Plus court chemin dans un graphe orienté avec longueur des arcs aléatoires

Edouard LONGUEVILLE

30 avril 2021 - 30 septembre 2021
Ce projet a été partiellement financé par la région Paris Ile-de-France Region via une subvention du DIM RFSI (projet EPINE).

Table des matières

0.1 Remerciements

Je remercie mon professeur et maître de stage Jean-Michel Fourneau de m'avoir proposé ce stage et pour ses suggestions durant le stage ainsi que le relecture de ce rapport.

Chapitre 1

Introduction

1.1 Cadre et objectif du stage

Le stage s'est déroulé dans le laboratoire DAVID de l'UFR des Sciences de l'université de Versailles Saint-Quentin en Yvelines avec pour encadrant Jean-Michel Fourneau. Dû aux conditions sanitaires le stage a été principalement en télétravail avec une réunion une fois par semaine pour faire un point. On m'a également fourni un bureau sur place.

Le but du stage était de poursuivre un travail de recherche existant sur le plus court chemin dans un graphe orienté avec longueur des arcs aléatoires. Ce problème étant un problème NP la première approche était de trouver des bornes inférieures et supérieures en utilisant des méthodes mathématiques. Ensuite, nous comparons les meilleures bornes obtenues avec les résultats exacts en utilisant des graphes série-parallèles qui ont la particularité de rendre un résultat exact calculable avec une complexité moindre.

Le travail effectué s'est divisé en deux parties. Tout d'abord la vérification des bornes obtenues avant mon arrivé en implémentant un programme permettant de calculer la valeur exacte pour un graphe SP et comparer cette valeur avec les bornes. La deuxième partie du travail commence avec un programme permettant de reconnaître si un graphe est SP et d'en retourner l'arbre correspondant. Ensuite, si le graphe n'est pas SP on applique différentes heuristiques pour le transformer en graphe SP et faire le calcul exact du graphe SP nous donnant une borne inférieure puis de comparer cette borne a celle obtenue précédemment.

Chapitre 2

État du projet

Cette partie résume tout le travail effectué avant mon arrivé au laboratoire dont la compréhension était nécessaire pour le stage.

2.1 Distance avec arc aléatoire discret

Le problème à résoudre est de trouver la plus petite distance dans un graphe orienté $G = (V, E)$ acyclique et, où à chaque arc (i, j) est associé un temps aléatoire $W(i, j)$ pour rejoindre i depuis j. On les noteras W_m où m est le nom de l'arc. On suppose que ces variables aléatoire sont discrètes et indépendantes. Ce graphe contient N sommets et M arcs. Dans ces sommets, on distinguera la source notée s. On veut pouvoir calculer la distance entre s et un sommet quelconque t de ce graphe. On représentera les distances sous la forme d'une distribution X_s où chaque atome a une probabilité associé et la somme des probabilités est égale à 1.

La difficulté du problème provient du fait que W(i,j) est aléatoire et que même si les distances des arcs sont indépendante les chemins ne le sont pas. Par conséquent, un calcul simple d'addition ou de minimum va demander un conditionnement. De plus, une convolution de deux distributions de taille S peut donner une distribution de taille S^2 ce qui pourrait augmenter fortement le nombre d'atomes de la distribution finale avec l'ajout d'un seul arc.

On a donc décidé de développer plusieurs algorithmes dans le but de trouver des bornes stochastiques sur la distribution de la distance. On ira donc chercher une borne stochastique forte (strong stochastic bound notée st) et une borne croissante concave (increasing concave bound notée icv). En se basant sur la

monotonie de la distance entre s et t notée $d(s,t)$, on propose deux algorithmes basé sur une réduction de la taille de $W(i,j)$.

2.2 Ordre Stochastique

Definition 1 (Ordre stochastique fort st) *Soit X et Y deux variables aléatoire, $X <_{st} Y$ si, pour toutes les fonctions croissantes Φ, on a $E[\phi(X)] \leq E[\phi(Y)]$.*

Une comparaison de deux variables implique également une stricte inégalité entre les variables tel que :

Property 1 *Soit X et Y deux variables aléatoires tel que $X <_{st} Y$. Si $E[X] = E[Y]$ alors, $X =_{st} Y$.*

On utilise également des ordres associés a la variabilité de certaines variables aléatoire pour obtenir de meilleures bornes.

Definition 2 (Ordre stochastique convexe) *Soit X et Y deux variables aléatoires, $X \preceq_{cx} Y$ si $E[X] = E[Y]$ et, pour toutes les fonctions convexe ϕ, on a : $E[\phi(X)] \leq E[\phi(Y)]$ si l'espérance existe.*

Nous utilisons en particulier l'ordre croissant concave qui est dérivé de l'ordre convexe.

Definition 3 (Ordre stochastique concave) *Soit X et Y deux variables aléatoire, $X \preceq_{cv} Y$ si $Y \preceq_{cx} X$.*

Definition 4 (Ordre stochastique croissant concave icv) *Soit X et Y deux variables aléatoire, $X \preceq_{icv} Y$ si, pour toutes les fonctions concave croissante ϕ, on a : $E[\phi(X)] \leq E[\phi(Y)]$.*

On veut la distance minimum pour un chemin entre s et t et la longueur d'un chemin et la somme des longueurs des arcs constituant ce chemin. Par conséquent, on calcule cette distance en utilisant les opérateurs "Min" et "+" qui sont tous les deux croissant et concave. On peut donc utiliser l'ordre icv et l'ordre st.

2.3 Approche monotone

On peut définir la monotonie de cette façon :

Definition 5 ($\Psi-$Monotonie) *Une fonction f est $\Psi-$monotone si pour toutes variables aléatoire X et Y telle que $X \preceq_\psi Y$, on a $f(X) \preceq_\psi f(Y)$.*

Dans notre cas, on peut donc déduire la propriété :

Property 2 *Si une fonction f est croissante, elle est $st-monotone$. De même, si une fonction f est croissante et concave alors elle est monotone pour l'ordre croissant concave icv.*

Cette propriété sur la monotonie nous permet de calculer les bornes suivantes.

2.3.1 Borne st monotone

Pour trouver les bornes st, on cherche d'abord a réduire la taille des distributions initiales.

Pour réduire ces distributions initiales, on les divise en K groupes de taille égale et on considère le plus grand atome(ou le plus petit selon la borne) de chaque groupe uniquement. Cela permet de réduire la taille d'une distribution à K quelque soit la distribution de départ.

Il suffit ensuite de calculer la distance $d(s,t)$ obtenue avec les nouvelles distributions pour obtenir une borne st.

2.3.2 Borne icv monotone

La démarche pour trouver les bornes icv est similaire a celle des bornes st. D'abord on cherche a réduire la taille des distributions. Pour réduire ces distributions, on fixe une taille K que nos distributions doivent atteindre.

Ensuite, si on cherche une borne supérieure on va réduire deux atomes par un nouvel atome ayant pour valeur la moyenne de ces deux atomes.

Pour une borne inférieure, on considère un groupe d'au moins 3 atomes dans la distribution et on les remplace par les deux atomes occupant les extrémités du groupe.

On répète ces étapes jusqu'à ce que le nombre d'atomes de la distributions soit inférieur ou égale à K.

Ensuite, on calcule la distance $d(s,t)$ avec les nouvelles distributions pour obtenir la borne icv.

2.4 Distance sur variable aléatoire associé

Dans notre graphe, les distances entre les arcs sont indépendantes mais les distances entre deux sommets non connectés ne le sont pas car les chemins peuvent passer par des arcs en commun. Il y a donc une dépendance positive entre les chemins. On veut utiliser cette dépendance pour établir des nouveaux algorithmes de borne.

2.4.1 Association de variable aléatoire

Definition 6 (PQD) *Deux v.a. X et Y sont Positively Quadrant Dependent (PQD) si $\forall x, y$:*

$$H(x, y) = Pr(X > x, Y > y) - Pr(X > x)Pr(Y > y) \geq 0$$

On considère un graphe orienté sans circuit $G = (V, E)$ avec des distances sur chaque arc $W(i, j)$ pour l'arc (i, j). Ces distances sont des variables aléatoires indépendantes dont les supports sont strictement positifs.

La longueur L d'un chemin du sommet s au sommet t est notée $P(t)$.

Lemme 2.1 *Soit Y, $Z1$ et $Z2$ trois v.a indépendantes (2 à 2), alors $Y+Z1$ et $Y+Z2$ sont associées.*

Property 3 *Posons $Y_P = L(P(t))$ une v.a. réelle donnant la durée du chemin $P(t)$ de s à t $Y = (Y_P)_{P \in \mathcal{P}(s,t)}$ est un vecteur de v.a. associées.*

Preuve : Soit $P1$ et $P2$ deux chemins de s à t. On a deux cas à examiner :

1. $P1$ et $P2$ sont arcs disjoints
2. L'intersection de $P1$ et $P2$ contient des arcs.

Dans le premier cas, $l(P1)$ et $l(P2)$ sont indépendantes puisqu'il s'agit de deux sommes de variables aléatoires indépendantes et distinctes. Dans le second cas, posons Q le sous-ensemble maximal d'arcs en commun dans $P1$ et $P2$ (ie l'intersection de $P1$ et $P2$ vu comme deux ensembles d'arcs). Comme la longueur des chemins est une propriété additive et commutative, on peut séparer chaque chemin entre deux ensembles d'arcs pas nécessairement incidents tel que

$$l(P1) = l(Q) + l(P1 \setminus Q) \;\; et \;\; l(P2) = l(Q) + l(P2 \setminus Q)$$

Comme $Q = P1 \cap P2$, on a $(P1 \setminus Q) \cap (P2 \setminus Q) = \emptyset$, donc $l(P1 \setminus Q)$ et $l(P2 \setminus Q)$ sont des variables indépendantes. De même $l(Q)$ est indépendantes de ces deux v.a. On peut donc appliquer le Lemme 2.1 pour en déduire que $l(P1)$ et $l(P2)$ sont associées.

2.4.2 Borne inférieure

On suppose que $d^k(t)$ est la v.a. distance de s à t à l'iteration k de l'algo. Au départ on pose pour tout t, $d^1(t) = W(s,t)$ si il existe un arc (s,t), sinon $d^1(t) = INFINIE$. On suppose que le graphe est un DAG et que les numéros de sommet sont ceux de l'ordre topologique. Donc le sommet source $s = 1$. L'étape k de l'algorithme consiste à modifier les distances pour prendre en compte tous les voisins du sommet $k-1$. A la fin de cette étape, la distance de s au sommet numéro k est connue (si on arrive à faire les calcul). Attention l'indice d'étape est aussi un indice de sommet.

On utilise l'association pour calculer la borne mais pour cela, il faut prouver que l'on calcule la borne a chaque étape de l'algorithme.

1. Init $l^1(u) = d^1(u)$ pour tout u voisin de $s = 1$.
2. Boucle sur k numéro de sommet (de 2 à N).
   (a) Pour tous les voisins u du sommet $k-1$
   (b) $l^k(u) = min\left(\overline{l^{k-1}(u)}, \overline{l^{k-1}(k-1) + W(k-1,u)}\right)$

Par définition $\overline{l^{k-1}(u)}$ a la même distribution et est indépendante. Et on sait calculer le min de deux distributions indépendantes.

Théorème 2.2 *Pour tout k et tout u, $l^k(u) <_{st} d^k(u)$*

On a donc une borne inférieure.

2.4.3 Borne supérieure

Pour calculer une borne supérieure, on utilise 2 propriétés :

— Quand deux chemins sont arcs disjoint, la longueur de ces chemins est indépendante et on sait les calculer.

— Si on considère seulement une partie des chemins, on calcule une borne supérieure st, si $\mathcal{A}(s,t) \subset \mathcal{P}(s,t)$, alors

$$d(s,t) = Min_{P \in \mathcal{P}(s,t)} L(P) <_{st} m(s,t) = Min_{P \in \mathcal{A}(s,t)} L(P).$$

On déroule donc l'algorithme :

1. Rechercher les chemins arcs disjoints. Pour cela adjoindre à chaque arc une capacité de 1 et rechercher par un algorithme classique, le flot maximal sous contrainte de capacité déterministe. Les chemins augmentants sont par construction arcs disjoints (le capacités sont 1) et l'algorithme retourne le nombre max de ces chemins et les chemins.

2. Calculer la distribution de la distance de chaque chemin par convolution des distances des arcs (les arcs sont indépendants)
3. Calculer la distribution du minimum des longueur de ces chemins (les longueurs sont indépendantes)

2.5 Application des algorithmes

2.5.1 Exemple 1

Comme premier exemple, on prend un graphe simple où on peut calculer rapidement la valeur exacte.

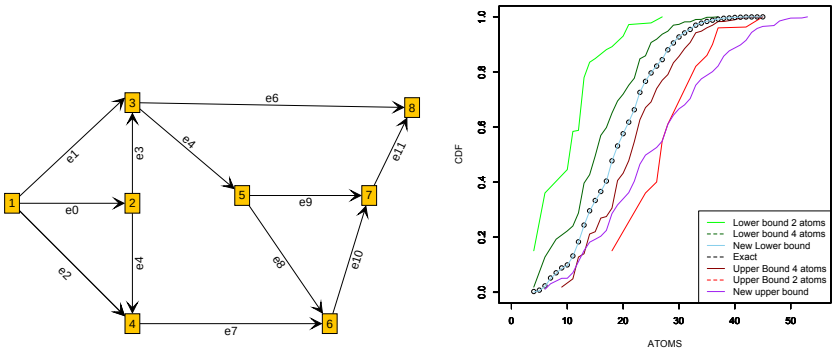

FIGURE 2.1 – Graphe pour le modèle 1 (gauche), bornes stochastiques (droite).

On remarque que sur toutes les bornes obtenues, la borne inférieure obtenue avec la deuxième méthode (la méthode de distance sur variable aléatoire associé) donne une très bonne borne.

Method		Association	St Monotonicity		"icv" Monotonicity		Fulkerson	Exact
			2 atoms	4 atoms	2 atoms	4 atoms		
Exp.	Lower bound	19.35	10.88	16.19	15.02	18.80	.	19.385
	Upper bound	26	28.29	21.99	21.23	20.83	23	.

TABLE 2.1 – Résultats attendu sur les distributions (Modèle 1).

2.5.2 Exemple 2

On prend ensuite un exemple de plus grande taille avec 26 arcs et 14 sommets où on ne peut pas calculer de résultat exact.

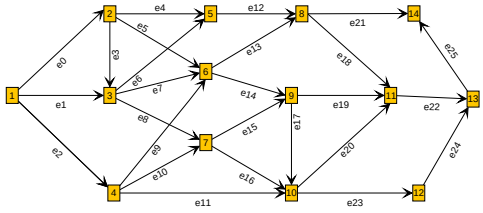

FIGURE 2.2 – Graphe pour le modèle 2.

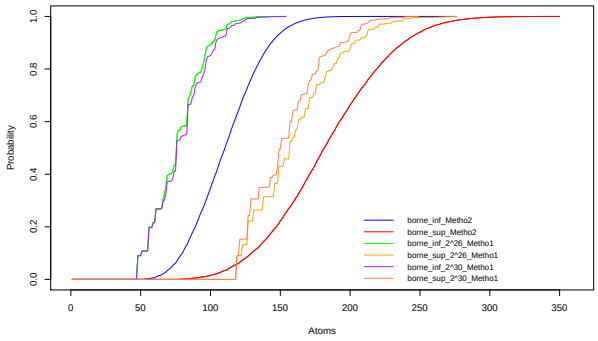

FIGURE 2.3 – Bornes stochastiques (Modèle 2).

On remarque que dans cet exemple, la borne inférieure de la méthode 2 se distingue fortement des autres bornes inférieures et pourrait rester une très bonne borne. Cependant nous ne pouvons pas vérifier cette borne avec un graphe classique.

Chapitre 3

Vérification des bornes

Le travail que j'ai effectué durant le stage commence avec cette partie.

3.1 Graphe Série-Parallèle

Pour pouvoir vérifier cette borne nous utilisons un type de graphe spécial appelé série-parallèle ou SP.

Definition 7 (Graphe Série-Parallèle) *Un graphe est un graphe série-parallèle ou SP, s'il peut être transformé en K_2, le graphe complet à deux sommets, par une séquence d'opérations suivantes :*

- *remplacement d'une paire d'arêtes parallèle par une seule arête qui relie leurs extrémités communes*
- *remplacement d'une paire d'arêtes incidentes à un sommet de degré 2 autre que s ou t par une seule arête*

Plus simplement,

- *une arête (i, j) est remplacé par deux arêtes (i, k) et (k, j) avec k un nouveau sommet (série)*
- *une arête (i, j) est dupliquée (parallèle)*

Ce type de graphe est construit de manière récursive ce qui nous permet de la calculer la distribution de la distance avec un algorithme récursif. Par contre, il est toujours soumis à l'explosion de la taille des distributions due aux opérations de convolution entre les distributions.

3.2 Qualité inf/sup

3.2.1 Algorithme utilisé

Étant donné que les graphes SP ont une structure récursive, il est alors possible de les écrire sous la forme d'un arbre.

On va donc représenter notre graphe sous la forme d'un arbre binaire où chaque feuille représentera un arc et chaque noeud une réduction en série ou parallèle. Pour calculer la valeur exacte de la distribution, on fait un calcul de convolution entre les deux feuilles qui ont pour père un noeud série et un calcul de min pour les feuilles qui auront pour père un noeud parallèle.

Pour calculer la valeur exacte de la distribution pour un graphe SP, nous avons utilisé l'algorithme suivant :

Algorithm 1 Résultat exact SP

```
Init pile p = parcourspostfixe(arbre)
while p n'est pas vide do
  n = pop(p)
  if n est une feuille then
    push(p2, n)
  end if
  if n est un noeud then
    if n = SERIE then
      result = convolution(pop(p2), pop(p2))
      push(p2, result)
    end if
    if n = PARALLELE then
      result = minimum(pop(p2), pop(p2))
      push(p2, result)
    end if
  end if
end while
result = pop(p2)
return result
```

3.2.2 Résultats

Pour cette partie, étant donné du grand nombre d'arc que nous avons, nous allons seulement calculer les bornes obtenues à l'aide de la méthode d'association.

On va prendre 2 graphe SP comme exemple. Tout d'abord un graphe de taille moyenne avec 16 sommets et 19 arcs. Ensuite un graphe de taille plus conséquente avec 48 sommets et 54 arcs. Ces graphes ont tous des arêtes de même distributions. C'est à dire 8 atomes (1,2,5,6,12,19,28,34) et une probabilité de (0.1,0.1,0.05,03,0.05,0.1,0.2,0.1).

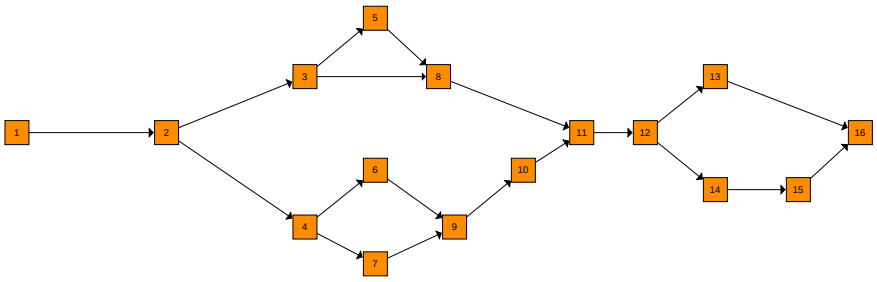

FIGURE 3.1 – Graphe 1

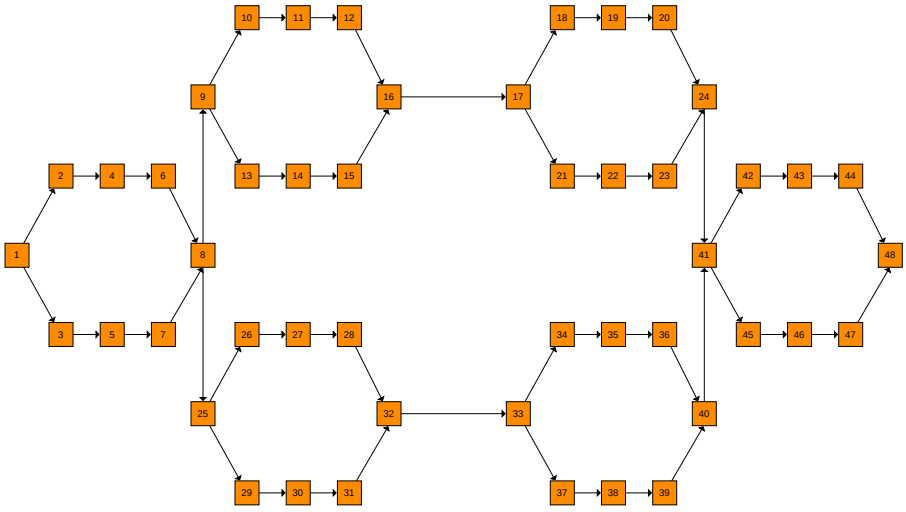

FIGURE 3.2 – Graphe 2

On s'intéresse d'abord au graphe 1. En plus d'être un graphe série-parallèle, ce graphe a pour particularité de n'avoir qu'un seul chemin arc disjoint.

Au niveau des temps d'exécution, l'algorithme donnant la valeur exacte s'exécute en 1.4 secondes contre 0.02 et 0.2 pour respectivement la borne supérieure et la borne inférieure. Le temps d'exécution plus élevé de la méthode exacte est dû à l'explosion de la taille des distributions. Cependant, ce temps d'exécution reste négligeable comparé a la méthode exacte dans le cas d'un graphe non SP qui prenait plus de 2 heures pour un graphe à 8 sommets et 11 arêtes.

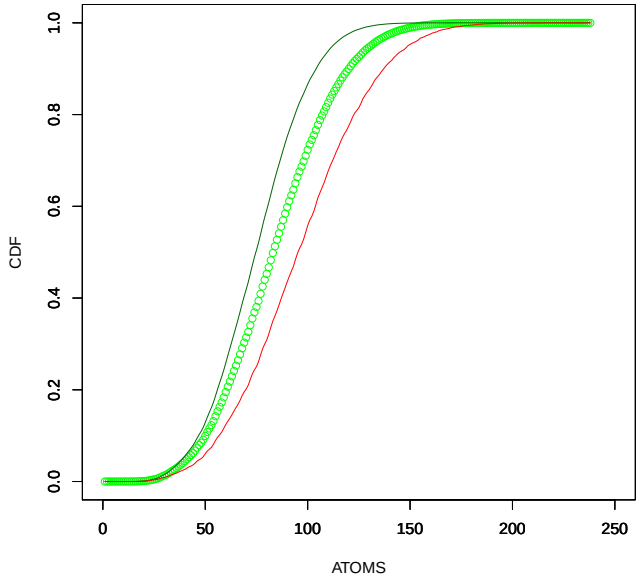

FIGURE 3.3 – Bornes st et valeurs exacte du graphe 1

Sur ce graphique, nous avons en vert la courbe représentant la borne inférieure, en vert clair la valeur exacte et en rouge la courbe de la borne supérieure.

Nous remarquons que la borne inférieure n'est pas aussi bonne que dans les autres exemple sur des tous petits graphe. Elle reste cependant meilleure que la borne supérieure sur cet exemple.

On regarde ensuite les résultats obtenus pour le graphe 2 qui a une taille plus conséquente.

Au niveau du temps d'exécution, on trouve un rapport similaire a ceux du graphe 1 avec 1m6s pour le calcul exact contre 1.7 secondes et 3.8 secondes respectivement pour la borne inférieure et la borne supérieure.

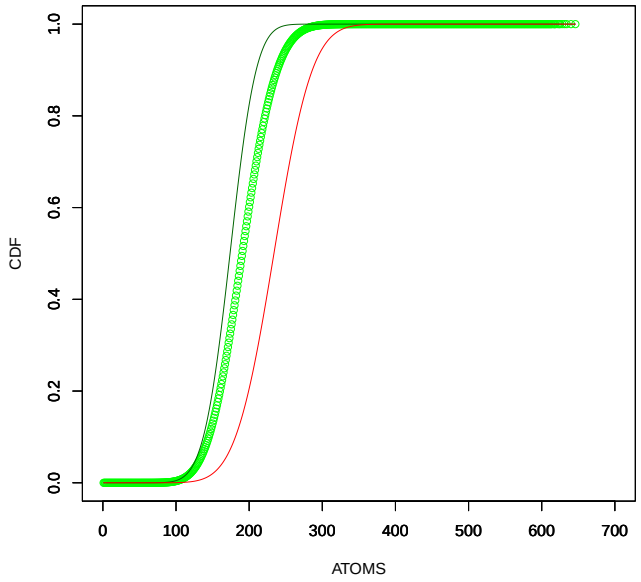

FIGURE 3.4 – Bornes st et valeurs exacte du graphe 2

Sur ce graphique, on remarque que la borne inférieure n'est toujours pas aussi proche du résultat exact que dans un petit graphe elle est plutôt précise. En revanche, la borne supérieure est très mauvaise sur cet exemple.

3.2.3 Conclusion

Avec ces résultats, on peut conclure que la borne inférieure obtenue grâce a la deuxième méthode est plus éloigné de la valeur exacte que ce que laissait imaginer les exemples sur des petits graphes mais reste une borne précise. La borne supérieure en revanche est mauvaise et reste trop éloigné de la valeur exacte et dépend probablement des propriétés du graphe.

Chapitre 4

Reconnaissance SP

4.1 Théorème sur les bornes st

4.1.1 Théorème

Théorème 4.1 *Soit G un graphe et G' un sous graphe de G, on a alors $d_{G'} >_{st} d_G$.*

Par conséquent, ce théorème nous permet de calculer une borne supérieure du graphe G en trouvant un sous graphe G' pour lequel on peut calculer la valeur exacte.

4.1.2 Graphe SP

Le but va donc être pour un graphe non-SP G, de trouver un sous graphe G' SP pour pouvoir calculer une nouvelle borne. Or, trouver un sous-graphe série-parallèle avec un maximum d'arête d'un graphe G est un problème NP difficile.

On va donc devoir faire plusieurs étape pour trouver un sous graphe SP :

- Utiliser un algorithme de reconnaissance pour savoir si G est SP ou non
- Avoir un algorithme permettant de construire l'arbre représentant G s'il est SP
- Trouver une heuristique donnant un sous graphe SP de G si G n'est pas SP

4.2 Reconnaissance du graphe SP et construction de l'arbre

4.2.1 Algorithme de reconnaissance

La première étape est de savoir si un graphe G est SP ou non. Pour savoir si un graphe est SP, il suffit de faire des réductions en série ou en parallèle sur ce graphe jusqu'à ce que l'on ne puisse plus en faire. Si le graphe résultant est le graphe complet K^2 alors G est un graphe SP.

Algorithm 2 transformSerie

Input : Graphe G, Sommet S1, Sommet S2
Output : Graphe G

Supprime arête $(S1, S2)$ de G
Cherche successeur $S3$ de $S2$
Ajoute arête $(S1, S3)$ à G
return G

Algorithm 3 transformPara

Input : Graphe G, Sommet S1, Sommet S2
Output : Graphe G

Supprime une arête $(S1, S2)$ de G
return G

Algorithm 4 Reconnaissance SP

```
while G est réductible en série ou parallèle do
  for all Sommet S1 de G do
    for all Voisins S2 de S1 do
      if (S1, S2) vérifie les conditions de réduction en série then
        G = transformSerie(G, S1, S2)
      end if
      if (S1, S2) vérifie les conditions de réduction parallèle then
        G = transformParallel(G, S1, S2)
      end if
    end for
  end for
end while
if G est constitué d'un seul arc then
  return G est SP
else
  return G n'est pas SP
end if
```

Au niveau de la complexité de cet algorithme, il faut distinguer deux cas :
- Le cas où G est SP
- Le cas où G n'est pas SP

Pour ces deux cas, on considère G un graphe à S sommets et K arcs.

Si G est SP, la boucle $while$ s'arrêtera lorsqu'il y aura eu $K-1$ réductions et fera donc dans le pire cas $K-1$ itérations. Dans la boucle $while$, on doit parcourir tous les sommets S de G ainsi que tous les voisins de chaque sommet. Or, dans le cas d'un graphe SP, un sommet a rarement un grand nombre de voisins. On peut donc considérer que le parcours des voisins d'un sommet est négligeable par rapport au parcours de S sommets quand S est grand. De même, les fonctions $transformSerie$ et $transformParallel$ ont un coût négligeable.

Dans le cas où G est SP, on arrive donc à une complexité de $O((K-1)*S)$.

Si G n'est pas SP, la boucle $while$ prendra seulement $K-2$ itérations dans le pire cas. Pour la première boucle $forall$, on a également S itérations. En revanche, pour le parcours des voisins dans un graphe non SP on aura un pire cas où le nombre d'itération sera de $S-1$.

Dans le cas où G n'est pas SP, on aura donc une complexité théorique de $O((K-2)*S*(S-1))$.

En revanche, cette complexité reste théorique puisque si le nombre de voisin de tous les sommets est de $S-1$, G est alors un graphe complet et fera une seule itération de la boucle $while$ car on ne peut pas faire de réduction en série ou parallèle sur un graphe complet.

On notera que théoriquement, la reconnaissance d'un graphe SP peut-être effectué en temps linéaire mais qu'il n'était pas utile d'avoir la meilleure complexité vu que nous ne pouvons pas travailler sur des très gros graphe en raison de l'explosion de la taille des distributions avec notre méthode de calcul exacte.

4.2.2 Construction de l'arbre

Pour la construction de l'arbre, la complexité théorique est également linéaire mais demande une implémentation précise. On utilise donc ici une méthode un peu spéciale pour construire l'arbre.

Cette méthode consiste a créer pour chaque réduction SP un noeud de l'arbre composé d'un père (représentant la réduction), deux feuilles(représentant les arcs supprimés) et un label donnant le nom du nouvel arc crée par la réduction.

Ainsi, lors d'une réduction en série entre les arcs $(3,5)$, $(5,8)$ donnant un nouvel arc $(3,8)$, on créera le noeud :

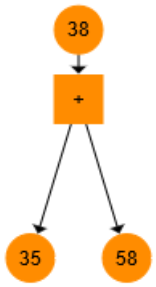

FIGURE 4.1 – Noeud individuel crée par la reconnaissance

Après avoir crée tous les noeuds, on parcours la liste de noeuds et on assemble l'arbre. Pour assembler l'arbre, il suffit de remplacer les fils par le noeud qui porte le nom de la distribution. Pour reprendre l'exemple plus haut, si on a un autre noeud tel que les arcs $(3, 8)$ et $(3, 8)$ forment une réduction parallèle, alors on aura un nouveau noeud :

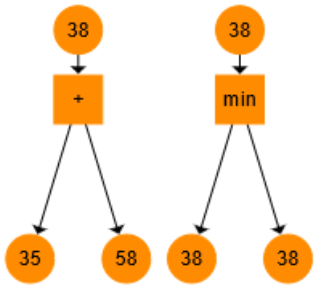

FIGURE 4.2 – Noeuds individuels crées par la reconnaissance

On remarque que l'on peut combiner ces noeuds vu que le label du noeud de gauche est 38 et que le noeud de droite a 38 pour feuille.

On va donc les combiner pour créer l'arbre :

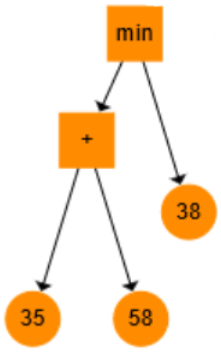

FIGURE 4.3 – Arbre crée a partir des noeuds individuels

De façon plus formelle, les algorithmes utilisés pour construire un arbre sont les suivants :

Algorithm 5 Création noeuds série

Input : Liste noeuds L_n, Sommet S1, Sommet S2, Sommet S3
Output : Liste noeuds L_n

Nouveau noeud N
N.filsdroit = $(S1, S2)$
N.filsgauche = $(S2, S3)$
N.type = SERIE
N.label = $(S1, S3)$
On ajoute N à L_n
return L_n

Algorithm 6 Création noeuds parallèle

Input : Liste noeuds L_n, Sommet S1, Sommet S2
Output : Liste noeuds L_n

Nouveau noeud N
N.filsdroit = $(S1, S2)$
N.filsgauche = $(S1, S2)$
N.type = PARALLELE
N.label = $(S1, S2)$
On ajoute N à L_n
return L_n

Algorithm 7 Construction arbre

```
Input : Liste noeuds L_n
for all Noeud N de L_n do
  for all Noeud N2 de L_n do
    if N! = N2 then
      if N.filsdroit == N2.label then
        N.filsdroit = N2
      end if
      if N.filsgauche == N2.label then
        N.filsgauche = N2
      end if
    end if
  end for
end for
```

En terme de complexité, ici les algorithmes de constructions de noeud ont un coût négligeable. Vu que chaque noeud correspond a une réduction, alors sur un graphe G sp à K arcs, l'algorithme de construction de l'arbre aura une complexité de $O((K-1)^2)$.

4.3 Heuristiques

Pour revenir au problème initial, on cherche un sous graphe G' de G tel que G non SP et G' est SP. Etant donné que c'est un problème NP-difficile, nous allons utiliser différentes heuristiques pour essayer de trouver les meilleures bornes possible. Les graphes G utilisés seront tous acycliques et contiendront une source et un puits.

4.3.1 Heuristiques utilisées

On va utiliser 3 heuristiques différentes pour transformer un graphe non-SP en graphe SP. Le concept sera d'appliqué l'algorithme de reconnaissance sur le graphe G jusqu'à ce que l'algorithme se bloque puis d'appliquer les heuristiques sur ce graphe pour supprimer un arc jusqu'à ce que l'algorithme de reconnaissance trouve un graphe SP. Ces 3 heuristiques ont une base commune qui consiste a calculer une valeur pour chaque sommet en fonction des arcs entrant et sortant de ce sommet.

Étant donné que les heuristiques cherchent a faciliter la réduction en série (les réductions parallèles se faisant naturellement une fois que les réduction en série sont terminé ce n'est pas nécessaire de chercher a les faciliter) le facteur qui nous importe le plus est le nombre d'arc entrant et sortant d'un sommet. En effet, une réduction en série est possible uniquement si le sommet que l'on "supprime" a un unique arc entrant et un unique arc sortant.

Vu que l'on travaille sur des graphes acyclique, un cas de figure qu'il faut absolument éviter est la création d'un sommet qui a soit un degré entrant de 0 soit un degré sortant de 0 (à l'exception de la source et du puits). Ces sommets soit ne menant plus au puits soit n'étant plus connecté à la source devront être supprimé entraînant potentiellement de nouveaux sommets avec des degrés nuls et potentiellement une suppression de chemins entier.

Par conséquent, une solution pour évaluer les sommets est d'assigner a chaque sommet la valeur obtenue en multipliant leur degré entrant par leur degré sortant et de regarder en premier les sommets avec les degrés les plus haut. Cela nous permet d'éviter au maximum les sommets avec un degré entrant ou sortant faible pouvant entraîner un sommet avec un degré a 0 après la suppression de l'arc.

La première heuristique est la plus simple. Elle est composé de plusieurs étapes principales :

- On calcule les valeurs des sommets
- On choisit le sommet avec la plus grande valeur
- On choisit le voisin de ce sommet avec la plus grande valeur de manière à ce que l'arc reliant le voisin et ce sommet puisse être supprimé sans créer de sommet ayant un degré nul
- On supprimer l'arc reliant ces deux sommets

Dans le cas où plusieurs sommets ont les même valeurs, on sélectionne le meilleur sommet en fonction des valeurs de ses voisins. C'est à dire que on va prendre le sommet dont la somme des valeurs de ses voisins est la plus grande. Si on a toujours des valeurs égales, alors on prend le sommet avec le plus petit label.

Pour la deuxième heuristique, nous voulons être sûr de ne pas détruire d'arc "contenant" une réduction. En effet, vu que entre exécution de l'heuristique nous appliquons l'algorithme de reconnaissance sur le graphe pour effectué des réductions, le meilleur arc a supprimer pourrait être un arc contenant un grand nombre de réduction et pourrait au lieu de supprimer un seul arc du graphe supprimer plusieurs arcs du graphe original. Nous gardons la même manière de calculer les valeurs des sommets mais nous prenons en priorité les arcs du graphe original, c'est à dire qui n'ont pas subis de réduction.

On applique donc les méthodes suivantes :

— On calcule les valeurs des sommets
— On choisit le sommet avec la plus grande valeur
— On choisit un arc relié à ce sommet qui a subit le minimum de réduction et qui ne déconnecte pas le graphe
— On supprime cet arc

Dans le cas où on a des sommets avec une valeur égale, on applique la même méthode que l'heuristique 1. Pour des arcs avec le même nombre de réduction, on choisis l'arc qui connecte le sommet avec la valeur maximale, en cas de même valeur maximale on applique la même méthode que l'heuristique 1.

La troisième heuristique est similaire a la deuxième heuristique, elle se concentre sur éviter de détruire des arcs que l'on a déjà réduit. Elle garde comme premier critère de sélection d'un sommet le même calcul de valeur que les autres heuristique mais elle rajoute un critère de sélection. Il faut que le sommet sélectionné soit en contact avec un minimum d'arc ayant été réduit.

On applique les méthodes suivantes :

— On calcule la valeur des sommets
— On sélectionne un sommet avec la plus grande valeur et le moins d'arc réduit
— On choisit un arc relié à ce sommet qui a subit le minimum de réduction et qui ne déconnecte pas le graphe
— On supprime cet arc

Même si cette heuristique paraît presque identique à la deuxième heuristique, les changements de critères de sélection du premier sommet changent fortement les résultats obtenus.

4.4 Exemples

On teste ces heuristiques avec 3 exemples de graphes.

4.4.1 Exemple 1

Ce graphe d'exemple est le graphe SP que l'on utilise pour comparé les bornes inf et sup dans la deuxième partie mais légèrement modifié pour le rendre non SP.

Modifier légèrement un graphe pour le rendre non SP possède deux avantages. Tout d'abord ça nous permet de voir rapidement quels arcs on devrait supprimer pour avoir le meilleur graphe SP possible et donc de juger facilement avec un support visuel de la qualité de notre heuristique. Ensuite, vu que notre algorithme exact calcule une distribution minimum, si on donne a notre graphe des délais très grand sur les arcs que l'on rajoute, le résultat exacte devrait être identique à celui du graphe SP ce qui nous permets de comparer nos heuristiques non seulement a la borne supérieure obtenue précédemment mais également à la valeur exacte.

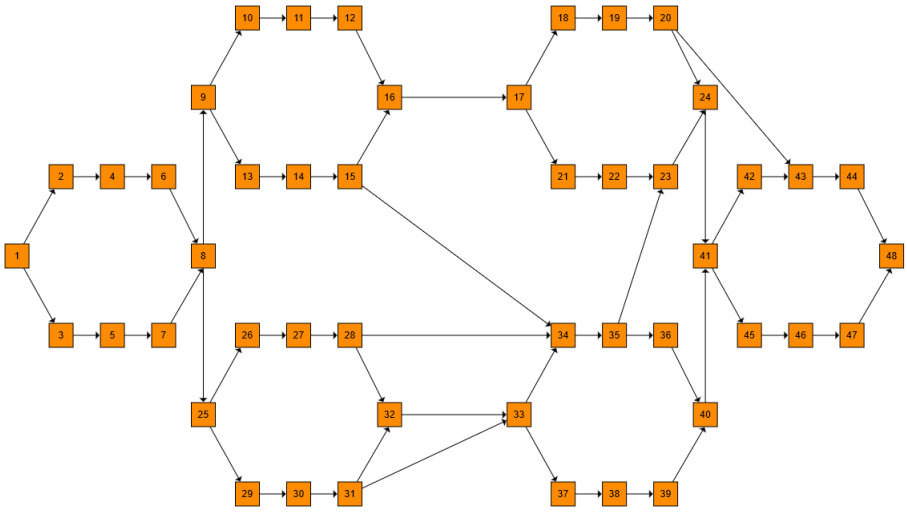

FIGURE 4.4 – Exemple 1

En partant de ce graphe et en appliquant les heuristiques, on obtient les graphes suivant :

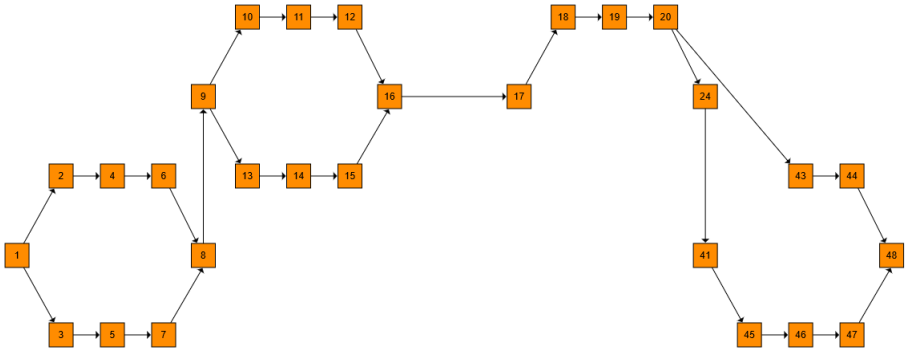

FIGURE 4.5 – Heuristique 1

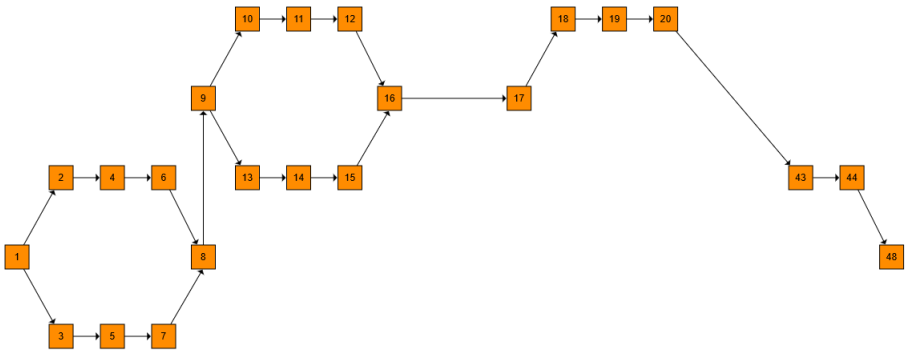

FIGURE 4.6 – Heuristique 2

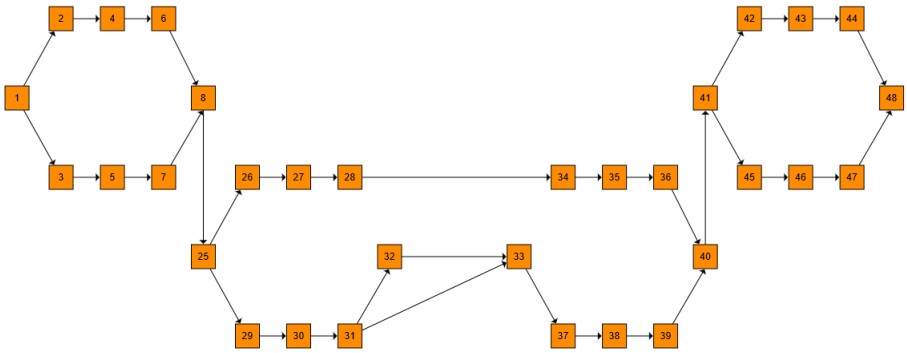

FIGURE 4.7 – Heuristique 3

On remarque tout de suite que l'heuristique 3 a mieux conservé le graphe original que les deux autres. En revanche, les trois heuristiques ont une grande partie du graphe originale qui a été supprimé dû en grande partie a des suppressions d'arc qui contenait des opérations de réduction et de sommets déconnecté du graphe.

Par conséquent, nous pouvons supposer que les bornes que vont donner ces graphes seront très mauvaise.

On compare donc les moyennes des distributions obtenue avec ces heuristiques par rapport à celle obtenue par l'ancien calcul de borne supérieure et la valeur exacte.

— La valeur exacte est 191.663645880637
— La moyenne de l'heuristique 1 est 236.627171688437
— La moyenne de l'heuristique 2 est 100000180.227171778679
— La moyenne de l'heuristique 3 est 236.627171688438
— La moyenne de la borne supérieure distance association est 194.448800855432

On voit donc que ces heuristiques donnent un très mauvais résultat de borne pour ce graphe. La borne de l'heuristique 2 étant mauvaise a ce point car le graphe résultant force le passage par une arête qui a été rajouté et qui par conséquent a une très grande distribution.

Les heuristiques 1 et 3 donnent un résultat identique en raison des spécificités

des poids des distributions dans cet exemple et du graphe utilisé. En effet, toutes les distributions sont identique à l'exception de celle des arcs rajouté qui elles sont très grande. Et, si on reconstruit ces deux graphes en enlevant les chemins qui passent forcément par un arc rajouté, on se retrouve au final avec deux graphes identiques.

4.4.2 Exemple 2

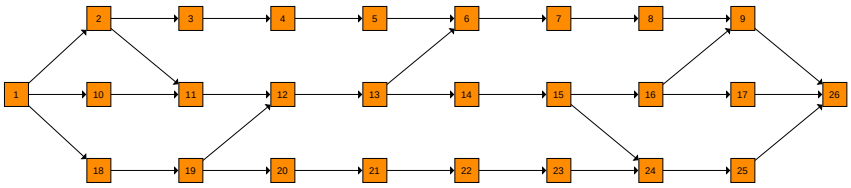

FIGURE 4.8 – Exemple 2

Cet exemple est encore un graphe SP modifié. Le graphe SP lui même était particulier vu qu'il était constitué de 3 chemins arcs-disjoints uniquement. La particularité de ce graphe vient du fait qu'il a originellement des degrés très faibles sur ses sommets et qu'il y a seulement certains arcs que l'on peut enlever sans le déconnecter.

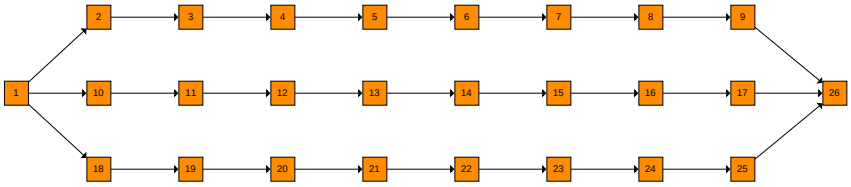

FIGURE 4.9 – Graphe après les heuristiques

Pour ce graphe, les trois heuristiques ont trouvé sans soucis les arcs rajouté et donc le meilleur graphe possible. La difficulté de pouvoir enlever des arêtes à ce graphe sans le déconnecter à rendu les 3 heuristiques très efficace vu que c'est la condition principale lorsque on décide de supprimer un arc. Le calcul de la borne supérieure nous donne également un résultat exact car les arcs ajoutés ont un poids très grand et seront donc ignorés. Le reste du graphe étant arc disjoint on tombe donc sur un résultat exact.

Au niveau des moyennes, on obtient donc la valeur exacte pour les 3 heuristiques et la borne supérieure : 124.650000000000

4.4.3 Exemple 3

Comme pour les deux autre exemple, on prend un graphe SP et on le modifie pour le rendre non SP.

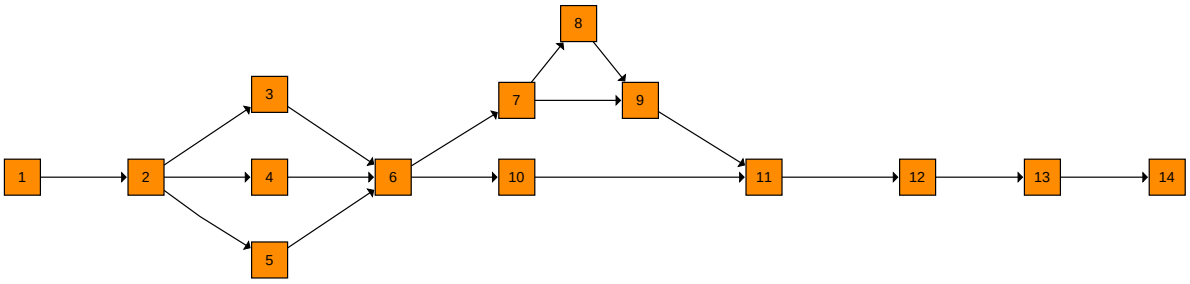

FIGURE 4.10 – Graphe SP avant modification

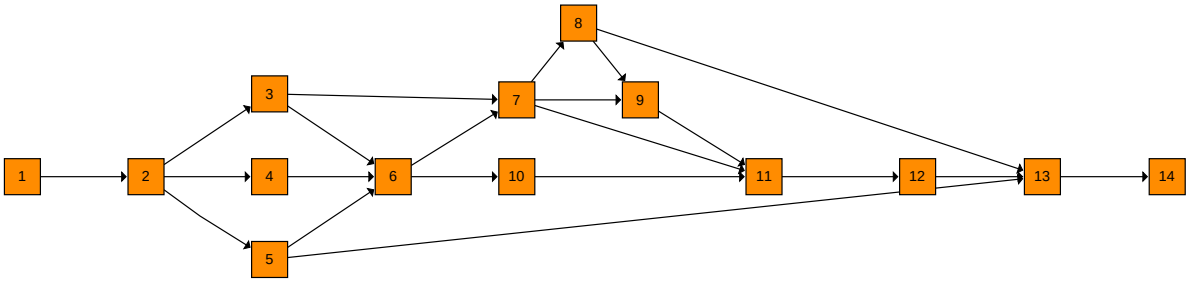

FIGURE 4.11 – Exemple 3

Cet exemple mélange les deux premiers exemples. On a d'un coté des arcs qui risquent d'être facile a identifier par nos heuristiques et de l'autre un groupe de sommet a fort degré ne risquant pas de déconnecter le graphe.

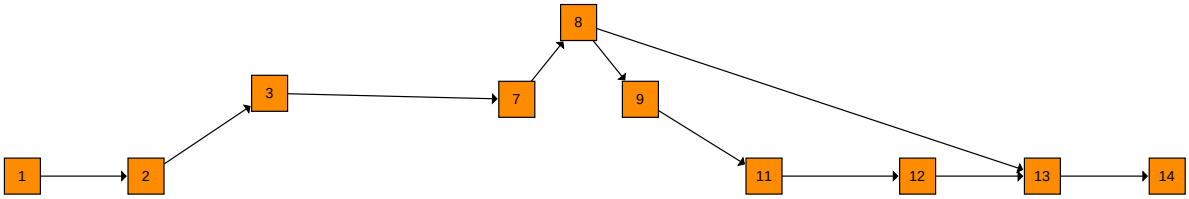

FIGURE 4.12 – Heuristique 1

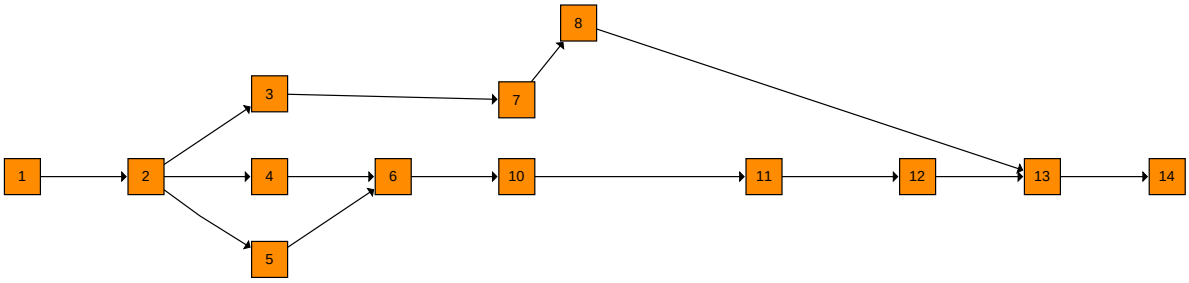

FIGURE 4.13 – Heuristique 2

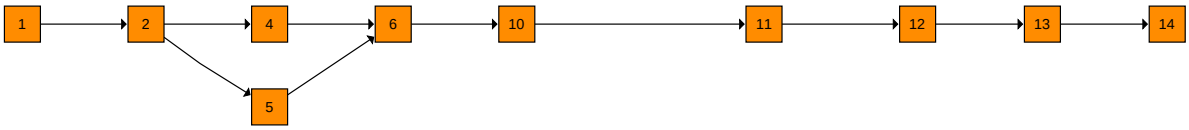

FIGURE 4.14 – Heuristique 3

On obtient les moyennes suivantes pour les distributions :

— Résultat exacte : 95.224254251563
— Heuristique 1 : 10000109.800000008196
— Heuristique 2 : 101.455137500000
— Heuristique 3 : 101.455137500000
— Borne supérieure distance associé : 103.747909791660

On remarque que comme dans l'exemple 1, la propriété des forts poids des arcs ajouté font que l'heuristique 2 et 3 donnent la même valeur. De même l'heuristique 1 donne maintenant une valeur très éloigné du résultat exact. En revanche, pour ce type de graphe les heuristique 2 et 3 donnent une borne plutôt précise et meilleure que celle que l'on trouve grâce à la méthode des distances sur variable aléatoire associé.

Chapitre 5

Conclusion

Pour conclure, j'ai effectué mon stage de master 2 au sein du laboratoire DAVID de l'UVSQ sur une problématique lié aux graphes. Les points clés du stage ont été la compréhension du travail effectué avant que j'arrive sur le projet, puis la vérification des bornes trouvé et enfin la mise en place d'une nouvelle méthode pour trouvé des bornes, en l'occurrence des heuristiques pour trouver un sous graphe SP.

La vérification des bornes a donné des résultats satisfaisant même si les données pré vérification tendaient vers une borne inférieure bien meilleure.
La reconnaissance d'un graphe SP et la construction de l'arbre associé on été une nécessité plutôt qu'un objectif mais on demandé un travail conséquent pour être réalisé.
L'utilisation d'heuristiques pour trouver un sous graphe SP a apporté des résultats satisfaisant même si les heuristiques trouvé sont sans doute améliorable pour avoir des résultats encore meilleurs.
Au niveau des pistes d'amélioration, il reste encore plusieurs méthode a tester pour trouver de meilleures bornes. Par exemple on pourrait supprimer des distributions toutes les valeurs plus grande que la plus grande valeur d'un plus court chemin pour réduire la taille des distributions et donc permettre des algorithmes de calcul exact plus performant.

Chapitre 6

Bibliographie

1. Valdes, Jacobo ; Tarjan, Robert E. ; Lawler, Eugene L. The recognition of series parallel digraphs. SIAM J. Comput. 11 (1982), no. 2, 298–313.
2. David Eppstein. Parallel Recognition of Series-Parallel Graphs. October 4, 1990
3. Hans L. Bodlaender and Babette de Fluiter. Parallel algorithms for series parallel graphs. 1996