faisables (peu probable mais possible tout de même).

6.3.2 Complexité

Tout comme l'algorithme précédent la complexité dans le pire cas est $\mathcal{O}(it \times nb\_noeud^4)$ cependant, pour les mêmes raisons, nous serons plus proche de $\mathcal{O}(it \times nb\_noeud^3)$ en pratique.
Toutefois même si la complexité des deux algorithmes est rigoureusement la même, après implémentation, nous avons un temps d'éxecution amélioré de 33% entre l'algorithme 4.2 et 4.3. Cette différence est expliquée par le fait que dans l'algorithme précédent on devait évaluer à chaque itération des voisins extrèmements proches de ce que l'on avait déjà visités. Et donc le 4.3 se rapproche encore plus d'une complexité de $\mathcal{O}(it \times nb\_noeud^3)$ par rapport au 4.2.

6.3.3 Résultats sur un graphe grille 5x5





Front Pareto avec comme référence l'algorithme complet

6.4 Non-deterministe : combinaison des deux recherches semi-deterministes

6.4.1 Algorithme

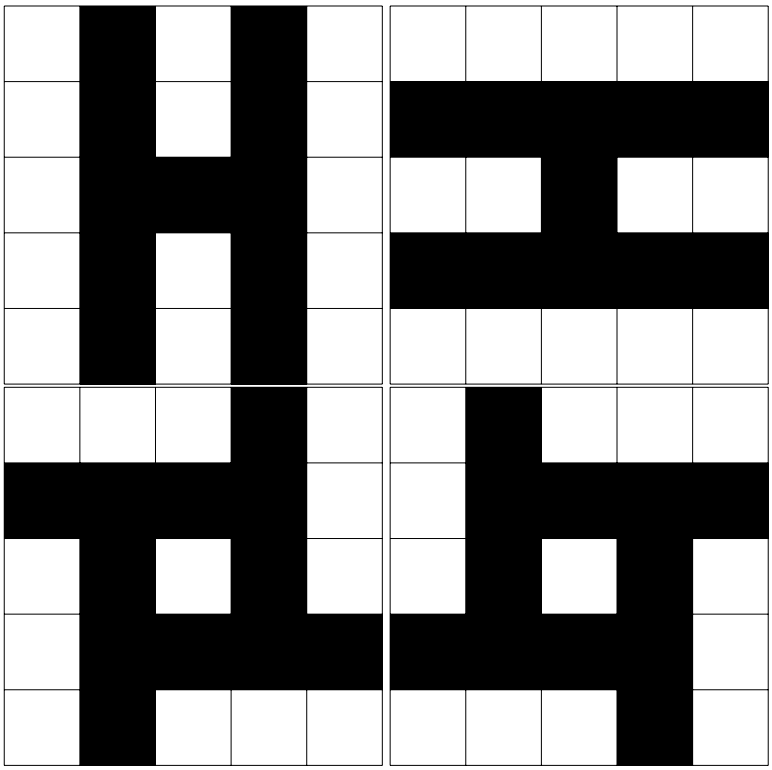
Soit c une solution aléatoire
Soit it le nombre d'iterations alloué
Pour i de 1 à it :
    On mélange les voisins de c
    Pour chaque voisin v de c :
        Si v domine c :
            On met à jour le front Pareto avec c
            c = v
            break ;
    Si c n'a pas changé :
        c = une solution aléatoire
On retourne le front Pareto

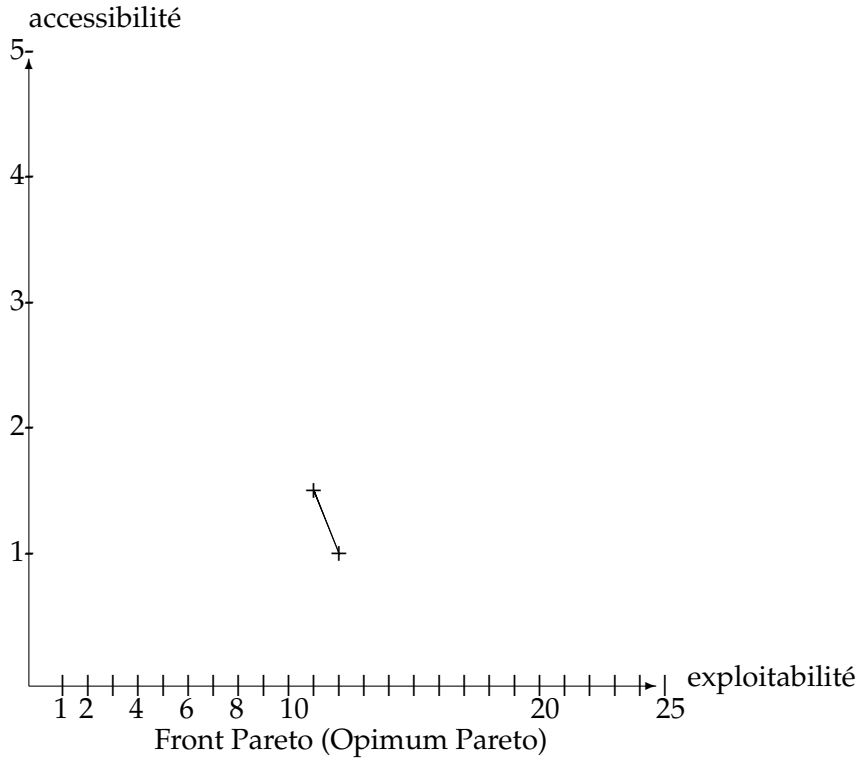
L'idée derrière cet algorithme est de pouvoir bénéficier des avantages des deux algorithmes précédents en une seule méthode. On choisit donc une solution aléatoire, le parcours de l'espace de recherche se fait dans un ordre aléatoire par premier méliorant. Une fois dans un minimum local on répète l'opération.
On sacrifie donc la garantie d'un ensemble de solutions faisables via la solution triviale pour une vitesse d'éxection accrue.

6.4.2 Complexité

La complexité reste donc $\mathcal{O}(it \times nb\_noeud^4)$ dans le pire cas et $\mathcal{O}(it \times nb\_noeud^3)$ en pratique, tout en se rapprochant plus de $\mathcal{O}(it \times nb\_noeud^3)$ que l'algorithme 4.2. Ainsi nous gardons l'amélioration de 33% de vitesse d'exécution.

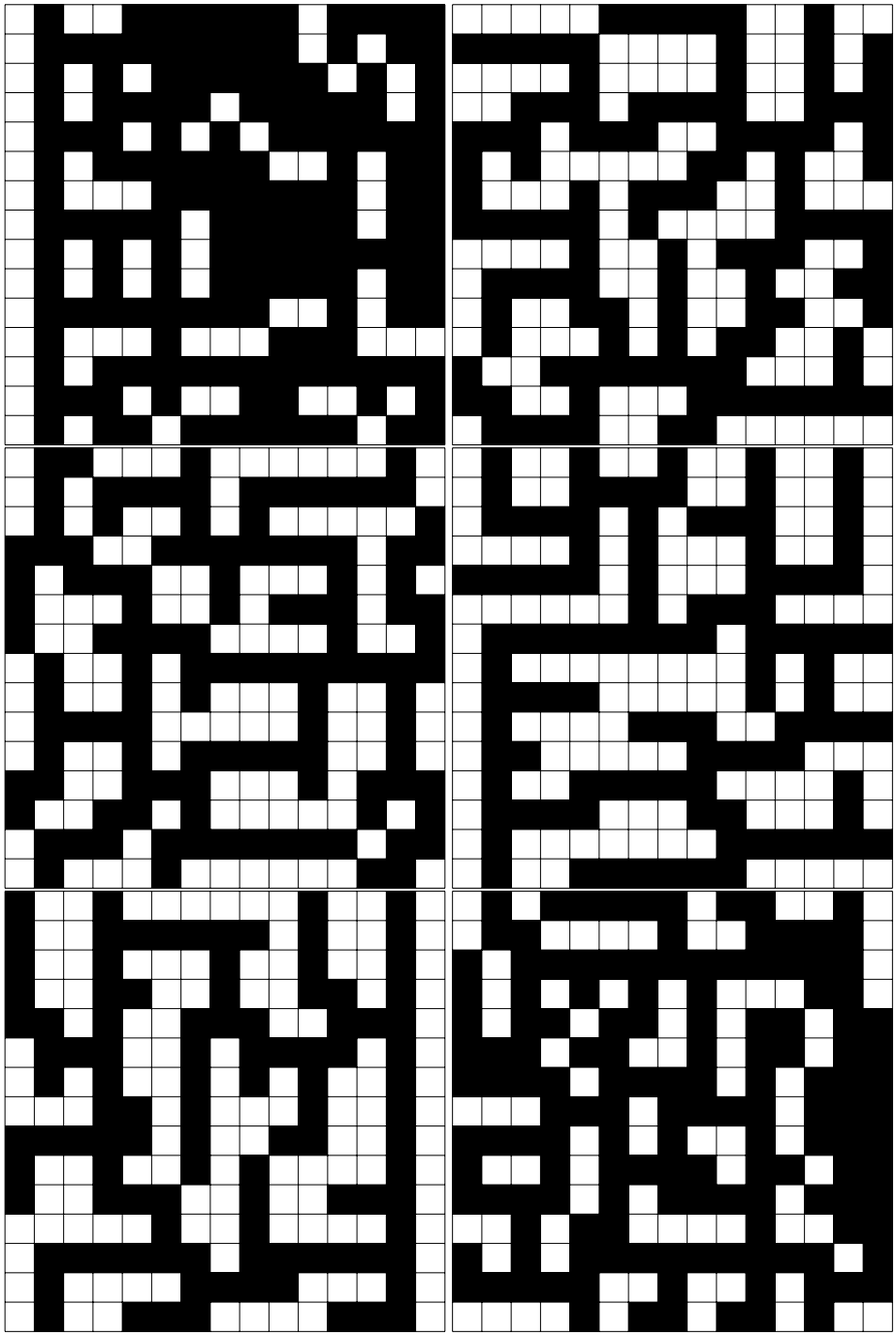
6.4.3 Résultats sur un graphe grille 5x5



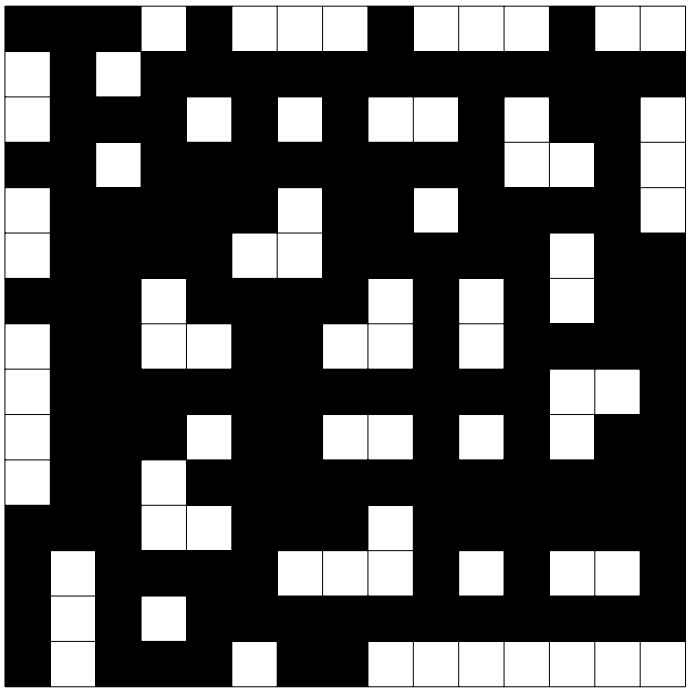


6.5 Résultats sur un graphe grille 15x15

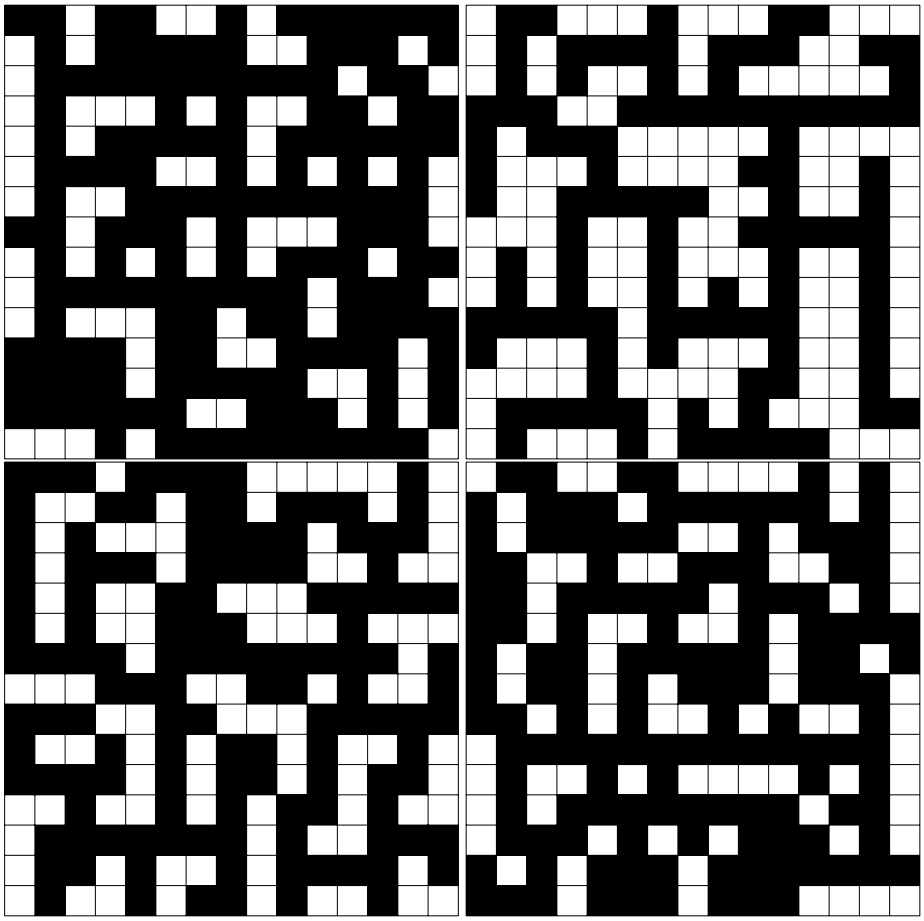
6.5.1 Algorithme 4.2

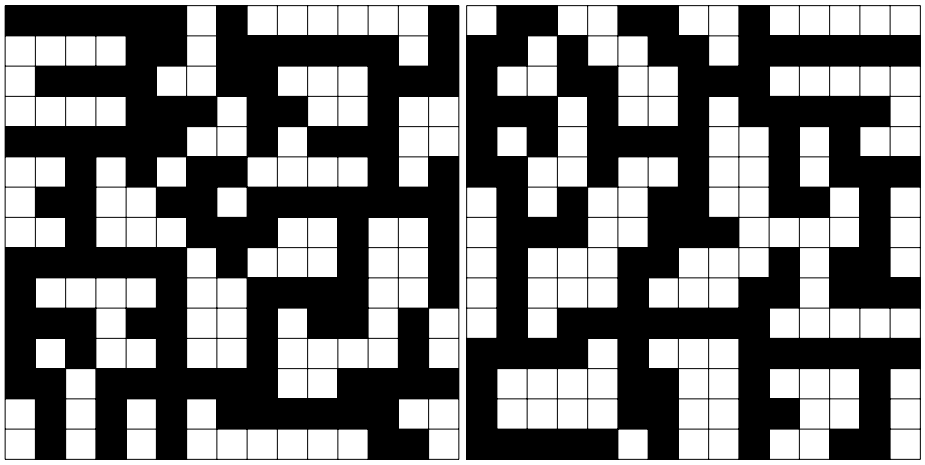


6.5.2 Algorithme 4.3

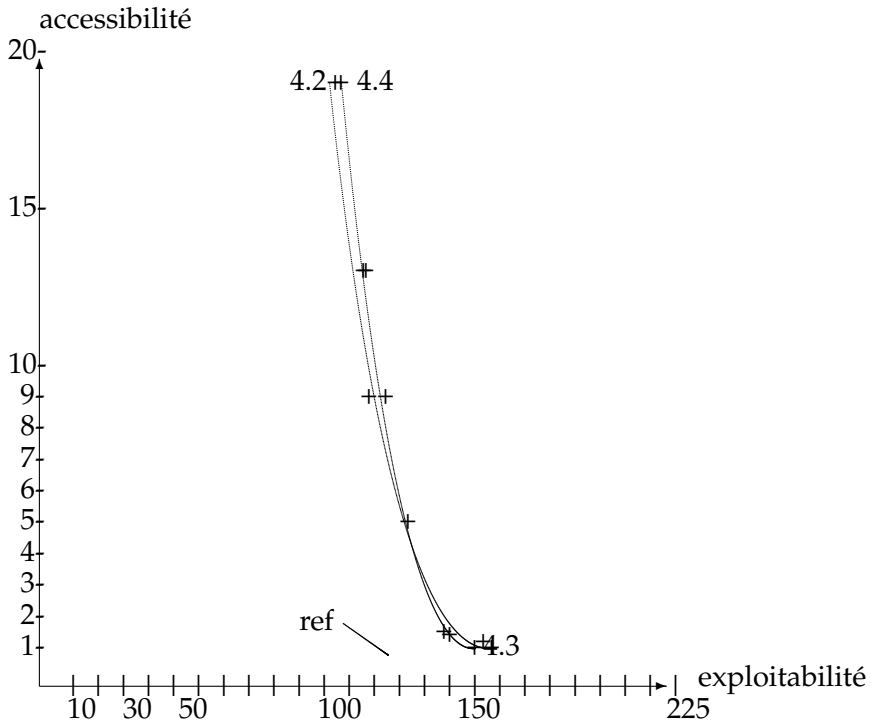


6.5.3 Algorithme 4.4





6.5.4 Front Pareto



Front Pareto avec comme référence des solutions trouvées à la main

Conclusion : Contrairement aux résultats sur le graphe grille 5x5 qui étaient tous similaires, ce graphe neuf fois plus grand nous aide à choisir quelle direction prendre pour la résolution du problème. Tout d'abord on peut voir que l'algorithme 4.3 ne nous donne pas un front Pareto mais une seule et unique solution qui, par ailleurs, minimise un seul des deux objectifs. Ceci veut tout simplement dire que cet algorithme est, dans sa recherche, un algorithme mono-objectif. C'est à rejeter.
Ensuite les fronts Pareto des méthodes 4.2 et 4.4 sont extrèmements proches voire égaux en certains points. On ne pourra alors pas les départager en étudiant la qualité des solutions générées. Cependant il est à noter que l'algorithme 4.2 a un temps d'éxecution 50% supérieur au 4.4. C'est en ça que l'on privilégiera la méthode 4.4 pour bénéficier d'un passage à l'échelle plus intéressant.

6.6 Améliorations possibles

Rugosité et déterminisme

Au vu des résultats des différents tests effectués avec l'algorithme 4.4 il semble évident que le paysage est bien trop rugueux pour pouvoir accéder à l'optimum Pareto avec une recherche locale simple. En rendant l'algorithme moins déterministe nous nous sommes permis d'être confrontés à des choix impossibles à atteindre auparavant.
La topologie du paysage de recherche est intrinsèquement liée à la fonction d'évaluation des solutions, cependant les scalaires des objectifs ne sont pas seulement inter-limitatifs, ce qui accorderait un réglage subtil des facteurs, mais hautement contradictoires.
De part ces conclusions il ne nous reste comme choix seulement des méthodes de recherches plus évoluées et moins guidées.

Chapitre 7

Recherche locale par recuit simulé

7.1 Algorithme

Soit c une solution aléatoire
Soit it le nombre d'iterations alloué
Soit τ_0 la température initiale
Soit α le facteur de τ (1^-)
Soit τ = τ_0
Tant que i est différent de it (i=0) :

On mélange les voisins de c
Pour chaque voisin v de c :

Si v domine c ou $e^{\frac{v-c}{\tau}} > random(0,1)$:

On met à jour le front Pareto avec c
c = v
break ;

$\tau = \tau \times \alpha$

Si c n'a pas changé :

c = une solution aléatoire
$\tau = \tau_0$
i = i + 1

On retourne le front Pareto

Cet algorithme est presque le même que le 4.4. La seule différence est qu'ici un recuit simulé est mis en place. Ce type de méthode de résolution est inspiré de la thermodynamique. τ est la température du milieu, ici de l'espace de recherche, et notre parcours un atome dans ce milieu. De sorte que, quand la température est élevée les mouvements sont chaotiques et peu guidés, et inversement quand la température est faible. Au fil des itérations la température diminue plus ou moins lentement selon le facteur choisi.
Ainsi on a une recherche locale qui explore l'espace des solutions plus ou moins aléatoirement mais toujours attirée vers les solutions optimales. Grâce à cette méthode la rugosité du problème devient un enjeux moins important même s'il reste le principal.

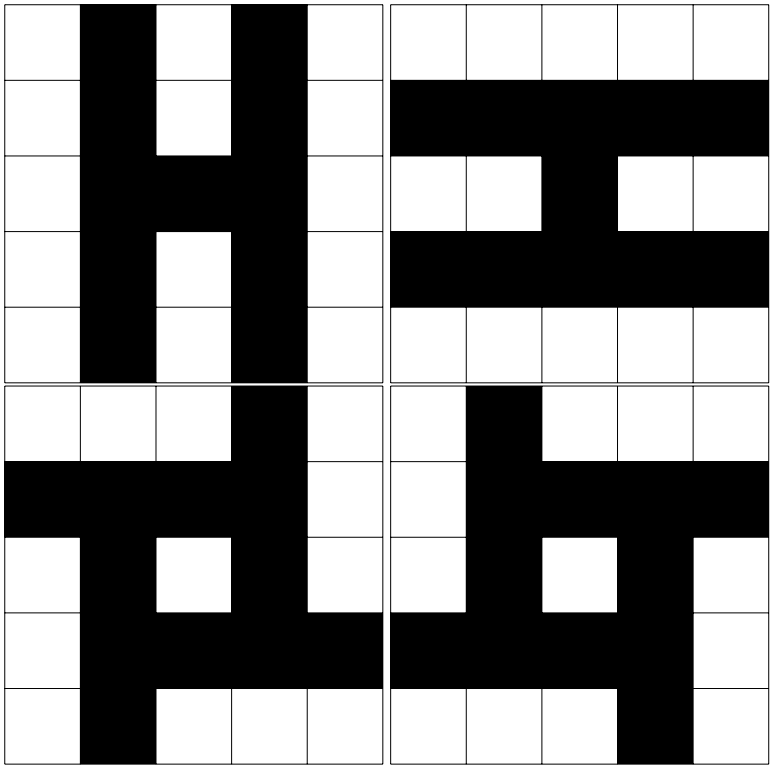
7.2 Complexité

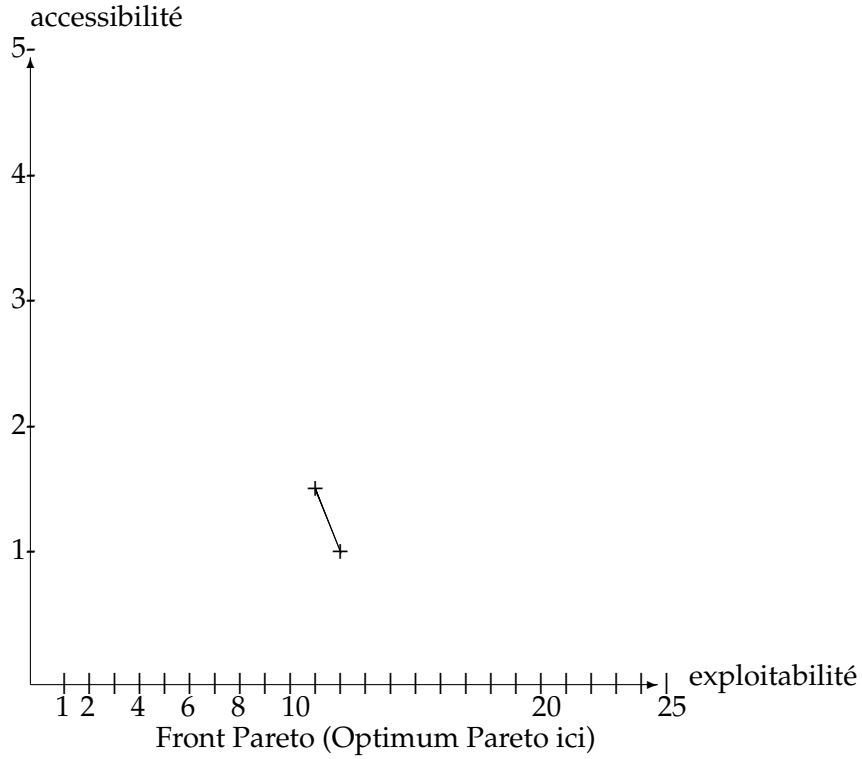
La base de la complexité de cet algorithme restera la même que les précédents, puisque la méthode est similaire, seulement plus indulgente avec des solutions moins bonnes mais tout de même proches en terme de qualité. C'est ce niveau d'indulgence qui est difficile à évaluer concrètement. En pratique le nombre d'itération semble suivre la fonction $1/(1-x)$ pour $x \in [0,1[$. Finalement la complexité de cet algorithme semble être de $\mathcal{O}(it \times nb\_noeud^4 \times \frac{1}{1-\tau})$ dans le pire cas en se rapprochant fortement de $\mathcal{O}(it \times nb\_noeud^3 \times \frac{1}{1-\tau})$ en pratique.

7.3 Améliorations possibles

Des améliorations possibles d'un recuit simulé ne sont pas évidentes à trouver. Cependant tout au long du T.E.R. et donc de ce rapport, une direction nette se fait remmarquer. C'est dans les méthodes avec une procédure d'execution faisant intervenir de l'aléatoire que l'on a les meilleurs résultats. De plus, plus l'aléatoire est influent sur l'algorithme plus les solutions générées sont de meilleure qualité. Sans aller jusqu'à un algorithme de Las Vegas, il est certain que cette direction est à suivre.

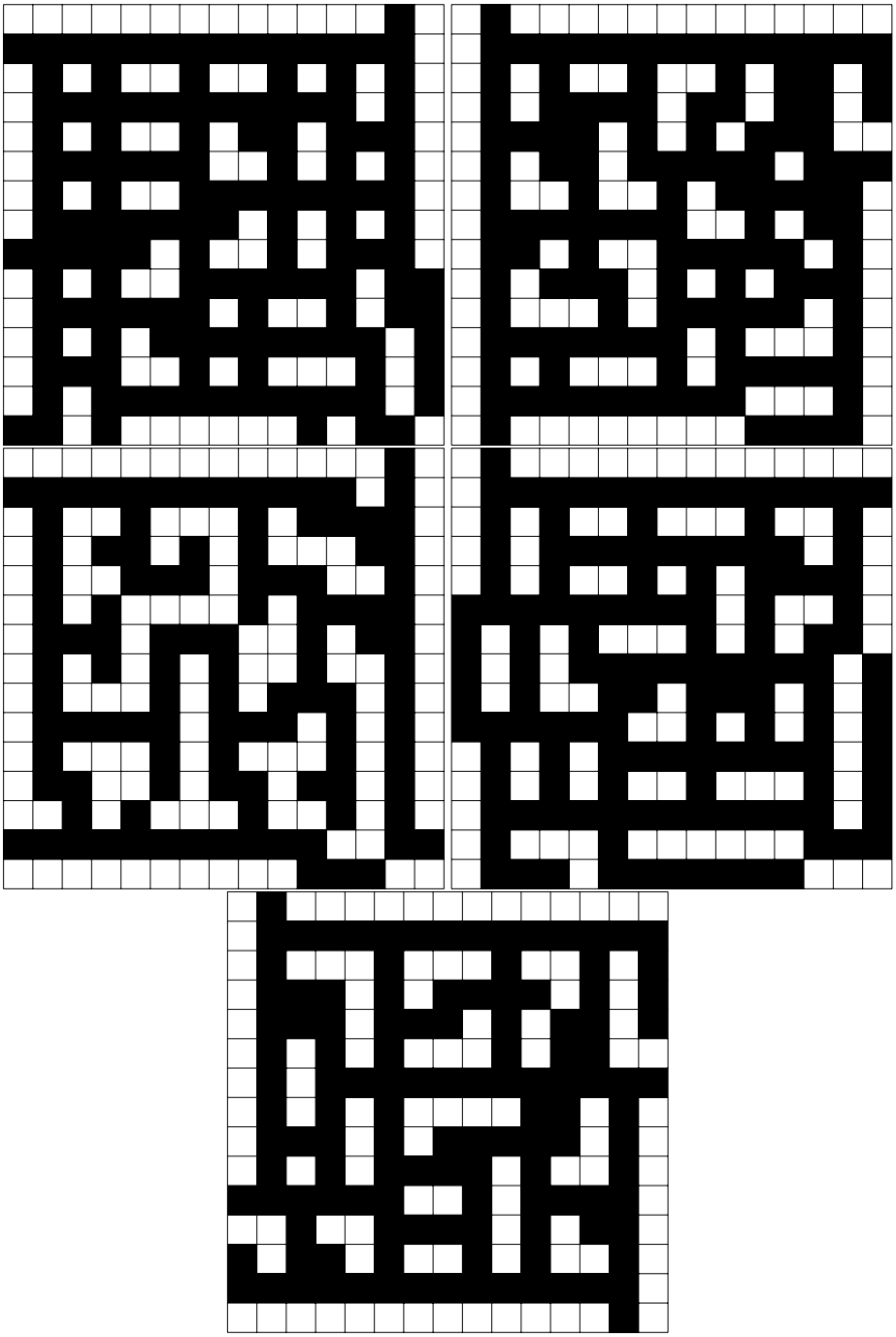
7.4 Résultats sur un graphe grille 5x5

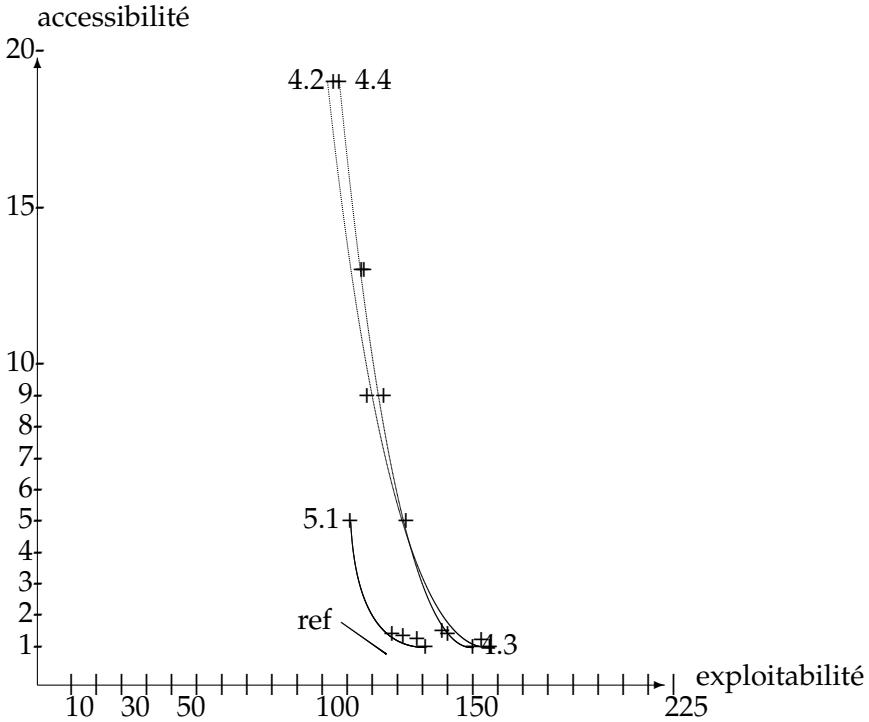




Front Pareto (Optimum Pareto ici)

7.5 Résultats sur un graphe grille 15x15





Front Pareto avec comme référence des solutions trouvées à la main

On voit sur cet exemple que même si le front Pareto de l'algorithme par recuit simulé est encore loin de l'optimum de Pareto, il reste une très bonne amélioration des méthodes précédentes. Ce résultat nous encourage à aller encore plus loin vers des méthodes plus stochastiques car, de toute évidence, c'est dans cette direction que les gains sont les plus grands.

Chapitre 8

Conclusion

Pour conclure sur l'état de ce T.E.R. et des connaissances que l'on a pû retirer de celui-ci, on peut commencer par le fait que sous son apparente simplicité le problème d'exploitabilité et d'accessibilité n'est pas un problème simple à résoudre dans le cas général.

De plus nous avons les réponses aux questions que l'on se posait en introduction. À savoir, les méthodes exhaustives sont beaucoup trop coûteuse en temps pour résoudre des problèmes de petite taille.

C'est donc pourquoi nous avons implémenté des algorithmes de recherche locale. D'abord déterministes en voulant privilégier la reproduction des résultats.
Puis non déterministes, non seulement pour plus se rapprocher d'une recherche locale classique, et surtout pour bénéficier d'un temps d'execution plus faible.
Enfin, en constatant des résultats nettement meilleurs atteints par ces méthodes, on a décidé de mettre en place un recuit simulé qui a montré sa supériorité sur toutes les autres approches testées.

Ce travail d'implémentation et d'interprétation des résultats a clairement dégagé une direction à suivre quant aux méthodes à implémenter plus tard. Les recherches locales fonctionnent, et la qualité des solutions croît plus le rôle de l'aléatoire est important.

Outre les méthodes de résolution, des changements sur la définition même du problème (tout en gardant l'essentiel) pourraient être à l'avant garde d'un changement radical de complexité algorithmique. Ou pourraient diminuer les contradictions entre objectifs. Et ce tout en gardant le même optimum Pareto.