

ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet Recherche & Développement
2016-2017

**Visualisation des procédures de
branchement**

Tuteur académique
Lei SHANG

Étudiant
Xiangyu MENG (DI5)

9 avril 2017



Liste des intervenants

Nom	Email	Qualité
Xiangyu MENG	xiangyu.meng@etu.univ-tours.fr	Étudiant DI5
Lei SHANG	lei.shang@univ-tours.fr	Tuteur académique, Département Informatique



Avertissement

Ce document a été rédigé par Xiangyu MENG susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Lei SHANG susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnaît assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



Pour citer ce document

Xiangyu MENG, *Visualisation des procédures de branchement*: , Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2016-2017.

```
@mastersthesis{
  author={MENG, Xiangyu},
  title={Visualisation des procédures de branchement: },
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2016-2017}
}
```


Table des matières

Liste des intervenants	a
Avertissement	c
Pour citer ce document	e
Table des matières	i
Table des figures	v
1 Introduction	1
1 Contexte	1
2 Objectifs	1
3 Bases méthodologiques	2
2 Description générale	3
1 Environnement du projet	3
2 Caractéristiques des utilisateurs	3
3 Fonctionnalités du système	4
4 Structure générale du système	5
3 Etat de l'art	7
1 Introduction.....	7
2 Synthèse de l'état de l'art.....	7
2.1 La communication avec le programme maître.....	8
2.2 Les informations du nœud	9
2.3 La disposition de l'arbre	9
3 Limites et Avantages de l'existant	10

4	Analyse et conception	13
1	Éléments principaux de l'analyse.....	13
1.1	La communication avec le programme maître.....	13
1.2	Les informations de nœud.....	14
1.2.1	L'état de nœud	14
1.2.2	La solution partielle.....	14
1.2.3	La borne inférieure	14
1.2.4	Le temps de calcul lié	14
1.3	La disposition de l'arbre	14
2	Choix de technique.....	15
3	Limites et conséquences de la solution	16
5	Mise en oeuvre	17
1	Les outils et bibliothèques utilisées.....	17
2	Architecture du système.....	17
2.1	Architecture générale	17
2.2	Architecture détaillée	18
3	Interface de l'application.....	19
4	Implémentation	20
4.1	Importation du fichier	20
4.2	Format de fichier	20
4.3	Entrée des paramètres de visualisation.....	21
4.4	Disposition de l'arbre	21
4.5	Mode statique	22
4.6	Mode dynamique	23
4.7	Coloration de nœud.....	23
4.8	Coloration d'arrêt	24
4.9	Affichage du label de nœud	24
4.10	Affichage du label d'arrêt	24
4.11	Replier et dérouler les nœuds	25
4.12	Zoom	26
4.13	Contrôle de vitesse	26
4.14	Pause	27
4.15	Redémarrer	27
4.16	Stop	27
5	Limite.....	27
5.1	Limite de la visualisation statique	27
5.2	Limite de la visualisation dynamique	27

6	Bilan	29
1	Conclution.....	29
2	Plan de développement	29
2.1	Tâches.....	29
2.2	Diagramme de Gantt	30
	Bibliographie	33
	Annexes	35
A	Description des interfaces externes du système	37
1	Interfaces matériel/logiciel	37
2	Interfaces homme/machine	37
2.1	Interface visiteur.....	37
2.2	Interface utilisateur	38
3	Interface logiciel/logiciel.....	38
B	Spécifications fonctionnelles	39
1	Enregistrement	39
2	Authentification	39
3	Importation de fichier	39
4	Coloration de nœud.....	40
5	Affichage de la solution partielle	40
6	Affichage de la borne inférieure	40
7	Affichage du temps calcul lié.....	40
8	Choix de mode d'exécution	40
9	Visualisation statique	40
10	Visualisation dynamique.....	41
11	Contrôle de vitesse	41
12	Pause et redémarrage.....	41
13	Replier et dérouler les nœuds	41
14	Zoom	41
15	Sauvegarde de fichier	41
16	Consultation l'historique de fichier	41
C	Spécifications non fonctionnelles	43
1	Contraintes de développement	43
2	Contraintes de fonctionnement.....	43
2.1	Performance	43
2.2	Capacité	43
2.3	Sécurité	44

D	Cahier du développeur	45
1	Utilisation des librairies	45
2	Environnement de développement	45
3	Versionning	45
4	Code documentation	45
E	Cahier d'utilisation	47
1	Fonctionnalités	47
2	Installation.....	48
3	Format de fichier	48
F	Cahier de test	49
1	Fiches de tests	49
2	La qualité	49
	Webographie	51
	Bibliographie	53

Table des figures

2 Description générale

1	Environnement du projet	3
2	Cas d'utilisations	4
3	Structure générale du système	5

3 Etat de l'art

1	La fichier sortie pour BAK.....	8
2	La fichier sortie pour VBCTOOL	8
3	La disposition de BAK	10
4	La disposition de VBCTOOL.....	10

4 Analyse et conception

1	La disposition de l'arbre par D3.js	15
2	La disposition de l'arbre radiatif par D3.js	16

5 Mise en oeuvre

1	Architecture du programme.....	18
2	Relations des fichier JS	19
3	Interface de l'application.....	19
4	Importation du fichier.....	20
5	Format de fichier	21
6	Paramètres de visualisation.....	22
7	Disposition de l'arbre	23
8	Coloration de nœud.....	23

9	Coloration d'arrêt	24
10	Affichage du label de nœud	25
11	Affichage du label d'arrêt.....	25
12	Replier les nœuds	25
13	L'arbre avant zoom	26
14	L'arbre après zoom.....	26
6	Bilan	
1	Diagramme de Gantt	31
A	Description des interfaces externes du système	
1	Interface visiteur.....	37
2	Interface utilisateur	38
E	Cahier d'utilisation	
1	Les fonctionnalités de l'application	47
F	Cahier de test	
1	Fiches de tests	49
2	Résultat de tests.....	50
3	Résultat de tests.....	50

1

Introduction

Le présent document a pour but de détailler la conception d'un outil de visualisation de la méthode PSE (Procédure par Séparation & Evaluation, ou Branch and Bound) afin que les utilisateurs puissent voir dynamiquement la méthode Branch and Bound via une application Web. Il devra visualiser dynamiquement une arborescence, étant donné un programme de Branch and Bound.

Ce projet est proposé et encadré par Lei Shang, doctorant à Polytech Tours.

1 Contexte

Dans la méthode Procédure par Séparation & Evaluation, un problème peut être vu comme un nœud dans l'arbre, la procédure de séparation génère plusieurs sous-problèmes qui peuvent être représentés comme les nœuds fils, certains nœuds sont ensuite coupés après être évalués et ceux qui restent vont continuer être traités par la procédure.

Cette représentation arborescente est très utile et intuitive pour bien comprendre un algorithme de branchement en le déroulant sur une instance d'exemple, par contre il n'existe pas encore un outil qui visualise dynamiquement cet arbre, étant donné le programme d'une procédure de branchement. Ce genre d'outil sersa très utile pour l'analyse d'un programme de branchement, il nous permettra de trouver des propriétés intéressantes dans l'arbre.

2 Objectifs

L'objectif du projet est donc de réaliser un tel outil, qui accompagne un programme de branchement, en tant qu'une application web, qui permet aux utilisateurs de voir dynamiquement la visualisation de leur programme Branch & Bound. Par exemple, notre application web devra dessiner un nœud quand un sous-problème est créé par le programme maitre, marquer un nœud comme "coupé" quand un sous-problème est éliminé par le programme maitre. Les utilisateurs pourront aussi contrôler la vitesse de visualisation, demander un arrêt temporaire, zoomer l'arbre, replier et dérouler les nœuds, stocker leur fichier et consulter l'historique etc.

En bref, l'objectif est de développer une application web, qui devra visualiser dynamiquement la méthode Branch & Bound et aussi réaliser des fonctionnalités correspondantes pour faire mieux la visualisation.

3 Bases méthodologiques

Il s'agit d'une application web qui devra donc nécessiter d'un hébergement spécialisé. Nous avons choisi la solution Node.js et MySQL. Pour la côté front-end, nous avons décidé d'utiliser les langages HTML, CSS, JavaScript.

Pour faciliter la partie visualisation, nous avons choisi la librairie D3.js. C'est une bibliothèque graphique JavaScript qui permet l'affichage de données numériques sous une forme graphique et dynamique. Il s'agit d'un outil important pour la conformation aux normes W3C qui utilise les technologies courantes SVG, JavaScript et CSS pour la visualisation de données.

Concernant la modélisation du système, nous utiliserons le langage UML nous permettant de réaliser différents diagrammes tels que des diagrammes de cas d'utilisation ou encore d'activités. Ceci dans le but d'améliorer la compréhension de nos modélisations par l'ensemble des intervenants sur le projet.

Les sources du logiciel seront versionnées grâce à Git qui permet de gérer l'avancement du projet, de récupérer les modifications, d'avoir un suivi sur les bugs, etc.

Afin de conduire et gérer ce projet, l'outil MS Visio sera utilisé pour réaliser la planification des tâches et le suivi de celle-ci. En termes de méthodologie, l'utilisation de SCRUM (méthode AGILE) et son fonctionnement en sprint a été optée afin de garantir la livraison régulière de livrables.

2

Description générale

1 Environnement du projet

Lors de la réalisation de ce projet, aucun existant n'est présent.

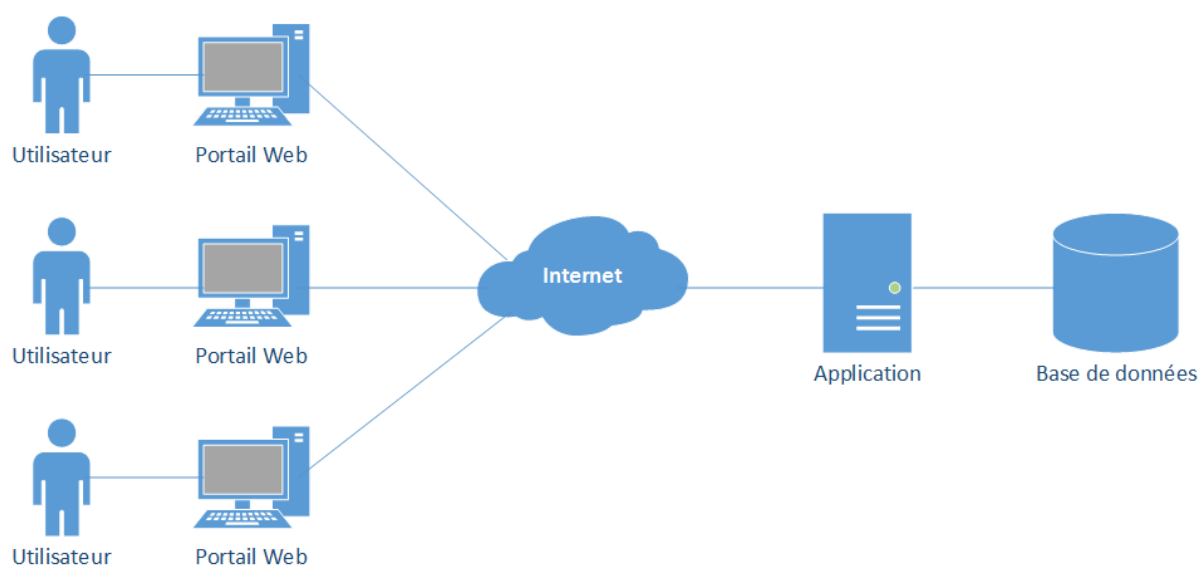


Figure 1 – Environnement du projet

Il s'agit d'une application web accessible depuis Internet permettant de l'utiliser sur bureau grâce à un navigateur Web. Elle est stockée sur un serveur communiquant avec une base de données qui contient toutes les informations concernant les informations d'utilisateur enregistré.

2 Caractéristiques des utilisateurs

Au cours de ce projet, trois types d'utilisateurs sont présents : le visiteur et l'utilisateur enregistré et l'informaticien.

- Le visiteur : Il accède l'application sans enregistrer. Il peut importer son fichier branch and bound et puis le visualiser. Il peut aussi enregistrer pour avoir plus de fonctionnalités.
- L'utilisateur enregistré : Il possède un compte de type « Utilisateur ». Il peut importer son fichier branch and bound et puis le visualiser. Il peut aussi sauvegarder le fichier et consulter l'historique.
- L'informaticien : Il est responsable de la base de données. Il peut gérer l'ensemble de ces données : fiches utilisateur, comptes utilisateur. L'Informaticien a le rôle de maintenir le bon fonctionnement de l'application mais aussi de son déploiement sur un serveur. Il a donc besoin de connaissances en Web, base de données et réseau.

3 Fonctionnalités du système

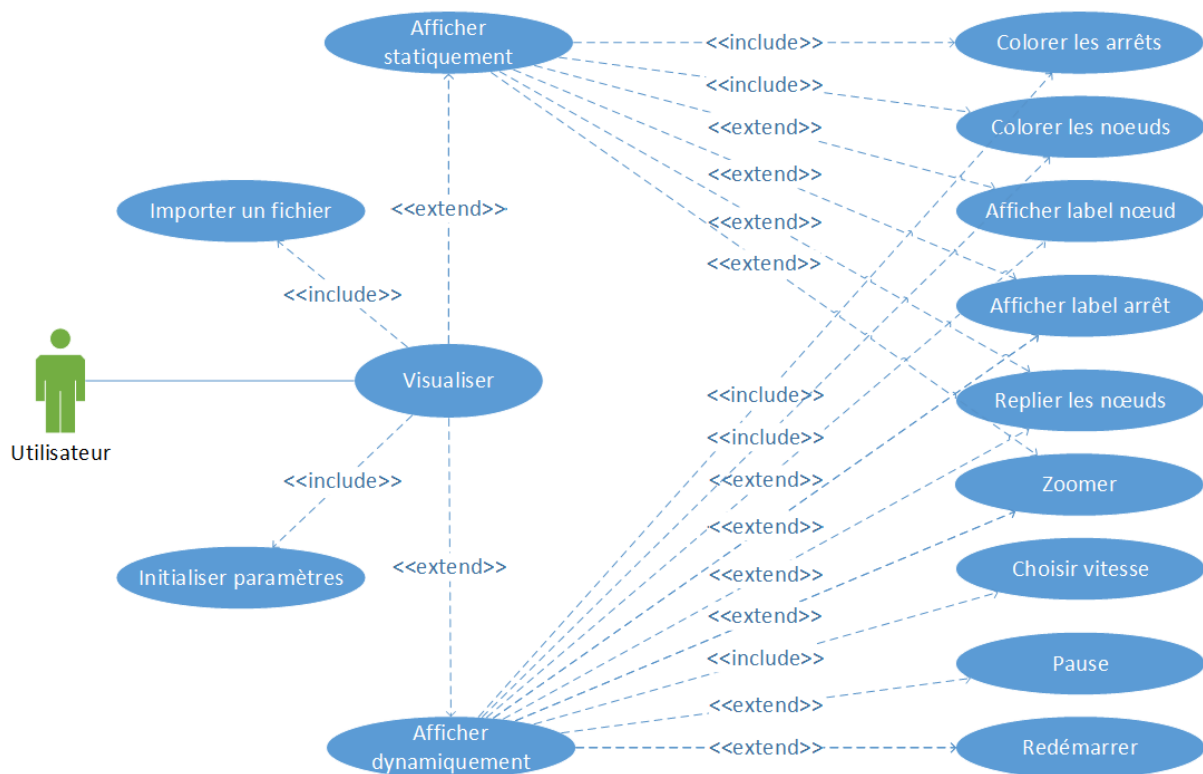


Figure 2 – Cas d'utilisations

Le fonctionnement de l'application est divisé en deux parties : la partie de visualisation et la partie de gestion d'utilisateur.

La fonctionnalité principale, c'est de visualiser la méthode branch and bound. Tout d'abord, l'utilisateur devra importer un fichier texte qui est exporté par son programme branch and bound dans l'application. Le format de fichier sera selon notre règle. L'utilisateur pourra choisir un mode d'exécution (dynamique ou statique). Ensuite, notre application va visualiser un arbre qui contient les informations correspondantes. Par exemple, l'état de chaque nœud (attendu, exploré, coupé etc.), la valeur de LB, le temps de calcul lié et la solution partielle. Les états différents de nœud seront représentés par les couleurs différentes qui rempliront les nœuds. La solution partielle sera affichée dans un label quand l'utilisateur met la souris sur un nœud. Le temps de calcul lié et la valeur de LB seront affichés dans un champ à l'extérieur de l'arbre quand l'utilisateur double clique un nœud. L'utilisateur pourra replier un nœud avec tous ses fils par cliquer le nœud. Après, il pourra dérouler le nœud par cliquer une autre fois. L'utilisateur

pourra aussi zoomer cet arbre. Pour le mode dynamique, l'utilisateur pourra choisir la vitesse de visualisation, demander un arrêt temporaire et redémarrer la visualisation.

Dans notre système, il existe deux types d'utilisateurs. Le visiteur et l'utilisateur enregistré. Tous les deux types d'utilisateurs pourrons avoir l'autorisé de visualiser. Mais le visiteur ne pourra pas sauvegarder le fichier et consulter l'historique. Par contre, Il peut créer un compte afin de devenir l'utilisateur enregistré.

4 Structure générale du système

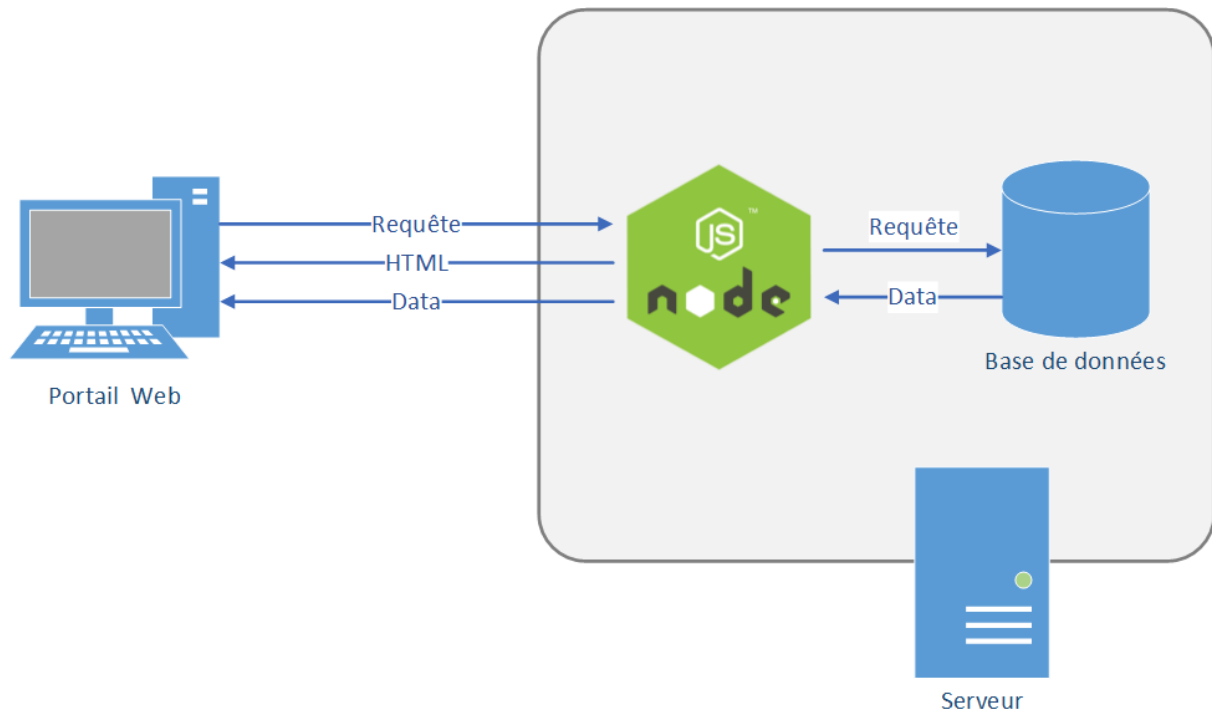


Figure 3 – Structure générale du système

Afin de manipuler (lecture, modification, insertion) les données nécessaires au fonctionnement, le portail web de l'utilisateur interagit avec des services web. Ces services web se trouvent sur notre serveur aussi la base de données. Les services web interagissent avec la base de données afin d'exécuter les requêtes SQL appropriées à la demande effectuée. Une fois les résultats de la requête SQL obtenus le web service renvoi les données au demandeur au format JSON.

3

Etat de l'art

1 Introduction

Branch-and-Bound a été proposé par A. H. Land en 1960.[4] Il utilise une formulation d'arbre pour résoudre un problème donné. Chaque feuille de cet arbre peut être considérée comme une solution complète et chaque nœud interne représente une solution partielle.

Par exemple, dans le cas du problème du sac à dos, pour chaque article, nous devons décider si le prendre ou non. Nous pouvons assigner 1 à un article si nous décidons de le prendre et 0 si nous décidons de ne pas le prendre. Si nous ne décidons pas de le prendre ou non, nous lui assignons un point d'interrogation. Une solution partielle serait une solution que nous n'avons pas décidé sur tous les articles, ce signifie que notre décision a quelques points d'interrogation. Une solution complète à ce problème serait une solution pour chaque article, 1 ou 0 est affecté. Une feuille dans l'arbre représenterait une solution complète et un nœud interne représenterait une solution partielle.

L'algorithme Branch-and-Bound essaie d'accélérer le processus de recherche de cette énumération arbre en utilisant une stratégie de délimitation. B&B maintient une seule meilleure borne inférieure et calcule la borne extérieure à chaque nœud de l'arbre. Si la borne extérieure calculée dépasse la borne inférieure, alors nous savons que le sous-arbre qui est enraciné à ce nœud ne peut pas contenir la solution optimale. Donc nous n'avons pas besoin d'explorer ce sous-arbre. C'est ce qu'on appelle l'élagage de l'arbre.

En effet, pour la plupart des problèmes, de grandes portions de l'arbre peuvent être taillées par cette façon, ce qui rend l'espace de recherche plus petit et diminue considérablement le temps de calcul.

2 Synthèse de l'état de l'art

En termes de l'état de l'art, il n'y pas beaucoup de recherches sur la visualisation de la méthode Branch-and-Bound. Une des premières tentatives a été faite par Forrest et al.[3] (1974), qui a présenté une représentation alternative de l'arbre B & B qui est analogue aux diagrammes de dispersion des LP-Bound et des inféasibilités entières. Applegate et al.[1] (2006) ont affiché un arbre B&B lors de la résolution TSP. Pour l'instant, il y a que 3 outils existants. Ils sont VBnB[8] (Visual Branch and Bound), BAK[6] (Branch and Cut Analysis Kit) et VBCTOOL[5] (The Tree

Interface for Visualization of Branch Cut algorithms). On va les analyser par les trois aspects : la communication avec le programme maître, les informations du nœud et La disposition de l'arbre.

2.1 La communication avec le programme maître

Pour la visualisation dynamique, la première chose qu'il faut considérer, c'est comment communiquer avec le programme maître.

```
# CBC
0.040003 heuristic -28.000000
2.692169 branched 0 -1 N -39.248099 16 0.169729
2.692169 pregnant 2 0 R -39.248063 14 105.991922
2.708170 pregnant 3 0 L -38.939929 6 0.105246
2.764173 pregnant 5 2 R -39.244862 12 49.115388
2.764173 branched 2 0 R -39.248063 14 105.991922
```

Figure 1 – La fichier sortie pour BAK

La version actuelle de BAK se compose d'environ 1500 lignes de code Perl. Il a modifié les codes sources de CBC 1.01 (Forrest, 2007), SYMPHONY 5.1 (Ralphs, 2007) et GLPK 4.15 (Makhorin, 2007) afin qu'ils sortent les messages d'état requis par BAK. Ces messages sont constitués certaines lignes de production indiquant la création ou le changement d'état de chaque nœud dans l'arbre. Le cœur de BAK est un script Perl qui analyse la sortie de texte du solveur et extrait des informations par les messages d'état BAK qui sont incorporés dans cette sortie. Après analyse la sortie du solveur, BAK prépare les données et les fichiers de script pour la visualisation et puis BAK appelle Gnuplot 4.0 (Williams et Kelley, 2004), un paquet graphique open-source, pour générer le graphe.

```
#TYPE: COMPLETE TREE
#TIME: SET
#BOUNDS: SET
#INFORMATION: STANDARD
#NODE_NUMBER: NONE
00:00:00.00 N 0 1 5
00:00:00.00 U 200.1
00:00:00.00 L 0
00:00:00.01 I 1 \inode 1\information included by node 1
00:00:00.50 N 1 2 5
00:00:00.50 I 2 \inode 2\information included by node 2
00:00:02.00 N 1 3 5
00:00:02.00 I 3 \inode 3\information included by node 3
00:00:03.00 N 1 4 5
00:00:03.00 I 4 \inode 4\information included by node 4
00:00:06.00 P 1 15
00:00:06.00 U 178.52
```

Figure 2 – La fichier sortie pour VBCTOOL

Le VBCTOOL est un outil spécialement conçu pour dessiner des arbres binaires et généraux. Il contient donc deux modes principaux : stimuler le processus de la création de l'arbre après le processus de calcul, dessiner l'arbre au cours d'un processus de calcul. Pour le premier mode, il a pris le fichier de la sortie du programme maître comme l'entrée afin de réaliser la

communication avec le programme maître. Pour le deuxième mode, il a deux approches. Le premier est de simplement rediriger la sortie de programmes au VBCTOOL. Cela est assez facile à manipuler, l'utilisateur doit simplement s'assurer que sa sortie obéit à quelques règles. La deuxième option est d'inclure le programme de l'utilisateur dans l'environnement VBCTOOL afin que le programme puisse être lancé en cliquant sur un bouton dans la barre de menu. Cela implique certain travail pour l'utilisateur et se limite à implémentations C++ uniquement.

2.2 Les informations du nœud

Dans la visualisation de méthode B&B, le nœud peut contenir plein de informations. C'est important de préciser les informations afin qu'on peut les afficher pendant la visualisation.

Le BAK considère certains différents états du nœud, chaque état sera attaché avec une couleur unique :

- Branché : Réalisable incomplet nœud avec les enfants existants, donc pas plus de traitement sera nécessaire.
- Candidat pour traiter : Nœud ayant un LP-bound, mais n'a pas encore été prouvée comme étant optimale.
- Candidat pour brancher : Nœud ayant un LP-bound prouvé et ayant des enfants potentiels, mais pas tous les enfants existent encore.
- Insonore : Nœud ayant un LP-bound pire que la titulaire actuel.
- Irréalisable : Nœud dont la relaxation de programmation linéaire a été prouvée irréalisable.
- Complet : Nœud ayant une complète réalisable optimale solution.

Par contre, le VBCTOOL n'a pas précisé les états du nœud. Donc, l'utilisateur devra définir les différents états lui-même, et puis les attacher avec différentes couleurs fournis par le VBCTOOL.

2.3 La disposition de l'arbre

Au cours de la visualisation dynamique, chaque fois un nouveau nœud est créé, on devra refaire la disposition de l'arbre. C'est-à-dire qu'on devra recalculer la position de chaque nœud. Donc, une bonne stratégie de disposition est essentielle.

Le BAK a utilisé le Gnuplot pour dessiner l'arbre, les nœuds sont positionnés verticalement selon leurs LP-bound. Il existe deux options pour l'espacement horizontal des nœuds. Le premier est l'espacement fixe, qui attribue un espace égal aux enfants du nœud sans considérer son nombre de descendants. La deuxième option est l'espacement proportionnel, qui affecte l'espace horizontal à chaque nœud proportionnel à son nombre de descendants. L'espacement fixe permet de mieux montrer si l'arbre est équilibré, tandis que l'espacement proportionnel permet de mieux afficher tous les nœuds de l'arbre.

La disposition du VBCTOOL est basée sur l'algorithme proposé par Walker[7]. Elle satisfait aux règles esthétiques suivantes tandis que le dessin de l'arbre occupe le plus peu d'espace possible :

- Les nœuds au même niveau de l'arbre doivent se trouver dans une ligne droite.
- Un parent devrait être centré sur ses fils.
- Un arbre et son image en miroir doivent produire des dessins qui sont des réflexions les uns des autres.
- Un sous-arbre doit être tracé de la même manière, n'importe où il se trouve dans l'arbre.

Chaque nœud est identifié par un ensemble de coordonnées (x, y) , déterminant un point dans le plan. Ces coordonnées des nœuds sont calculées en fonction des règles esthétiques énumérées ci-dessus en respectant les valeurs importantes suivantes :

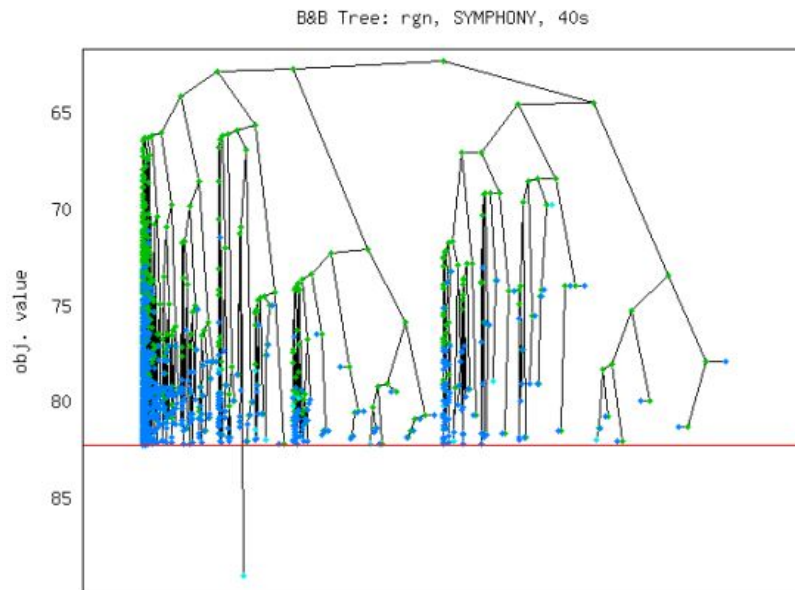


Figure 3 – La disposition de BAK

- Niveau séparation : la distance fixe des niveaux adjacents de l'arbre. Cette valeur est utilisée pour déterminer les coordonnées y des nœuds.
- Frères séparation : la distance minimale entre les frères adjacents de l'arbre.
- Sous-arbre séparation : la distance minimale entre les sous-arbres adjacents d'un arbre.

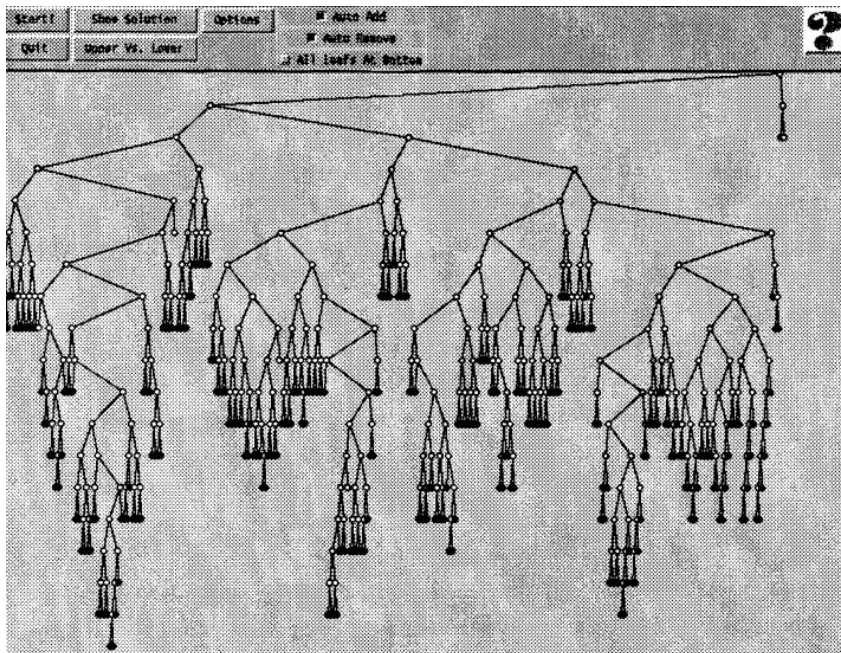


Figure 4 – La disposition de VBCTOOL

3 Limites et Avantages de l'existant

Pour les outils existants, il existe des défauts importants. C'est nécessaire de développer un meilleur outil.

Tout d'abord, ils considèrent très peu d'informations du nœud. Par exemple, la solution partielle et le temps calcul lié ne sont pas considérés. Ils sont les attributs très importants pour la méthode Branch-and-Bound. En plus, l'expérience utilisateur n'est pas bien. La disposition de l'arbre du BAK n'est pas du tout jolie et il suppose que l'arbre B&B est binaire. Le VBCTOOL ne peut pas être utilisé dans le système de l'exploitation Windows.

Mais le VBCTOOL nous inspire l'approche de la communication avec le programme maître. C'est de rediriger la sortie vers l'application de visualisation, cela sera utile pour réaliser la visualisation en temps réel. Le BAK nous fournit une claire définition de différent état du nœud. Tout cela sera utile pour notre application.

4

Analyse et conception

1 Éléments principaux de l'analyse

1.1 La communication avec le programme maître

Pour la visualisation dynamique, la première chose qu'il faut considérer, c'est comment communiquer avec le programme maître. Notre choix est de prendre le fichier de la sortie du programme maître comme l'entrée de notre application de visualisation. C'est une solution efficace qui peut nous fournir les attributs selon nos besoins pour la visualisation.

C'est une approche de simuler le processus Branch and Bound quand il est fini. Cela pourrait être préférable qu'une simulation en temps réel pour plusieurs raisons. La première raison, un processus Branch and Bound peut durer longtemps et l'utilisateur est obligé de garder un œil sur l'écran alors que rien ne se passe réellement. La deuxième raison, un processus peut être très rapide et produit un grand arbre, de sorte qu'il n'est pas possible de suivre l'action qui a lieu. Donc, au lieu de dessiner l'arbre pendant le processus de Branch and Bound, nous avons choisi l'approche de visualiser le processus quand il est fini. Notre application permettra l'utilisateur de contrôler la vitesse de visualisation selon son besoins.

Le fichier de sortie contiendra les attributs nécessaires pour la visualisation. Pour obtenir un tel fichier, l'utilisateur devra insérer certaines lignes de code dans son programme quand un nœud est créé, branché et coupé. Cela sera simple pour un utilisateur qui connaît bien son programme. Le format de fichier sera selon notre règle. Il devra contenir les attributs suivants :

- L'état de nœud.
- La valeur LB (Lower bound) de nœud.
- La solution partielle de nœud.
- Le parent de nœud.
- Le temps.

Notre application va prendre le fichier comme l'entrée, et puis l'analyser et extraire les informations pour la visualisation dynamique.

1.2 Les informations de nœud

Dans la visualisation de méthode B&B, le nœud peut contenir plein de informations. Par rapport aux anciens outils, notre application va essayer de considérer plus de informations.

1.2.1 L'état de nœud

Au cours du processus de la méthode B&B, l'état de nœud changera avec le temps. C'est important de distinguer les différents états de nœud. Dans notre application, on définit les trois états suivants :

- L'état attendu : un nœud a déjà été créé mais n'a pas encore été branché ou été coupé.
- L'état exploré : un nœud a déjà été branché.
- L'état coupé : un nœud a été coupé.

Pour distinguer les 3 différents états, on donnera chaque état une couleur unique : l'état attendu (bleu), l'état exploré (vert), l'état coupé (rouge). Quand l'état du nœud est changé, on devra changer aussi le couleur du nœud.

1.2.2 La solution partielle

Pour le nœud, la solution partielle est une information très importante. Pour les anciennes recherches, très peu d'outils la considère. Dans notre application, on va l'afficher dans un label quand l'utilisateur met la souris sur un nœud.

1.2.3 La borne inférieure

Pour le nœud, la borne inférieure est une information très importante. Pour les anciennes recherches, il n'y a pas beaucoup d'outils la considère. Dans notre application, on va l'afficher un champ à l'extérieur de l'arbre quand l'utilisateur double clique un nœud.

1.2.4 Le temps de calcul lié

Au cours du processus de la méthode B&B, chaque nœud a son temps de calcul lié. Pour les anciennes recherches, aucun outil ne le considère. Dans notre application, on va calculer le temps et puis l'afficher un champ à l'extérieur de l'arbre quand l'utilisateur double clique un nœud.

1.3 La disposition de l'arbre

Pour les anciens outils, Il existe un même problème : la disposition. Au cours de la visualisation dynamique, chaque fois un nouveau nœud est créé, on devra refaire la disposition de l'arbre. C'est-à-dire qu'on devra recalculer la position de chaque nœud. Donc, une bonne stratégie de disposition est essentielle. Pour les dispositions de l'ancien outil, évidemment, l'expérience utilisateur n'est pas bien.

Afin de garantir une expérience utilisateur optimale, on devra faire une disposition plus claire et plus jolie. Après plein de recherches, on a trouvé un outil très efficace. Notre choix est la

librairie D3.js. C'est une bibliothèque graphique JavaScript qui permet l'affichage de données numériques sous une forme graphique et dynamique. Il s'agit d'un outil important pour la conformation aux normes W3C qui utilise les technologies courantes SVG, JavaScript et CSS pour la visualisation de données. La librairie D3.js nous fournit une série de dispositions fortes.

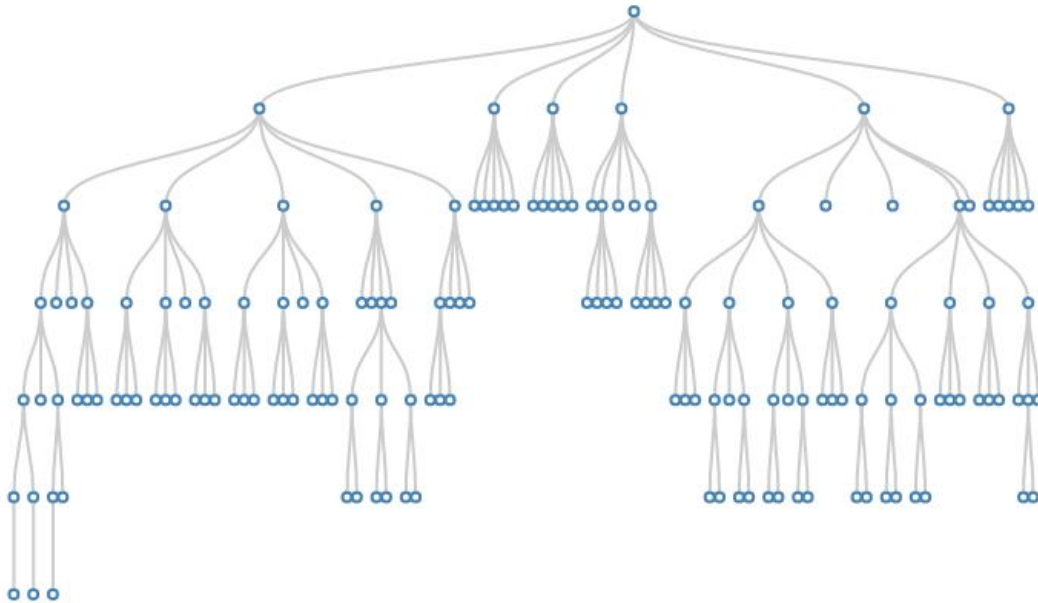


Figure 1 – La disposition de l'arbre par D3.js

Parmi eux, la disposition de l'arbre conforme complètement à nos besoins. De plus, elle permet aussi d'attacher les données avec les éléments SVG. Cela va faciliter d'attacher les informations de nœud avec le nœud correspondant. Voici une disposition simple de l'arbre qui est construit par D3.js. Évidemment, elle est plus jolie que les anciennes dispositions.

2 Choix de technique

Après une suite d'analyse, nous avons choisi de réaliser notre outil sous la forme de l'application web. Cela permettra tous les utilisateurs d'utiliser notre application sans installer, sans considérer leur système d'exploitation. Tout ce que les utilisateurs devront faire, c'est juste de accéder notre site et puis importer leur fichier.

Le choix d'application web pourra aussi faciliter la disposition de l'arbre, la librairie D3.js est un outil très forte qui conforme complètement à nos besoins. Pour les anciennes recherches, la disposition est réalisée par C++ ou Gnuplot. Dans le cas C++, chaque fois un nouveau nœud est créé, on devra recalculer la position de chaque nœud par nous-même. C'est compliqué de trouver une bonne stratégie de disposition. Dans le cas Gnuplot, la disposition est pas du tout joli.

Par rapport à C++ et Gnuplot, la librairie D3.js nous fournit une suite de dispositions qui sont plus claires et jolies. De plus, elle permet aussi d'attacher les données avec les éléments SVG. Cela va faciliter d'attacher les informations de nœud avec les nœuds correspondants. Cela sera efficace pour notre visualisation. Si le temps permet, on pourra aussi visualiser la méthode Branch and Bound par d'autre disposition de D3.js, par exemple l'arbre radiatif[2].



Figure 2 – *La disposition de l'arbre radiatif par D3.js*

3 Limites et conséquences de la solution

L'application web est presque un choix parfait sauf que la vitesse. La vitesse maximum de visualisation dépend le nombre de nœuds. Si le nombre est trop, le page web devra rendre beaucoup d'éléments SVG, la vitesse maximum sera ralentie. La performance de navigateur diminuera aussi quand le nombre de nœuds est trop. Pour garantir la performance de l'application, c'est mieux que le nombre de nœuds ne dépasse pas cent mille.

5

Mise en oeuvre

1 Les outils et librairies utilisées

Pour faciliter le développement du projet, notre choix de technique est comme suivant.

- Les langages de programmation :
 - HTML5
 - CSS3
 - JavaScript
- Les librairies utilisées :
 - D3.js 3.1.0
 - JQuery.js 3.1.1
- L'environnement de développement :
 - Navigateur : Google Chrome 56.0.1 et Mozilla Firefox 51.0.1
 - Système d'exploitation : Windows 8 et Ubuntu 12.04.1
 - Éditeur de texte : Sublime Text 2.

2 Architecture du système

2.1 Architecture générale

Afin de bien structurer notre programme, les codes source sont composés par 4 parties principales :

- Le dossier **lib** : il contient les librairies JavaScript que nous avons utilisées. Voici les différentes librairies qu'il contient :
 - **D3.js** : la librairie que nous avons utilisée pour faciliter la partie visualisation. Les parts de nos fonctionnalités ont besoins de D3.js.
 - **JQuery.js** : la librairie que nous avons utilisée pour manipuler les attributs DOM.
- Le dossier **style** : il contient nos fichiers CSS. Voici les différents fichiers qu'il contient :

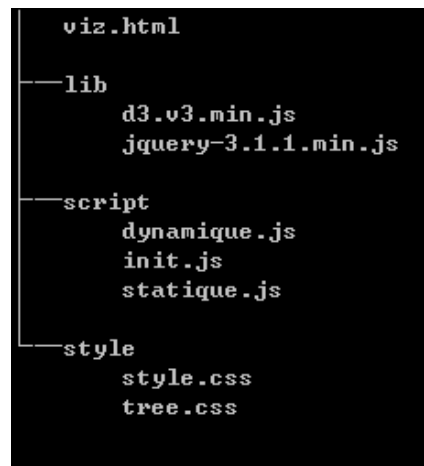


Figure 1 – Architecture du programme

- **style.css** : le fichier CSS pour initialiser le style de la page Web.
 - **tree.css** : le fichier CSS pour initialiser le style de l'arbre que nous avons visualisé.
- Le dossier **scripts** : il contient nos fichiers JavaScript. Voici les différents fichiers qu'il contient :
- **init.js** : le fichier JS pour traiter les informations entrées par la formule, importer le fichier de l'utilisateur, passer les paramètres de visualisation au statique.js ou dynamique.js.
 - **statique.js** : le fichier JS pour exécuter la visualisation statique.
 - **dynamique.js** : le fichier JS pour exécuter la visualisation dynamique.
- Le fichier **viz.html** : le fichier HTML pour initialiser la disposition de notre page web.

2.2 Architecture détaillée

Toutes les fonctionnalités de la visualisation sont réalisées par les 3 fichiers JavaScript. Ils sont les parties les plus importantes de notre programme. Leur relations et méthodes comprises est montré dans la [Figure 2](#).

Le **init.js** contient les méthodes suivantes :

- **uploadFile()** : importer et lire fichier.
- **setParameters()** : traiter les informations entrées par l'utilisateur et puis les initialiser comme les paramètres de visualisation. Enfin, passer les paramètres à la méthode suivante.

Le **statique.js** contient les méthodes suivantes :

- **formatData()** : formaliser toutes les données aux formes des nœuds et puis construire un array arborescent avec ces nœuds. Cette méthode est appelé qu'une seule fois.
- **setTreeLayout()** : initialiser la disposition de l'arbre.
- **draw()** : dessiner statiquement l'arbre. Cela comprend la coloration de nœud, la coloration d'arrêt, l'initialisation du label de nœud, l'initialisation du label d'arrêt.
- **clickNode()** : replier et dérouler un nœud quand nous le clique.
- **zoom()** : zoomer l'arbre.

Le **dynamique.js** contient les méthodes suivantes :

- **formatData()** : formaliser les données aux formes des nœuds et puis construire un array arborescent avec ces nœuds. Cette méthode est appelé chaque fois nous traitons une ligne de données.

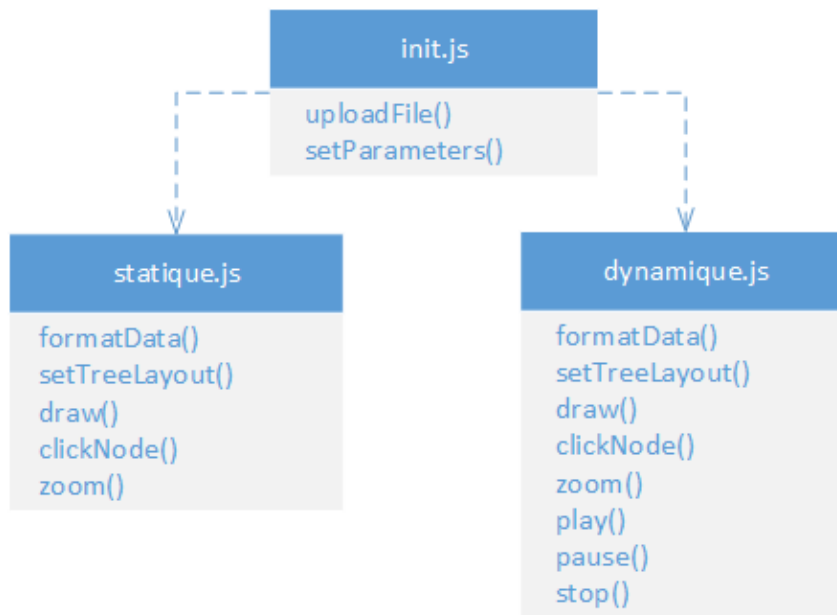


Figure 2 – Relations des fichier JS

- **setTreeLayout()** : initialiser la disposition de l'arbre.
- **draw()** : dessiner dynamique statiquement l'arbre. Cela comprend le changement de la coloration de nœud, le changement de la coloration d'arrêt, le changement du label de nœud, le changement du label d'arrêt.
- **clickNode()** : replier et dérouler un nœud quand nous le clique.
- **zoom()** : zoomer l'arbre.

3 Interface de l'application

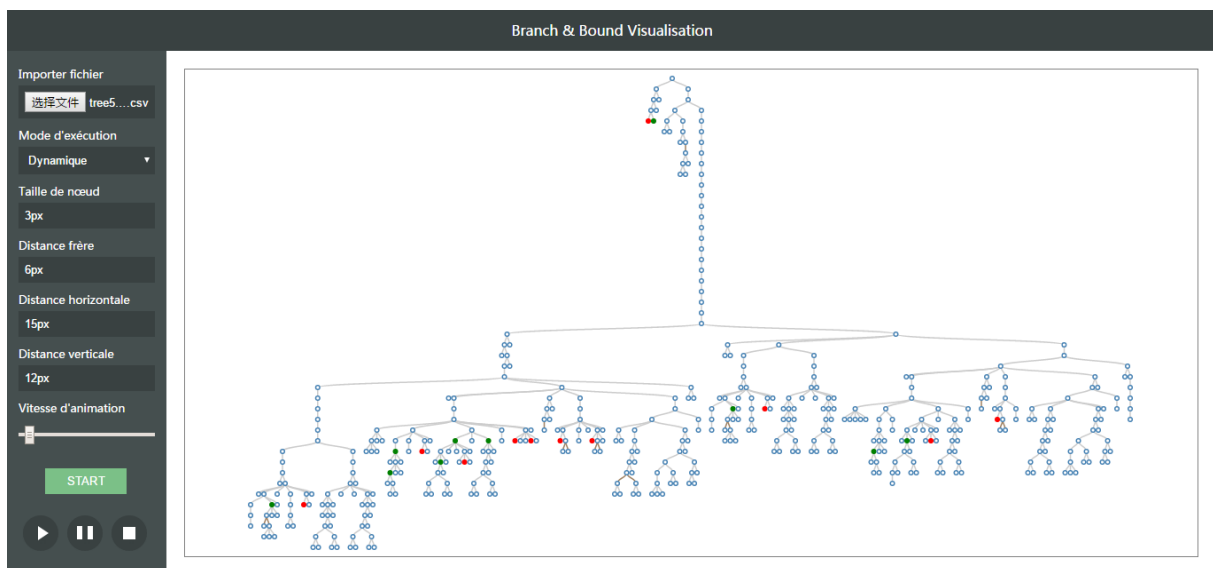


Figure 3 – Interface de l'application

L'interface de l'application est montré dans Figure 3. Elle contient 3 parties. Voici les différentes parties et leurs fonctionnalités :

- La barre de navigation : afficher l'information de l'application.
- La barre d'outils : permettre aux utilisateurs d'entrer les paramètres de visualisation et contrôler le processus de visualisation.
- La zone de visualisation : afficher le résultat de visualisation un arbre B&B.

La barre d'outils changera avec les différents modes de visualisation. Pour la mode dynamique, il aura l'entrée de vitesse et les boutons de contrôle en plus.

4 Implémentation

4.1 Importation du fichier

Notre application permet aux utilisateurs d'importer un fichier texte qui est exporté par son programme branch and bound.

Le HTML5 nous fournit une méthode **FileReader().onload** qui permet de charger directement le fichier local. Donc, nous n'avons plus besoins d'importer le fichier sur le serveur. Après, nous avons utilisé la méthode **FileReader().readAsText()** pour retourner un string ce qui est le contenu du fichier. Le résultat de réalisation est montré dans la **Figure 4**.

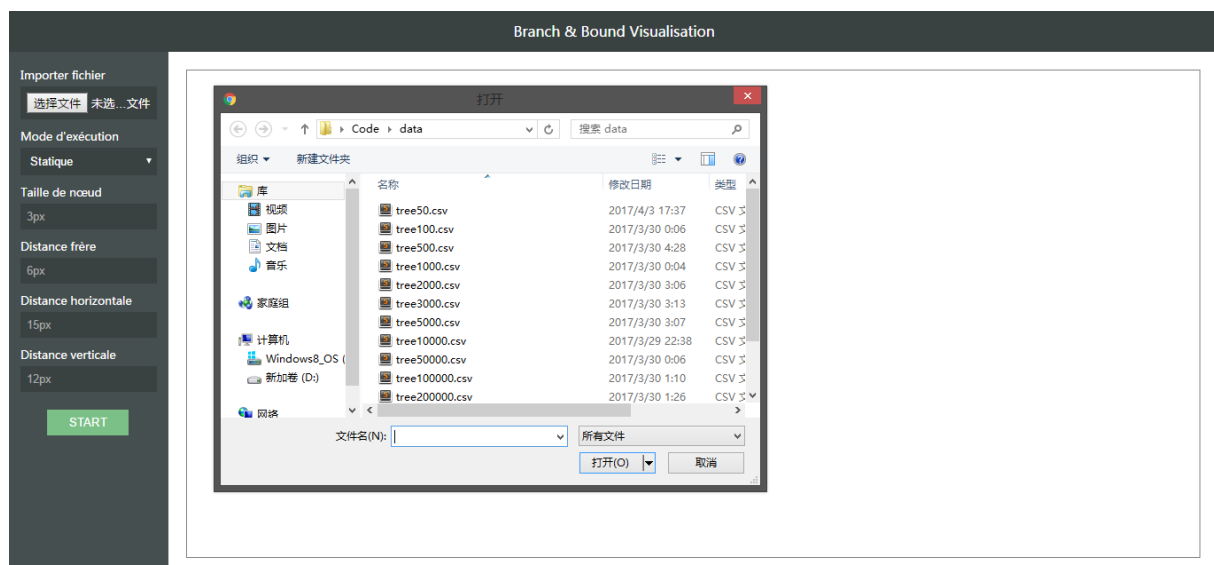


Figure 4 – Importation du fichier

4.2 Format de fichier

Le format de fichier sera forcément selon notre règle. Chaque ligne sera sous un format **Action, Id, PID, LabelNode, LabelLink** :

- Action : la commande de l'action. Notre application fournit une suite de commande pour préciser les actions différentes. Voici les commandes :
 - **C** : Un nœud est créé. Cela peut être **C1, C2** pour distinguer les différentes façons de création.
 - **B** : Un nœud est branché.

```

Action,Id,PId,LabelNode,LabelLink
C2,1,0,,
B,1,,Branched,
C2,2,1,,
B,2,,Branched,
C2,3,2,,
B,3,,Branched,
C1,4,3,,
K2,4,,Cut by 1,
U,1,,Used by 4,
C2,5,3,,
B,5,,Branched,
C2,6,5,,
B,6,,Branched,
C2,7,6,,

```

Figure 5 – Format de fichier

- **K** : Un nœud est coupé. Cela peut être **K1**, **K2** pour distinguer les différentes façons de coupe.
- **U** : Un nœud est utilisé pour couper un autre nœud.

— Id : le numéro de nœud.

— PId : le numéro de parent de nœud. Cela peut être vide si le nœud a été créé.

— LabelNode : un String pour décrire le nœud. Cela peut être une borne inférieure ou extérieure, une solution partielle, un temps calcul lié etc. Cela peut être vide.

— LabelLink : un String pour décrire l'arrêt. Cela peut être vide.

Afin de structurer les données, nous avons formalisé les données sous une forme d'un array arborescent de nœuds. Le nœud racine sera utilisé pour générer l'arbre. Nous avons utilisé la méthode `d3.layout.tree().nodes(root)` pour générer l'ensemble de nœuds, la méthode `d3.layout.tree().links(nodes)` pour générer l'ensemble de nœud.

4.3 Entrée des paramètres de visualisation

Notre application permet aux utilisateurs d'entrer les paramètres pour initialiser une visualisation selon leurs besoins. Cela contient :

- Le fichier de données
- La mode d'exécution
- La taille de nœud
- Les paramètres de disposition
- Il aura la vitesse d'animation en plus pour la mode dynamique.

Nous avons utilisé JQuery.js pour faciliter le contrôle et la manipulation DOM. Cette fonctionnalité est réalisé par la fonction `setParameters()`. Après, les paramètres récupérés seront envoyés à la méthode `statique()` ou `dynamique()` pour faire la visualisation. Le résultat de réalisation est montré dans la Figure 6.

4.4 Disposition de l'arbre

Notre application permet aux utilisateurs de personnaliser la disposition de l'arbre selon leurs besoins. Pour fixer une disposition, il y a 4 paramètres à préciser :



Figure 6 – Paramètres de visualisation

- La taille du nœud : le demi-diamètre de nœud.
- La distance entre les nœuds frère : la distance entre les centres des nœuds frère.
- La distance horizontale : la distance entre un nœud et son voisin.
- La distance verticale : la distance entre les niveaux.

La méthode `d3.layout.tree().nodeSize([nodeSize]).separation([separation])` fourni par D3.js a été utilisé pour fixer la disposition de l'arbre. la Figure 7 montre un arbre de 500 nœuds visualisé avec notre disposition par défaut, les 4 paramètres sont 3px,6px,15px et 12px.

4.5 Mode statique

La mode statique affichera un arbre entier dans une seule fois. Cela va permettra aux utilisateurs de voir leur résultat de visualisation avec peu d'attente. Si le nombre de nœuds est trop, par exemple cent mille, la visualisation dynamique va prendre longtemps. Dans le cas, la visualisation est un bon choix.

La méthode `selection.enter().append()` fourni par D3.js a été utilisé pour entrer les nouveaux nœuds aux éléments SVG. Tous les nœuds sont entrés dans une seule fois.

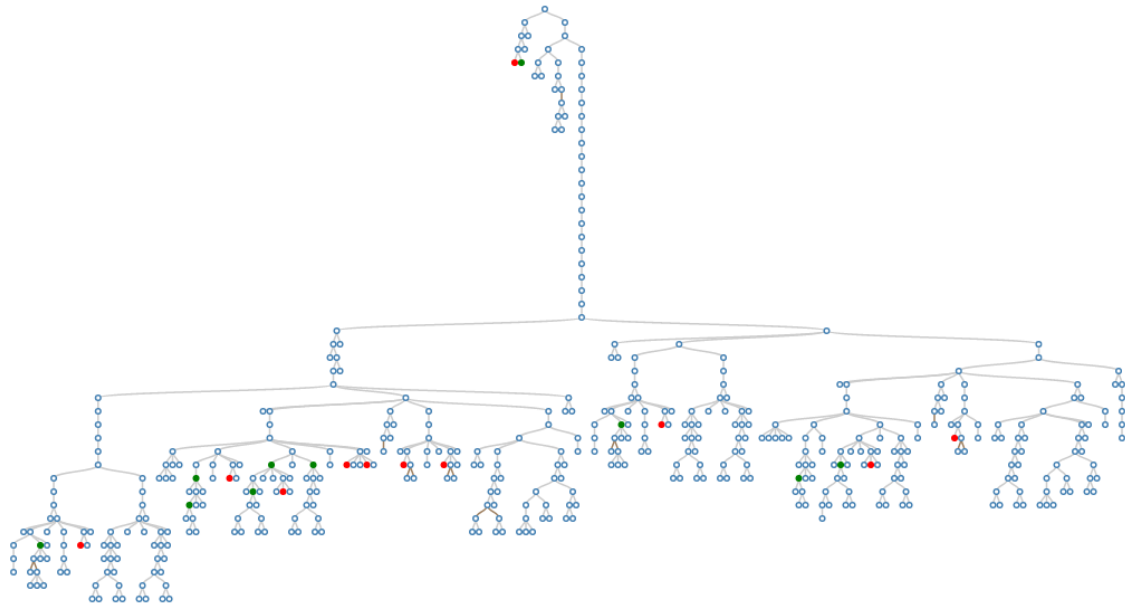


Figure 7 – Disposition de l'arbre

4.6 Mode dynamique

La mode dynamique affichera dynamiquement un arbre, nœud par nœud. Cela permettra aux utilisateurs de bien regarder et comprendre tout le processus de la visualisation.

La méthode `selection.enter().append()` fourni par D3.js a été utilisé pour entrer les nœuds aux éléments SVG. Un seul nouveau nœud ou aucun nœud est entrée chaque fois. S'il y a un nouveau nœud entré, il se cachera dans l'ancienne position de son parent. Enfin, nous avons utilisé la méthode `selection.transition().duration()` pour faire la transition.

4.7 Coloration de nœud

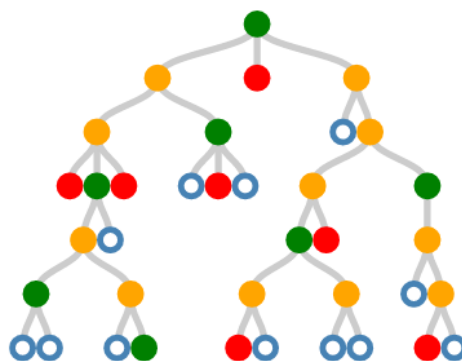


Figure 8 – Coloration de nœud

Notre application va colorer les nœuds en différentes couleurs selon leurs actions.

Pour la visualisation statique, la coloration est faite dans une seule fois. La couleur représentera l'action dernière effectuée sur ce nœud. Pour la visualisation dynamique, la couleur d'un nœud changera avec le changement de son action.

Voici les couleurs et leurs actions liée :

- Transparent : nœud créé (action C1, C2)
- Orange : nœud branché (action B)
- Rouge : nœud coupé (action K1, K2)
- Vert : nœud utilisé (action U)
- Bleu clair : nœud qui a des nœuds fils repliés.

La méthode `seleciton.attr("r",).style("fill",).style("stroke-width",)` a été utilisé pour initialiser ou changer le style de nœud. Les trois attribut sont : la demi diamètre de nœud, la couleur de nœud et le largeur de stroke. Le résultat de réalisation est montré dans la [Figure 8](#).

4.8 Coloration d'arrêt

Notre application va colorer aussi les arrêts en différentes couleurs selon leurs façons d'être créé (action C1, C2).

Voici les couleurs et leurs actions liée :

- Brun : arrêt créé par la méthode 1. (action C1)
- Gris : arrêt créé par la méthode branch and bound. (action C2)

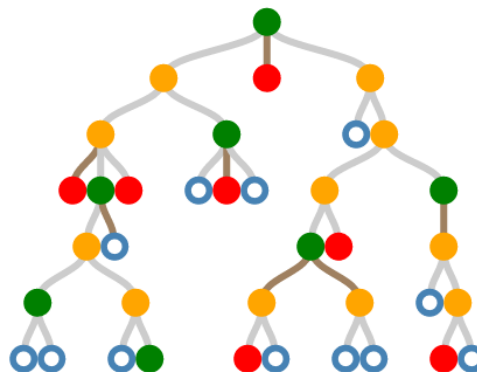


Figure 9 – Coloration d'arrêt

Nous avons utilisé la méthode `seleciton.attr(,).style("stroke",)` pour colorer les arrêts. Le résultat de réalisation est montré dans la [Figure 9](#).

4.9 Affichage du label de nœud

Notre application va afficher un label à côté du nœud quand nous mettons la souris sur ce nœud. Le contenu du label est le numéro du nœud et le String donné par le fichier d'entrée. Cela contiendra et les informations du nœud, comme **la borne inférieure ou extérieure, la solution partielle, le temps calcul lié** etc.

Le résultat de réalisation est montré dans la [Figure 10](#).

4.10 Affichage du label d'arrêt

Notre application va afficher un label à côté de l'arrêt quand nous mettons la souris sur cette arrêt . Le contenu du label est le numéro du nœud et le String donné par le fichier d'entrée. Cela contiendra et les informations de l'arrêt.

Le résultat de réalisation est montré dans la [Figure 11](#).

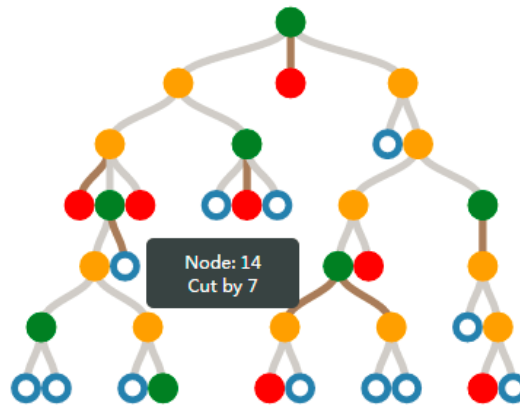


Figure 10 – Affichage du label de nœud

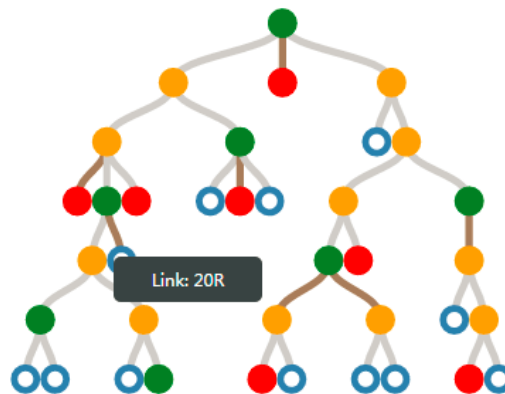


Figure 11 – Affichage du label d'arrêt

4.11 Replier et dérouler les nœuds

Notre application permet aux utilisateurs de replier les nœuds fils d'un nœud et puis dérouler les nœuds par cliquer un nœud. Un nœud qui a des nœuds fils sera coloré en bleu clair.

Le résultat de réalisation est montré dans la Figure 12.

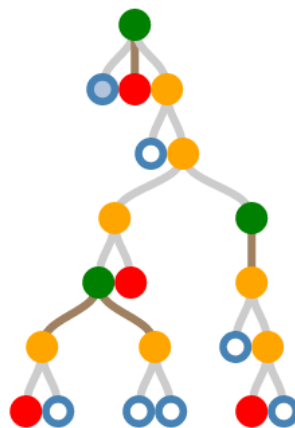


Figure 12 – Replier les nœuds

4.12 Zoom

Notre application permet aux utilisateurs de zoomer l'arbre avec rouler la souris (mousewheel). Le grossissement est de 0.1 à 10.

La méthode `d3.behavior.zoom().scaleExtent()` fourni par D3.js a été utilisé. Le résultat de réalisation est montré dans la Figure 13 et la Figure 14.

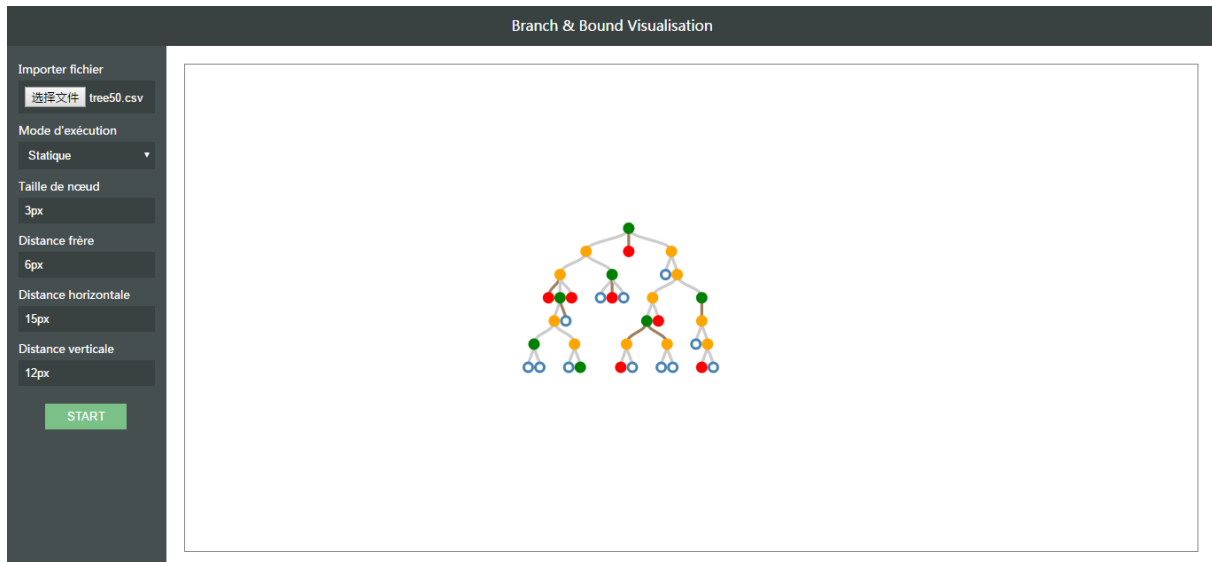


Figure 13 – L'arbre avant zoom

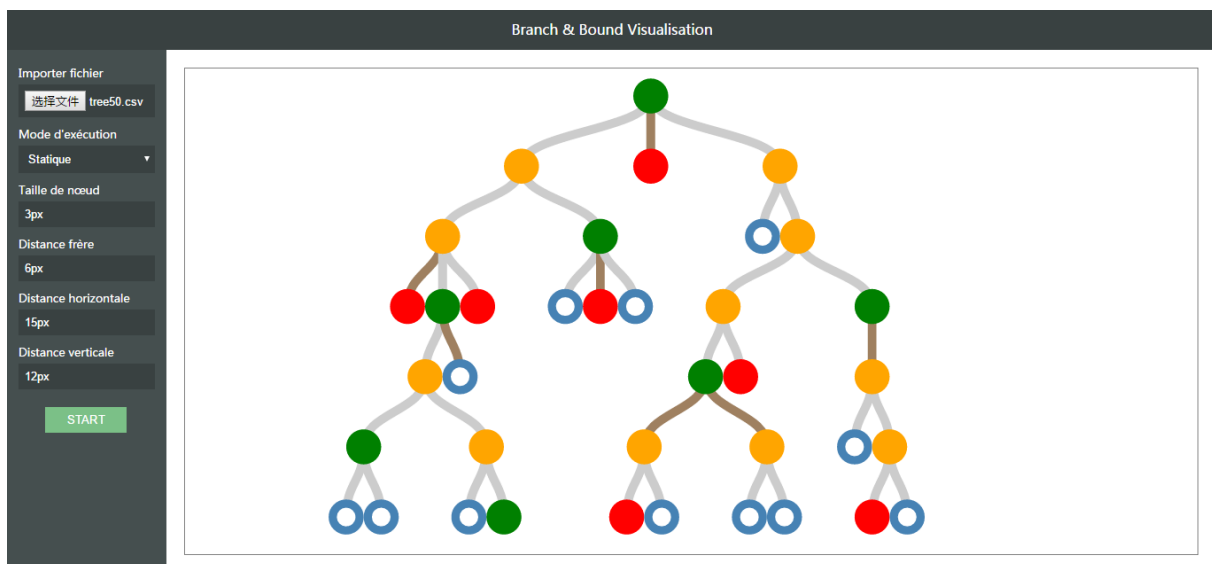


Figure 14 – L'arbre après zoom

4.13 Contrôle de vitesse

La mode dynamique permet aux utilisateurs de contrôler la vitesse de visualisation par entrer un paramètre quand ils initialisent la visualisation.

Nous avons utilisé la méthode **setInterval(draw, durationDraw)** pour exécuter la fonction **draw()** chaque durationDraw milli second.

4.14 Pause

Notre application permet aux utilisateurs de demander un arrêt temporaire au cours de la visualisation dynamique.

Nous avons utilisé la fonction **clearInterval(timer)** pour terminer le timer.

4.15 Redémarrer

Notre application permet aux utilisateurs de redémarrer la visualisation dynamique après l'arrêt temporaire.

Nous avons utilisé la méthode **setInterval(draw, durationDraw)** pour reexécuter la fonction **draw()** chaque durationDraw milli second.

4.16 Stop

Notre application permet aux utilisateurs de terminer une visualisation dynamique.

Nous avons utilisé la fonction **clearInterval(timer)** pour terminer le timer. Après , faut vider le champ de visualisation.

5 Limite

5.1 Limite de la visualisation statique

Le temps de réponse de visualisation statique changera avec le nombre de nœuds. Quand le nombre est inférieur à cent mille, le temps ne dépasse pas une minute. Quand le nombre dépasse 400 000, le temps est extérieur à 10 minutes. Afin de garantir une expérience d'utilisateur, c'est mieux que le nombre de nœuds ne dépasse pas 400 000.

5.2 Limite de la visualisation dynamique

La vitesse maximum de visualisation dynamique dépend fortement le nombre de nœuds. Si le nombre est trop, la vitesse maximum sera ralentie. La performance de navigateur diminuera aussi. Pour garantir la performance de l'application, c'est mieux que le nombre de nœuds ne dépasse pas dix mille.

6

Bilan

1 Conclusion

La réalisation de cette application Web de visualisation de B&B nous a permis d'acquérir de nombreuses compétences techniques et d'organisation. Nous avons pu développer nos compétences dans le domaine du HTML, CSS, Javascript. De plus nous avons pu améliorer nos compétences de gestion de projet.

2 Plan de développement

2.1 Tâches

Lors de la réalisation de ce projet, nous allons devoir effectuer différentes tâches avant de pouvoir livrer l'application :

- Importation de fichier : Mise en place la fonctionnalité d'importer fichier dans l'application.
- Structuration de données : Structurer les données d'entrée afin de construire l'arbre.
- Affichage de l'arbre : Mise en place la fonctionnalité d'afficher un arbre selon les données d'entrée.
- Coloration de l'arbre : Mise en place la fonctionnalité de colorer les nœuds et les arrêts selon leurs états.
- Affichage de label : Mise en place la fonctionnalité d'afficher le label nœud, le label arrêt.
- Transition de couleur de nœud : Mise en place la fonctionnalité de changer la couleur de nœud pendant la visualisation dynamique
- Transition de label : Mise en place la fonctionnalité de changer le label de nœud pendant la visualisation dynamique.
- Contrôle de vitesse : Mise en place la fonctionnalité de contrôler la vitesse pour la visualisation dynamique.
- Replier et dérouler les nœuds : Mise en place la fonctionnalité de replier et dérouler les nœuds.

- Implémentation d'interface : Mise en place l'interface.
- Pause et redémarrage : Mise en place la fonctionnalité de demander un arrêt et redémarrer.
- Zoom : Mise en place la fonctionnalité zoom.
- Tests : Mise en place le test.
- Livraison et rapport : Rendu des livrables, réalisation du rapport et soutenance.

2.2 Diagramme de Gantt

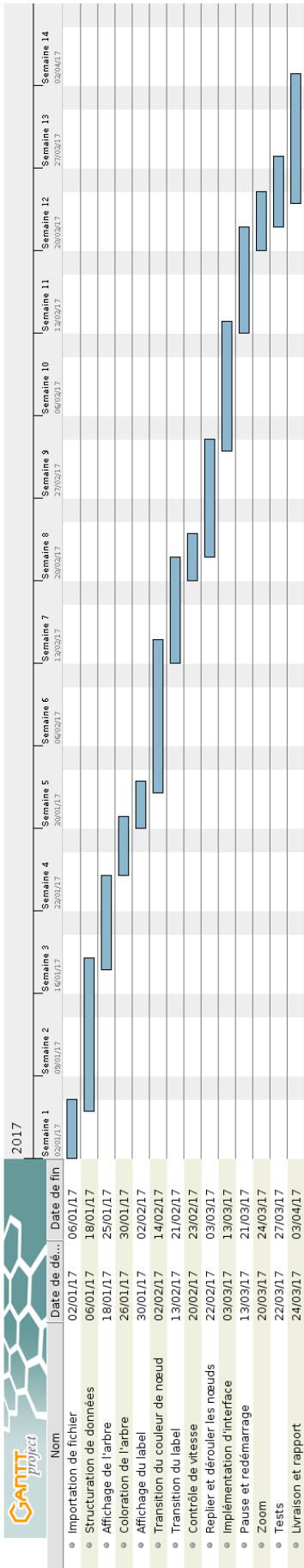


Figure 1 – Diagramme de Gantt

Bibliographie

- [1] David L APPLEGATE, Robert E BIXBY, Vasek CHVATAL et William J COOK. *The traveling salesman problem : a computational study*. Princeton university press, 2011.
- [2] Mike BOSTOCK. *Radial Tidy Tree*. URL : <http://bl.ocks.org/mbostock/4063550>.
- [3] JJH FORREST, JPH HIRST et JOHN A TOMLIN. « Practical solution of large mixed integer programming problems with UMPIRE ». In : *Management Science* 20.5 (1974), p. 736–773.
- [4] A. H. LAND et A. G. DOIG. « An Automatic Method of Solving Discrete Programming Problems ». In : *Econometrica* 28.3 (1960), p. 497–520. issn : 00129682, 14680262. URL : <http://www.jstor.org/stable/1910129>.
- [5] Sebastian LEIPERT. *VBCTOOL - a graphical interface for Visualization of Branch Cut algorithms*. URL : http://www.informatik.uni-koeln.de/lis_juenger/vbctool/.
- [6] Osman Y OZALTIN, Brady HUNSAKER et Ted K RALPHS. « Visualizing Branch-and-Bound Algorithms ». In : URL <http://www.rosemaryroad.org/projects/BAK> (2007).
- [7] John Q WALKER. « A node-positioning algorithm for general trees ». In : *Software : Practice and Experience* 20.7 (1990), p. 685–705.
- [8] Tong ZHANG, Gaofeng HUA et Arika LIGMANN-ZIELINSKA. « Visually-driven parallel solving of multi-objective land-use allocation problems : a case study in Chelan, Washington ». In : *Earth Science Informatics* 8.4 (2015), p. 809–825.

Annexes

A

Description des interfaces externes du système

1 Interfaces matériel/logiciel

Dans ce projet, nous allons développer une interface web. Pour cela, l'utilisateur devra être muni d'un ordinateur et plus précisément d'un navigateur (avec JavaScript activé) et d'un service d'accès Internet afin de pouvoir accéder à l'application.

2 Interfaces homme/machine

2.1 Interface visiteur

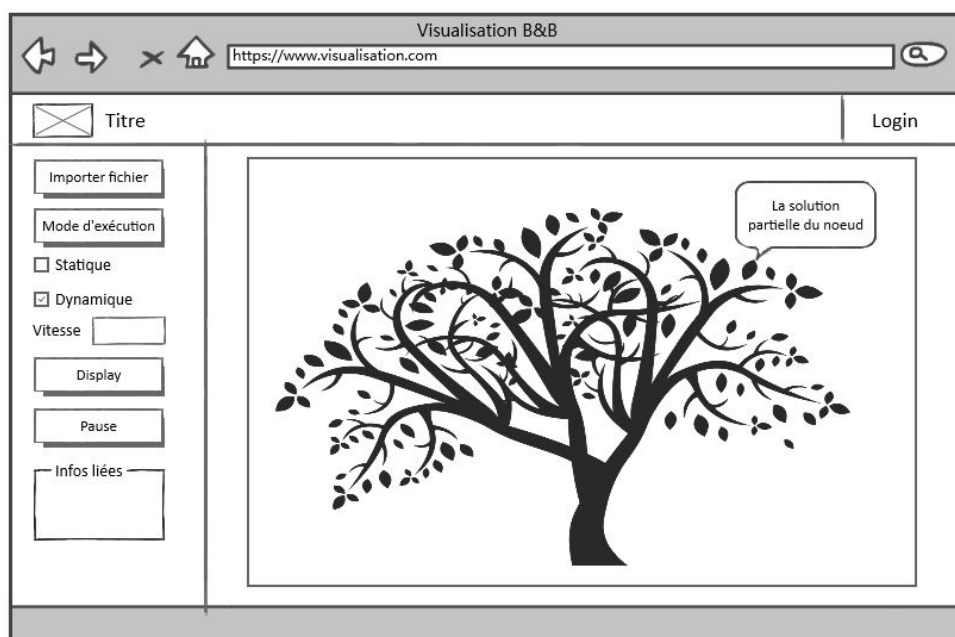


Figure 1 – Interface visiteur

L'interface du visiteur permettra d'avoir accès à l'ensemble des fonctionnalités liées à la visualisation : import de fichier, choix de mode d'exécution, demande d'un arrêt temporaire et redémarrage de la visualisation. Elle permettra aussi d'enregistrer ou se connecter.

2.2 Interface utilisateur

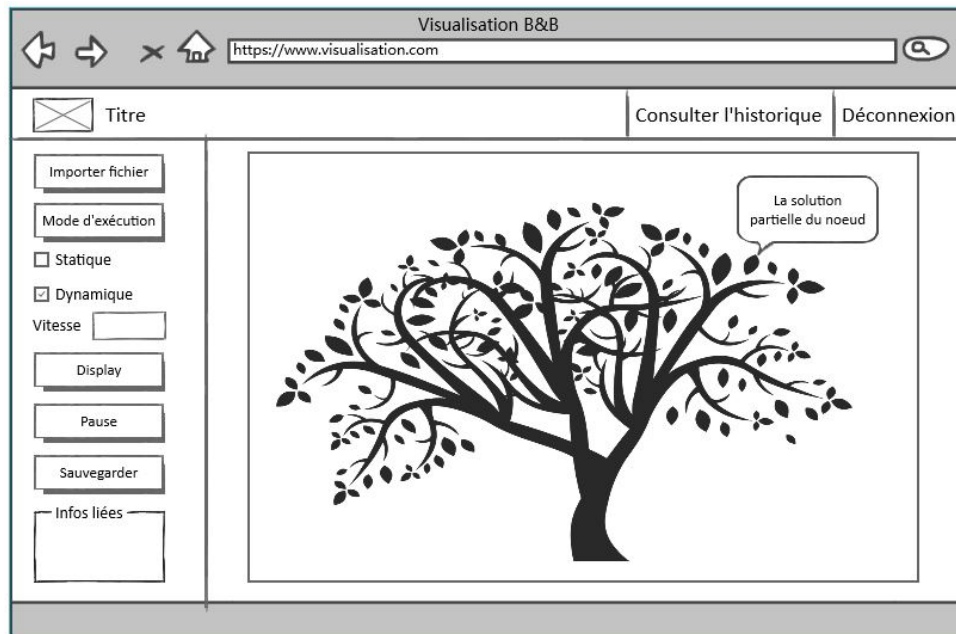


Figure 2 – Interface utilisateur

L'interface de l'utilisateur aura les mêmes fonctionnalités comme l'interface du visiteur. En plus, elle permettra de sauvegarder le fichier et consulter l'historique précédent.

3 Interface logiciel/logiciel

Concernant les interfaces logiciel/logiciel, il s'agit de la liaison entre la base de données et notre application (situées sur le même serveur). En effet, étant donné que nous manipulons une base de données il faudra régulièrement communiquer avec celle-ci.

B

Spécifications fonctionnelles

1 Enregistrement

Le visiteur pourra enregistrer dans l'application pour accéder à plus de fonctionnalités (sauvegarder le fichier et consulter l'historique), l'enregistrement aura besoin d'une adresse e-mail et d'un mot de passe. Les données seront stockées dans notre base de données. Après, il pourra se connecter à l'application avec son compte.

2 Authentification

L'authentification va permettre aux utilisateurs de se connecter. L'utilisateur devra, à chaque fois qu'il tente d'accéder à l'application, rentrer son adresse e-mail et son mot de passe. Si les informations sont correctes, l'accès sera autorisé ; dans le cas contraire, l'accès lui sera refusé. Pour savoir si l'utilisateur a bien accès, cela se fera via une requête à la base de données.

3 Importation de fichier

L'utilisateur devra importer un fichier csv qui est exporté par son programme branch and bound. Le format de fichier sera selon notre règle. Pour obtenir les attributs nécessaires pour la visualisation, l'utilisateur devra insérer certaines lignes de code dans son programme quand un nœud est créé, branché et coupé. Le fichier devra contenir au moins les attributs suivants :

- L'état de nœud (attendu, exploré, coupé).
- La valeur LB (Lower bound) de nœud.
- La solution partielle de nœud.
- Le parent de nœud.
- Le temps.

4 Coloration de nœud

Cette fonctionnalité va permettre aux utilisateurs de bien connaître l'état de chaque nœud. Dans notre application, un nœud représentera une solution partielle. La couleur de nœud représentera l'état de nœud. Au cours de la visualisation, il existe 3 états de nœud (attendu, exploré, coupé). Après, on pourra replier et dérouler un nœud. Quand un nœud est replié, on l'appelle dans un état « replié ». Quand l'état du nœud est changé, on devra changer aussi la couleur du nœud. Pour distinguer les 4 différents états, on donnera chaque état une couleur unique.

- L'état attendu (un nœud a déjà été créé mais n'a pas encore été branché ou été coupé) : Bleu.
- L'état exploré (un nœud a déjà été branché) : Vert.
- L'état coupé (un nœud a été coupé) : Rouge.
- L'état replié (un nœud a été fusionné avec ses fils) : Orange.

5 Affichage de la solution partielle

Cette fonctionnalité va permettre aux utilisateurs de savoir la solution partielle. Quand on met la souris sur un nœud (mouseover), un label qui contient l'information de la solution partielle du nœud sera affiché à côté du nœud.

6 Affichage de la borne inférieure

Cette fonctionnalité va permettre aux utilisateurs de savoir la borne inférieure du nœud. Quand l'utilisateur double clique un nœud, la borne inférieure du nœud sera affichée dans un champ à l'extérieur de l'arbre.

7 Affichage du temps calcul lié

Cette fonctionnalité va permettre aux utilisateurs de savoir le temps dépensé par un nœud. Quand l'utilisateur double clique un nœud, le temps calcul lié du nœud sera affiché dans un champ à l'extérieur de l'arbre.

8 Choix de mode d'exécution

Notre application permettra l'utilisateur de choisir le mode de visualisation (dynamique ou statique).

9 Visualisation statique

La visualisation statique va afficher un arbre entier dans une seule fois. Cette fonctionnalité va permettre aux utilisateurs de voir leur résultat de visualisation avec peu d'attente. Si le nombre de nœuds est trop, par exemple cent mille, la visualisation dynamique va prendre longtemps. Dans le cas, la visualisation est un bon choix.

10 Visualisation dynamique

La visualisation dynamique va afficher dynamiquement un arbre, nœud par nœud. Cette fonctionnalité va permettre aux utilisateurs de bien regarder et comprendre tout le processus de la visualisation.

11 Contrôle de vitesse

Quand l'utilisateur est dans le mode d'exécution dynamique. Il pourra choisir une vitesse d'affichage (milli-second par nœud). Cette fonctionnalité va permettre aux utilisateurs de contrôler la vitesse de visualisation selon leur besoins.

12 Pause et redémarrage

Quand l'utilisateur est dans le mode d'exécution dynamique. Au cours de la visualisation, il pourra demander un arrêt temporaire et puis redémarrer. Cette fonctionnalité va permettre aux utilisateurs de bien contrôler le processus de visualisation selon leur besoins.

13 Replier et dérouler les nœuds

Quand le processus de visualisation est fini, l'utilisateur aura un arbre entier avec tous les nœuds sont affichés. Dans le cas, s'il y a des branche ou des nœuds qu'il s'intéresse pas, il pourra fusionner un nœud et tous ses fils par cliquer le nœud. Après, il pourra dérouler le nœud par cliquer une autre fois. Cette fonctionnalité va permettre aux utilisateurs d'ajuster la structure de l'arbre selon leur besoins.

14 Zoom

Quand le processus de visualisation est fini ou dans une pause, l'utilisateur pourra zoomer l'arbre avec rouler la souris (mousewheel). Parce que si le nombre de nœuds est trop, par exemple un million, on ne pourra pas voir clairement tous les nœuds. Cette fonctionnalité va bien résoudre ce problème.

15 Sauvegarde de fichier

Cette fonctionnalité va permettre aux utilisateurs enregistrés de sauvegarder leur fichier dans notre serveur afin de faciliter leur visualisation la prochaine fois. Un utilisateur pourra sauvegarder au maximum trois fichiers.

16 Consultation l'historique de fichier

Cette fonctionnalité va permettre aux utilisateurs enregistrés de consulter l'historique de fichier importé.

C

Spécifications non fonctionnelles

1 Contraintes de développement

Il s'agit d'une application Web qui devra donc nécessiter d'un hébergement spécialisé. Nous avons choisi la solution Node.js et MySQL. L'ensemble des développements sera effectué avec les langages du Web tel que HTML5, CSS3, JavaScript pour le développement de l'application. Le langage SQL sera utilisé pour la construction et gestion de la base de données.

Pour faciliter la partie visualisation, nous avons choisi la librairie D3.js. C'est une bibliothèque graphique JavaScript qui permet l'affichage de données numériques sous une forme graphique et dynamique. Il s'agit d'un outil important pour la conformation aux normes W3C qui utilise les technologies courantes SVG, JavaScript et CSS pour la visualisation de données.

Le développement s'effectuera sur le système d'exploitation Windows. Toutefois, des tests d'intégration réguliers seront effectués sur un environnement pré-production tournant sous un système d'exploitation Unix : Ubuntu 14.04 LTS. L'application sera testée à l'aide de deux navigateurs web qui sont Google Chrome et Mozilla Firefox.

2 Contraintes de fonctionnement

2.1 Performance

La vitesse maximum de visualisation dépend fortement le nombre de nœuds. Si le nombre est trop, la vitesse maximum sera ralentie. La performance de navigateur diminuera aussi quand le nombre de nœuds est trop. Pour garantir la performance de l'application, c'est mieux que le nombre de nœuds ne dépasse pas cent mille.

2.2 Capacité

Dans le but de stocker les données relatives à notre application, nous utilisons le système de gestion de base de données MySQL permettant de supporter jusqu'à 4 294 967 295 connexions simultanées. Ce qui est amplement suffisant pour le bon fonctionnement de notre application.

2.3 Sécurité

L'application web, nécessitant une authentification de la part des utilisateurs, devra forcer l'utilisation du protocole sécurisé HTTPS. L'hébergement devra en conséquence supporter une implémentation de la suite SSL. La communication avec les API tierces devra aussi s'effectuer par ce protocole pour éviter une fuite ou une falsification des données.

D

Cahier du développeur

1 Utilisation des librairies

- D3.js 3.1.0
- JQuery.js 3.1.1

2 Environnement de développement

- Navigateur : Google Chrome 56.0.1 et Mozilla Firefox 51.0.1
- Système d'exploitation : Windows 8 et Ubuntu 12.04.1
- Éditeur de texte : Sublime Text 2.

3 Versionning

- Online repository : GitHub.
- Client management : GitHub Desktop.

4 Code documentation

La documentation du code est générée avec JSDuck. Les commentaires sont forcément respectés les spécifications de JSDoc.

1 Fonctionnalités

Les fonctionnalités de l'application sont montrées dans la **Figure 1**.

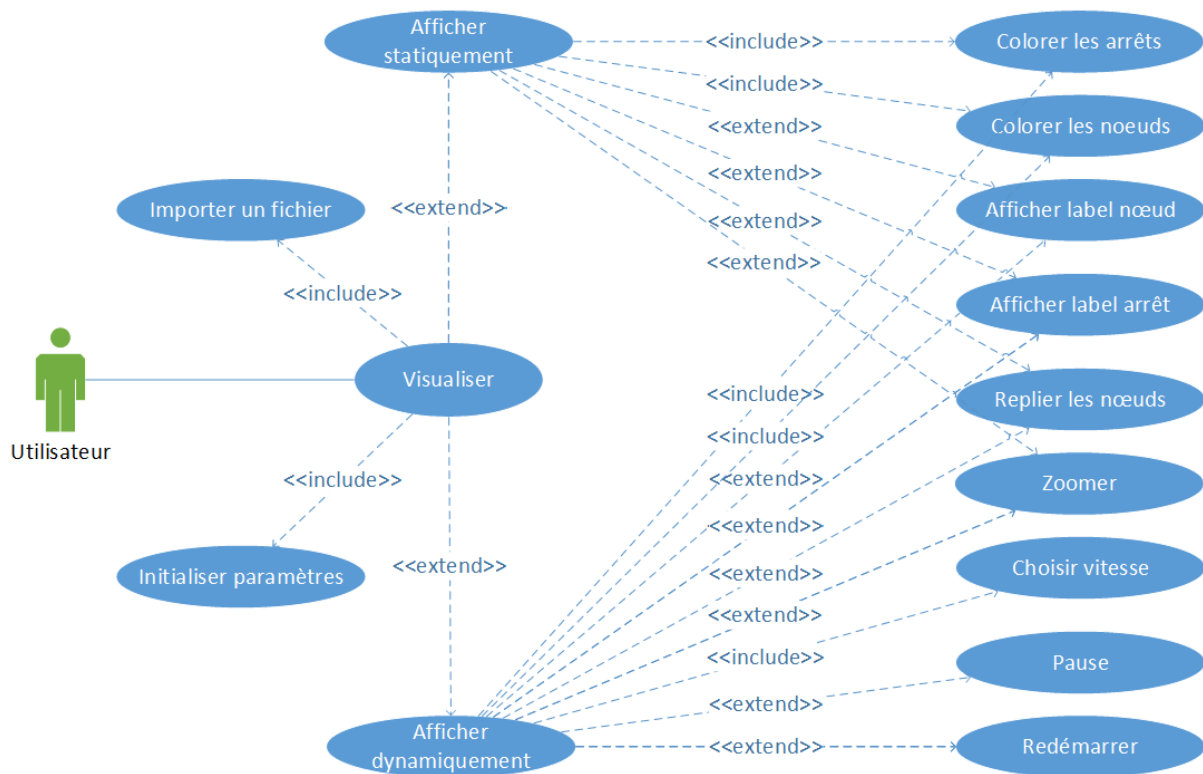


Figure 1 – Les fonctionnalités de l'application

2 Installation

Notre application n'a pas besoins d'installer. C'est juste d'entrer dans le répertoire de l'application et ouvrir le fichier **viz.html** dans le navigateur.

3 Format de fichier

Le format de fichier sera forcément selon notre règle. Chaque ligne sera sous un format **Action, Id, PId, LabelNode, LabelLink** :

- Action : la commande de l'action. Notre application fournit une suite de commande pour préciser les actions différentes. Voici les commandes :
 - **C** : Un nœud est créé. Cela peut être **C1**, **C2** pour distinguer les différentes façons de création.
 - **B** : Un nœud est branché.
 - **K** : Un nœud est coupé. Cela peut être **K1**, **K2** pour distinguer les différentes façons de coupe.
 - **U** : Un nœud est utilisé pour couper un autre nœud.
- Id : le numéro de nœud.
- PId : le numéro de parent de nœud. Cela peut être vide si le nœud a été créé.
- LabelNode : un String pour décrire le nœud. Cela peut être une borne inférieure ou extérieure, une solution partielle, un temps calcul lié etc. Cela peut être vide.
- LabelLink : un String pour décrire l'arrêt. Cela peut être vide.

F

Cahier de test

1 Fiches de tests

Afin de générer les données pour tester, nous avons écrit un programme pour simuler le résultat de la méthode B&B. Le résultat est complètement aléatoire.

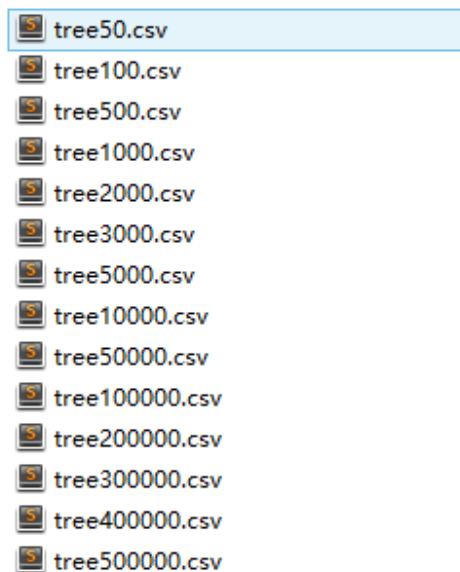


Figure 1 – Fiches de tests

2 La qualité

Le temps d'exécution augmente avec le nombre de nœuds. Quand le nombre de nœuds est très grand, la performance de l'application va diminuer.

Afin de garantir une expérience utilisateur optimale. Pour la mode statique, c'est mieux que le nombre de nœuds ne dépasse 200000. Le résultat de test est montré dans la [Figure 2](#).

Nombre de nœuds	Temps d'exécute
100	9ms
500	37ms
1000	73ms
5000	410ms
10000	1043ms
50000	13295ms
100000	51267ms
200000	175382ms
300000	370348ms
400000	634291ms
500000	Navigateur bloqué

Figure 2 – Résultat de tests

Pour la mode dynamique, c'est mieux que le nombre de nœuds ne dépasse 5000. Le résultat de test est montré dans la Figure 3.

Nombre de nœuds	Temps d'exécute
100	563ms
500	15115ms
1000	46678ms
2000	206449ms
3000	493034ms
5000	1248431ms
10000	5132262ms

Figure 3 – Résultat de tests



Webographie

- [2] Mike BOSTOCK. *Radial Tidy Tree*. URL : <http://bl.ocks.org/mbostock/4063550>.
- [5] Sebastian LEIPERT. *VBCTOOL - a graphical interface for Visualization of Branch Cut algorithms*. URL : http://www.informatik.uni-koeln.de/ls_juenger/vbctool/.



Bibliographie

- [1] David L APPLEGATE, Robert E BIXBY, Vasek CHVATAL et William J COOK. *The traveling salesman problem : a computational study*. Princeton university press, 2011.
- [3] JJH FORREST, JPH HIRST et JOHN A TOMLIN. « Practical solution of large mixed integer programming problems with UMPIRE ». In : *Management Science* 20.5 (1974), p. 736–773.
- [4] A. H. LAND et A. G. DOIG. « An Automatic Method of Solving Discrete Programming Problems ». In : *Econometrica* 28.3 (1960), p. 497–520. issn : 00129682, 14680262. URL : <http://www.jstor.org/stable/1910129>.
- [6] Osman Y OZALTIN, Brady HUNSAKER et Ted K RALPHS. « Visualizing Branch-and-Bound Algorithms ». In : URL <http://www.rosemaryroad.org/projects/BAK> (2007).
- [7] John Q WALKER. « A node-positioning algorithm for general trees ». In : *Software : Practice and Experience* 20.7 (1990), p. 685–705.
- [8] Tong ZHANG, Gaofeng HUA et Arika LIGMANN-ZIELINSKA. « Visually-driven parallel solving of multi-objective land-use allocation problems : a case study in Chelan, Washington ». In : *Earth Science Informatics* 8.4 (2015), p. 809–825.

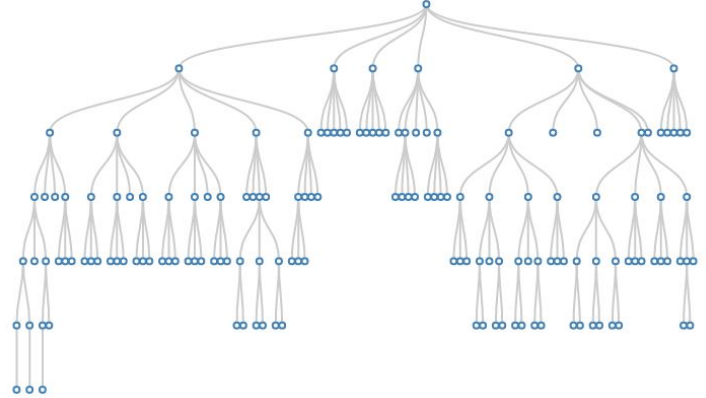
Visualisation des procédures de branchement :

Xiangyu MENG

Encadrement : Lei SHANG

Objectifs

- Visualiser la méthode Branch and Bound.
- Trouver des propriétés intéressantes dans l'arbre Branch and Bound.



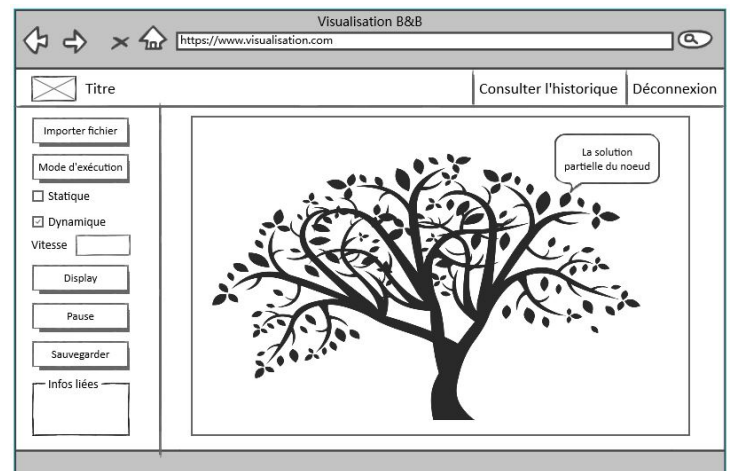
Mise en œuvre

- Développer un Application Web.
- Réaliser la visualisation dynamique et statique.



Résultats attendus

- Visualisation dynamique.
- Gestion de fichier.
- Gestion de compte.



Visualisation des procédures de branchement :

Xiangyu MENG

Encadrement : Lei SHANG

Objectifs

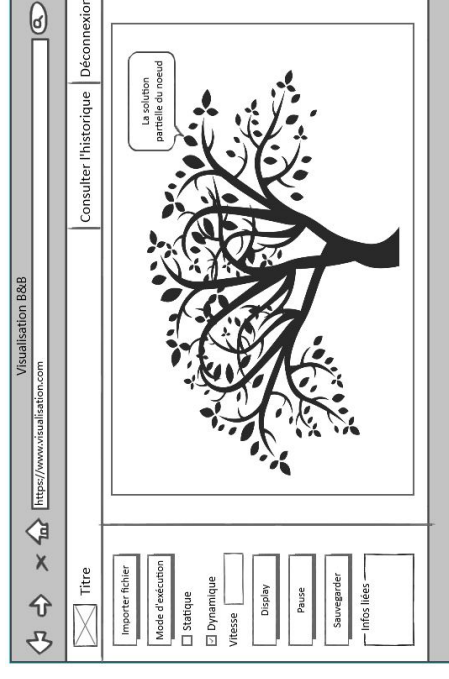
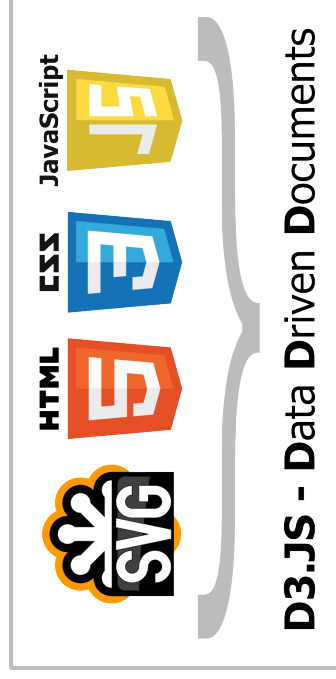
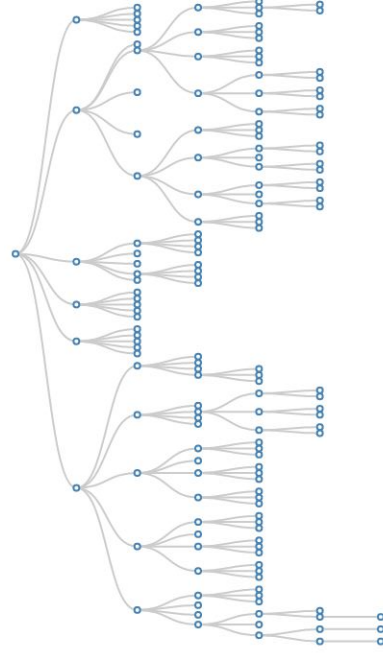
- Visualiser la méthode Branch and Bound.
- Trouver des propriétés intéressantes dans l'arbre Branch and Bound.

Mise en œuvre

- Développer un Application Web.
- Réaliser la visualisation dynamique et statique.

Résultats attendus

- Visualisation dynamique.
- Gestion de fichier.
- Gestion de compte.



Visualisation des procédures de branchement

Résumé

Dans le cadre du projet recherche et développement , nous travaillons sur le développement d'une application web pour visualiser la Procédure par Séparation & Evaluation. Ce document a pour but de présenter le plus précisément possible les spécifications, l'état de l'art, l'analyse et la mise en œuvre de ce projet.

Mots-clés

Procédure par Séparation & Evaluation, Visualisation, Arbre

Abstract

As part of this Project of Research & Development, we work on the development of a web application to visualize the algorithm Branch and Bound. The aim of this document is to present the specifications, the state of the art, the analysis and the implementation of this project as precisely as possible.

Keywords

Branch and Bound, Visualization, Tree