Rapport de stage

Optimisation multicritère de l'extension de réseaux

Auteur :
Millian Poquet

Responsables :
Victor Essayan
Hervé Noel

Septembre 2014

Remerciements

Je tiens tout d'abord à remercier Victor Essayan qui nous a encadré tout au long du stage.

Je remercie également Sébastien Limet et Sophie Robert pour l'aide qu'ils nous apporté dans la recherche de stage ainsi que dans l'établissement de l'état de l'art initial.

Je remercie aussi Kévin Bourgeois dont le stage était complémentaire au mien pour sa sympathie, ce qui a permis de créer un environnement de travail agréable.

Table des matières

Introduction

Afin de valider leur Master recherche en informatique spécialité Visualisation, Image et Performance (VIP), les étudiants de l'université d'Orléans doivent mettre en pratique les connaissances acquises lors de leurs années de formation grâce à un stage de six mois ayant une forte thématique recherche.

Le sujet de stage qui m'a été proposé à Géo-Hyd, membre d'Antea Group convient à ces critères puisqu'il consiste à résoudre un problème complexe d'optimisation qui ne semble pas avoir été étudié dans la littérature. Ce stage est également l'occasion pour moi de découvrir le monde de l'entreprise, qui m'était inconnu puisque mes précédents stages et travaux ont tous été effectués dans le milieu académique.

1 Sujet de stage

1.1 Définition initiale du sujet

Géo-Hyd est une société d'études et de services à double compétence Informatique et Environnement. Depuis 2013, Géo-Hyd fait partie de Antea Group France. Dans ce cadre, le projet Optipipe mis au point par Géo-Hyd a remporté le concours d'innovation du groupe international Antea. Ce projet s'intéresse à un problème complexe d'optimisation. Nous souhaitons étendre les réseaux de surfaces existants (pipelines de pétrole et de gaz, routes, électricité, eau...) d'un champ pétrolifère dense et complexe afin d'y exploiter de nouveaux puits pétroliers et gaziers.

L'extension de ces réseaux est soumise à de nombreuses contraintes : respect de distances de sécurité entre les différents éléments des réseaux, minimisation du coût et du temps de construction, minimisation du temps de trajet des employés, minimisation du temps d'intervention sur puits en cas d'urgence, maximisation de la robustesse et de l'extensibilité des réseaux résultants... Ces contraintes sont parfois contradictoires, il nous faut donc trouver un compromis. Nous devons également être capables de planifier différents scénarios comme le forage initial d'un ensemble de puits (en minimisant le trajet parcouru par la foreuse) ou l'intervention d'urgence sur un puits (qui peut nécessiter des modifications des réseaux, comme couper une ligne électrique survolant une route par exemple).

Les données peuvent être de deux types : matriciel (matrice des altitudes...) et vectoriel (réseau des routes, réseau des pipelines, obstacles...). Certains algorithmes, pour résoudre certains sous-problèmes, sont plus adaptés à une structure de données qu'à une autre. Il nous faut ainsi créer un méta-algorithme qui prenne en compte la différente nature des algorithmes utilisés ainsi que le caractère hétérogène des données.

Au vu de la complexité du problème et des grands volumes de données traités, il est peu probable qu'une solution optimale soit calculable en temps raisonnable. Nous allons donc probablement concevoir et utiliser des heuristiques pour accélérer le calcul. De plus, le méta-algorithme que nous allons concevoir devra prendre en compte sa parallélisation, afin d'accélérer le calcul en exploitant les ressources du cluster disponible à Géo-Hyd.

1.2 Analyse du sujet et déroulement du stage

Le début du stage a consisté à effectuer un état de l'art autour de ce problème, afin de savoir s'il avait déjà été étudié ou non. Nous n'avons pas trouvé d'article étudiant ce problème en particulier ni de sujet proche. Cette recherche initiale a permis de voir que le problème rencontré était très complexe. En effet, même en ne prenant en compte qu'une petite partie des contraintes, le problème obtenu restait NP-difficile. Deux exemples de sous-problèmes NP-difficiles sont décrits dans les deux paragraphes suivants.

Dans un premier cas, nous nous intéressons uniquement à la création de chemins optimaux entre un ensemble de puits de pétrole et une station de traitement. Nous souhaitons que tous les puits soient connectés à la station de traitement et minimiser la longueur totale des chemins. Ceci revient à créer un arbre de longueur minimale reliant tous les points (les puits et la station de traitement), ce qui est un problème de Steiner dans un plan euclidien, problème NP-difficile.

Le deuxième cas concerne la planification du forage initial des nouveaux puits de pétrole. Nous pouvons créer le graphe $G = (V, E)$ tel que V soit l'ensemble des puits à forer et E l'ensemble des chemins reliant ces puits. Chaque chemin dispose d'un poids : sa longueur. Nous souhaitons créer un chemin passant par tous les nœuds du graphe et qui soit de longueur minimale. Ceci revient au problème de voyageur de commerce, autre problème NP-difficile.

Cet état de l'art a permis de situer le problème dans les problèmes d'aide à la décision à référence spatiale, ce qui est détaillé dans la section 1.3. Au cours des discussions avec notre maître de stage, nous nous sommes rendus compte que l'extension du réseau des pipelines était le sous-problème le plus important pour Géo-Hyd. Nous nous sommes ainsi concentrés sur ce sous-problème tout au long du stage.

1.3 Optimisation multicritère à référence spatiale

Le problème général du stage entre dans le cadre des problèmes d'aide à la décision à référence spatiale. Le premier chapitre de la thèse[1] m'a servi d'introduction à ce domaine. On y apprend notamment quelques différences caractéristiques entre les problèmes spatiaux et les problèmes aspatiaux : la prise en compte des structures spatiales existantes, les conséquences d'une décision sont d'une portée rarement locale, les décisions sont évaluées sur différents critères (quantitatifs ou qualitatifs) qui sont assez souvent conflictuels. Ce chapitre détaille également pourquoi les problèmes de décision à référence spatiale sont complexes. Cette complexité vient en partie du type de données utilisées : les différentes entités géographiques entretiennent des relations topologiques et sont caractérisées par des dépendances spatiales et temporelles.

Cette thèse introduit également l'aide multicritère à la décision. Les problèmes de décision classiques sont généralement de la forme :

$$opt\{f(x) : x \in D\}$$

Nous souhaitons optimiser la fonction critère f dans le domaine D. Ce problème est bien défini mathématiquement, ce qui facilite sa résolution. Lorsque nous souhaitons optimiser plusieurs critères définis par les fonctions critères $f_1, f_2, \ldots, f_n$ ceci se représente sous la forme :

$$opt\{f_1(x), f_2(x), \ldots, f_n(x) : x \in D\}$$

Malheureusement, ce problème est mal posé au sens mathématique. En général, il n'existe pas d'action permettant d'améliorer tous les critères simultanément. Ainsi, le concept de solution optimale n'a pas de sens dans un contexte multicritère, nous recherchons plutôt une solution de compromis.

Le problème étudié comporte une problématique de choix. En effet, nous souhaitons sélectionner un sous-ensemble de solutions S^0 satisfaisant les différents critères. Dans l'idéal, $\left|S^0\right| = 1$ mais la nature multicritère du problème nous pousse à vouloir présenter différentes *bonnes* solutions au décideur pour qu'il puisse en choisir une.

Si nous disposons d'un grand nombre de solutions, il peut être intéressant d'agréger les évaluations de chaque critère dans une seule mesure. L'article[2] propose différentes fonctions d'agrégation pour la décision.

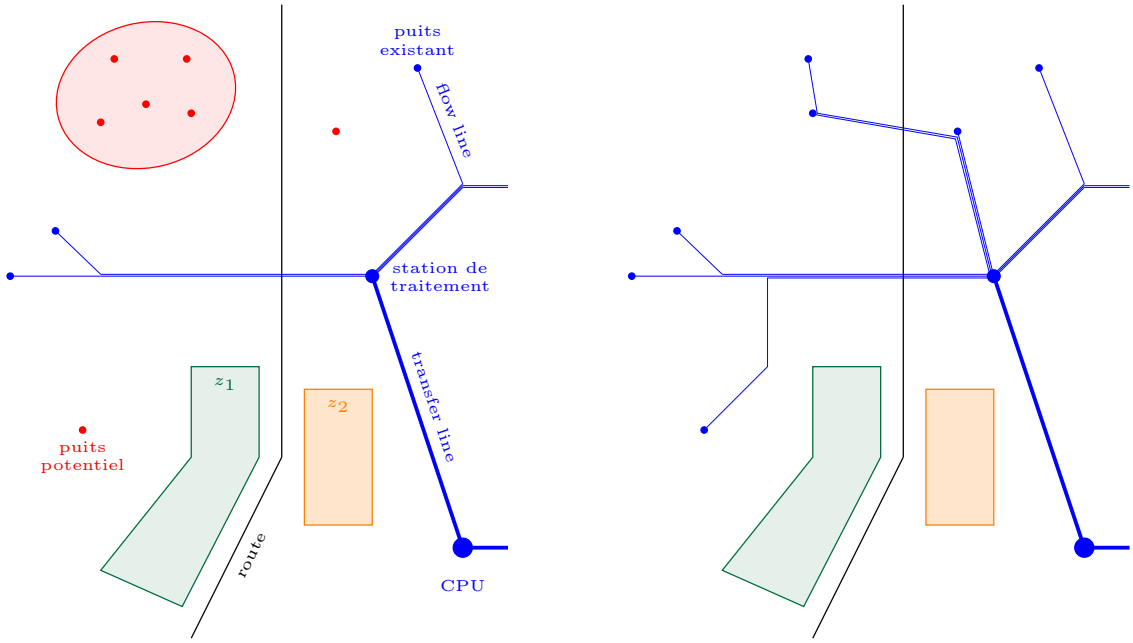
2 Extension du réseau de pipelines

Le réseau de pipelines est le principal réseau que l'on souhaite étendre. Ce réseau est un graphe orienté acyclique où les puits de pétrole et de gaz sont reliés à des stations intermédiaires de traitement (RS pour *remote station*), elles-mêmes reliées à une station centrale de traitement (CPU pour *central process unit*). Les pipelines entre les puits et les stations intermédiaires sont appelés des *flow lines* alors que ceux entre les stations et le CPU sont appelés des *transfer lines*. Les pipelines sont regroupés en corridors : ils suivent le même chemin en étant proches les uns des autres (tout en respectant la distance de sécurité pipeline/pipeline).

Grouper les chemins en corridors permet *a priori* une bonne extensibilité puisque ceci évite de saturer l'espace de travail. Cette approche peut cependant créer des problèmes de sécurité (et donc de robustesse) puisqu'elle centralise différentes ressources, ce qui augmente l'impact que peut avoir un accident ou une attaque. Le réseau existant de pipelines utilise largement des corridors, nous avons donc choisi de faire de même. Cependant, afin de limiter la perte de robustesse du réseau, nous avons choisi de limiter la taille des corridors que nous allons générer.

Les puits de pétrole et de gaz extraient leur fluide du sol et alimentent les pipelines à un débit donné. De plus, les stations de traitement ont un débit de traitement donné. Nous souhaitons ajouter des nouveaux puits de pétrole et de gaz dans ce réseau. L'emplacement d'un puits est déterminé par des contraintes géologiques, nous avons ainsi des problèmes du type "ajouter x puits dans une région donnée, délimitée par un polygone p". De plus, il peut être intéressant de savoir si de nouvelles stations de traitement doivent être rajoutées ou non, où les placer et comment les relier au CPU.

Le coût de la construction de l'extension du réseau des pipelines dépend donc des puits de pétrole et de gaz qui seront créés, des stations intermédiaires construites, de la longueur des chemins des pipelines ainsi que des croisements entre les nouveaux pipelines et les autres réseaux, notamment des routes. Créer un croisement entre un pipeline et une route est très coûteux puisque cela nécessite de creuser un chemin assez solide sous la route pour que le poids des voitures et des foreuses passant sur la route ne soit pas un problème pour les pipelines. De plus, nous pouvons remarquer que le coût d'un tel passage n'est pas beaucoup plus elevé lorsque nous voulons faire passer plusieurs pipelines plutôt qu'un seul, ce qui nous conforte sur l'utilisation de corridors.



(a) L'entrée du problème : réseau existant de pipelines (bleu), zones de coûts z_1 et z_2 et puits potentiels (rouge). Certains puits potentiels sont regroupés en sous-ensembles au sein desquels une quantité de flot doit être extraite (ellipse rouge).

(b) Une solution possible au problème. Tous les puits potentiels ne sont pas forcément construits. Afin de minimiser le coût des croisements et la saturation de l'espace de travail, les pipelines sont ici regroupés en deux corridors.

FIGURE 1 – Extension du réseau des pipelines : ajout de flow lines entre des nouveaux puits potentiels et une station de traitement existante.

2.1 Méthode générale

Cette sous-section décrit la méthode générale utilisée pour réaliser une extension du réseau de pipelines. Cette sous-section introduit également les problèmes de flot[3] qui sont utilisés dans l'extension de ce réseau.

La méthode que nous proposons permet de choisir quelles installations doivent être construites (puits de pétrole ou de gaz, stations intermédiaires) et comment les relier au réseau existant de pipelines. Cette méthode ne génère pas des chemins réels mais des couloirs au sein desquels les pipelines se situent. Cette méthode se base essentiellement sur des graphes. La méthode est illustrée en figure 2.

Les contraintes de débit des puits de pétrole/gaz, des stations de traitement et des pipelines m'ont fait penser à un problème de flot. En effet, si nous disposons d'un graphe comprenant les différentes installations ainsi que des nœuds correspondant à l'espace de travail étudié, il est possible de modéliser l'extension du réseau de pipelines via un problème de flot sous contraintes.

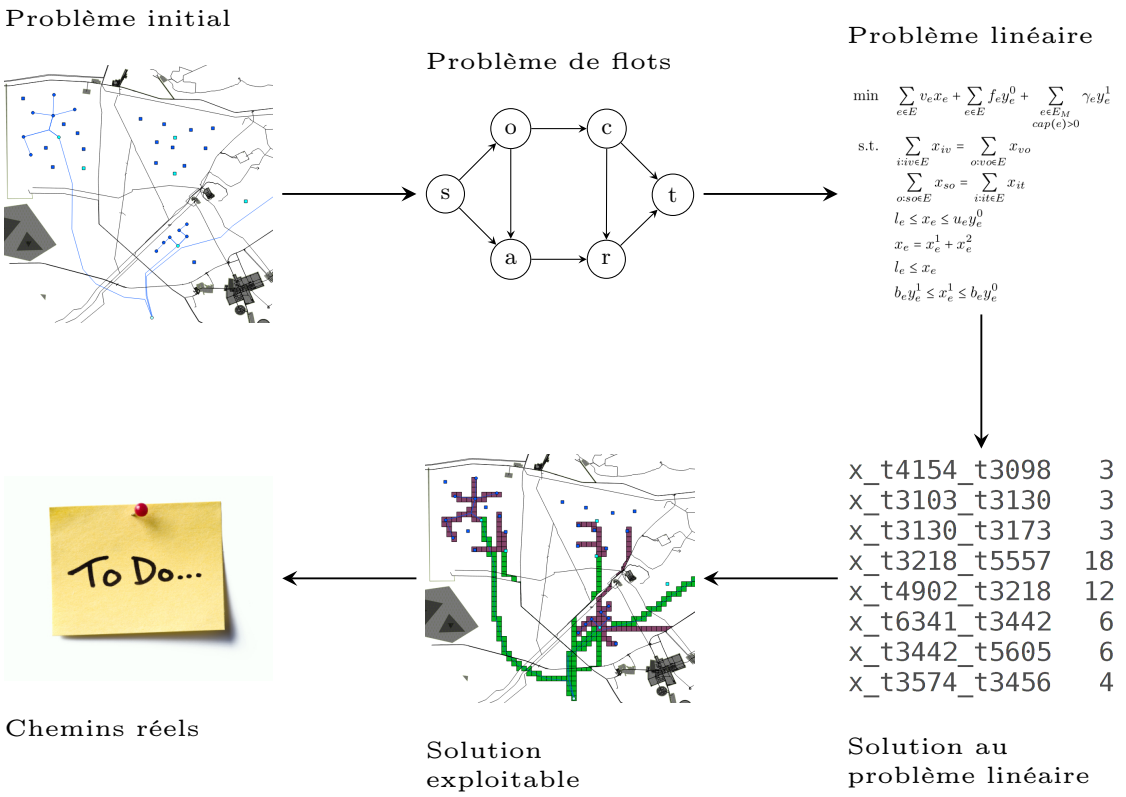


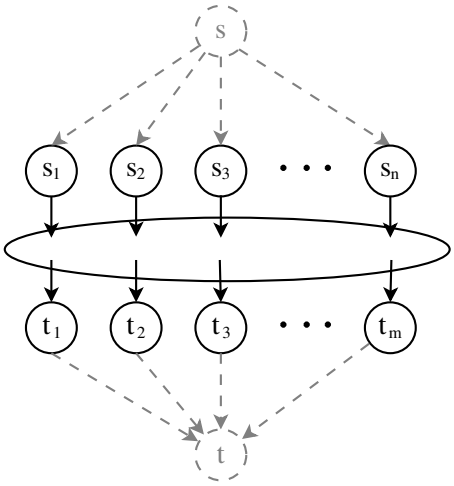
FIGURE 2 – La méthode générale que nous proposons pour effectuer une extension du réseau des pipelines.

Un graphe de flot est un graphe orienté où un flot peut passer par chaque arête. Chaque arête a une capacité : la borne maximale du flot passant par l'arête. La plupart des nœuds d'un tel graphe ont une contrainte de conservation du flot énonçant que le flot entrant d'un nœud est égal à son flot sortant. Si le flot entrant d'un nœud est supérieur à son flot sortant, on dit que ce nœud est une *source*. Sinon si le flot sortant d'un nœud est supérieur à son flot entrant, on dit que ce nœud est un *puits*[1].

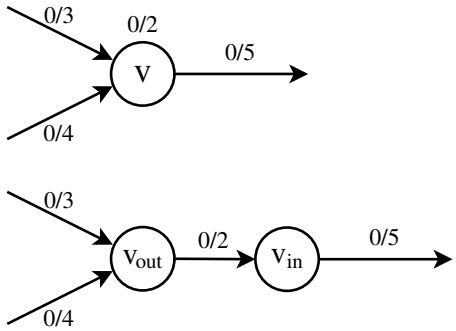
Différents problèmes existent dans les graphes de flot comme le problème de flot maximum qui consiste à utiliser le plus de flot possible dans un graphe de flot comportant un seul nœud source et un seul nœud puits. Le problème qui va nous intéresser est un problème de flot de coût minimum : un coût est associé aux arêtes en fonction du flot qui y passe et certaines arêtes sont contraintes d'avoir un flot. On cherche ensuite la solution qui minimise le coût tout en satisfaisant les contraintes.

1. **Attention** : il ne faut pas confondre un puits en terme de flots et un puits de pétrole/gaz. Puisqu'un puits de pétrole ou de gaz émet un flot, il serait plutôt considéré comme une source.

Il existe différentes astuces pour transformer un problème proche d'un problème de flot en un problème de flot connu. La première de ces astuces (cf. figure 3a) permet de n'avoir qu'un seul nœud source au lieu de plusieurs. C'est le cas dans l'extension des pipelines puisqu'il existe plusieurs puits de pétrole/gaz qui correspondent tous à des nœuds sources. On crée pour cela un nouveau nœud source nommé *supersource* et on le connecte à tous les nœuds sources précédents. La même astuce existe pour les nœuds puits : le nœud puits créé est nommé *superpuits*. Une autre astuce (cf. figure 3b) est utile si l'on souhaite borner le flot d'un nœud et non d'une arête. Le nœud borné peut être divisé en deux nœuds distincts connectés par une arête de même borne. Cette technique permet de mettre une capacité aux stations intermédiaires.



(a) Les nœuds sources $s_1, s_2, s_2, \ldots, s_n$ sont connectés à une unique supersource s. Il en va de même pour les puits $t_1, t_2, t_3, \ldots, t_m$ connectés à un unique superpuits t.



(b) Le nœud borné v est divisé en deux nœuds v_{out} et v_{in} non bornés. Une arête bornée est ajoutée entre les deux nœuds ainsi créés.

FIGURE 3 – Deux astuces pour créer un graphe de flots.

La nature des coûts que nous avons modélisé ainsi que celle des contraintes nous a conduit à utiliser l'optimisation linéaire pour résoudre le problème de flot minimum. Plus particulièrement, nous utilisons la programmation linéaire mixte : certaines variables sont continues, d'autres sont binaires.

Afin de trouver des solutions au problème linéaire nous nous servons de solveurs linéaires existants. Ces solveurs nous retournent un fichier qui associe à chaque variable non nulle sa valeur. Ce fichier solution permet de générer un résultat exploitable qui contient les installations créées ainsi que les zones spatiales dans lesquelles elles sont connectées au réseau existant.

L'étape finale de la méthode proposée est de générer des chemins réels pour chaque pipeline. Cette étape n'a pas été implémentée. Ce problème est également complexe puisqu'il faut éviter les croisements inutiles au sein des zones spatiales générées à l'étape précédente. Des contraintes concernant la largeur d'un pipeline et la distance de sécurité à respecter entre les pipelines sont également à prendre en compte. Si l'on souhaite obtenir une solution optimale à ce problème, il peut être intéressant de le modéliser comme un problème d'arbre de Steiner contraint. On peut également utiliser des méthodes heuristiques telles des recherches de chemins dans une discrétisation sous forme de graphe des zones spatiales générées, ou alors s'orienter vers des algorithmes multi-agents tels les algorithmes de colonies de fourmis ou d'abeilles, dans un espace discret ou continu.

2.2 Extension simple

Cette sous-section modélise l'extension du réseau des pipelines dans le cas où nous ne souhaitons pas ajouter de nouvelles stations de traitement. Puisqu'il n'y a qu'un nombre très faible de stations de traitement dans les données dont nous disposons, l'ajout d'une station de traitement est probablement un phénomène rare. Ainsi, les techniques décrites ici permettent d'ajouter des puits de pétrole et de gaz et de les relier aux stations de traitement existantes via des flow lines. Une modélisation plus complexe permettant de créer de nouvelles stations de traitement est décrite dans la sous-section 2.3.

2.2.1 Modélisation par un problème de flots

J'ai modélisé cette partie du sujet par un problème de flot capacitif de coût minimum sous contraintes, les coûts étant à la fois variables et fixes. Soit $G = (V, E)$ un graphe orienté dont les nœuds (mis à part s et t) sont géoréférencés. Afin de faciliter la lecture des prochaines pages, veuillez vous référer à la figure 15 (page 39) qui représente les différents nœuds de G et la manière dont ses arêtes sont crées. V est composé d'une supersource s, d'un superpuits t, d'un ensemble W contenant un nœud pour chaque puits existant, d'un ensemble S contenant un nœud pour chaque station de traitement existante, d'un ensemble W^n des puits potentiels à ajouter qui sont regroupés en sous-ensembles et d'une discrétisation de l'espace de travail sous la forme d'un graphe $G_M = (V_M, E_M)$.

Le graphe $G_M = (V_M, E_M)$ respecte certaines propriétés. Il est tout d'abord planaire : les arêtes de E_M ne se croisent pas. Ceci implique, puisque les nœuds V_M sont géoréférencés, que les chemins correspondant aux arêtes de E_M (des chemins en ligne droite entre certains nœuds de V_M) ne se croisent pas. Ces chemins représentent l'emplacement des corridors que l'on souhaite créer et il est important qu'ils ne puissent pas se croiser pour éviter d'imputer des coûts de croisements entre les corridors que l'on crée. Le graphe G_M est également fortement connexe : il existe un chemin entre toute paire de nœuds de V_M. Le graphe G_M est aussi symétrique, c'est à dire que $\forall i, \forall j,\ (\exists\, ij \in E_M) \Leftrightarrow (\exists\, ji \in E_M)$. Dans le graphe de flots, V contient un ensemble de nouveaux puits potentiels qui sont regroupés en sous-ensembles. W^n est l'ensemble des sous-ensembles $W^{n_1}, W^{n_2}, \ldots, W^{n_k}$, où chaque sous-ensemble contient des puits. Par exemple, le sous-ensemble W^{n_k} contient les puits $w_1^{n_k}, w_2^{n_k}, \ldots, w_m^{n_k}$. Un exemple de zones de puits potentiels se trouve à la figure 4. À chaque sous-ensemble W^{n_k} est associé un flot minimum $L_W^{n_k}$ qui doit être atteint par la somme des flots extraits par les puits du sous-ensemble, afin de forcer la création de certains puits mais pas tous dans chaque sous-ensemble.

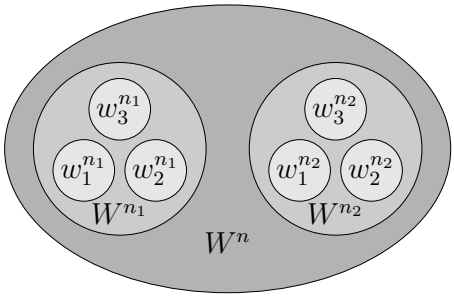


FIGURE 4 – Les puits potentiels W^n regroupés dans les sous-ensembles W^{n_1} et W^{n_2}.

Cette notion de sous-ensemble contraint est pratique puisqu'elle permet de modéliser différents cas. Si nous souhaitons ajouter avec une probabilité $p = 1$ un ensemble de puits, il suffit de les regrouper dans un sous-ensemble W^{n_k} avec $L_W^{n_k} = \sum_i emit(w_i^{n_k})$. Si nous souhaitons ajouter θ puits dans une zone délimitée par un polygone, nous pouvons discrétiser le polygone en un ensemble de puits potentiels regroupés dans le sous-ensemble W^{n_p}. Tous les puits de W^{n_p} ont le même débit d'extraction de flot f_p et $L_W^{n_p} = \theta \cdot f_p$. Enfin, si nous disposons d'un large choix de puits candidats regroupés dans le sous-ensemble W^{n_l} mais un budget limité, nous pouvons fixer un flot f_l qui doit être extrait de cet ensemble de puits via $L_W^{n_l} = f_l$.

Le coût des arêtes est modélisé de deux manières différentes. La plupart des arêtes ont un coût décrit par la fonction (1) et représenté sur la figure 5. Ce type de coût permet de payer un coût fixe de construction K_0 lorsqu'une arête est utilisée et un coût variable α en fonction de la quantité de flot utilisée sur l'arête. La fonction est définie sur le même intervalle que la quantité de flot x, c'est-à-dire dans $[0, b]$. Le coût fixe permet de représenter le coût de construction des puits et des croisements entre les pipelines et les autres réseaux, notamment les routes.

$$c(x, \alpha, K_0, b) = \begin{cases} 0 & \text{si } x = 0 \\ K_0 + \alpha x & \text{si } x \in]0, b] \end{cases} \tag{1}$$

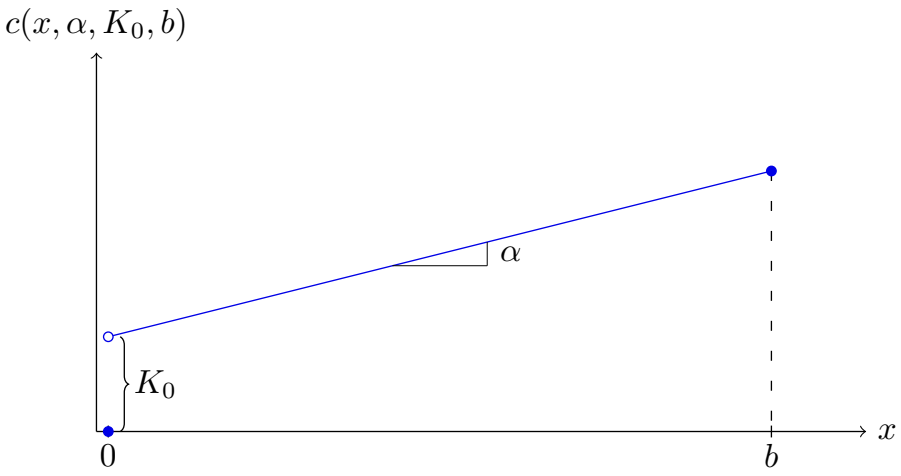


FIGURE 5 – Fonction de coût simple : utilisation d'un coût variable α et d'un coût fixe K_0.

Les arêtes de G_M où des pipelines sont déjà présents ont une fonction de coût plus avancée décrite par l'équation (2) et représentée sur la figure 6. Cette fonction permet d'appliquer un coût variable α, un coût fixe de construction K_0 ainsi qu'un coût fixe d'extension K_1 si le flot de l'arête dépasse b, le flot qui peut passer par l'arête sans nécessiter d'extension. La fonction de coût est définie sur le même intervalle que la quantité de flot x, c'est-à-dire $[0, c]$. Enfin, on peut remarquer que si $b = c$ la fonction de coût (2) se ramène à la fonction de coût (1). La présence d'un coût fixe d'extension permet d'être plus réaliste quant à l'extension des croisements entre les pipelines et les autres réseaux. Si un croisement déjà construit a été conçu assez grand pour accueillir 5 pipelines mais qu'il ne contient que 3 pipelines, on peut fixer $b = 5$ et ainsi ajouter 2 pipelines à moindre coût. De plus, la manière dont un croisement a été construit peut influencer le coût de son extension, ce qui peut se paramétrer au cas par cas via K_1.

$$c(x, \alpha, K_0, K_1, b, c) = \begin{cases} 0 & \text{si } x = 0 \\ K_0 + \alpha x & \text{si } x \in]0, b] \\ K_0 + K_1 + \alpha x & \text{si } x \in]b, c] \end{cases} \tag{2}$$

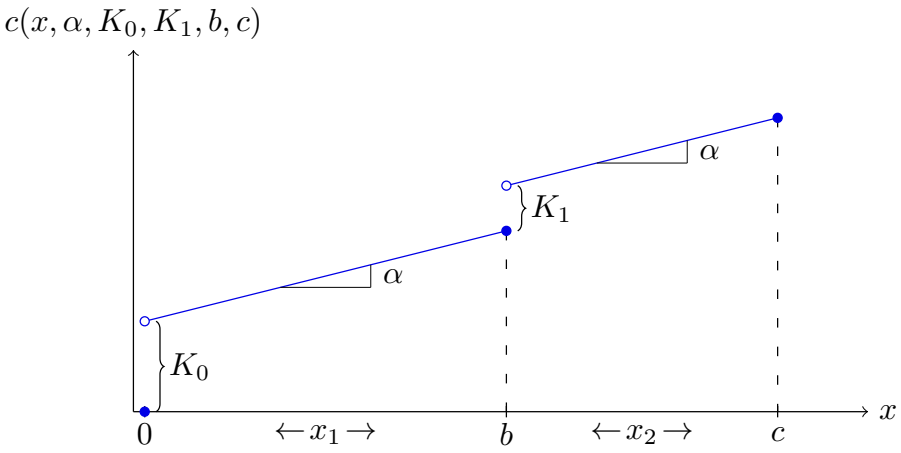


FIGURE 6 – Fonction de coût plus avancée, comprenant un coût variable α, un coût fixe initial K_0 et un coût fixe d'extension K_1.

Chaque arête e de E a différents attributs : un coût variable v_e qui est imputé proportionnellement à la quantité d'utilisation de l'arête, un coût fixe de construction f_e qui est imputé si et seulement si l'arête est utilisée, une borne inférieure l_e et une borne supérieure u_e pour la quantité de flot passant par l'arête. Les arêtes de G_M ayant une quantité de flot initiale ($l_e > 0$) ont également un coût fixe d'extension γ_e et une borne b_e correspondant au seuil devant être franchi pour que le coût γ_e soit imputé. La manière dont les arêtes sont construites est décrite dans la table 1 et illustrée en figure 15 (page 39). Cette table crée certaines variables (première colonne) qui permettent d'ajouter des arêtes dans le graphe G (deuxième colonne) en leur fixant certaines valeurs d'attributs (colonnes 3 à 8). D'autres notations existent sur les arêtes : x_e est la capacité de l'arête e, y_e^0 est une variable booléenne vraie si et seulement si l'arête est utilisée, c'est-à-dire $y_e^0 = (x_e > 0)$. Enfin, y_e^1 est une variable booléenne indiquant, pour les arêtes de G_M ayant une capacité initiale, si une extension de l'arête est nécessaire ou non, c'est-à-dire $y_e^1 = (x_e > b_e)$.

TABLE 1 – Règles de création des arêtes du graphe

		$e \in E$	v_e	f_e	γ_e	l_e	b_e	u_e
(3)	$\forall w \in W$	sw	0	0		$cap(w)$		$emit(w)$
(4)	$\forall w_i^{n_k} \in \bigcup_k W^{n_k}$	$sw_i^{n_k}$	0	$cost(w_i^{n_k})$		0		$emit(w_i^{n_k})$
(5)	$\forall w \in (W \cup \bigcup_k W^{n_k})$	wm_w	0	0		$cap(w)$		$emit(w)$
(6)	$\forall e_m \in E_M, cap(e_m) = 0$	e_m	$var(e_m)$	$cross(e_m) + \lambda$		0		$min(size_m(e_m), \chi)$
(7)	$\forall e_m \in E_M, cap(e_m) > 0$	e_m	$var(e_m)$	0	$ext(e_m)$	$cap(e_m)$	$size(e_m)$	$max(size(e_m), min(size_m(e_m), \chi))$
(8)	$\forall \sigma \in S$	$m_\sigma\sigma$	0	0		$cap(\sigma)$		$recv(\sigma)$
(9)	$\forall \sigma \in S$	σt	0	0		$cap(\sigma)$		$recv(\sigma)$

Les différents termes utilisés dans la table 1 sont explicités ici :

— $cap(w) \in \mathbb{R}^+$, $cap(\sigma) \in \mathbb{R}^+$ et $cap(e_m) \in \mathbb{R}^+$ correspondent respectivement à la capacité initiale d'un puits w, d'une station de traitement σ et d'une arête discrète e_m. Pour les arêtes de E_M, cette capacité initiale est calculée en projetant les pipelines existants sur la discrétisation spatiale.
— $emit(w) \in \mathbb{R}^{+*}$ correspond à la quantité de flot maximale émise par le puits w, c'est-à-dire la quantité de flot extraite depuis le sol par ce puits.
— $cost(\alpha) \in \mathbb{R}^+$ correspond au coût de construction de l'élément α.
— $recv(\sigma) \in \mathbb{R}^{+*}$ correspond au flot maximal traité par la station de traitement σ.
— $var(e) \in \mathbb{R}^+$ correspond au coût variable de l'arête e, qui dépend notamment de la longueur de l'arête e et de la zone de coût dans laquelle e est située.
— $cross(e) \in \mathbb{R}^+$ correspond, pour les arêtes qui croisent des structures existantes, au coût devant être imputé pour construire un croisement permettant de faire passer une quantité de flot χ sur l'arête e. Si e ne représente pas un croisement, $cross(e) = 0$.
— $\lambda \in \mathbb{R}^+$ est le coût attribué par défaut aux corridors. Ainsi, il est plus coûteux de créer de nouveaux corridors que de saturer ceux déjà existants.
— $ext(e) \in \mathbb{R}^+$ correspond au coût de l'agrandissement d'un corridor pour qu'il puisse accueillir une quantité de flot χ.
— $size(e) \in \mathbb{R}^+, size(e) \geq cap(e)$ est la taille, en terme de quantité de flot, que peut contenir l'installation existante de l'arête e.
— $size_m(e) \in \mathbb{R}^+$ est la taille maximale, en terme de quantité de flot, que peut contenir l'arête e. Cette borne peut être approximée dans certaines discrétisations spatiales.
— $\chi \in \mathbb{R}^{+*}$ est la quantité de flot maximale autorisée dans un corridor.
— m_α représente le nœud de V_M auquel doit être relié l'élément α. C'est généralement le point le plus proche mais la manière dont la discrétisation de l'espace de travail a été faite peut conduire à des contraintes topologiques plus avancées (voir section 3).

Les différentes lignes de la table 1 sont expliquées dans les paragraphes ci-dessous. La ligne (3) permet de relier la supersource à tous les puits existants. Ces puits n'ont aucun coût ($v_{sw} = f_{sw} = 0$) puisqu'ils existent déjà. Leur capacité est déjà connue et fixée grâce à $l_{sw} = u_{sw} = emit(w)$, ce qui implique $x_{sw} = emit(w)$.

La ligne (4) relie la supersource aux nouveaux puits potentiels regroupés en sous-ensembles. La borne inférieure du flot de chaque arête $l_{sw_i^{n_k}}$ est fixée à 0, ce qui ne force pas la création des puits. De plus, afin d'utiliser une quantité de flots d'au moins $L_W^{n_k}$ dans chaque sous-ensemble, les contraintes (10) doivent être satisfaites.

$$\forall W^{n_k} \in W^n, \ \sum_i x_{sw_i^{n_k}} \geq L_W^{n_k} \tag{10}$$

Les lignes (5) et (8) font le lien respectivement entre les puits et l'espace de travail d'un côté et l'espace de travail et les stations intermédiaires de l'autre. On peut remarquer l'utilisation de m_α, qui est le nœud de V_M auquel l'élément α doit être rattaché.

La ligne (6) effectue des connexions dans la discrétisation de l'espace de travail pour les arêtes qui n'ont pas de capacité initiale, c'est-à-dire les arêtes sur lesquelles des pipelines ne sont pas présents. Cette ligne est importante puisqu'elle permet de prendre en compte la distance entre les différents nœuds, la création de corridors, le coût des croisements ainsi que la limite à la taille des corridors. Le coût variable $v_{e_m} = var(e_m)$ dépend de la longueur de l'arête e_m. Le fait que ce coût soit variable implique qu'il est imputé proportionnellement à l'utilisation de l'arête. Ainsi, plus le flot est fort dans un corridor et plus son coût variable augmente, ce qui est réaliste puisque un flot plus fort implique soit d'utiliser plus de pipelines, soit d'utiliser des pipelines plus larges. Le coût fixe de construction f_{e_m} prend en compte le coût de construction d'un croisement $cross(e_m)$ entre un corridor placé sur cette arête et les installations existantes. La présence d'un coût fixe λ même en l'absence de croisement permet aux chemins de se regrouper : en effet, il est plus rentable de créer moins de chemins mieux *remplis* en terme de flots pour éviter de payer trop de coûts fixes.

La prise en compte des corridors existants sur la discrétisation spatiale est effectuée à la ligne (7), qui se base sur la ligne (6). La présence des corridors initiaux est forcée dans le problème via $l_{e_m} = cap(e_m)$ et on limite leur capacité maximale $u_{e_m} = max(size(e_m), min(size_m(e_m), \chi))$. On permet ainsi d'agrandir les corridors non saturés en interdisant l'agrandissement de ceux étant déjà saturés. De plus, on respecte les contraintes de la discrétisation de l'espace de travail en limitant la quantité de flot présente dans l'arête grâce à $size_m(e_m)$. La construction initiale de l'arête e permet d'accueillir la quantité de flot $size(e)$ à moindre coût. Si on souhaite étendre cette quantité jusqu'à χ, il faut alors imputer le coût fixe d'extension $ext(e)$. Ces arêtes sont les seules dont le coût est calculé par la fonction (2).

Veuillez noter que tous les coûts des lignes (6) et (7) peuvent être modifiés par des zones de coût (cf. section 3.3). Les valeurs de $var(e)$, $cross(e), \lambda$ et $ext(e)$ sont ainsi sensibles aux zones de coût dans lesquelles les arêtes sont situées.

Enfin, la ligne (9) permet de relier toutes les stations de traitement existantes vers le superpuits.

2.2.2 Optimisation mathématique

Une fois ce graphe construit, nous pouvons essayer d'appliquer des méthodes pour résoudre le problème de flots sous contraintes. Dans la littérature, les méthodes d'optimisation mathématique en utilisant des programmes linéaires mixtes (nombres réels et binaires) semblent être la voie privilégiée[4][5]. Je propose ainsi le programme suivant :

$$\min \quad \sum_{e \in E} v_e x_e + \sum_{e \in E} f_e y_e^0 + \sum_{\substack{e \in E_M \\ cap(e)>0}} \gamma_e y_e^1 \tag{11}$$

$$\text{s.t.} \quad \sum_{i:iv \in E} x_{iv} = \sum_{o:vo \in E} x_{vo} \qquad \forall v \in V \setminus \{s,t\} \tag{12}$$

$$\sum_{o:so \in E} x_{so} = \sum_{i:it \in E} x_{it} \tag{13}$$

$$l_e \leq x_e \leq u_e y_e^0 \qquad \forall e \in E, \neg(e \in E_M \wedge cap(e) > 0) \tag{14}$$

$$x_e = x_e^1 + x_e^2 \qquad \forall e \in E_M, cap(e) > 0 \tag{15}$$

$$l_e \leq x_e \qquad \forall e \in E_M, cap(e) > 0 \tag{16}$$

$$b_e y_e^1 \leq x_e^1 \leq b_e y_e^0 \qquad \forall e \in E_M, cap(e) > 0 \tag{17}$$

$$0 \leq x_e^2 \leq (u_e - b_e) y_e^1 \qquad \forall e \in E_M, cap(e) > 0 \tag{18}$$

$$y_e^0 \in \{0,1\} \qquad \forall e \in E \tag{19}$$

$$y_e^1 \in \{0,1\} \qquad \forall e \in E_M, cap(e) > 0 \tag{20}$$

$$\sum_{w_i^{n_k} \in W^{n_k}} x_{sw_i^{n_k}} \geq L_W^{n_k} \qquad \forall W^{n_k} \in W^n \tag{21}$$

$$x_{ij} + x_{ji} \leq min(u_{ij}, u_{ji}) \qquad \forall ij \in E_M \tag{22}$$

(11) minimise le coût total, qui est la somme des coûts variables, des coûts fixes de construction et des coûts fixes d'extension.

(13) est la contrainte de conservation du flot entre la supersource et le superpuits : il faut que le flot sortant de la supersource soit égal au flot entrant dans le superpuits.

(12) représente les contraintes de conservation du flot pour tous les nœuds différents de la supersource et du superpuits. Pour chaque nœud, le flot entrant doit être égal au flot sortant.

(14) permet de borner la capacité du flot pour chaque arête ne correspondant pas à un corridor existant. La contrainte $x_e \leq u_e y_e^0$ force la variable binaire y_e^0 à prendre la valeur 1 si $x_e > 0$ (car $u_e > 0$).

Les contraintes (15), (16), (17) et (18) sont la formulation linéaire correspondant à la fonction de coût (2), qui a à la fois un coût fixe de construction et un coût fixe d'extension. La fonction de coût (2) est une fonction affine par morceaux (cf. figure 6), l'idée est donc d'utiliser deux variables x_e^1 et x_e^2 qui représentent la quantité de flot utilisée sur les deux *morceaux* de la fonction. x_e^1 représente la quantité de flot utilisée dans $[0, b_e]$ et x_e^1 la quantité de flot utilisée dans $]b_e, u_e]$. La contrainte (15) permet de créer ces variables telle que leur somme soit égale à x_e. La contrainte (16) impose un flot minimal. La partie droite de la contrainte (17) $x_e^1 \leq b_e y_e^0$ force $y_e^0 = 1$ si $x_e^1 > 0$, c'est-à-dire si l'arête est utilisée. La contrainte (18) borne x_e^2 entre 0 et $(u_e - b_e)$ et force $y_e^1 = 1$ si $x_e^2 > 0$, c'est-à-dire quand une extension doit être réalisée. Enfin, la partie gauche de la contrainte (17) $b_e y_e^1 \leq x_e^1$ force x_e^1 à être saturée ($x_e^1 = b_e$) dès que $y_e^1 = 1$, c'est-à-dire lorsqu'une extension est réalisée. Ceci permet de respecter la contrainte de découpe de x_e en deux intervalles : pour que $x_e^2 > 0$ il faut que $x_e^1 = b_e$.

(19) indique que les variables y_e^0 sont binaires. De manière similaire, (20) indique que les variables y_e^1 sont binaires.

La contrainte (21) permet de spécifier pour chaque ensemble de puits potentiels W^{n_k} le flot minimal $L_W^{n_k}$ qui doit être atteint par la somme du flot des puits de l'ensemble.

Enfin, (22) limite la taille des corridors quel que soit le sens dans lequel les pipelines y passent.

Dans la solution de ce problème linéaire, la valeur de la fonction objectif à minimiser nous importe peu. Nous voulons par contre connaître la nouvelle valeur du flot x_e passant par chaque arête. Pour les arêtes construites via la ligne (4), si $x_{sw_i^{n_k}} > 0$ alors le puits $w_i^{n_k}$ doit être construit et doit extraire la quantité de flot $x_{sw_i^{n_k}}$. Pour les arêtes construites via les lignes (6) et (7), si $x_{e_m} > 0$ alors un corridor de flot x_{e_m}, de direction le chemin que représente e_m et de même sens que e_m est présent dans la solution.

Cette modélisation ne permet pas directement de construire des pipelines mais permet de savoir dans quels corridors lesdits pipelines sont regroupés. Si nous ne conservons que les arêtes e respectant $x_e > 0$ dans la solution optimale de ce problème linéaire, nous obtenons un graphe orienté acyclique R. Ce graphe peut être utilisé pour créer des chemins réels pour les pipelines puis pour affiner ces chemins afin de limiter l'impact de la discrétisation de l'espace de travail.

2.3 Extension avec ajout de stations de traitement

La modélisation précédente est suffisante lorsque nous souhaitons ajouter un petit nombre de puits de pétrole et de gaz. Cependant, si nous souhaitons faire une extension de taille importante du réseau, il peut être intéressant de savoir si créer de nouvelles stations de traitement est souhaitable ou non. Cette sous-section décrit un enrichissement de la modélisation précédente qui permet de créer des stations de traitement et de les relier au CPU via des transfer lines.

2.3.1 Graphe de flots

Cette modélisation utilise également un graphe orienté de flots $G = (V, E)$. Ce graphe est représenté sur la figure 16 (page 40). Ce graphe contient, comme dans la modélisation précédente, une supersource s, un superpuits t, des puits existants W, des stations existantes S et des puits potentiels W^n qui sont regroupés en sous-ensembles $W^{n_1}, W^{n_2}, \ldots, W^{n_k}$ où chaque sous-ensemble W^{n_k} contient les puits $w_1^{n_k}, w_2^{n_k}, \ldots, w_m^{n_k}$ et est contraint par $L_W^{n_k}$, le flot minimum devant être extrait par la somme des flots extraits par les puits de W^{n_k}.

À ces ensembles s'ajoutent les stations de traitement potentielles S^n regroupées en sous-ensembles $S^{n_1}, S^{n_2}, \ldots, S^{n_k}$, où chaque sous-ensemble S^{n_k} contient les stations $s_1^{n_k}, s_2^{n_k}, \ldots, s_m^{n_k}$ et est contraint par $L_S^{n_k}$, le flot minimum devant être reçu par la somme des flots entrants dans les stations de traitement de S^{n_k}. Les stations potentielles ont été modélisées d'une manière similaire aux puits potentiels car cela permet de créer plusieurs cas comme forcer l'existence de certaines stations, en construire un nombre défini dans une zone donnée ou de faire un choix parmi plusieurs candidats (cf. section 2.2.1).

Nous avons ici aussi besoin d'une discrétisation de l'espace de travail sous la forme d'un graphe planaire, orienté, symétrique et fortement connexe $G_M = (V_M, E_M)$. Ce graphe est dupliqué afin de séparer la création de flow lines et de transfer lines. Cette duplication est représentée par des couches tridimensionnelles distinctes, afin de faciliter la compréhension du processus. La première couche $\mathcal{L}_f$ représente les flow lines et la seconde couche $\mathcal{L}_t$ les transfer lines. On obtient ainsi le graphe $G_M^f = (V_M^f, E_M^f)$ sur la couche $\mathcal{L}_f$ et le graphe $G_M^t = (V_M^t, E_M^t)$ sur la couche $\mathcal{L}_t$, tels que $G_M = G_M^f \cup G_M^t$.

TABLE 2 – Règles de création des arêtes du graphe

		$e \in E$	v_e	f_e	γ_e	l_e	b_e	u_e
(23)	$\forall w \in W$	sw	0	0		$cap(w)$		$emit(w)$
(24)	$\forall w_i^{n_k} \in \bigcup_k W^{n_k}$	$sw_i^{n_k}$	0	$cost(w_i^{n_k})$		0		$emit(w_i^{n_k})$
(25)	$\forall w \in (W \cup \bigcup_k W^{n_k})$	wm_w^f	0	0		$cap(w)$		$emit(w)$
(26)	$\forall e_m^f \in E_M^f, cap(e_m^f) = 0$	e_m^f	$var(e_m^f)$	$cross(e_m^f) + \lambda$		0		$min(size_m(e_m^f, \chi^f)$
(27)	$\forall e_m^f \in E_M^f, cap(e_m^f) > 0$	e_m^f	$var(e_m^f)$	0	$ext(e_m^f)$	$cap(e_m^f)$	$size(e_m^f)$	$max(size(e_m^f), min(size_m(e_m^f), \chi^f))$
(28)	$\forall \sigma \in S$	$m_\sigma^f \sigma$	0	0		$cap(\sigma)$		$recv(\sigma)$
(29)	$\forall \sigma \in S$	σm_σ^t	0	0		$cap(\sigma)$		$recv(\sigma)$
(30)	$\forall \sigma_i^{n_k} \in \bigcup_k S^{n_k}$	$m^f_{\sigma_i^{n_k}} \sigma_i^{n_k}$	0	$cost(\sigma_i^{n_k})$		0		$recv(\sigma_i^{n_k})$
(31)	$\forall \sigma_i^{n_k} \in \bigcup_k S^{n_k}$	$\sigma_i^{n_k} m^t_{\sigma_i^{n_k}}$	0	0		0		$recv(\sigma_i^{n_k})$
(32)	$\forall e_m^t \in E_M^t, cap(e_m^t) = 0$	e_m^t	$var(e_m^t)$	$cross(e_m^t) + \lambda$		0		$min(size_m(e_m^t, \chi^t)$
(33)	$\forall e_m^t \in E_M^t, cap(e_m^t) > 0$	e_m^t	$var(e_m^t)$	0	$ext(e_m^t)$	$cap(e_m^t)$	$size(e_m^t)$	$max(size(e_m^t), min(size_m(e_m^t), \chi^t))$
(34)	$\exists\ cpu$	$m_{cpu}^t t$	0	0		$\sum_w cap(w)$		$\sum_w emit(w)$

La table 2 permet de construire les arêtes du graphe de flot de cette extension et est illustrée en figure 16 (page 40). Elle se lit de la même manière que la table 1 (cf. section 2.2.1). Les termes utilisés diffèrent cependant légèrement à cause de la séparation de la discrétisation de l'espace de travail en deux couches. Généralement, pour un objet o, o^f représente o dans la couche $\mathcal{L}_f$ des flow lines et o^t le même objet dans la couche $\mathcal{L}_t$. Ainsi, $cap(e_m^f)$ représente la capacité initiale de l'arête e_m^f dans la couche des flow lines, cette capacité est donc obtenue en projetant uniquement les flow lines sur la discrétisation de l'espace de travail. En projetant uniquement les transfer lines, on obtient $cap(e_m^t)$. De manière similaire, $cross(e_m^t)$ représente le coût de croisement entre une transfer line située sur l'arête e_m^t et les réseaux existants. Il en va de même pour $ext(e_m^t)$.χ^f est le flot maximum pouvant être dans un corridor de flow lines alors que χ^t est le flot maximum possible dans un corridor de transfer lines. Il est probable que $\chi^t \gg \chi^f$. Une brève explication des lignes de la table 2 est donnée ci-dessous.

(23) relie la supersource aux puits existants. (24) relie la supersource aux nouveaux puits potentiels. (25) relie tous les puits à la couche des flow lines $\mathcal{L}_f$.

(26) et (27) connectent les arêtes de G_M^f dans la couche $\mathcal{L}_f$ grâce respectivement à la fonction de coût (1) et à la fonction de coût (2).

La jonction entre la couche $\mathcal{L}_f$ et la couche $\mathcal{L}_t$ se fait en passant par les stations intermédiaires. (28) et (29) permettent de passer par les stations existantes alors que (30) et (31) font passer le flot par les nouvelles stations potentielles.

(32) et (33) connectent les arêtes de G_M^t dans la couche $\mathcal{L}_t$ grâce respectivement à la fonction de coût (1) et à la fonction de coût (2).

Enfin, (34) permet de relier l'emplacement du CPU dans la couche $\mathcal{L}_t$ au superpuits. Pour des raisons de lisibilité, les bornes du flot de cette arête ont été simplifiées. Ces bornes sont décrites de manière plus formelle dans les équations (35) et (36).

$$l_{m_{cpu}^t t} = \sum_{w \in (W \cup \bigcup_k W^{n_k})} cap(w) \tag{35}$$

$$u_{m_{cpu}^t t} = \sum_{w \in (W \cup \bigcup_k W^{n_k})} emit(w) \tag{36}$$

2.3.2 Optimisation mathématique

Le graphe de flot décrit dans la section 2.3.1 permet lui aussi de créer un problème linéaire mixte.

$$\min \quad \sum_{e \in E} v_e x_e + \sum_{e \in E} f_e y_e^0 + \sum_{\substack{e \in E_M \\ cap(e)>0}} \gamma_e y_e^1 \tag{37}$$

$$\text{s.t.} \quad \sum_{i:iv \in E} x_{iv} = \sum_{o:vo \in E} x_{vo} \qquad \forall v \in V \setminus \{s,t\} \tag{38}$$

$$\sum_{o:so \in E} x_{so} = \sum_{i:it \in E} x_{it} \tag{39}$$

$$l_e \leq x_e \leq u_e y_e^0 \qquad \forall e \in E, \neg(e \in E_M \wedge cap(e) > 0) \tag{40}$$

$$x_e = x_e^1 + x_e^2 \qquad \forall e \in E_M, cap(e) > 0 \tag{41}$$

$$l_e \leq x_e \qquad \forall e \in E_M, cap(e) > 0 \tag{42}$$

$$b_e y_e^1 \leq x_e^1 \leq b_e y_e^0 \qquad \forall e \in E_M, cap(e) > 0 \tag{43}$$

$$0 \leq x_e^2 \leq (u_e - b_e) y_e^1 \qquad \forall e \in E_M, cap(e) > 0 \tag{44}$$

$$y_e^0 \in \{0,1\} \qquad \forall e \in E \tag{45}$$

$$y_e^1 \in \{0,1\} \qquad \forall e \in E_M, cap(e) > 0 \tag{46}$$

$$\sum_{w_i^{n_k} \in W^{n_k}} x_{s w_i^{n_k}} \geq L_W^{n_k} \qquad \forall W^{n_k} \in W^n \tag{47}$$

$$\sum_{\sigma_i^{n_k} \in S^{n_k}} \sigma_{s \sigma_i^{n_k}} \geq L_S^{n_k} \qquad \forall S^{n_k} \in S^n \tag{48}$$

$$x_{ij} + x_{ji} \leq min(u_{ij}, u_{ji}) \qquad \forall ij \in E_M \tag{49}$$

$$x_{e^f} + x_{e^t} \leq max(size_m(e^f), size_m(e^t)) \qquad \forall (e^f, e^t) \in E_M^f \times E_M^t,\ e^f \parallel e^t \tag{50}$$

Ce problème est très proche de celui de la section 2.2.2. On remarque cependant l'ajout de (48), qui permet de respecter la contrainte : la somme du flot des stations dans chaque groupe de stations S^{n_k} doit être supérieure à une borne $L_S^{n_k}$ donnée.

La contrainte (50) permet de limiter le flot passant dans une arête de l'espace de travail discrétisé. $e^f \parallel e^t$ est vrai si et seulement si e^f et e^t représentent le même chemin discret. Cette contrainte peut être remplacée par la contrainte (51) si on souhaite éviter la présence simultanée de flow lines et de transfer lines sur une arête discrète.

$$y^0_{e^f} + y^0_{e^t} \leq 1 \qquad \forall (e^f, e^t) \in E^f_M \times E^t_M,\ e^f \parallel e^t \tag{51}$$

2.4 Analyse de la méthode

Cette sous-section analyse la méthode générale proposée pour effectuer une extension du réseau de pipelines. Elle cite les limites qui lui sont inhérentes et en propose quelques extensions.

Dans cette méthode, l'espace de travail est représenté par une discrétisation en tant que graphe planaire. Ce graphe peut être obtenu par de nombreuses méthodes dont une est détaillée dans la section 3. La précision des chemins générés dépend donc de la finesse de cette discrétisation. Ceci devient vite problématique puisque l'espace de travail est grand (de l'ordre de 10000 km^2). Le problème à résoudre étant assez complexe, les temps de calcul nécessaires pour donner une solution optimale au problème linéaire mixte seront très grands si la discrétisation est fine. De plus, l'optimalité du résultat ne sera vraie que pour une discrétisation donnée de l'espace de travail et non pour l'espace de travail lui-même. Il peut ainsi être intéressant de se servir de cette modélisation pour générer une solution approximative puis d'affiner les chemins obtenus grâce à d'autres techniques moins coûteuses. On peut pour cela arrêter le solveur linéaire après un temps donné ou utiliser une discrétisation moins fine.

Cette méthode ne permet d'étendre que le réseau des pipelines et ignore les modifications à faire sur les autres réseaux, ce qui nuit à la qualité d'un résultat au problème général posé par le stage. Une contrainte a notamment été énoncée concernant le besoin d'accéder aux puits par la route pour qu'une foreuse puisse y réaliser un forage initial ou au cours de la vie du puits. Nous pouvons traiter ces deux problèmes de manière indépendante (par exemple obtenir une solution au problème linéaire pour savoir quels éléments construire, puis ajouter des routes) au prix d'une perte d'optimalité. Je pense que si nécessaire, on peut modéliser les deux problèmes en même temps en utilisant un problème de flot à plusieurs commodités (plusieurs types de flots qui ne se mélangent pas). Le graphe des corridors possibles $G_M = (V_M, E_M)$ et celui des routes possibles $G_R = (V_R, E_R)$ seront probablement duaux et auront des croisements entre eux. Il faudra alors ajouter le coût de croisements entre ces chemins dans la fonction objectif à minimiser. On peut par exemple utiliser

$$\sum_{e_m \in E_M} \sum_{e_r \in E_R} y_{e_m} \cdot y_{e_r} \cdot cross(e_m, e_r) \tag{52}$$

où $cross(e_m, e_r)$ représente le coût de croisement imputé si un corridor est présent sur e_m et si une route est présente sur e_r. Le coût devient alors quadratique, ce qui risque d'augmenter considérablement le temps de résolution d'un tel problème. Cependant, la contrainte quadratique porte sur des variables binaires, ce qui à mon avis est optimisé par certains solveurs quadratiques.

Afin de modéliser plus finement le coût de construction entre un pipeline et une installation gênante (une route par exemple), il est possible d'utiliser une fonction affine par morceaux. Ce type de fonction permet d'appliquer des coûts différents en fonction de plusieurs paliers. En effet, les méthodes de construction de ces croisements permettent probablement de faire des croisements de diverses largeurs en fonction des machines utilisées. On peut ainsi imaginer qu'une machine M_1 puisse construire un croisement pour 5 pipelines et un coût c_1 alors qu'une machine M_2 peut construire un croisement pour 10 pipelines et un coût c_2, $c_2 > c_1$. On obtient ainsi une fonction affine par morceaux contenant un morceau sur $]0, 5]$ et un autre sur $]5, 10]$. Ce type de fonction se représente aisément dans un programme linéaire (cf. section 2.2.2 où une fonction affine par morceaux est utilisée pour représenter les coûts d'extension).

3 Construction du graphe de l'espace de travail

Dans le problème de flots, l'espace de travail est représenté sous la forme d'un graphe G_M. Cette section décrit comment obtenir ce graphe à partir des données d'entrée. La figure 17 (page 41) résume la méthode utilisée en liant les différents sous-problèmes entre eux. La plupart des sous-problèmes sont issus du domaine de la géométrie algorithmique.

3.1 Arrangement de segments

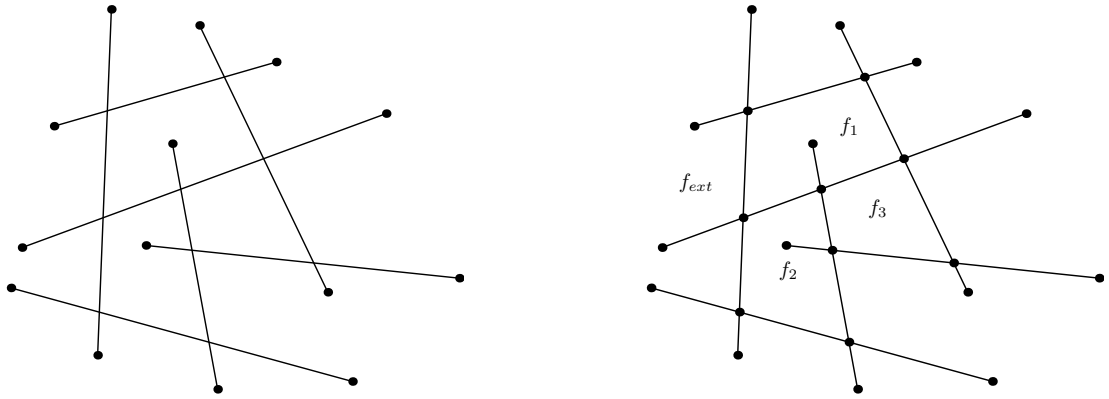
L'espace dans lequel nous travaillons peut être vu comme un plan en deux dimensions. Cet espace est composé de points (puits, stations de traitement, cpu), de polylignes (routes, pipelines) et de polygones (zones où il est interdit de construire des pipelines, zones où un coût est différent qu'ailleurs...). Une polyligne est un ensemble de segments placés les uns à la suite des autres. Une polyligne indique donc un chemin et un sens sur ce chemin. Ce sens est important pour le réseau de pipelines (qui vont des puits vers les stations de traitement, puis vers le CPU) mais pas pour les routes, puisque seule l'information de croisement nous intéresse.

Nous souhaitons partitionner le plan $\mathcal{P}$ en un ensemble de faces F, de telle sorte que $\mathcal{P} = \bigcup_i F_i$ et que $\forall (f_1, f_2) \in F \times F,\ f_1 \neq f_2 \implies f_1 \cap f_2 = \varnothing$. Autrement dit, nous souhaitons que le plan $\mathcal{P}$ soit découpé en zones de telle manière que tout point soit dans une zone et que les zones ne se chevauchent pas. Les faces F forment donc une partition du plan.

Puisque nos données sont *in fine* des points et des segments, nous pouvons nous intéresser à un type particulier de partitionnement du plan : l'arrangement de segments.

Un arrangement de segments est une partition du plan en cellules de dimension 0 (sommets), de dimension 1 (arêtes) et de dimension 2 (faces). Afin de manipuler un arrangement, la structure de données DCEL (*doubly-connected edge list*) est généralement utilisée (cf. rapport de Kévin Bourgeois qui a implémenté une telle structure). Celle-ci permet de stocker les sommets, les arêtes et les faces ainsi que les relations d'incidence entre ces cellules. Une relation d'incidence est une relation entre cellules qui ne sont pas de même dimension. On parle ainsi des arêtes incidentes à un sommet ou des faces incidentes à une arête. Lorsque les cellules sont de même dimension, on parle de relation d'adjacence, *e.g.* deux faces sont adjacentes si elles sont incidentes à une même arête.

On s'intéresse donc au partitionnement du plan à partir d'un ensemble de segments comme entrée. Ce partitionnement est représenté sur la figure 7 où la sous-figure 7a représente les segments en entrée et la sous-figure 7b représente le partionnement du plan résultant.



(a) Les segments d'entrée au problème.

(b) Le partitionnement du plan correspondant comprenant des nœuds, des arêtes et des faces.

FIGURE 7 – Un arrangement de segments

Ce problème est lié à celui de l'intersection d'un ensemble de segments. En effet, chaque point d'intersection entre plusieurs segments sera un nœud de l'arrangement final. L'algorithme de Bentley-Ottmann[6] est souvent pris comme référence en dépit de son manque d'optimalité en raison de sa simplicité. Les algorithmes de Clarkson[7] et de Mulmuley[8] ont une composante aléatoire (de la même manière que l'algorithme de tri quicksort) et devraient en moyenne se comporter de manière optimale. Enfin, l'algorithme de Chazelle-Edelsbrunner[9] est à la fois optimal et déterministe mais utilise des notions plus complexes et est plus difficile à implémenter[10].

Nous pouvons noter qu'aucun de ces articles ne décrit de manière suffisante la manière dont les cas dégénérés doivent être gérés. Les cas dégénérés comprennent les segments de longueur nulle, deux points se trouvant sur la même abscisse, une intersection entre trois segments ou plus au même point, un segment finissant à l'intérieur d'un autre... Ce manque de description rend les algorithmes plus simples à prouver et à décrire mais plus difficiles à impleménter en pratique, les cas dégénérés n'étant pas rares. Il existe des méthodes lourdes pour ne pas avoir à gérer les cas dégénérés mais elles ne sont pas efficaces en pratique[11].

Puisque ce problème n'est pas très coûteux en temps par rapport au problème du stage, j'ai décidé d'implementer une méthode naïve qui consiste à calculer l'intersection entre toutes les paires de segments. Cette méthode peut être facilement optimisée en segmentant l'espace, ce qui permet de ne calculer que l'intersection des segments étant dans la même zone. Cette méthode est également facilement parallélisable puisqu'on peut calculer les différentes intersections dans chaque unité de calcul de manière indépendante puis fusionner les résultats. La méthode est décrite par l'algorithme 1 de complexité $\mathcal{O}(n^2 log(n) + nk)$ avec $n = |S|$ et k le nombre de points d'intersection dans S.

Algorithme 1 : IntGraph calcule le graphe de l'intersection d'un ensemble de segments.

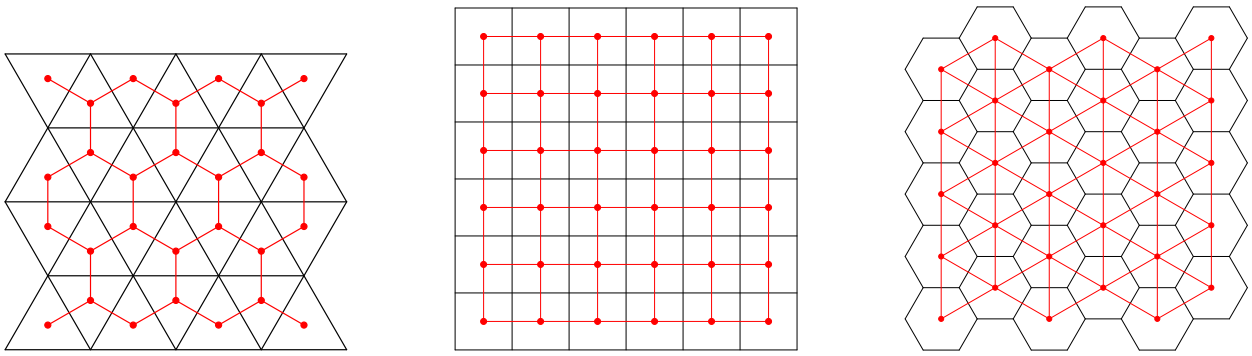
Entrées : S, l'ensemble des segments
Sorties : $G = (V, E)$, le graphe non orienté tel que :
- V contienne tous les points terminaux de S et tous les points d'intersection de S,
- $\bigcup_{e \in E} e = S$,
- $\forall (e_1, e_2) \in E \times E,\ e_1 \neq e_2 \implies e_1 \cap e_2 = \varnothing$.

$vm \leftarrow map(Point \rightarrow Noeud)$
$lm \leftarrow map(Segment \rightarrow set(Point))$
$em \leftarrow map((Point, Point) \rightarrow Arete)$
pour chaque $s \in S$ **faire**
    $insert(lm[s], source(s))$
    si $source(s) \notin vm$ **alors**
        $vm[source(s)] \leftarrow ajouterNoeud(G, source(s))$
    $insert(lm[s], cible(s))$
    si $cible(s) \notin vm$ **alors**
        $vm[cible(s)] \leftarrow ajouterNoeud(G, cible(s))$
pour chaque $(s_1, s_2) \in S \times S,\ s_1 \neq s_2$ **faire**
12 **pour chaque** $p \in pointsIntersection(s_1, s_2)$ **faire**
        $insert(lm[s_1], p)$
        $insert(lm[s_2], p)$
        si $p \notin vm$ **alors**
            $vm[p] \leftarrow ajouterNoeud(G, p)$
pour chaque $(s, P) \in lm$ **faire**
18 **pour chaque** $p_1 \rightarrow p_2 \in P$ **faire**
        si $(p_1, p_2) \notin em$ **alors**
            $em[(p_1, p_2)] \leftarrow ajouterArete(G, p_1, p_2)$
21          $data(em[(p_1, p_2)]) \leftarrow set(TypeArete)$
22      $insert(data(em[(p_1, p_2)]), type(s))$
retourner (G)

Ce paragraphe décrit l'algorithme 1. Les structures de données *map* utilisées sont des tableaux associatifs tels des arbres équilibrés (AVL, arbre rouge-noir...). La structure de données *set* est un ensemble ordonné, on peut utiliser un arbre équilibré tel un AVL en pratique. La fonction $pointsIntersection(s_1, s_2)$ de la ligne 12 renvoie les points d'intersection des deux segments s_1 et s_2. Cette fonction peut ne renvoyer aucun point si les segments ne se croisent pas, un point s'ils ont un croisement classique et deux points p_1 et p_2 si s_1 et s_2 sont confondus entre p_1 et p_2. À la ligne 18, P est une suite ordonnée de points (puisqu'un *set* est ordonné). On la parcourt en prenant des paires de points consécutifs, c'est ce que signifie la notation $p_1 \to p_2 \in P$. La ligne 21 permet d'ajouter des métadonnées aux arêtes afin de pouvoir connaître pour chaque arête les types dont elle est issue. Ces informations sont stockées dans un ensemble de types d'arêtes $set(TypeArete)$. Enfin, la ligne 22 permet de mettre à jour les métadonnées des arêtes, $type(s)$ étant le type de l'arête s. Le nombre de types d'arêtes différents étant petit en pratique, nous pouvons utiliser la représentation des nombres (en associant à chaque bit un type d'arête) plutôt qu'un ensemble de types d'arêtes. Cette technique permet de faire des opérations sur cet ensemble en temps constant.

3.2 Discrétisation de l'espace de travail

L'algorithme 1 est appelé sur un ensemble de segments composé des infrastructures gênantes pour les pipelines (les routes), des délimitations des zones de coût (cf. section 3.3) ainsi que d'un ensemble de segments permettant de discrétiser l'espace. Ces segments discrets forment un pavage pouvant avoir plusieurs types de forme. Puisque nous souhaitons partitionner l'espace de travail en un ensemble de zones d'aire et de forme similaire, nous avons choisi d'utiliser des pavages réguliers. Les trois pavages réguliers du plan euclidien sont représentés sur la figure 8.



(a) Un pavage triangulaire. (b) Un pavage carré. (c) Un pavage hexagonal.

FIGURE 8 – Les trois types de pavages réguliers du plan euclidien. On peut voir sur chaque schéma le pavage (noir) et son dual (rouge).

Le pavage carré est très simple à construire puisqu'il suffit de fournir un ensemble de lignes verticales et horizontales à l'arrangement qui se chargera de les découper en sous-parties. Le pavage triangulaire n'est guère plus difficile puisqu'il suffit d'utiliser un ensemble de lignes horizontales, de lignes inclinées d'un angle $\alpha = \frac{\pi}{3}$ et de lignes inclinées d'un angle $-\alpha = -\frac{\pi}{3}$. Le pavage hexagonal est par contre un petit peu plus difficile à construire.

Le dual d'un pavage carré reste un pavage carré. Les pavages triangulaires et hexagonaux sont duaux l'un de l'autre. Ces trois types de pavages réguliers peuvent être utilisés pour discrétiser l'espace de travail. On remarque cependant que plus le nombre de sommets du polygone régulier utilisé est grand, plus le degré des sommets du graphe dual au pavage est grand, ce qui améliore la précision des chemins mais augmente le nombre d'arêtes du graphe et donc le temps de calcul.

3.3 Zones de coût définies par l'utilisateur

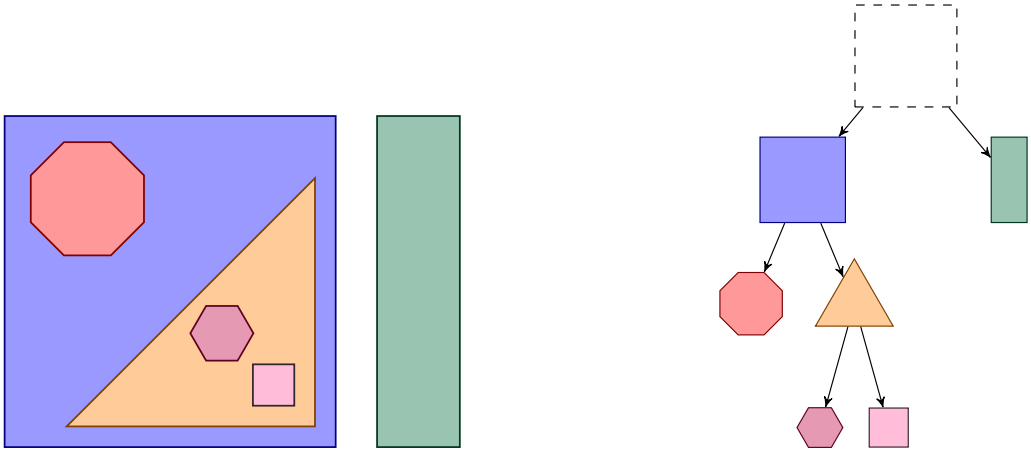
L'espace de travail étudié pendant le stage est situé dans le désert des Émirats arabes unis. Cet espace n'a qu'un seul type de sol, peu d'obstacles et un nivellement faible. Ceci se traduit par une isotropie des coûts : ils ne dépendent pas (plutôt peu) de l'orientation dans lequel le déplacement est effectué. Ceci nous a conduit à ignorer ce type de coût. Une reflexion sur la prise en compte du dénivelé en terme de contraintes et de coûts est faite en section 3.5.

Afin de pouvoir appliquer des coûts différents en fonction de l'emplacement d'un pipeline, j'ai choisi d'utiliser des zones qui permettent de définir différemment certains coûts. Une zone est représentée par un polygone qui en définit la bordure extérieure. Une zone peut être comprise dans une autre, mais deux zones ne peuvent se chevaucher. De plus, deux zones ne peuvent pas être strictement égales. Ces zones peuvent être de trois types différents :

- **absolue** : les coûts de la zone z ont une valeur définie uniquement par la zone z : les coûts ne dépendent pas de la zone dans laquelle z est située,
- **additive** : les coûts de la zone z sont additionnés à ceux de la zone dans laquelle z est située,
- **multiplicative** : les coûts de la zone z sont multipliés à ceux de la zone dans laquelle z est située.

Il existe une zone absolue de taille infinie et de type fixe qui définit les coûts de l'espace de travail. Si la zone z_1 est dans la zone z_2 alors $z_1 \subset z_2$. Ainsi, les zones Z sont organisées sous la forme d'un arbre dont des nœuds sont des zones et $\forall (z_1, z_2) \in Z^2,\ z_1 \subset z_2 \ \equiv \ z_1 \in fils(z_2)$. La figure 9 illustre la notion de zones de coûts. Les zones sont représentées de manière spatiale dans la figure 9a et sous forme d'arbre sur la figure 9b. La zone absolue de taille infinie est toujours la racine de l'arbre des zones.

L'utilisateur définit un ensemble de zones de coût à utiliser. Nous organisons ensuite ces zones en arbre, ce qui permet d'accélérer l'attribution des coûts aux faces de l'arrangement. L'algorithme 2 définit comment obtenir un arbre à partir d'un ensemble de zones et d'une zone infinie. La notation $Z[z_1..]$ signifie que l'on itère Z à partir de z_1 jusqu'au dernier élément de Z. L'idée de cet algorithme est de trier les polygones par aire croissante. Ceci permet d'être sûr, pour chaque polygone p, que le premier polygone rencontré englobant p est le plus petit des polygones qui englobent p, c'est-à-dire son nœud père dans l'arbre. Cet algorithme est parallélisable puisqu'on peut rechercher le père de chaque polygone de manière indépendante. Enfin, si le nombre de polygones devient grand, on peut segmenter l'espace en plusieurs parties pour éviter les calculs inutiles.



(a) Représentation des zones dans l'espace de travail.

(b) Représentation sous forme d'arbre.

FIGURE 9 – Différentes zones de coût.

Une fois l'arbre obtenu, nous pouvons déterminer à quelle zone appartient chaque face de l'arrangement. Puisque les segments des polygones font partie des segments d'entrée de l'arrangement, nous sommes assurés que chaque face de l'arrangement est soit complètement couverte par une zone, soit n'est pas couverte du tout par ladite zone. Ainsi, nous cherchons à calculer pour chaque face de l'arrangement l'ensemble des zones qui la couvrent. On peut pour cela se servir de la structure arborescente des zones : si une face n'est pas couverte par la zone z, il est impossible qu'elle soit couverte par une des zones filles de z. On parcourt ainsi, pour chaque face de l'arrangement, l'arbre des zones en ne continuant le parcours que si la face est couverte par la zone. Au final, on obtient pour chaque face de l'arrangement une liste de zones de coûts triée de la zone la plus englobante à la plus petite. Enfin, les coûts de la face sont calculés en enchaînant les opérations de calcul (addition, multiplication, affectation) en fonction du type de zone et des coûts associés à chaque zone.

3.4 Prise en compte des pipelines existants

Afin de remplir le graphe de flots correctement, il faut savoir où se situent les pipelines existants dans le partitionnement de l'espace qui a été créé. Un parcours de la DCEL permet d'obtenir une suite de cellules (sommets, arêtes ou faces) correspondant à chaque pipeline. Cette suite de cellules est ensuite traitée pour générer une suite de faces. Puisqu'à chaque face de l'arrangement correspond un sommet du graphe de flots, il suffit de modifier la capacité minimale des arêtes correspondant à la suite de faces de l'arrangement. Il faut également s'assurer que les contraintes correspondant aux arêtes modifiées restent satisfaisables (notamment la contrainte de borne maximale sur la somme du flot d'une arête et de son arête symétrique).

Algorithme 2 : ArbreZones calcule l'arbre d'inclusion d'un ensemble de zones de coût.

Entrées : Z, l'ensemble des zones telles que :
- Il n'existe pas deux zones couvrant exactement la même surface
  $\forall (z_1, z_2) \in Z^2,\ z_1 \neq z_2$,
- Si une zone chevauche une autre alors une des zones couvre toute la surface de l'autre
  $\forall (z_1, z_2) \in Z^2,\ (z_1 \cap z_2 \neq \varnothing) \ \equiv\ \big((z_1 \subset z_2) \veebar (z_2 \subset z_1)\big)$.

  z_{inf}, la zone absolue infinie

Sorties : $T = (Z, E)$, un arbre tel que :
- $\forall (z_1, z_2) \in Z^2,\ z_1 \subset z_2 \ \equiv\ z_2 z_1 \in E$.

$Z \leftarrow Z \cup z_{inf}$
Trier Z par aire croissante
pour chaque $z \in Z$ **faire**
  $ajouterNoeud(T, z)$
pour chaque $z_1 \in Z,\ z_1 \neq z_{inf}$ **faire**
  pour chaque $z_2 \in Z[z_1..]$ **faire**
    si $z_1 \subset z_2$ **alors**
      $ajouterArete(T, z_2 z_1)$
      Arrêter d'itérer z_2
retourner (T)

Des cas dégénérés peuvent se présenter lorsque l'on souhaite transformer la suite de cellules en suite de faces. Il est en effet possible qu'un pipeline ne passe par aucune face (en longeant les arêtes et en passant par les sommets) ou qu'il passe par un point en reliant deux faces qui ne sont pas incidentes à la même arête. Les arêtes du graphe de flot étant construites à partir des arêtes de l'arrangement, ces cas posent problème. Ces cas n'ont pas été gérés (ils ne sont jamais apparus en pratique).

3.5 Analyse et améliorations

La méthode de partitionnement du plan utilisée permet de créer un maillage hybride. En effet, un maillage régulier est appliqué sur tout l'espace de travail grâce à un arrangement de segments. Les zones précédemment vides suivent alors un maillage régulier alors que celles contenant des infrastructures gênantes existantes ou des délimitations de zones de coût suivent un maillage irrégulier. Cette méthode permet ainsi de prendre en compte les infrastructures existantes et a l'avantage de créer facilement des faces dont l'aire a une borne supérieure. Les faces générées ne sont cependant pas convexes ni sans trou, ce qui crée des problèmes dans l'approximation de la distance entre deux faces adjacentes. En effet, la distance entre deux faces est approximée par la distance entre les centroïdes de ces faces, ce qui n'a un sens que si les faces sont convexes. Nous effectuons un traitement sur les faces ainsi obtenues afin de les rendre strictement convexes, ce qui augmente considérablement le nombre de faces. Un trop grand nombre de faces est problématique puisque la complexité en temps et en espace du problème dépend du nombre d'arêtes du graphe de flots, qui est directement lié au nombre de faces du partitionnement du plan. Nous fusionnons ainsi les petites faces en faisant en sorte que la face résultante reste presque convexe.

Avant de réaliser cette méthode, nous avons essayé de partitionner le plan grâce à une triangulation de Delaunay contrainte. Cette méthode est une généralisation de la triangulation de Delaunay qui force certaines arêtes à être présentes dans la triangulation (les délimitations de zones de coût et les infrastructures gênantes dans notre cas). Cette méthode a l'avantage de créer des faces convexes. Cependant, le maillage obtenu est irrégulier et les triangles générés ont une grande variance d'aire. En effet, le centre des grandes faces initiales est composé d'un petit nombre de très grands triangles alors que les bordures entre plusieurs faces comportent de nombreux très petits triangles. Les grandes faces sont pratiques pour faire des recherches de chemins mais elles représentent mal l'occupation du sol par les pipelines générés. Les très petites faces, quant à elles, agrandissent considérablement la taille du graphe de flots, ce qui pose des problèmes à la fois en espace et en temps lors de la résolution du problème. Nous n'avons pas retenu cette méthode mais nous aurions pu nous en servir en effectuant un post-traitement sur les faces, afin d'en limiter le nombre.

Nous nous sommes également intéressé à différentes techniques de création de *mesh polygonaux*. En effet, il serait intéressant d'obtenir des faces dont l'aire suit un processus Gaussien, c'est-à-dire des faces telles que leur aire est proche de la moyenne de l'aire des faces. Bubble mesh[12] propose une méthode basée sur une simulation physique de bulles qui permet de créer une triangulation d'un solide. Le placement initial des bulles est effectué par un arbre binaire sur les arêtes et par un quadtree pour les surfaces. Les bulles sont ensuite modifiées et déplacées par une simulation physique (qui peut également supprimer des bulles ou en faire apparaître). Une fois la simulation terminée, on utilise une triangulation de Delaunay contrainte pour obtenir la triangulation finale. NETGEN[13] est une autre méthode de génération de mesh triangulaire de solides. Cette méthode est constructive : elle fait avancer un front qui commence sur la bordure de chaque domaine et progresse jusqu'à fermer chaque domaine par des triangles. Les triangles sont créés par des règles abstraites de construction. Nous n'avons pas utilisé ces méthodes car elles semblaient être plus difficiles à implémenter que la solution retenue.

Les routes ont été modélisées comme des segments, ce qui simplifie la création de croisements mais perd l'information concernant l'empreinte spatiale des routes. Il peut être intéressant d'utiliser des polygones pour représenter les routes. Les routes ne seront ainsi plus des segments mais des faces. Ces zones de routes peuvent être intégrées facilement à la solution retenue (arrangement de segments) puisque cette méthode est très générale. La création du graphe à partir des faces de l'arrangement va cependant changer pour les croisements pipeline/route. Il faudrait en effet connecter les faces vides (celles qui ne correspondent pas à des routes ni à des obstacles) qui sont adjacentes à une même face correspondant à une route. Pour créer ces arêtes, il suffirait, pour chaque face f_r correspondant à une route, de lister les faces F_{f_r} vides et adjacentes à f_r puis de créer une arête pour chaque paire de faces de F_{f_r}. D'autres contraintes pourraient être ajoutées afin de ne pas créer d'arêtes invalides. Connaître l'empreinte spatiale des routes permettrait d'améliorer l'approximation du coût des croisements puisque le coût d'un croisement dépend de la largeur de la route qu'il faut traverser.

Si l'on souhaite étendre les méthodes développées à d'autres cadres spatiaux, le dénivelé peut devenir un élément important. Il fait en effet varier le coût de construction des différents éléments, construire dans une forte pente pouvant être plus cher que de construire sur un terrain plat. Le dénivelé influence également le phénomène physique connu en tant que *perte de charge* dans les pipelines, qui impose d'utiliser des pompes plus puissantes pour un même débit de fluide dans un tuyau non horizontal que dans un tuyau horizontal. Ceci se traduit probablement en terme de coût puisque l'ajout de pression dans les pipelines n'est pas gratuit. Afin de gérer le dénivelé, il est possible d'associer à chaque face de l'arrangement une altitude. On peut ainsi calculer la différence d'altitudes entre deux faces adjacentes et modifier en conséquence les coûts de l'arête reliant les faces. De plus, le dénivelé implique probablement des coûts anisotropiques : aller dans un sens n'a pas forcément le même coût qu'aller dans le sens opposé. Il est également possible de supprimer les arêtes ayant un dénivelé trop fort afin de respecter les contraintes de construction. Nous manquons d'information sur les coûts associés au dénivelé, nous n'avons donc pas proposé de modélisation de ce type de coût. Les fonctions utilisées seront cependant situées quelque part entre une fonction linéaire et l'équation de Dercy-Weisbach[14] en fonction de la qualité de l'approximation souhaitée.

4 Résultats

Cette section montre un cas d'utilisation de la méthode que nous proposons et des outils que nous avons développé. On s'intéresse à une extension du réseau des pipelines en pouvant créer des stations intermédiaires, des flow lines et des transfer lines.

Le problème considéré est illustré en figure 10. Les routes y sont représentées en noir et les pipelines existants en bleu. Les objets ronds sont des objets existants alors que ceux carrés sont potentiels. Les ronds bleu foncé sont des puits de pétrole/gaz. Les ronds bleu clair sont des stations intermédiaires et le rond bleu pâle au sud est l'unique CPU. Les carrés bleu foncé sont des puits de pétrole/gaz potentiels et les carrés bleu clair des stations intermédiaires potentielles. Les zones de coût et les obstacles sont représentés par des polygones grisés. La création des stations n'est pas forcée (elle aurait pu l'être), la méthode que nous proposons peut créer des stations intermédiaires ou non en fonction de la capacité des stations à réduire le coût total de l'extension.

Tous les puits potentiels ont le même flot minimal (0) et le même flot maximal (1). Les puits potentiels sont regroupés en trois sous-ensembles. Le premier est situé au nord-ouest de l'espace de travail et comporte 6 puits dont 3 doivent être construits. Le second sous-ensemble est situé au nord de l'espace de travail et comporte 9 puits dont 5 doivent être construits. Enfin, le dernier sous-ensemble comporte un puits à l'est et un puits au sud qui doivent tous deux être construits.

Toutes les stations intermédiaires (existantes et potentielles) ont une capacité maximale de 10 puits : on ne peut pas relier plus de 10 puits à une seule station intermédiaire.

Le partitionnement du plan obtenu par notre méthode se trouve en figure 11. Le graphe de flot résultant est illustré en figure 12. On peut remarquer que ce graphe ne comporte pas de nœud à l'intérieur des obstacles. On évite ainsi de créer des chemins invalides tout en réduisant les temps de calcul et l'espace mémoire occupé. Des arêtes existent à l'intérieur et entre les différentes zones de coût (à l'ouest). On remarque également que toutes les arêtes ne sont pas utilisées (au sud-ouest notamment), ce qui est le résultat d'une heuristique qui marque les nœuds trop éloignés des chemins optimaux (en ne prenant en compte que la distance entre les nœuds) comme nœuds non utilisés dans le programme linéaire.

Une première solution au problème se trouve en figure 13. Les coûts choisis pour générer cette solution sont les suivants. Le coût d'extension des corridors de pipelines dans le vide (en l'absence de croisement) est nul. Le coût de construction d'un corridor de pipelines dans le vide vaut 1. Le coût variable est faible. Le coût de construction d'un nouveau croisement pipeline/route est de 10. On considère que les croisements existants ont été construits assez larges pour y faire passer des pipelines sans coût d'extension supplémentaire. Ces faibles coûts font que la création de station intermédiaire est chère (50), ainsi aucune station intermédiaire n'est créée dans ce cas.

Une autre solution au problème se trouve en figure 14. Cette solution a les coûts suivants. Le coût variable est faible. Le coût de construction d'un corridor de pipeline dans le vide est de 2. Le coût d'extension d'un corridor de pipelines dans le vide est de $\frac{1}{2}$. Le coût de construction d'un nouveau croisement pipeline/route est de 10. Les croisements pipeline/route existants sont considérés comme étant saturés. Ainsi, le coût d'extension de ces croisements est de 10, c'est-à-dire le même coût que lorsqu'on crée un nouveau croisement.

Dans la solution de la figure 14, on peut remarquer l'ajout de transfer lines entre la station intermédiaire existante au nord-ouest et le CPU. Cette nouvelle route a été créée car il s'avère qu'il est moins cher de créer ces nouvelles transfer lines que d'étendre le chemin existant. Ceci n'a pas de sens dans ce cas puisque la transfer line entre ladite station et le CPU existe déjà, il suffit d'y faire passer plus de flot. C'est cependant logique au sens de la modélisation puisque la méthode de résolution ne sait pas d'où vient le flot, ce nouveau chemin aurait ainsi été pertinent si le flot provenait d'une nouvelle station de traitement, ce que la méthode de résolution ne peut savoir. L'étape finale de notre méthode qui consiste à générer des chemins réels devrait ainsi détecter ce genre de cas.

Dans la solution de la figure 14, on remarque qu'une nouvelle station intermédiaire a été créée au nord. Le solveur linéaire ayant été arrêté avant de trouver une solution optimale, on voit qu'il hésite sur la manière de relier cette station au CPU (il existe un chemin longeant une route vers le sud et un autre vers l'est). Là encore, la génération de chemins réels permettrait de trancher. Il est cependant intéressant de montrer ce type de divergence dans le cadre de l'aide à la décision multicritère. En effet, si nous disposons de plusieurs solutions nous pouvons les présenter à un décideur qui sera chargé de prendre la décision finale. Afin de faciliter cette tâche il est possible d'effectuer divers calculs sur chaque solution afin de la confronter à chaque critère comme le coût de l'extension, l'accessibilité par la route de l'extension ou son impact environnemental. Afin d'obtenir plusieurs solutions, on peut indiquer au solveur linéaire d'enregistrer toutes les solutions intermédiaires trouvées.

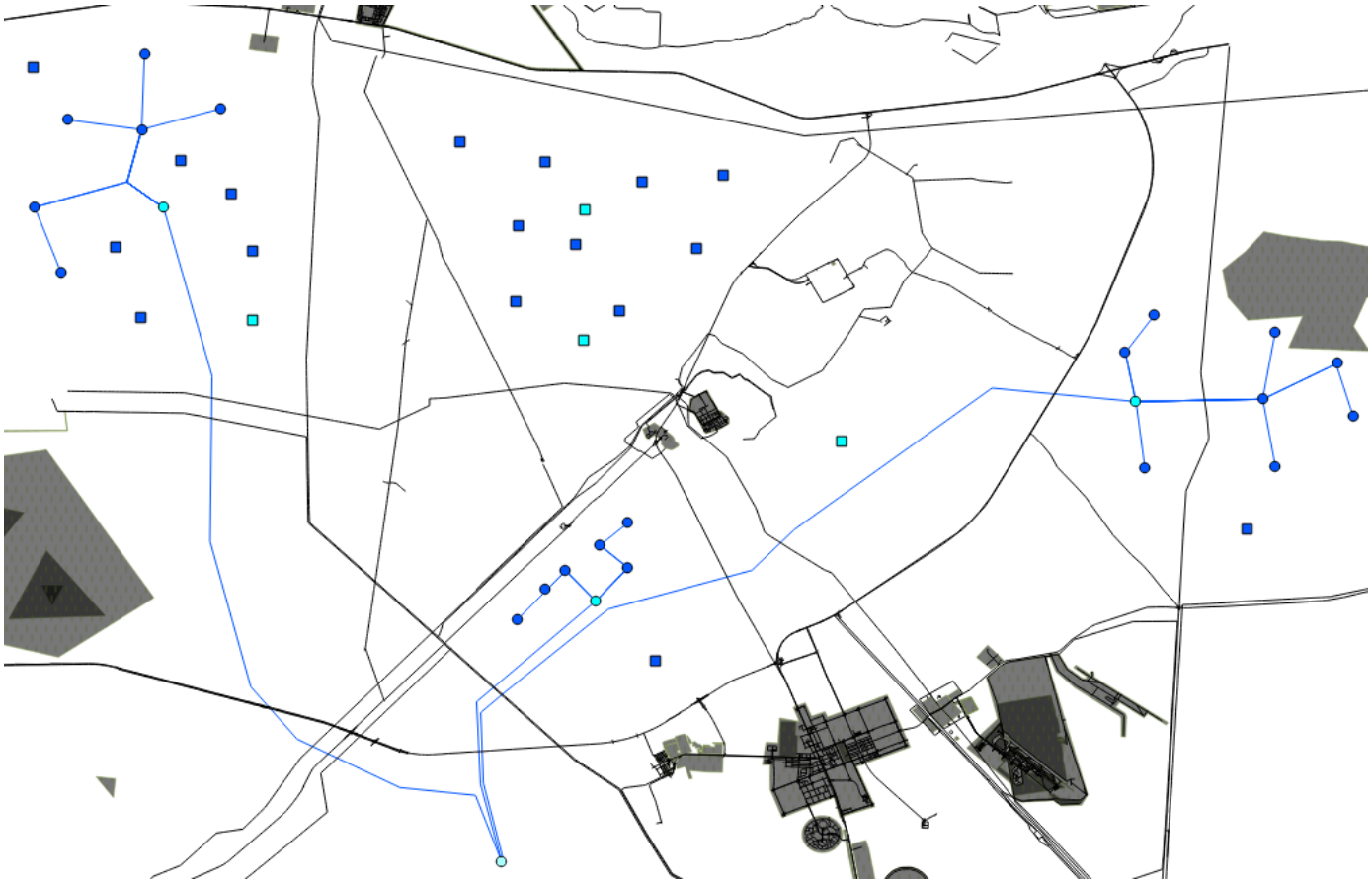


FIGURE 10 – Le problème considéré.

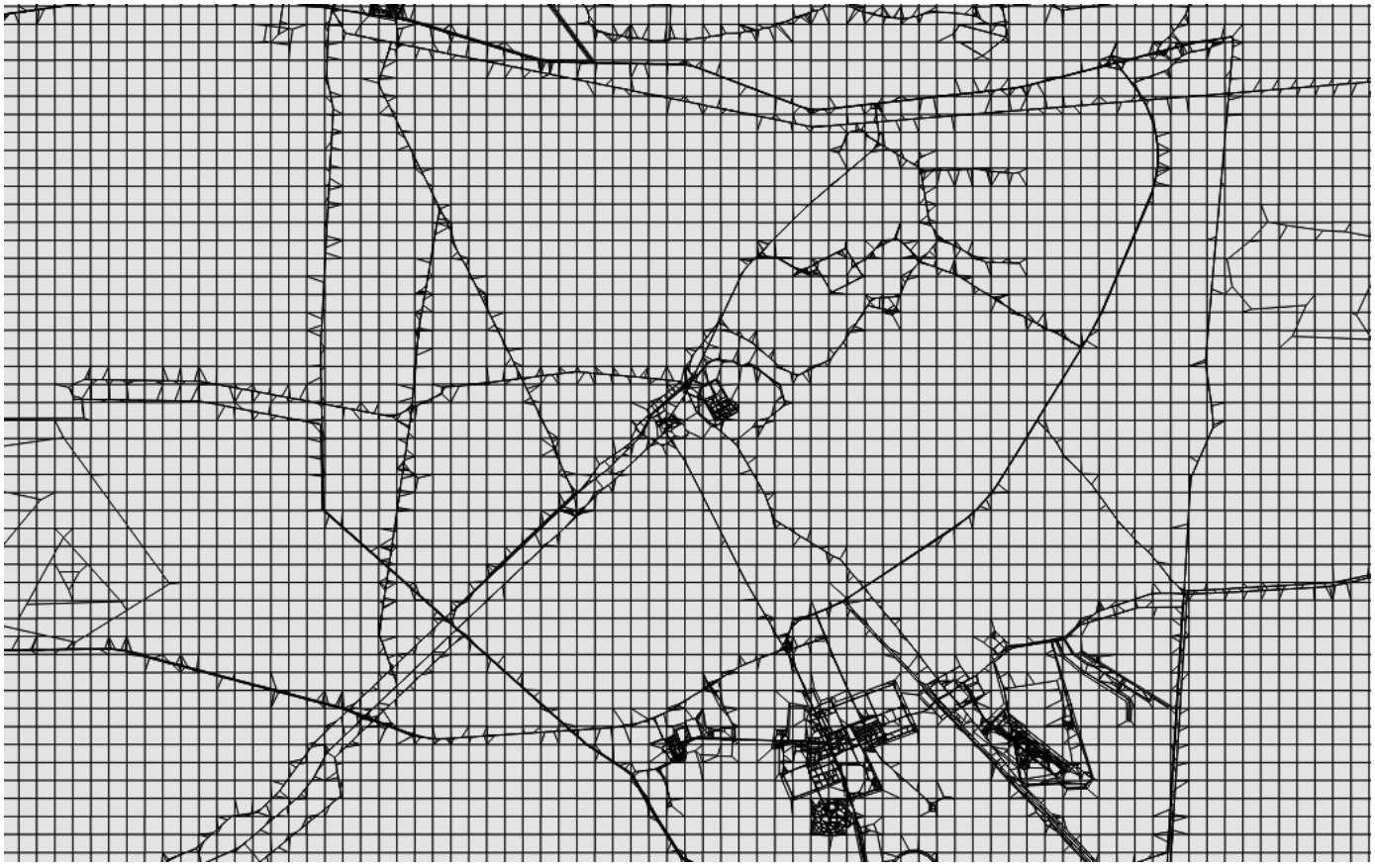


FIGURE 11 – Le partitionnement du plan.

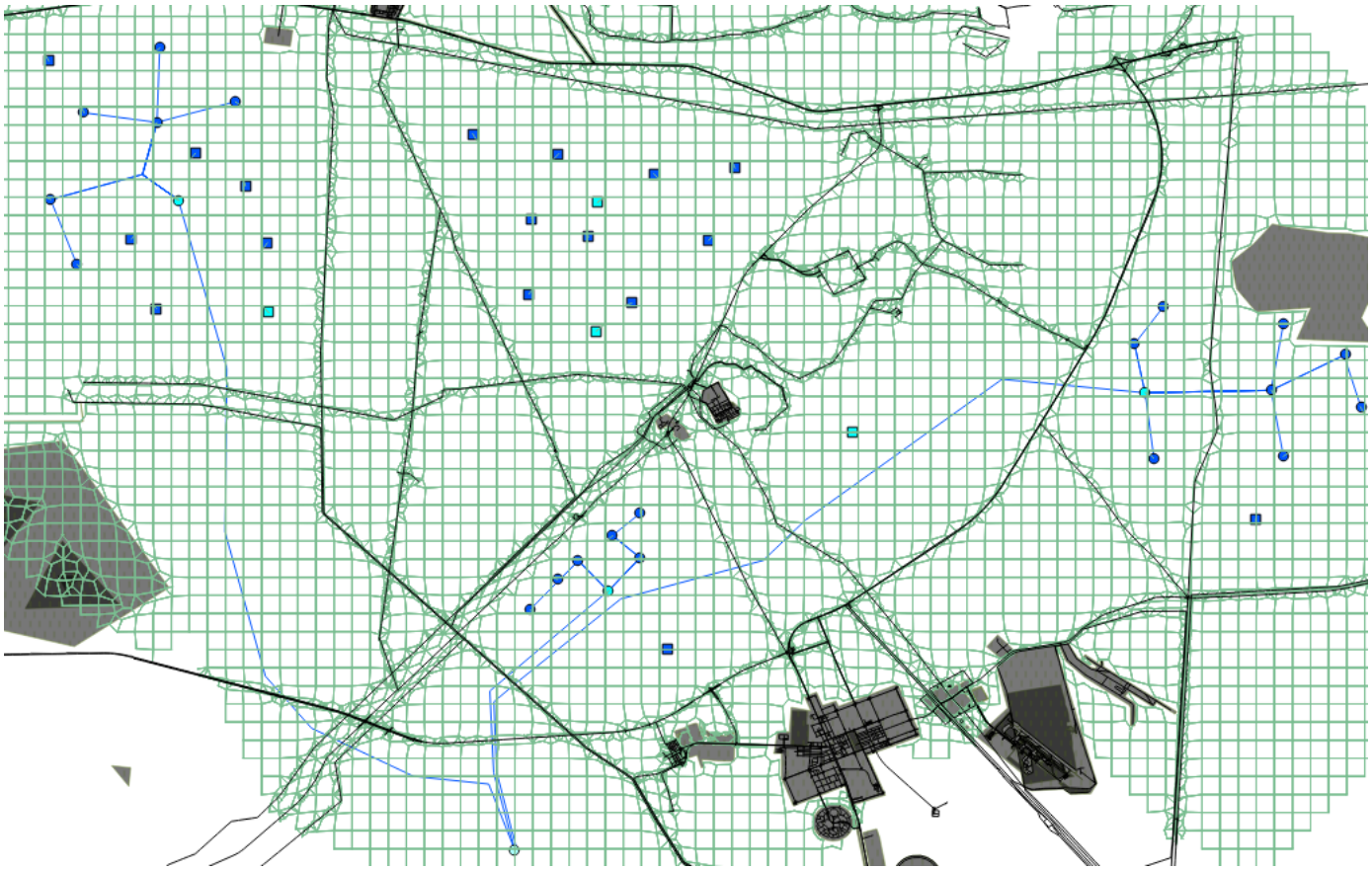


FIGURE 12 – Le graphe de flot (en bleu-vert).

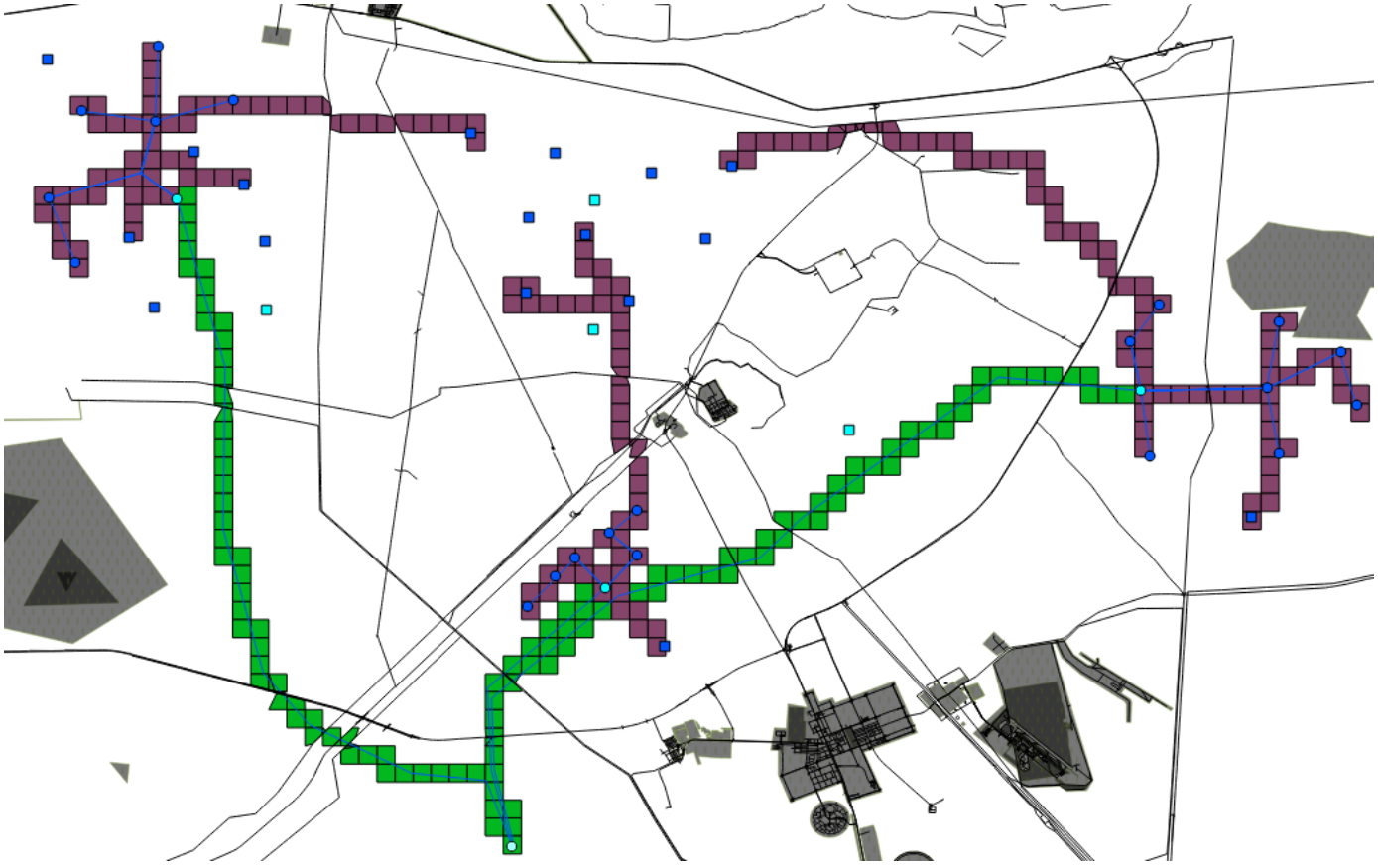


FIGURE 13 – Une solution au problème. Les zones au sein desquelles les flowlines sont construits sont en violet. Les mêmes zones concernant les transfer lines sont en vert.

FIGURE 14 – Une autre solution au problème. Les zones au sein desquelles les flowlines sont construits sont en violet. Les mêmes zones concernant les transfer lines sont en vert.

Conclusion

Au cours de ce stage, j'ai proposé une méthode qui permet de réaliser une extension du réseau des pipelines. Cette méthode construit un graphe de flot correspondant à l'espace de travail puis résout un problème de flot de coût minimum sous contraintes grâce à une modélisation en programme linéaire mixte. La méthode de construction de graphe proposée est assez générique pour discrétiser d'autres domaines et peut permettre d'étendre d'autres types de réseaux comme le réseau électrique ou celui d'eau. Cette méthode a été conçue dans le but de gérer les coûts de croisements de manière réaliste (ce sont les coûts principaux de cette extension), ce qui n'était pas fait dans les prototypes précédents à Géo-Hyd. Enfin, l'intérêt d'utiliser un solveur linéaire est de pouvoir obtenir une très bonne solution si on utilise une discrétisation fine et qu'on s'autorise un long temps de résolution. La méthode proposée est compatible avec une analyse multicritère puisqu'elle permet de générer plusieurs solutions mais elle n'est pas directement orientée en ce sens.

Ce stage a été l'occasion pour moi de compléter mon initiation à la recherche par une problématique réelle. Le sujet d'étude de ce stage étant situé sur plusieurs domaines de recherche, ce stage m'a permis de découvrir la géométrie algorithmique et d'apprendre à modéliser un problème. Cela a également été l'occasion pour moi de me renseigner sur les méthodes d'optimisation mathématique et sur la conception de programmes linéaires mixtes. Les sous-problèmes posés par ce stage étant nombreux, j'ai appris à lire rapidement des articles de recherche afin de savoir s'ils étaient applicables aux problèmes posés ou non.

Ce stage m'a également permis de découvrir le monde de l'entreprise. Cette expérience fut enrichissante mais me conforte sur mes choix qui sont de vouloir travailler en tant que chercheur au sein d'un laboratoire publique. Je trouve en effet que les contraintes commerciales sont moins fortes dans ce cadre, ce qui me laisse une plus grande liberté de recherche.

Références

[1] Salem Chakhar. *Cartographie décisionnelle multicritère : formalisation et implémentation informatique*. PhD thesis, Université Paris Dauphine-Paris IX, 2006.

[2] Jean-Luc Marichal. Fonctions d'agrégation pour la décision.

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows : Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[4] Hak-Jin Kim and John N Hooker. Solving fixed-charge network flow problems with a hybrid optimization and constraint programming approach. *Annals of Operations Research*, 115(1-4) :95–124, 2002.

[5] Mike Hewitt, George L Nemhauser, and Martin WP Savelsbergh. Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing*, 22(2) :314–325, 2010.

[6] Jon L Bentley and Thomas A Ottmann. Algorithms for reporting and counting geometric intersections. *Computers, IEEE Transactions on*, 100(9) :643–647, 1979.

[7] Kenneth L Clarkson and Peter W Shor. Applications of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4(1) :387–421, 1989.

[8] Ketan Mulmuley. A fast planar partition algorithm, i. *Journal of Symbolic Computation*, 10(3) :253–280, 1990.

[9] Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM (JACM)*, 39(1) :1–54, 1992.

[10] John D Hobby. Practical segment intersection with finite precision output. *Computational Geometry*, 13(4) :199–214, 1999.

[11] Christoph Burnikel, Kurt Mehlhorn, and Stefan Schirra. On degeneracy in geometric computations. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 16–23. Society for Industrial and Applied Mathematics, 1994.

[12] Kenji Shimada and David C Gossard. Bubble mesh : automated triangular meshing of non-manifold geometry by sphere packing. In *Proceedings of the third ACM symposium on Solid modeling and applications*, pages 409–419. ACM, 1995.

[13] Joachim Schöberl. Netgen an advancing front 2d/3d-mesh generator based on abstract rules. *Computing and visualization in science*, 1(1) :41–52, 1997.

[14] Glenn O Brown. The history of the darcy-weisbach equation for pipe flow resistance. *Environmental and Water Resources History*, 38(7) :34–43, 2002.

[15] Hua-Yang Lin, Ping-Yu Hsu, and Gwo-Ji Sheen. A fuzzy-based decision-making procedure for data warehouse system selection. *Expert systems with applications*, 32(3) :939–953, 2007.

[16] Mehdi Samadi, Ariel Felner, and Jonathan Schaeffer. Learning from multiple heuristics. In *AAAI*, pages 357–362, 2008.

[17] Rafia Inam. A* algorithm for multicore graphics processors. Master's thesis, Chalmers University of Technology, 2010.

[18] Fernando Kuipers, Turgay Korkmaz, Marwan Krunz, and Piet Van Mieghem. Performance evaluation of constraint-based path selection algorithms. *Network, IEEE*, 18(5) :16–23, 2004.

[19] Adi Botea, Martin Müller, and Jonathan Schaeffer. Near optimal hierarchical path-finding. *Journal of game development*, 1(1) :7–28, 2004.

[20] Daniel Harabor and Adi Botea. Hierarchical path planning for multi-size agents in heterogeneous environments. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 258–265. IEEE, 2008.

[21] Roland Geraerts. Planning short paths with clearance using explicit corridors. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1997–2004. IEEE, 2010.

[22] Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. Theta* : Any-angle path planning on grids. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1177. Menlo Park, CA ; Cambridge, MA ; London ; AAAI Press ; MIT Press ; 1999, 2007.

[23] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2) :100–107, 1968.

[24] Natthaporn Buaphut and Nanthi Suthikarnnarunai. Effects of pipeline extension and network robustness evaluation : the case study of oil distribution to the northern region of thailand. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 2, 2014.

[25] Hyoungshick Kim and Ross Anderson. An experimental evaluation of robustness of networks. *IEEE Systems Journal*, 7(2) :179–188, 2013.

[26] Ali Sydney, Caterina Scoglio, Phillip Schumm, and Robert E Kooij. Elasticity : topological characterization of robustness in complex networks. In *Proceedings of the 3rd International Conference on Bio-Inspired Models of Network, Information and Computing Sytems*, page 19. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[27] Boris Aronov, Kevin Buchin, Maike Buchin, Bart Jansen, Tom de Jong, Marc van Kreveld, Maarten Loffler, Jun Luo, Rodrigo I Silveira, and Bettina Speckmann. Connect the dot : Computing feed-links for network extension. *Journal of Spatial Information Science*, pages 3–31, 2014.

[28] Tom de Jong and T Tillema. Transport network extensions for accessibility analysis in geographic information systems. In *Proceedings of AfricaGIS*, 2005.

[29] Marta SR Monteiro, Dalila BMM Fontes, and Fernando ACC Fontes. An ant colony optimization algorithm to solve the minimum cost network flow problem with concave cost functions. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 139–146. ACM, 2011.

[30] Kenneth E Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 277–286. ACM Press/Addison-Wesley Publishing Co., 1999.

[31] Aydin Buluç and Kamesh Madduri. Parallel breadth-first search on distributed memory systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 65. ACM, 2011.

[32] Steven J Owen. A survey of unstructured mesh generation technology.

[33] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational geometry*, 22(1) :21–74, 2002.

[34] Ulrike Bartuschka, Kurt Mehlhorn, and Stefan Näher. A robust and efficient implementation of a sweep line algorithm for the straight line segment intersection problem. In *in proc. workshop on algorithm engineering*, pages 124–135, 1997.

[35] Timothy M Chan. A simple trapezoid sweep algorithm for reporting red/blue segment intersections. In *CCCG*, pages 263–268. Citeseer, 1994.

A Grandes figures

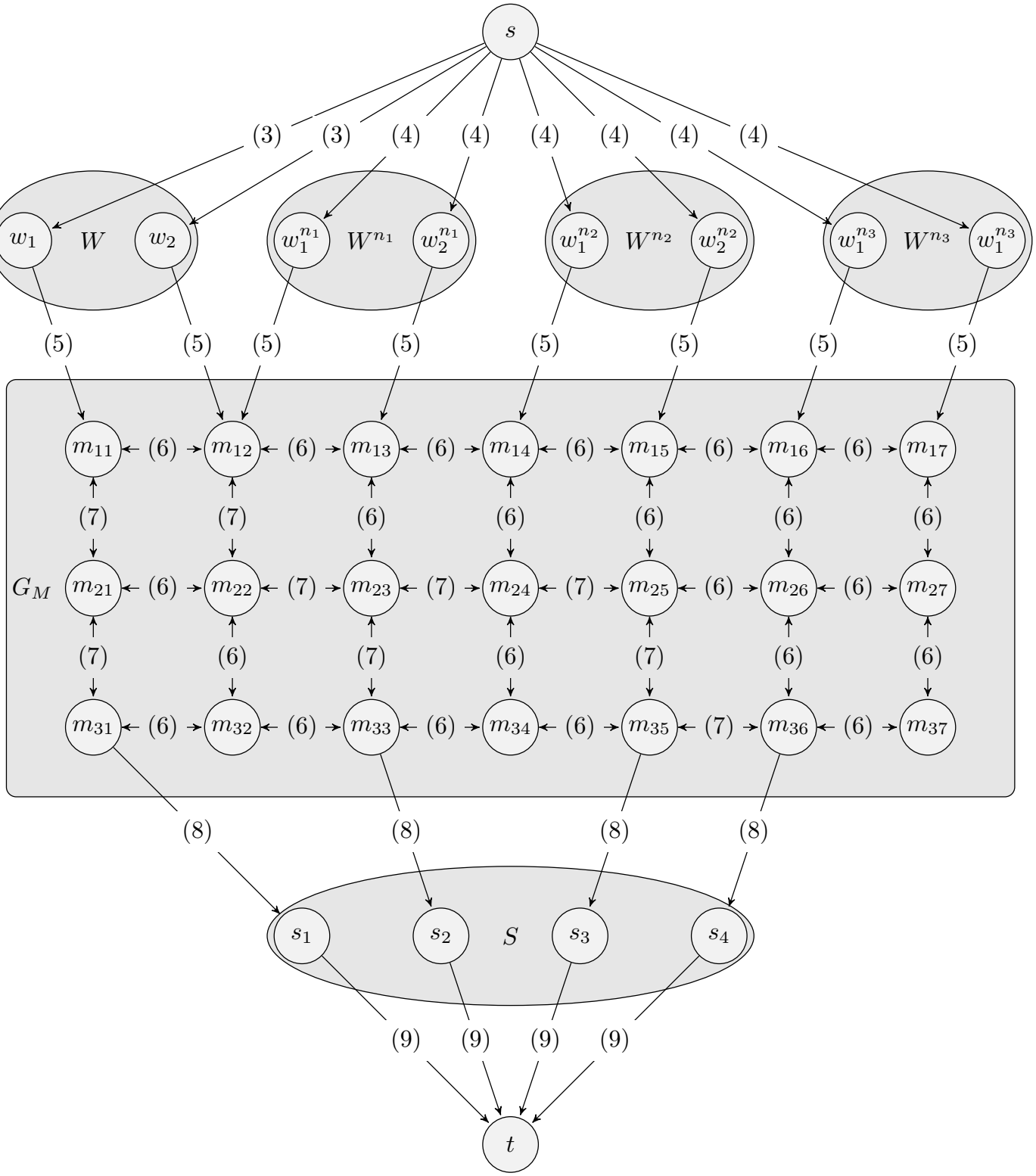


FIGURE 15 – Le graphe de flots $G = (V, E)$ de l'extension simple du réseau des pipelines.

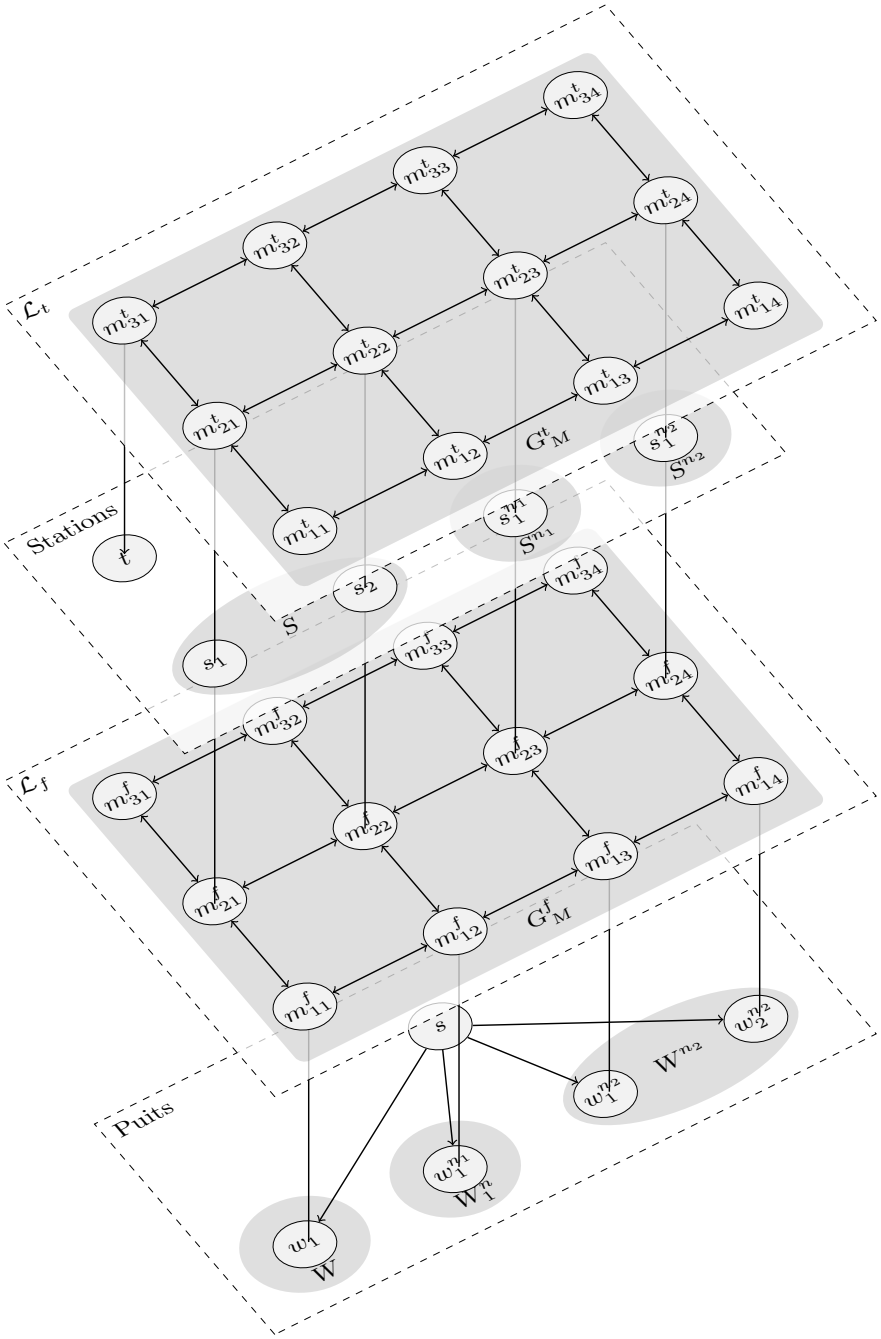


FIGURE 16 – Le graphe de flot permettant de construire des stations intermédiaires et des transfer lines.

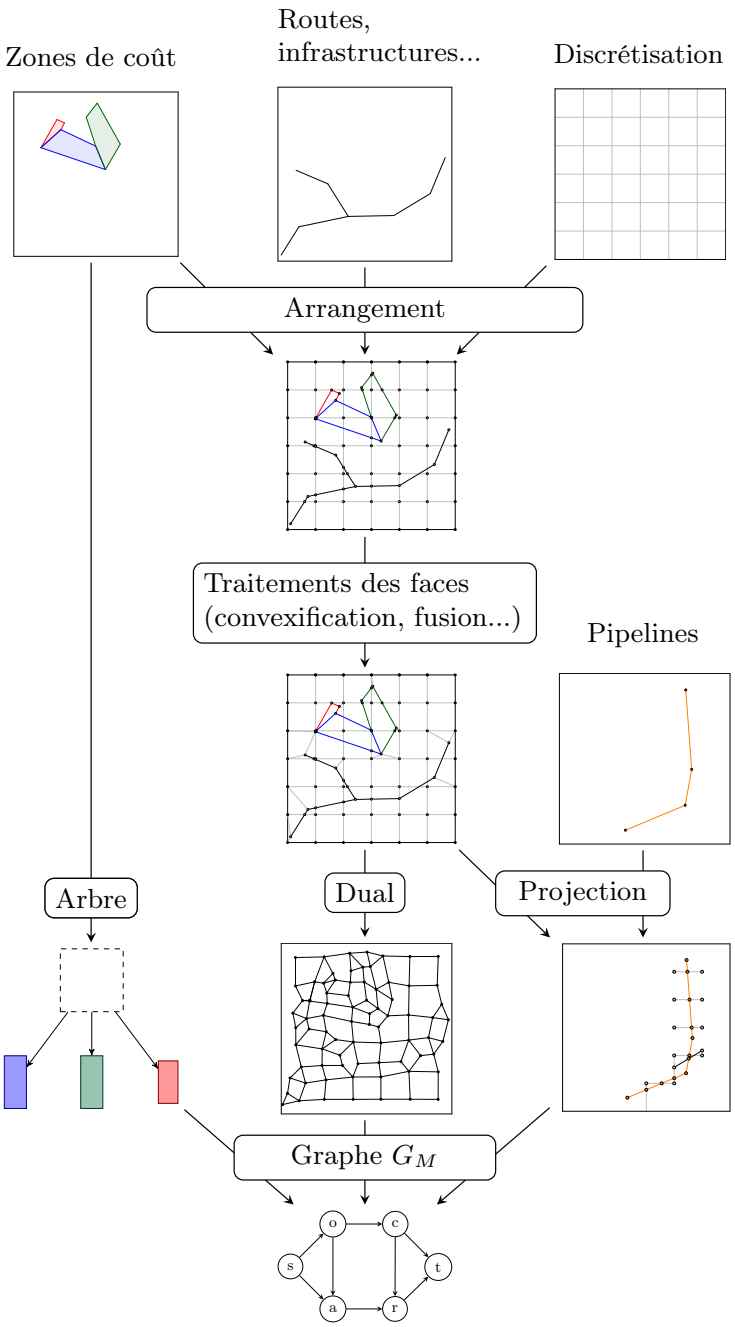


FIGURE 17 – Le processus de création du graphe G_M qui représente l'espace de travail.