

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 2017-2018

Optimisation du chargement d'un camion



Entreprise

KeenSaas



KeenSaaS

Tuteurs entreprise

Frédéric PERRIN

Bruno BOUDINSKI

Étudiant

Jean CELLIER (DI5)

Tuteur académique

Jean-Charles BILLAUT

31 mars 2018

Liste des intervenants

Entreprise

KeenSaas
1 Avenue du Champ de Mars, 45100 Orléans
www.keensaas.fr/



Nom	Email	Qualité
Jean CELLIER	jean.cellier@etu.univ-tours.fr	Étudiant DI5
Jean-Charles BILLAUT	jean-charles.billaut@univ-tours.fr	Tuteur académique, Département Informatique
Frédéric PERRIN	frederic.perrin@keensaas.fr	Tuteur entreprise
Bruno BOUDINSKI	bruno.boudinski@keensaas.fr	Tuteur entreprise



Avertissement

Ce document a été rédigé par Jean Cellier susnommé l'auteur.

L'entreprise KeenSaas est représentée par Frédéric Perrin et Bruno Boudinski susnommés les tuteurs entreprise.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Jean-Charles Billaut 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

Jean Cellier, *Optimisation du chargement d'un camion*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2017-2018.

```
@mastersthesis{
  author={Cellier, Jean},
  title={Optimisation du chargement d'un camion},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2017-2018}
}
```

Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iv
Liste des Algorithmes	vi
Remerciements	1
I Introduction	2
1 Acteurs et contexte	2
1.1 KeenSaas	2
1.2 Contexte	3
2 Présentation du problème	3
2.1 Contraintes liées aux éléments	3
2.2 Contraintes liées aux conteneurs	4
3 Objectifs et enjeux	4
4 Bases méthodologiques	5
II Description générale	7
5 Environnement du projet et structure du système	7

6	Caractéristiques des utilisateurs	8
7	Fonctionnalités du système	8
III	Etat de l'art et veille	10
8	Le problème dans la littérature	10
8.1	Définition et déclinaisons	10
8.2	Les contraintes	11
8.2.1	Contraintes relatives aux objets	11
8.2.2	Contraintes relatives aux conteneurs	12
8.2.3	Autres contraintes	12
9	Les solutions existantes	13
9.1	Une heuristique utilisant la méthode du Wall Building Approach	13
9.2	Un algorithme de parcours d'arbre utilisant la méthode du Block Building Approach	14
9.3	Un modèle mathématique à implémenter sur un solveur	16
10	Conclusion et choix	17
IV	Analyse et conception	18
11	Phase de recherche	18
11.1	Entités	18
11.2	Les solutions	19
11.3	Essais sous LocalSolver	20
12	Phase de Développement	21
12.1	Les classes et leurs liaisons	21
12.2	Les limites de cette modélisation	23
V	Mise en œuvre	25
13	L'algorithme et les procédures majeures	26
14	Technologies et instances de données	30
15	Contraintes prises en compte	32
16	Les solutions et les indicateurs de qualité	34
16.1	Multiples solutions	34
16.2	Qualité des solutions	36
17	Tests et étude des résultats	37
18	Limites de l'algorithme	39
VI	Bilan et conclusion	42
19	Perspectives	43
20	Bilan personnel et conclusion	44

Annexes	45
A Reste du cahier de spécifications	46
1 Interfaces	46
1.1 Interfaces matériel/logiciel	46
1.2 Interfaces logiciel/logiciel	46
1.3 Interfaces homme/machine.....	46
2 Spécifications fonctionnelles.....	48
3 Spécifications non fonctionnelles.....	49
3.1 Contraintes de développement et conception.....	49
3.2 Contraintes de fonctionnement et d'exploitation	49
3.2.1 Performances	49
3.2.2 Capacités et contrôlabilité.....	50
B Gestion de projet	51
1 Phase de recherche	51
1.1 Gantt et plan de développement	51
1.2 Cycles et outils	56
2 Phase de développement	57
2.1 Ecart des plannings réel et prévisionnel	57
2.2 Gitlab	58
Comptes rendus hebdomadaires	60
Bibliographie	63

Table des figures

Remerciements

1	Système de KsFulltruck.....	7
2	Diagramme de cas d'utilisation	9
3	Wall Building Approach.....	14
4	Notion de bloc.....	15
5	Les espaces résiduels après placement d'un objet.....	15
6	L'entité Element	19
7	LocalSolver	20
8	Diagramme de classe	21
9	Format de fichier d'instance.....	31
10	Calcul de la hauteur de chargement	33
11	Stratégie de swap	35
12	400 objets avec α à 1	37
13	400 objets avec α à 0.9	38
14	400 objets avec α à 0.6	38
15	278 objets avec α à 1	39
16	278 objets avec α à 0.3	39

A Reste du cahier de spécifications

1	Visualisateur 3D développé par KeenSaas.....	47
2	Format de fichier d'entrée pour le visualisateur.....	47

B Gestion de projet

1	Diagramme de Gantt	55
---	--------------------------	----

2	CRAH : Compte-rendu d'activité hebdomadaire	56
3	Ecart des plannings	57
4	Gitlab	58
5	Une issue sur Gitlab	59



Liste des Algorithmes

1	Compute	27
2	doesObjectFit	29
3	placeObject	29



Remerciements

Je tiens à remercier Monsieur Jean-Charles Billaut, enseignant chercheur à Polytech Tours, pour son rôle d'encadrant dans ce projet. Ses conseils, son expérience et ses connaissances ont été nécessaires pour réaliser les tâches les plus ardues.

J'adresse également mes remerciements à Messieurs Bruno Boudinski et Frédéric Perrin, fondateurs de l'entreprise KeenSaas, et clients de ce projet, pour la confiance qu'ils m'ont accordé dans cette réalisation, et leur gentillesse lors de nos différentes rencontres.

Première partie

Introduction

Ce document a pour but de présenter le Projet de Recherche et Développement “Optimisation de chargement d’un camion”. Ce projet est proposé par l’entreprise KeenSaas, basée à Orléans et spécialisée dans les solutions Web. KeenSaas est donc la Maîtrise d’Ouvrage de ce projet. La Maîtrise d’Oeuvre comporte trois étudiants, encadrés par Monsieur Jean-Charles Billaut : Jean Cellier, étudiant assigné au PRD et auteur de ce document, Morad Sanba et Omar Elhachimi, deux étudiants de quatrième année qui vont rapidement travailler sur la réalisation. Ce projet se déroule sur l’ensemble de l’année, avec un rythme de deux journées, Mercredi et Jeudi, de travail par semaine.

1 Acteurs et contexte

1.1 KeenSaas

KeenSaas est une société qui édite des logiciels en mode SaaS ("Software As A Service"), basée à Orléans. Elle a été fondée en 2013, par Frédéric Perrin et Bruno Boudinsky.

Elle propose des logiciels uniquement destinés aux entreprises, dans des secteurs divers et variés, tels que l’artisanat, les TPE, la logistique, et d’autres.

Les solutions réalisées sont donc orientées Web, qu’elles soient de simples sites vitrines ou des application plus complexes. Les technologies utilisées sont assez variées, mais tournent souvent autour de Ruby on Rails et Bootstrap.

En plus de cela, KeenSaas propose des services d’hébergement ou d’administration d’applications open-source.

L’un de leur dernier projet est KsFulltruck, qui consiste à proposer une solution Web permettant de calculer et visualiser le chargement optimal de camions, sur laquelle est basée le travail de ce projet.

1.2 Contexte

En optimisation et recherche opérationnelle, le problème d'optimisation de chargement d'un conteneur est très commun, et a été étudié à maintes reprises depuis les années 1980. On le qualifie souvent de “Container loading problem”, et possède de nombreuses déclinaisons, qui diffèrent selon le contexte d'application, les contraintes ou bien les choix algorithmiques.

Dans la littérature, le problème est couramment exprimé comme suit, en version simple : on a une liste d'éléments, objets, palettes, cartons, de forme rectangulaire ou non, qu'on souhaite ranger dans un ou plusieurs camions de formes rectangulaires. L'objectif est simplement d'optimiser le chargement des éléments afin de réduire au maximum le nombre de camions à utiliser.

La société KeenSaas souhaiterait développer un algorithme permettant de résoudre le “container loading problem”, mais avec des contraintes bien particulières.

2 Présentation du problème

Commençons par définir deux termes importants : les éléments et les conteneurs. Les éléments représentent les objets à “ranger” dans le conteneur. Il s'agit dans notre problème d'un carton ou d'une palette. On considérera qu'il a une forme parallélépipédique dans un premier temps. En cas d'avancement rapide dans le traitement des éléments rectangulaires, on pourra considérer des éléments d'autres formes (cylindriques, forme de T, etc.). Les conteneurs sont, eux, assimilés à des camions, trains, bateaux, où l'on stocke ces éléments. On considère qu'il s'agit de camions de forme rectangulaires dans notre problème. L'objectif de l'algorithme est de dresser un plan de chargement où les objets sont rangés d'une telle manière, tel que le nombre de conteneurs mobilisés soit le plus faible possible.

Souvent dans la littérature, le “container loading problem” est divisé en deux sous problèmes : la construction des palettes, et le chargement optimisé. Dans notre cas, on ne considérera que la seconde partie, dans un premier temps. On suppose que les palettes sont déjà constituées.

2.1 Contraintes liées aux éléments

- Degré de liberté sur les 3 axes : cela représente la possibilité de tourner un élément sur les trois axes dimensionnels. On envisagera sûrement d'ajouter un attribut booléen pour chaque axe et pour chaque élément, disant si oui ou non ce même élément peut effectuer une rotation sur cet axe. Par exemple, on ne transportera jamais un frigo à plat, ou une machine à laver sur le côté.
- Gerbable : si un élément peut accepter sur sa face supérieure un autre élément, soit identique soit différent.

- Obligatoire : valeur booléenne disant si un élément doit obligatoirement être livré lors de cette livraison. Cette valeur passe à TRUE si l'élément en question n'a pas pu être livré lors de la dernière livraison, alors qu'il était dans la liste des éléments en entrée, pour cause de manque d'espace dans un camion par exemple. Il est donc impératif de le livrer en priorité à la prochaine livraison pour ne causer aucun retard.
- Priorité dans l'ordre de chargement : souvent, les objets sont rangés suivant différentes catégories, pour chaque client. On essaiera d'abord d'organiser le camion en n parties, s'il y a n client. Bien sûr, la partie accessible en premier sera celle du premier client à livrer. Pour chaque partie, on la divisera en sous-parties, représentant les différentes catégories d'objets. Par exemple, d'abord le gros électroménager, puis le petit électroménager, puis l'ameublement.
- Poids maximum : le poids maximal qu'un objet peut porter sur sa face supérieure, si il est gerbable.
- Hauteur de chargement maximum : la hauteur maximale où un objet peut être placé.

2.2 Contraintes liées aux conteneurs

- Poids total : la charge maximale que peut supporter le camion. La somme des charges des éléments chargés doit être inférieure à cette valeur.
- Poids sur chaque essieu : un camion possède plusieurs essieux. Il est important de répartir le poids total équitablement sur chaque essieux. Néanmoins, un essieu possède aussi un poids maximal qu'il peut supporter. Il faut donc réussir à répartir la charge sur les essieux sans dépasser leur capacité de charge individuelle.
- Centrage de la charge : si on prend le camion dans sa largeur, il est important de centrer, ou équilibrer la charge. Alors que la répartition de la charge sur les essieux est plus une répartition sur la longueur du camion, le centrage est une répartition sur la largeur. Cette contrainte est importante pour éviter tout penchement lors de la conduite, dans les virages par exemple.
- Chargement par le côté ou arrière : les ouvertures d'un camion peuvent être diverses. Le plus souvent, on a une porte arrière et on charge en remplissant du fond vers l'avant. Cependant, dans notre cas, on pourra également considérer le cas d'une porte latérale, ce qui change complètement un plan de chargement classique.

3 Objectifs et enjeux

L'objectif principal du projet consiste donc à élaborer un algorithme permettant de résoudre ce problème de chargement de camion, en respectant l'ensemble des contraintes citées plus haut. Pour ce faire, on va probablement s'inspirer d'algorithmes établis de la littérature. Cependant, l'ensemble de nos contraintes font que notre problème est original, et qu'aucune solution y répondant parfaitement n'existe. C'est pour cela qu'on viendra adapter ces algorithmes, peut être en combiner, et apporter notre partie, nouvelle, afin de réaliser complètement cet objectif.

On pourrait s'interroger sur l'utilité d'un tel outil. Premièrement, il apporterait un gain de temps global non négligeable aux entreprises effectuant des livraisons régulières. Plus précisément, le fait d'avoir un plan de chargement généré automatiquement évite de le construire mentalement, par les chargeurs ou les logisticiens, potentiels utilisateurs de cette application. Même si l'expérience de certains chargeurs est parfois suffisante à garantir un chargement optimisé, les nouveaux dans ce métier gagneraient un temps significatif. Si la contrainte du plan de chargement n'est plus à prendre en compte, on gagne déjà le temps de la réflexion. De plus, avec un plan établi à l'avance, si on connaît par exemple à l'avance ce que l'on va charger, on peut s'organiser plus efficacement pour réunir l'ensemble des éléments au préalable, juste avant le chargement. Ainsi, on gagne encore du temps.

Si on met l'aspect temporel de côté, on a évidemment l'aspect financier qui rentre en compte. En effet, l'un des objectifs principaux de l'algorithme est d'optimiser le chargement afin de réduire au maximum le nombre de camions à utiliser. Moins nous avons de camions mobilisés, moins nous avons de carburant à utiliser. Moins nous consommons de carburant, plus nous économisons de l'argent. C'est également plus écologique. Enfin, dans le cas où on doit louer des camions pour effectuer nos livraisons, le gain financier est évident si on en a moins à louer.

Sur le plan personnel, l'enjeu pour moi est ici de réussir à élaborer un algorithme suffisamment performant pour permettre de répondre aux besoins émis, tout en rendant le projet KsFulltruck le plus attractif possible. Cela permettrait à KeenSaas d'imposer leur nouveau produit et d'attirer un maximum de potentiels clients.

4 Bases méthodologiques

D'un point de vue méthodologie, je souhaite mettre en place différents outils afin d'optimiser le développement. Tout d'abord, une méthodologie plutôt orientée agile sera utilisée. En effet, tout comme nous fonctionnons actuellement, je ferais un compte rendu de l'avancement du travail toutes les deux semaines à mon encadrant et aux professionnels. En phase de recherche, aucune démonstration n'était vraiment nécessaire. Cependant, nous considérerons pour la phase de développement d'éventuelles démonstrations de l'algorithme. Tout cela dans le but de garantir une vision commune et harmonisée des objectifs par rapport à la réalisation, et ce tout au long du projet.

Deuxièmement, je mettrai en place un outil de versionning du code dès l'entrée en phase de développement. Typiquement, je pense fortement à Gitlab. Cet outil permet, en plus de versionner, de créer et visualiser des issues, et ainsi garder constamment un backlog de fonctionnalités/-tâches à accomplir. C'est un outil de gestion de projet très riche que je souhaite utiliser. Pour plus de détails concernant les méthodes de gestion de projet, je vous oriente vers l'annexe en question.

La communication avec les deux étudiants de quatrième année intervenant sur le projet se fait de manière assez régulière. Comme leur travail consiste à tester des algorithmes directement, je vais souvent aux nouvelles pour savoir comment les choses avancent, et les résultats qu'ils obtiennent. De plus, des réunions avec M. Billaut sont parfois organisées pour faire le point ensemble. Si leur travail propose des résultats satisfaisants, je repartirai de là pour entamer la partie orientée développement.

Deuxième partie

Description générale

5 Environnement du projet et structure du système

Ce projet s'inscrit directement dans le projet global de l'entreprise KeenSaas appelé KsFulltruck. Ce dernier a pour objectif de fournir un outil accessible depuis une URL web, permettant l'optimisation d'un chargement d'éléments divers dans un camion. Plus précisément, l'utilisateur fournit sa liste d'objets à l'outil, et celui-ci, après traitement, lui renvoie un plan de chargement optimisé en trois dimensions, où l'opérateur peut visualiser où il doit placer chaque objet.

Les développeurs de KeenSaas sont plutôt spécialisés dans la création d'éléments Web, et ont ainsi déjà créé la plateforme de visualisation du plan de chargement, accessible directement depuis un navigateur internet. Cependant, l'algorithme permettant de générer ce plan de chargement reste à réaliser.

Le périmètre de mon projet porte sur l'élaboration d'un algorithme permettant de répondre à la problématique imposée. L'ensemble du processus est représenté dans le schéma suivant :

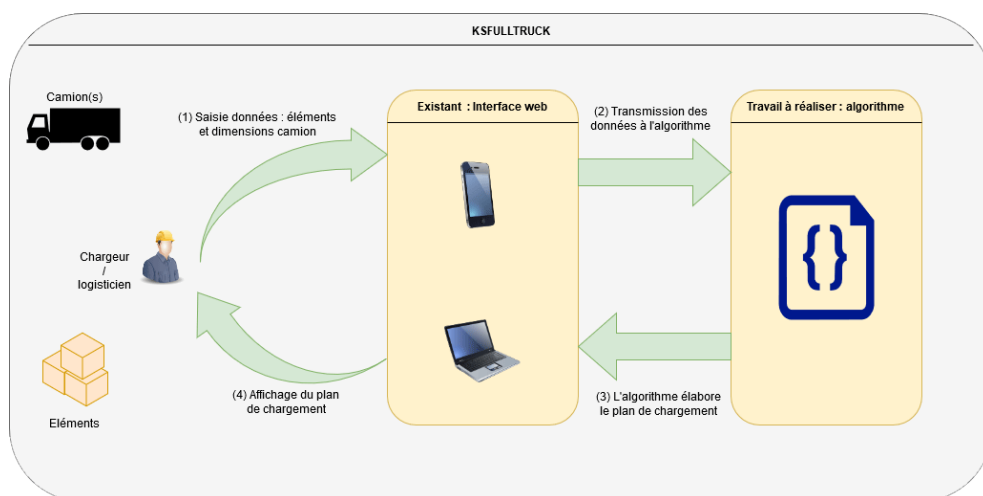


Figure 1 – Système de KsFulltruck

On distingue donc trois blocs importants dans KsFulltruck. Le premier représenterait l'utilisateur de l'application, ainsi que les données d'entrées, c'est-à-dire les éléments et les camions. Ces données sont injectées par l'utilisateur dans le second bloc, déjà existant : l'interface de visualisation du plan de chargement. Enfin, le troisième bloc représente l'algorithme de calcul du plan de chargement à implémenter. Le second et le troisième bloc interagissent donc ensemble : après passage des données de l'interface vers l'algorithme, ce dernier renvoi un plan de chargement optimisé à la vue, qui l'affiche directement à disposition de l'utilisateur.

6 Caractéristiques des utilisateurs

L'algorithme à développer va servir aux chargeurs et logisticiens des entreprises voulant effectuer les livraisons. Typiquement, des chargeurs sont responsables de plusieurs tâches :

- aller chercher les différents éléments à livrer dans le ou les entrepôts.
- charger tous ces éléments dans le ou les camions de livraison.
- effectuer les livraisons aux clients.

C'est donc à ces chargeurs qu'est destinée l'application utilisant l'algorithme voulu. On imagine qu'avant l'étape de chargement, un chargeur transmet tous les éléments qu'il doit charger à l'application, et celle-ci lui retourne un plan de chargement optimisé.

On suppose donc que les utilisateurs ne sont pas familiers avec l'informatique. On peut également supposer que certains chargeurs peuvent être nouveaux dans le métier, et d'autres avoir beaucoup d'expérience. Ces derniers peuvent parfois effectuer un chargement optimal sans aucune aide du fait de leur connaissance et habitudes. Au contraire, l'outil à développer prend tout son sens pour les nouveaux chargeurs, à qui un plan de chargement optimal sera immédiatement proposé, ce qui fera gagner un temps considérable. Si on prend par exemple But qui fait livrer tous ses magasins en France trois fois par semaine, on peut s'imaginer que les utilisateurs de l'application seront relativement réguliers, si l'outil est utilisée à chaque fois.

7 Fonctionnalités du système

Dans notre cas, les fonctionnalités du système sont très peu nombreuses. On peut les retrouver sur le diagramme de cas d'utilisation représenté sur la [Figure 2](#).

Deux cas d'utilisation sont ici exécutables à chaque utilisation. D'abord, "Saisir données", où l'utilisateur de l'application, typiquement un chargeur, viendra saisir l'ensemble des données nécessaires à la résolution de l'algorithme, c'est-à-dire les dimensions de tous les éléments à charger, et les dimensions du camion. Le second est "Visualiser plan de chargement", et ne peut s'effectuer que lorsque l'algorithme a terminé de générer le plan de chargement.

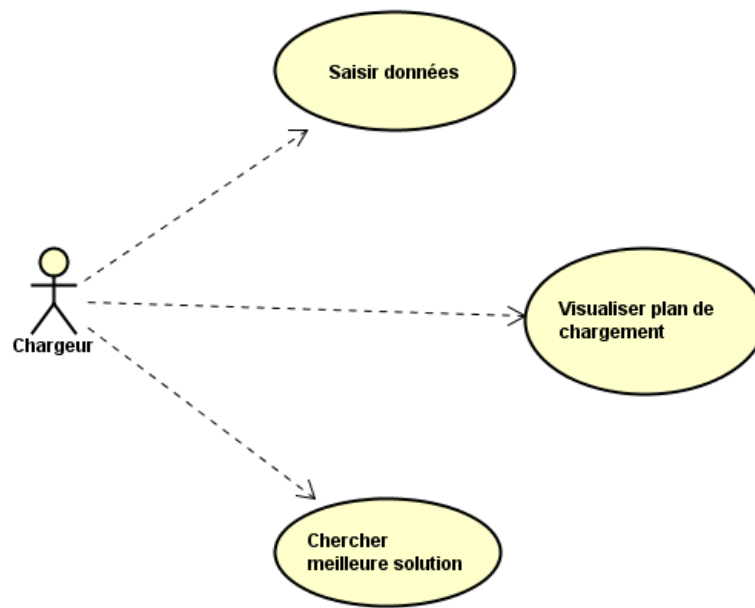


Figure 2 – *Diagramme de cas d'utilisation*

Un troisième cas d'utilisation peut être identifié, "Chercher meilleure solution". En effet, dans le cas où la solution de plan de chargement proposée par l'algorithme ne plaît pas à l'utilisateur, ce dernier peut éventuellement demander à l'outil de recommencer à chercher une autre solution, différente de la première.

Troisième partie

Etat de l'art et veille

Le Container Loading Problem reste l'un des problèmes les plus traités et répandus dans le domaine de la recherche opérationnelle. Malgré cela, il est tellement complexe et varié qu'aucune solution proposée dans l'histoire ne permet de répondre à tous les cas particuliers. Comme l'ont attesté Bischoff et Ratcliff(1995), "les solutions proposées dans la littérature ne répondent qu'à une étroite partie du spectre des possibilités des situations de ce problème". Cette partie a pour but de faire un état de l'art à propos de la littérature concernant le Container Loading Problem. Nous allons d'abord présenter en détail ce problème : sa définition et ses grandes déclinaisons. Ensuite, nous listerons toutes les contraintes, en essayant d'être le plus exhaustif possible. Dans un second temps, nous nous pencherons sur certaines publications de la littérature, explicitant les approches et algorithmes principaux permettant de fournir une réponse au problème.

8 Le problème dans la littérature

8.1 Définition et déclinaisons

Le Container Loading Problem, ou Problème de Chargement de Conteneur en français, est un problème géométrique spatial. Typiquement, on cherche à "ranger" des objets tri-dimensionnels, appelés items, dans un conteneur tri-dimensionnel de forme parallélépipédique. Ranger l'ensemble, ou un sous ensemble des objets, génère ce que l'on appelle un plan de chargement. Ce dernier représente tout simplement la position exacte de chaque objet chargé dans le conteneur. Très souvent, le but des algorithmes permettant de résoudre ce problème est de charger le maximum d'objets dans un conteneur d'un certain volume.

On peut venir classer ce problème en trois catégories principales. Chaque catégorie est définie par une nature de l'ensemble des objets à charger différente. En effet, on parlera d'un problème homogène si les objets à charger sont identiques. On parlera ensuite de problème faiblement hétérogène si les objets à charger ne possèdent que peu de types différents. Enfin, on parlera de problème fortement hétérogène si les objets à charger sont très différents, et donc que le nombre de types d'objet est très grand. Dans certaines applications, on peut également parfois disposer de plusieurs conteneurs. Dans ce cas, on pourra également considérer des sous catégories de problème, avec un ensemble homogène de conteneurs, où tous les conteneurs sont identiques, ou bien un ensemble hétérogène de conteneurs où tous les conteneurs n'ont pas de volume identique.

Avec toutes ces possibilités, on peut très vite voir que le nombre de déclinaisons de ce problème est extrêmement grand, rien qu'en faisant varier l'homogénéité des ensembles d'objets et/ou des conteneurs. Et nous n'avons encore même pas évoqué les contraintes possibles. Selon la nature des ensembles d'objets ou de conteneurs, la littérature assimile parfois le problème à un Bin Packing, ou un Knapsack.

8.2 Les contraintes

Depuis les années 1980, on décompte presque 200 papiers différents publiés répondant au Container Loading Problem. Presque dans chaque papier, les contraintes relatives aux objets et au(x) conteneur(s) sont différentes. Par définition, un papier permet souvent de répondre à un problème unique en termes de contraintes, et il est donc difficile d'avoir une solution répondant à tout type de problème, comme dit précédemment.

Ainsi, on imagine bien que le nombre de problème différents, combinant des contraintes sur les objets et sur les conteneurs spécifiques, est extrêmement grand. Cependant, nous allons tenter de faire une liste relativement exhaustive des contraintes connues, qui sont déjà apparues dans la littérature.

Tout d'abord, deux conditions primordiales sont à prendre en compte lors de la résolution de ce problème. La première, est qu'aucun objet chargé ne doit chevaucher un autre, ou dépasser du conteneur. Chevaucher veut ici dire que deux objets occuperaient simultanément le même sous-volume d'un espace, ce qui est absurde et impossible pour des objets géométriquement simples tels que des objets rectangulaires. La seconde condition est qu'aucun objet à charger n'est initialement plus grand que le ou les conteneurs. Dans ce cas, l'objet n'a rien à faire dans l'ensemble des objets à charger car il ne pourra jamais rentrer dans le conteneur.

8.2.1 Contraintes relatives aux objets

- Priorité de chargement d'un objet : il peut arriver dans certains cas que des objets soient considérés prioritaires dans la chargement par rapport à d'autres. Typiquement, l'algorithme les chargera toujours d'abord, et ensuite les autres. Prenons un exemple pour illustrer. Imaginons que lors d'un chargement à une date X , un objet qui devait être chargé ne l'a pas été, car il n'y avait plus de place dans le conteneur. On applique alors à cet objet un attribut de priorité par rapport aux autres. Par exemple, disons que sa valeur de priorité passe de 0 à 1. Lors d'un prochain chargement $Y > X$, l'algorithme de chargement prendra en priorité les objets avec une valeur de priorité plus élevée, en l'occurrence l'objet que nous n'avions pas pu charger la première fois.
- Orientation : chaque objet est défini par des dimensions, une largeur, une hauteur et une longueur. En théorie, chaque dimension peut servir de hauteur. Si on prend des objets parallélépipédique, on peut tourner l'objet comme on veut, et définir son orientation verticale avec n'importe quelle dimension. Cependant, il arrive dans certains cas que ces manipulations soient restreintes, et que les objets ne soient pas orientables dans un certain

sens. Par exemple, si on a un carton contenant un objet fragile devant absolument être à plat, aucune manipulation ne sera permise, et la hauteur sera quoi qu'il arrive la hauteur. Second exemple, si on a un carton de largeur très fine, on peut ne pas vouloir le poser sur cette face très fine pour des raisons de sécurité.

- Superposition : on pourra souvent considérer que tous les types d'objets ne sont pas superposables. En effet, si on pose un objet très lourd car très dense, sur un objet, de même dimensions, mais beaucoup moins dense et donc plus léger, ce second objet se verra sans doute écrasé par le premier, et sera potentiellement abîmé. On imposera alors des règles de superposition entre les différents types d'objets. Les règles de superposition peuvent également intervenir sur les dimensions des objets, même si c'est souvent lié au poids de ceux-ci. En effet, imaginons poser un carton de face supérieure moyenne. Sur ce carton, imaginons poser un carton deux fois plus long. Alors, la moitié du second objet ne reposera potentiellement sur rien de solide. On voudra donc imposer des règles disant qu'un certain pourcentage de la face inférieure d'un objet placé au dessus d'un ou plusieurs autres objets doit reposer sur quelque chose de solide.
- Positionnement : les contraintes de positionnement se divisent en deux groupes, le positionnement relatif et absolu. Le positionnement relatif implique que certains objets doivent être placés proches ou loin les uns des autres. Dans le cas d'une livraison à plusieurs clients, on voudra par exemple regrouper tous les objets d'un même client proches les uns des autres, afin de faciliter le déchargement quand on arrive chez ce client. Le positionnement absolu, quant à lui, implique qu'un objet doit être placé dans une position particulière ou dans une certaine zone du conteneur. Par exemple, si on prend le cas de la construction d'une palette, où la palette représente le conteneur. On placera des objets tels que des liquides volatiles ou des explosifs sur les extérieurs de la palette, afin de pouvoir les retirer rapidement en cas d'urgence.

8.2.2 Contraintes relatives aux conteneurs

- Poids limite total : le conteneur ne peut supporter qu'une certaine charge maximale. Ainsi, la valeur de la somme des poids des objets chargés dans le conteneur ne peut excéder cette limite.
- Répartition de la charge : cette contrainte implique que la charge soit répartie sur l'ensemble de la surface du conteneur. En effet, il est important d'avoir une charge équilibrée, afin d'éviter que le conteneur, comme un camion par exemple, "penche" en roulant. Également, le problème de la répartition de la charge peut intervenir dans certains cas où on veut répartir la charge sur les différents essieux du conteneurs, encore une fois par exemple un camion. Dans ce cas, chaque essieu ne peut supporter qu'une certaine charge, et il est important de la répartir pour garantir une certaine stabilité et sécurité.

8.2.3 Autres contraintes

- Les groupes d'objets : il peut arriver que lors d'un chargement, on veuille manipuler des groupes d'objets. Un groupe d'objet représente un ensemble d'objets qui sont liés dans le chargement. En d'autres termes, aucun objet du groupe ne pourra être chargé seul : si un objet du groupe est chargé, alors on doit charger tout le groupe. A l'inverse, si un des objets du groupe ne peut pas être chargé, alors on ne chargera aucun objet du groupe. Cependant, ce genre de contrainte est très rarement considérée dans la littérature.

- Problème d'allocation : un peu dans le même esprit que la contrainte précédente, le problème d'allocation peut survenir dans deux cas. Le premier, est celui où deux objets doivent impérativement être chargés dans le même conteneur. Typiquement, cette contrainte ne peut apparaître que dans le cas où nous gérons plusieurs conteneurs. Ainsi, le problème d'allocation n'est pas forcément un problème de groupe d'objets, car on peut charger un groupe d'objets dans plusieurs conteneurs, et ils seront toujours tous chargés. Le second cas est le cas inverse, c'est à dire quand deux types d'objets doivent impérativement être dans des conteneurs différents. On peut penser à la nourriture, qui ne doit pas être chargée avec les cosmétiques, par exemple.
- Patterns de chargement : pour certains chargement, on veut pouvoir appliquer un pattern à notre plan de chargement. Par exemple, pour un entrepôt d'électroménager, on voudra d'abord charger tout le gros électroménager, puis le petit électroménager et enfin l'ameublement.

9 Les solutions existantes

Une multitude de solutions différentes sont apportées dans les papiers de la littérature, qu'elles soient des méthodes exactes ou des heuristiques. Encore une fois, ces solutions varient selon la nature du problème qui, nous l'avons vu, est souvent unique. Cependant, certaines méthodes et algorithmes sont parfois récurrents, et nous allons tenter de présenter ces quelques grandes approches dans cette seconde partie. La liste des éléments de résolution présentée ci-après n'est donc pas exhaustive.

9.1 Une heuristique utilisant la méthode du Wall Building Approach

Le paragraphe suivant s'inspire du papier "Heuristics for the container loading problem", publié par David Pisinger en 2002 ([9]).

Cette solution va mettre en oeuvre ce que l'on appelle la "Wall Building Approach", ou l'approche de construction de murs en français. Cette approche a été introduite par George et Robinson en 1980. Elle représente une méthode de remplissage du conteneur bien spécifique. En l'occurrence, on cherchera ici à remplir le conteneur "mur par mur", c'est à dire construire à la suite des niveaux verticaux d'objets, que l'on vient mettre côte à côte, en commençant à remplir le conteneur par le fond. Pour illustrer le concept, observons la [Figure 3](#)

Cette figure représente bien le concept de "layer" ou niveau en français. Si on imagine un camion comme conteneur, on visualise bien les niveaux de cartons. Chaque niveau est en fait basé sur une largeur d sur le schéma. On met alors les niveaux côte à côte jusqu'à ce qu'on atteigne la profondeur totale du conteneur, D sur le schéma.

L'algorithme ainsi proposé fonctionne par itération, et à chaque itération, on vient ajouter un niveau dans le conteneur. Il se décompose en deux étapes clés. La première est celle de choisir une largeur. En effet, une fois la largeur du niveau de l'itération choisie, on n'ajoutera à ce niveau que des objets pouvant prendre cette largeur. En d'autres termes, l'ensemble des objets d'un même niveau possèdent la même largeur. Il est donc nécessaire de choisir cette largeur intelligemment. Une fois la largeur choisie, la seconde étape est de remplir le niveau avec des

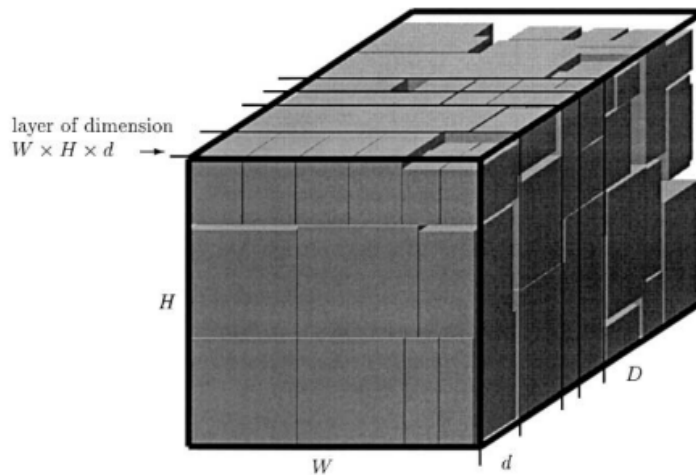


Figure 3 – Wall Building Approach

objets. Puis, une fois le niveau créé, on l’ajoute dans le conteneur et l’itération est terminée. L’algorithme s’arrête lorsque soit nous n’avons plus d’objets, soit la largeur de profondeur du camion restante est inférieure à la largeur minimale des objets restants, autrement dit qu’aucun niveau de la largeur restante n’est constructible avec les objets restants.

Cette solution a été testée sur un problème où aucune contrainte n’était imposée, c’est donc un simple Knapsack tri-dimensionnel. Le but est de remplir l’espace du conteneur au maximum. Les résultats sont que l’heuristique proposée permet de remplir 95% du volume du conteneur en moyenne, et ce même sur des grandes instances. Ces résultats sont extrêmement satisfaisants. Cependant, on peut se demander si l’approche ne présentera pas rapidement de limites dès qu’on voudra ajouter des contraintes sur les objets ou le conteneur, comme la répartition de la charge par exemple, qui est très souvent prise en compte.

9.2 Un algorithme de parcours d’arbre utilisant la méthode du Block Building Approach

Le paragraphe suivant s’inspire du papier “A Tree Search Algorithm for Solving the Container Loading Problem”, publié par Tobias Fanslau et Andreas Bortfelden en 2008 ([3]).

Il m’a semblé intéressant de citer ce papier, car il met en oeuvre deux concepts assez récurrents dans les méthodes de résolutions de ce type de problème. Premièrement, parlons du Block Building Approach, qui est comme son prédécesseur le Wall Building Approach, une méthode de remplissage de conteneur. Simplement, cette fois ci, nous allons raisonner en blocs, et non en “murs”.

Tout d’abord, encore une fois, nous ne raisonnerons ici qu’avec des objets de forme parallélépipédique, type cartons. Une des premières étapes de l’algorithme est de créer une liste de blocs B. Un bloc est un ensemble d’objets regroupés ensemble, comme ceci :

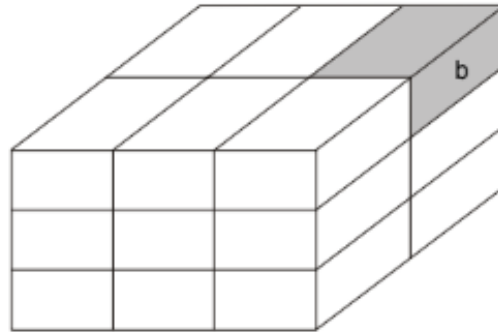


Figure 4 – *Notion de bloc*

Le principe de l'algorithme est assez simple. Un bloc est caractérisé par ses dimensions, et donc son volume. Nous supposons alors un ensemble de sous espaces libres du conteneur noté S . Chaque sous espace est de forme parallélépipédique. A la première itération de l'algorithme, tout le conteneur est le seul sous ensemble dans S . On commence par sélectionner le meilleur bloc de B qui remplit le sous espace sélectionné. En d'autres termes, on choisit le bloc dont le volume remplit au mieux le volume du sous espace. Une fois ceci fait, on le place, et l'espace résiduel initial est divisé en trois nouveaux espaces résiduels. Le schéma suivant explique bien le principe :

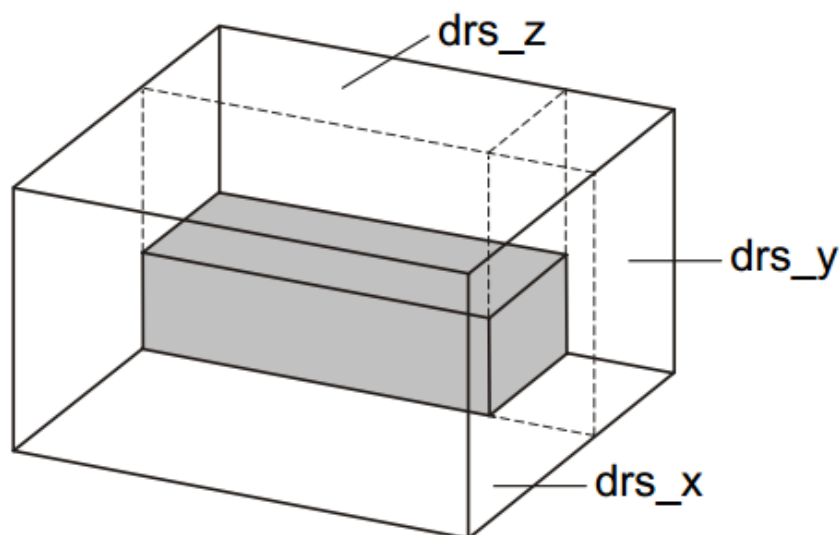


Figure 5 – *Les espaces résiduels après placement d'un objet*

Comme on peut le voir, nous avons placé un objet grisé dans un coin du conteneur. Alors, trois nouveaux sous espaces résiduels drs_z , drs_x et drs_y apparaissent, qu'on vient stocker dans S . Alors, nous itérons sur les éléments de S , jusqu'à ce qu'aucun bloc ne puisse remplir les espaces

résiduels restants, ou qu'il ne reste plus de blocs dans B. On peut noter que le choix et l'ordre de sélection du sous espace résiduel de S à chaque itération pourra avoir une influence sur la valeur de la solution finale. Ainsi, plusieurs solutions différentes sont explorées, et la meilleure est sélectionnée pour proposer le meilleur plan de chargement, d'où la méthode de parcours d'arbre initialement émise.

Au final, cet algorithme présente en moyenne d'excellents résultats. En effet, le but étant de maximiser l'utilisation du volume total, il présente une occupation moyenne de 92,7% du volume, avec 15 instances de tests différentes.

La technique de découpage de l'espace reste très efficace et répandue, et permet une approche orientée blocs plutôt que "murs". C'est certainement la caractéristique majeure de cette méthode de résolution. Cependant, encore une fois, on exclut ici toute complexité de forme d'objet, et toute contrainte de répartition de poids. Néanmoins, je pense pouvoir dire que cet algorithme serait une bonne base pour appréhender notre problème.

9.3 Un modèle mathématique à implémenter sur un solveur

Le paragraphe suivant s'inspire du papier "A linear programming approach for the three-dimensional Bin-Packing Problem", publié par Mhand Hifi, Imed Kacem, Stéphanie Nègre et Lei Wu en 2010 ([6]).

La solution abordée ici est assez différente des deux précédentes. On ne développe pas ici un algorithme, mais un modèle mathématique de programmation par contrainte, utilisant un solveur mathématique. Typiquement, les modèles mathématiques reposent sur des données, variables, contraintes et une fonction objectif. On passe ce modèle dans un solveur, qui viendra maximiser ou minimiser cette fonction objectif, tout en respectant l'ensemble des contraintes émises.

Voici l'expression du modèle mathématique proposé :

$$l_{ij} + l_{ji} + u_{ji} + u_{ij} + b_{ij} + b_{ji} + c_{ij} + c_{ji} = 1, \quad i < j = 1, \dots, n \quad (1)$$

$$x_i - x_j + W(l_{ij} - c_{ij}) - c_{ji} \leq W - w_i \quad i \neq j = 1, \dots, n \quad (2)$$

$$y_i - y_j + H(u_{ij} - c_{ij}) - c_{ji} \leq H - h_i \quad i \neq j = 1, \dots, n \quad (3)$$

$$z_i - z_j + D(b_{ij} - c_{ij}) - c_{ji} \leq D - d_i \quad i \neq j = 1, \dots, n \quad (4)$$

$$(\bar{\gamma} - 1)(l_{ij} + l_{ji} + u_{ij} + u_{ji} + b_{ij} + b_{ji}) + \gamma_i - \gamma_j + \bar{\gamma}c_{ij} \leq \bar{\gamma} - 1 \quad i \neq j = 1, \dots, n \quad (5)$$

$$l_{ij}, u_{ij}, b_{ij}, c_{ij} \in \{0, 1\} \quad i \neq j = 1, \dots, n$$

$$0 < x_i \leq W - w_i \quad i = 1, \dots, n$$

$$0 < y_i \leq H - h_i \quad i = 1, \dots, n$$

$$0 < z_i \leq D - d_i \quad i = 1, \dots, n$$

$$0 < \gamma_i \leq \gamma \leq \bar{\gamma} \quad i = 1, \dots, n$$

Explications

On veut ici ranger un ensemble de n éléments rectangulaires dans un ensemble de conteneurs identiques. Chaque item i ($i = 1, 2, \dots, n$) est caractérisé par une largeur w_i , une hauteur h_i , et une profondeur d_i ($i = 1, 2, \dots, n$). Les conteneurs ont la même largeur W , la même hauteur H , et la même profondeur D . On veut ici minimiser le nombre de conteneur utilisés.

On utilise ici une borne supérieure $\bar{\gamma}$ ($n \geq \bar{\gamma}$), représentant un nombre suffisant de conteneurs. On définit le vecteur $(x_i, y_i, z_i, \gamma_i)$, la position de l'élément i , où $x_i \geq 0, y_i \geq 0, z_i \geq 0$ et $\gamma_i \geq 1$. Enfin, γ_i est le numéro (ou label) du conteneur auquel l'élément i est assigné.

Dans le modèle proposé, $l_{ij} = 1$ si l'élément i est à gauche de l'élément j , $u_{ij} = 1$ si l'élément i est en dessous de l'élément j , $b_{ij} = 1$ si l'élément i est derrière l'élément j et $c_{ij} = 1$ si $\gamma_i < \gamma_j$.

Les contraintes 1, 2, 3 et 4 permettent d'assurer que les éléments ne se chevauchent pas.

J'ai implémenté ce modèle, et je détaille cette procédure et ses résultats dans la partie Analyse. Cependant, nous pouvons ici clairement dire que la solution proposée fonctionne pour un problème simpliste. Aucune contrainte sur la charge, l'ordonnancement des éléments, et même le support des éléments n'est exprimée. Ainsi, son utilité paraît très limitée quant à la résolution de notre problème. J'ai trouvé intéressant de citer cet exemple d'implémentation, car elle diffère des algorithmes classiques, et nous permet de voir que d'autres outils, tels que les solveurs mathématiques, existent pour résoudre ce genre de problème de manière simple.

10 Conclusion et choix

Comme nous avons pu le constater, des méthodes très variées existent, et nous n'en avons cité que quelques unes. Il est très difficile voir même impossible de trouver une méthode qui puisse résoudre exactement notre problème. Chaque approche abordée est pensée et construite en fonction des contraintes initiales du problème en question. A partir du moment où on ajoute des contraintes, tout le fondement de ces approches est chamboulé, et elles ne seront plus forcément aussi efficaces, voire obsolètes.

Ainsi, je ne pense pas que nous allons pouvoir réutiliser un des concepts précédemment présentés. L'utilisation d'un solveur mathématique présente des limites de performances, et la complexité des contraintes devient beaucoup trop grande lorsqu'on en ajoute de trop. Ce qu'on pourra envisager, cependant, est d'utiliser une méthode de remplissage similaire à celle du Wall Building ou Block Building approach. En effet, on peut, je pense, séparer le problème en deux aspects : la méthode de remplissage, et la sélection des objets pour effectuer ce remplissage. Dans notre cas, la majorité des contraintes porte sur la seconde partie, de sélection des objets. Donc, nous pourrions adopter une de ces méthodes de remplissage, et le cas échéant, reprendre la structure du code relatif à ces méthodes pour gagner du temps. Egalement, je trouve que le calcul des espaces résiduels dans le second exemple est très intéressant.

Encore une fois, l'originalité de notre problème fait que l'algorithme à créer devra sans doute être pensé de A à Z.

Quatrième partie

Analyse et conception

11 Phase de recherche

11.1 Entités

Dans ce projet, il est difficile d'identifier beaucoup d'entités. En fait, je pense qu'il n'y en aura qu'une, représentant les éléments. Les camions en eux-même n'ont pas à être décrits en tant qu'entités, étant donné que l'on utilisera toujours les mêmes caractéristiques de camions pour une exécution du programme (dimensions, distance des essieux, charge maximale, etc.). Ainsi, ces données seront globales dans le programme, sans être forcément stockées dans des objets en particulier.

Une première implémentation de l'entité "Element" pourrait ressembler à celle de la [Figure 6](#).

Cet objet contiendrait tous les attributs permettant de répondre aux contraintes imposées par KeenSaas. Ainsi, un objet Element lors du début de la phase de développement sera plus minimaliste.

On retrouve bien évidemment ses dimensions, largeur, longueur et hauteur. A cela, on pourra éventuellement ajouter un volume si nous en avons besoin, même si il est facilement calculable grâce aux trois premiers. Ensuite, il faudra renseigner le poids de l'élément. En effet, pour les contraintes de répartition de charge dans le camion, mais également de poids maximal supporté par les éléments, cet attribut est nécessaire.

On retrouve alors l'attribut de stackabilité, ou de gerbabilité, disant si oui ou non l'objet peut accepter d'autres objets sur sa face supérieure. Dans ce cas, on renseigne également une valeur pour l'attribut maxBearableWeight, représentant le poids maximal supporté par l'objet. Dans le cas contraire, cette valeur est égale à 0.

Un élément est caractérisé par un type, par exemple "Electroménager" ou "Cosmétique", dans le cas de la prise en compte de la contrainte de chargement par type d'objets. Un élément possède aussi une priorité, dont la valeur est initialement nulle. Si cet élément n'est pas chargé dans le premier chargement où il était censé l'être, on incrémente cette valeur. Lors des prochains chargements, on charge en priorité les éléments avec des valeurs de priorités élevées.

Enfin, un objet possède des degrés de liberté, que sont les trois attributs rotationAxis. Ces attributs disent si oui ou non un élément peut faire des rotations dans l'espace, sur les axes X,Y ou Z. Ces degrés de liberté nous servent à savoir si on peut tourner les objets dans un sens ou dans un autre afin d'optimiser certains chargements.

Element
<ul style="list-style-type: none"> - length : double - width : double - height : double - weight : double - volume : double - stackability : boolean - type : String - priority : int - rotationAxisX : boolean - rotationAxisY : boolean - rotationAxisZ : boolean - maxBearableWeight : double

Figure 6 – L'entité Element

11.2 Les solutions

L'algorithme que nous allons créer sera capable de prendre des décisions. Il va s'appuyer sur des données qui vont évoluer au fil de l'exécution du programme. Par exemple, je pense que nous allons constamment garder une liste de volumes "disponibles", dans lesquels nous n'avons pas encore placé d'objets, afin de pouvoir continuer à itérer jusqu'à ce que cette liste soit vide, où que nous ayons placé tous les objets. Ces volumes seront remplis par une méthode que nous allons définir, et cette méthode va suivre une même logique, caractéristique de l'algorithme en question. Suivre une certaine logique va permettre à l'algorithme de prendre facilement des choix, en fonction de la configuration du camion et des objets à un instant t.

Si on réfléchit en termes de solutions, cela a un effet conséquent. En effet, rappelons que l'algorithme prendra en entrée une liste d'éléments. Puis, il va être amené à constamment parcourir cette liste, dans l'ordre, pour savoir si tel ou tel objet peut être placé à tel ou tel endroit. Pour un ordonnancement de liste particulier, les choix que l'algorithme va faire seront toujours les mêmes.

Pour être plus clair, si nous exécutons deux fois l'algorithme avec exactement les mêmes objets, ordonnées dans exactement le même ordre dans la liste d'éléments de base, les choix, l'exécution et la solution proposée par l'algorithme seront deux fois les mêmes. Ce qui est normal, car il aura suivi une logique, qui l'aura amené à prendre les mêmes choix, aux mêmes moments.

Cependant, si nous considérons maintenant une liste d'éléments de base qui suit un ordre différent, par exemple le dixième et le vingtième objet échangent leur place. Dans ce cas, le vingtième objet sera considéré avant le dixième, et les choix effectués par l'algorithme seront potentiellement très différents, ce qui peut avoir un impact direct sur la solution.

Ainsi, il sera nécessaire d'adopter une logique dans l'exécution de l'algorithme, afin de garantir la meilleure solution selon un ensemble d'éléments donnés. On voudra en effet, si possible, tester le plus grand nombre de possibilités d'ordonnancement. Typiquement, on voudra échanger la place de deux éléments dans la liste, ou faire un "swap", pour faire varier les solutions. Pour une même exécution de l'algorithme, on viendra en fait l'exécuter autant de fois que de swaps sont possibles. Ainsi, beaucoup de solutions seront générées, théoriquement toutes différentes, et l'algorithme ne retiendra que la meilleure, ou les quelques meilleures. En effet, il peut parfois être intéressant de garder quelques autres solutions, au cas où l'unique meilleure solution ne convienne pas au chargeur, pour des raisons pratiques par exemples.

11.3 Essais sous LocalSolver

Environ au milieu de la phase de recherche, j'ai souhaité commencer à tester des algorithmes. Ainsi, j'ai tenté de reprendre le concept cité dans l'article [6], c'est à dire l'implémentation d'un modèle mathématique. Un tel modèle peut être passé dans un solveur, comme Cplex ou LocalSolver par exemple. Comme j'avais déjà utilisé LocalSolver l'année précédente, j'ai choisi celui-là. Plus précisément, j'ai utilisé l'API Java de LocalSolver pour le faire.

LocalSolver est une simple librairie à importer, et propose un solveur capable de résoudre des problèmes d'optimisation en parfois très peu de temps. Il est nécessaire d'acquérir une licence, gratuite pour les étudiants, pour pouvoir l'utiliser.



Figure 7 – LocalSolver

J'ai repris exactement les mêmes variables, contraintes et fonction objectif que le modèle proposé. Cependant, les résultats n'étaient pas vraiment satisfaisants. En effet, c'est un modèle extrêmement minimaliste, ne gérant pratiquement aucune contrainte, pas même le fait que les objets doivent reposer sur une surface solide. Je me suis donc vite retrouvé avec des objets volants.

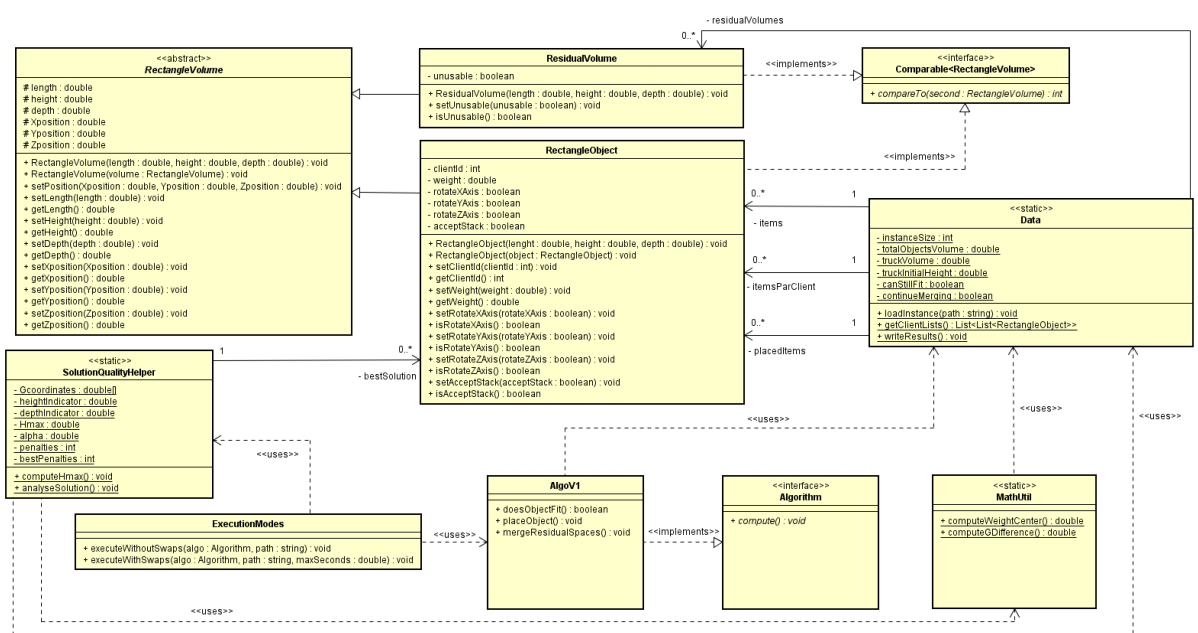
Même si j'aurais pu continuer, je me suis rapidement rendu compte que la rédaction d'une contrainte est souvent trop complexe. De plus, il faut que les contraintes rédigées ne soient pas en conflit avec les précédentes, ce qui rajoute une difficulté supplémentaire. Et dans ce cas, il restait beaucoup de contraintes à rédiger.

J'ai donc un peu écarté cette approche, qui m'a paru trop compliquée. Et, pour avoir échangé avec le patron de l'entreprise ayant développé LocalSolver par la suite, il m'a assuré que son solveur n'aurait pas pu être assez performant pour résoudre un problème aussi complexe que le nôtre. Cette méthode de résolution est donc probablement à écart, ou alors à entreprendre avec un solveur différent.

Tous ces essais avec LocalSolver ont quant même soulevé un problème important. En effet,

Ainsi, c'est là qu'a démarré l'échange avec KeenSaas concernant l'accès à leur visualisateur 3D. Je décris cette partie dans l'annexe contenant le reste du cahier de spécifications, dans la partie interfaces. Cependant, cette étape fut, je pense, importante, car elle a permis la mise en place d'un moyen de visualisation que me servira jusqu'à la fin du projet.

Dans cette partie, nous allons maintenant aborder la conception résultante du développement effectué. Il faut bien préciser que l'ensemble des classes et objets présentés ici n'ont pas été anticipés lors de la phase de conception, mais bien construits et améliorés au fur et à mesure de l'avancement du projet.



Étant abstraite, cette classe ne sera jamais instanciée. En effet, on vient en faire hériter deux autres classes, qui sont les réels objets de notre problème. La première est `RectangleObjet`, qui représente les objets que l'on souhaite charger dans notre camion, et qui peuvent être assimilés à des colis ou cartons, toujours uniquement parallélépipédiques. Cette classe est l'évolution de la classe `Elément` présentée un peu plus haut. On y retrouve des attributs utiles à la prise en compte de certaines des contraintes du problème :

- `clientId` : attribut entier représentant un ID de client auquel cet objet est destiné. On pourra l'utiliser pour trier les objets selon les clients.
- `weight` : valeur du poids de l'objet, qui peut être utilisé pour les contraintes de centrage ou répartition de la charge, ou également pour la contrainte de poids maximal sur la face supérieure d'un objet.
- `acceptStack` : attribut booléen disant si oui ou non un objet peut accepter d'autres objets sur sa face supérieure, qui peut être utilisé pour la contrainte de gerbabilité.
- `rotateAxis` : trois attributs booléens, un pour chaque axe, disant si oui ou non un objet peut effectuer une rotation dimensionnelle sur tel ou tel axe.

La seconde classe qui hérite de la classe abstraite `RectangleVolume` est `ResidualVolume`. Nous le verrons un peu plus tard dans la partie mise en œuvre, mais l'algorithme implémenté implique la gestion de volumes vides, étant aussi des parallélépipèdes. Ainsi, cette classe `ResidualVolume` représente les volumes restants à combler dans le camion. Ces objets possèdent un attribut `unusable`, booléen, que nous détaillerons dans la partie mise en œuvre.

Ensuite, nous pouvons identifier les classes qui vont contenir les algorithmes que nous devons développer. On retrouve donc une interface `Algorithm`, possédant une unique méthode `compute`. Cette méthode contient l'algorithme de résolution au `Container Loading Problem` implémenté. Comme plusieurs implémentations peuvent exister, pour une implémentation donnée, on vient hériter de cette interface. C'est le cas de la classe `AlgoV1`, qui en plus de la méthode `compute` de l'interface `Algorithm`, possède plusieurs méthodes bien propres à sa méthode de résolution.

Puis, nous avons la classe `Data`. Cette classe est un peu une classe utilitaire. Elle sera responsable de toute la gestion des données lors de l'exécution, mais également des instances de données à charger. C'est donc pour cela que l'on peut y retrouver des méthodes telles que `loadInstance`, ou `writeResults`, qui sont respectivement responsables du chargement de l'instance de données que l'on veut traiter, et de l'écriture des résultats obtenus par l'algorithme dans un fichier texte de sortie. On y retrouve également un ensemble d'attributs relatifs à l'instance de données chargée, qui seront utilisées tout au long de l'algorithme, comme la taille de l'instance, le volume total des objets à charger, ou encore les dimensions du camion.

On remarque que cette classe est statique, ce qui veut dire qu'elle ne peut pas être instanciée, et que ses méthodes et attributs sont accessibles depuis l'ensemble du programme. Ce choix de conception a été fait car cette classe, comme d'autres dans notre cas, contient des informations auxquelles on souhaiterait facilement accéder depuis n'importe où. En d'autres termes, cela évite de devoir créer une dépendance à un même objet `Data` plusieurs fois pour plusieurs autres classes. Ici, les données sont chargées une fois dans la classe `Data`, et une fois ceci fait, on peut les utiliser comme bon nous semble depuis toutes les autres classes. Même si ce choix peut

soulever des contraintes de sécurité et également d'évolutivité, il nous rendrait ici un grand service et permettrait un ensemble des données beaucoup plus facilement maniables à travers le programme. Enfin, cette classe Data contiendra différentes listes d'objets et de volumes, qui seront constamment manipulées dans les méthodes de résolution. Nous y reviendrons encore une fois dans la partie mise en œuvre.

Enfin, nous pouvons identifier le reste des classes que je qualifierais de purement utilitaires, même si chacune possède un rôle bien particulier. D'abord, la classe MathUtil. Comme son nom l'indique, elle contient quelques méthodes mathématiques bien spécifiques, telles que des calculs de coordonnées de centre de gravité, qui peuvent être utilisées pour résoudre les contraintes de centrage de la charge par exemple.

Ensuite, nous avons la classe ExecutionMode, à laquelle nous consacrerons une partie entière dans la partie mise en œuvre. Pour faire simple, cette classe nous permettrait, une fois l'instance chargée, de pouvoir exécuter le programme avec des modes différents. Ici, nous pouvons en observer deux : l'un avec swaps et l'autre sans swaps. Pour rappel, les swaps nous permettraient de faire varier les solutions, et ainsi pouvoir disposer de plusieurs solutions à proposer à l'utilisateur. Une classe telles que ExecutionMode nous permettrait donc de choisir entre différents modes d'exécution, selon ce que l'on souhaite obtenir.

Pour terminer, et dans la continuité de ce que l'on vient de dire, nous avons la classe SolutionQualityHelper. Dans le cas où l'on voudrait appliquer une stratégie de swaps sur notre ensemble d'objets de base, et obtenir plusieurs solutions, il faut être capable de comparer ces solutions selon certains indicateurs de qualité. Pour estimer ces indicateurs, nous aurons besoin dans notre programme d'une certaine quantité d'informations, qui seront stockées dans cette classe. De plus, on peut remarquer une méthode analyseSolution, qui permettrait d'afficher certaines informations relatives à la solution courante à l'utilisateur. Encore une fois, nous reviendrons sur cet aspect qualité de solution et solutions multiples dans une partie ultérieure. Tout comme la classe Data, les classes MathUtil et SolutionQualityHelper sont statiques, pour les mêmes raisons que citées précédemment.

12.2 Les limites de cette modélisation

La modélisation précédemment présentée fonctionne plutôt bien pour mon implémentation. Cependant, il est important de se demander si, en termes d'évolutivité, cette conception ne présente pas quelques limites.

On peut ici ajouter autant d'implémentations de méthodes de résolution différentes que l'on veut, juste en faisant une nouvelle classe héritant de l'interface Algorithm, ce qui est une bonne chose. Également, certaines méthodes et certains attributs de la classe Data pourraient être ré-utilisés, notamment les fonctions de lecture d'instance et d'écriture de résultats, et toutes les informations relatives à l'instance. Pareillement, les méthodes et attributs de la classe SolutionQualityHelper pourrait être ré-utilisées.

En revanche, dans le cas de SolutionQualityHelper, tout pourrait être ré-utilisé, mais seulement dans le cas où l'on utilise les mêmes indicateurs de qualité de solution. Si on souhaite en utiliser d'autres, on va devoir rajouter des attributs et méthodes dans cette classe, qui je le

rappelle est statique. Au bout d'un certain temps et de plusieurs indicateurs, on se retrouverait avec une classe possédant une quantité de méthodes et attribut beaucoup trop importante, et surtout écartée de toute notion de découpage fonctionnel. On ne voudrait pas d'une classe qui "peut gérer tout les indicateurs", on voudrais plutôt "autant de classes qu'il y a d'indicateurs".

On frappe ici selon moi une limite de l'utilisation d'une classe statique, cette limite étant le fait de ne pas pouvoir créer des instances et donc de spécialiser une instance dans une direction, et une autre instance dans une autre. Je pense que dans un aspect évolutif, on pourrait éventuellement créer, comme on l'a fait avec Algorithm, une interface ou simplement une classe mère, de laquelle différentes implémentations peuvent hériter. Par contre, la contrainte qui apparaîtrait serait le fait de devoir ajouter des dépendances entre ces différentes implémentations de SolutionQualityHelper et les implémentations de Algorithm.

Cinquième partie

Mise en œuvre

Nous allons maintenant aborder la partie concernant la mise en œuvre de ce projet. Elle comportera l'ensemble des réflexions et du travail effectués lors du second semestre dédié au développement.

13 L'algorithme et les procédures majeures

Avant de commencer à étudier l'algorithme, je vais vous expliquer sur quels principes il repose. Dans la partie état de l'art, nous avons mentionné un concept de découpage de l'espace particulier, qui peut se représenter sur la [Figure 5](#). Pour rappel, on suppose ici que nous travaillons uniquement avec des espaces en trois dimensions, de forme parallélépipédique. Ce principe nous dit que dès qu'on place un volume parallélépipédique dans un autre volume parallélépipédique, trois nouveaux sous volumes, également parallélépipédiques, se forment. L'idée de mon algorithme est de créer une procédure récursive qui itère sur cet ensemble de volumes. Il est important de comprendre quelles sont les données principales que nous allons manipuler, également visualisables sur le diagramme de classes présenté précédemment :

- `residualVolumes` : la liste des sous volumes résiduels. C'est la liste des volumes qu'il reste à occuper dans la camion. A l'itération 0, cette liste contient uniquement le volume correspondant au volume total du camion. A chaque itération, on sélectionne un volume de cette liste qu'on va combler. On le supprime donc de la liste, et on en ajoute trois nouveaux, inclus dans ce même volume.
- `items` : la liste des objets qu'il reste à placer dans le camion. A l'itération 0, cette liste contient tous les objets du chargement à placer. A chaque itération, après la sélection du sous volume à combler, on place un objet, qu'on supprime de cette liste.
- `placedItems` : la liste des objets que l'on a placé. A chaque itération, lorsqu'on place un objet et qu'on le retire de la liste `items`, on le place dans cette liste. L'intersection entre `items` et `placedItems` est nulle. Les objets sont placés dans cette liste dans leur ordre de placement dans le camion.

Ainsi, il est important de discerner les deux étapes clés de notre algorithme, que l'on répète à chaque itération. La première étape est la sélection du volume que l'on souhaite combler, à piocher dans la liste `residualVolumes`. Cette sélection peut s'effectuer selon différents critères. Par exemple, on peut choisir de prendre d'abord les volumes les plus gros, ou bien ceux qui se situent au fond du camion en priorité. La stratégie de sélection est à affiner en fonction des besoins. Dans mon cas, j'ai trié ma liste de sous volumes selon leur position sur l'axe Y, représentant le camion dans sa longueur, en plaçant d'abord les volumes les plus au fond du camion. Nous verrons comment ce tri influe la sélection une fois que nous aurons présenté l'algorithme.

La seconde étape est la sélection de l'objet à placer dans le volume que l'on a sélectionné. Encore une fois, cette sélection peut s'effectuer selon différents critères et différentes stratégies. Par exemple, on peut choisir de prendre l'objet qui remplit au mieux le volume, ce qu'on pourrait qualifier de `bestFit`. Aussi, nous pouvons choisir de prendre le premier objet qui rentre dans le volume, ce qu'on pourrait qualifier de `firstFit`. Dans mon cas, j'ai implémenté les deux solutions, chacune pour une situation différente, ce que j'expliquerais après la présentation de l'algorithme.

A chaque itération, les trois listes sont modifiées, et utilisées à l'itération d'après. Notre algorithme comporte trois conditions d'arrêt qui sont :

- lorsque tous les objets ont été placés, donc que `items` est vide
- lorsque tous les sous volumes ont été comblés, donc qu'il n'y a plus de place dans le camion. Cette condition d'arrêt est théorique, et ne pourra dans notre cas jamais arriver,

car des sous volumes sont créés à chaque itération. On se retrouvera plutôt dans le cas où il ne restera plus que des sous volumes beaucoup trop petits, ce qui rejoint la dernière condition d'arrêt.

- lorsque plus aucun des objets ne loge dans aucun des sous volumes restants, alors on ne peut plus rien placer.

Regardons maintenant la procédure récursive principale, qui sert à sélectionner le sous volume, l'objet à placer, place l'objet. Elle fait appel à certaines des fonctions des classes présentées dans le diagramme un peu plus haut. C'est le cœur de notre algorithme :

```

Data : residualVolumes, list of residualVolumes
items, list of items to be placed
placedItems, list of items already placed
canStillFit, boolean saying if there are still objects that can fit into one or several volumes

1 begin
2   while residualVolumes ≠ ∅ and items ≠ ∅ and canStillFit is true do
3     indiceVolume ← 0
4     boolVolume ← false; bestObjectIndex ← -1
5     bestVolume ← null; bestObject ← new RectangleObject(0,0,0)
6     while indiceVolume ≤ residualVolumes.length and boolVolume is false do
7       bestVolume ← residualVolumes.at(indiceVolume)
8       found ← false
9       if bestVolume.isUnusable() is false then
10        for i ∈ items do
11          if doesObjectFit(items.at(i), bestVolume) is true then
12            bestObject ← items.at(i); bestObjectIndex ← i
13            found ← true; break
14          else
15            copy ← items.at(i) with rotation on X axis
16            if doesObjectFit(copy, bestVolume) is true and copy.isRotateXAxis is true then
17              bestObject ← copy; bestObjectIndex ← i
18              found ← true; break
19            else
20              copy ← items.at(i) with rotation on Z axis
21              if doesObjectFit(copy, bestVolume) is true and copy.isRotateZAxis is true then
22                bestObject ← copy; bestObjectIndex ← i
23                found ← true; break
24              else
25                copy ← items.at(i) with rotation on Y axis
26                if doesObjectFit(copy, bestVolume) is true and copy.isRotateYAxis is true then
27                  bestObject ← copy; bestObjectIndex ← i
28                  found ← true; break
29                end
30              end
31            end
32          end
33        end
34      end
35      if found is false then
36        indiceVolume ← indiceVolume + 1
37      else
38        boolVolume ← true
39        break
40      end
41    end
42    if bestObject ≠ null and bestVolume ≠ null and bestObjectIndex ≠ -1 then
43      placeObject(bestObject, bestVolume, bestObjectIndex)
44      mergeResidualSpaces()
45    end
46    canStillFit ← boolVolume
47    compute()
48  end
49 end

```

Algorithme 1 : Compute

La procédure commence par une boucle, qui s'arrête si l'une des trois conditions d'arrêt précédemment décrites est vérifiée. L'objectif d'une itération est de trouver le volume et l'objet qui va le remplir. Je vais d'abord revenir sur la stratégie de sélection du volume que nous avons évoqué. La liste des sous volumes résiduels est ordonnée selon les valeurs sur l'axe Y, de telle sorte que les volumes les plus au fond du camion sont en premiers. Ainsi, à chaque itération, on vient prendre le premier volume de la liste, théoriquement le plus au fond du camion d'après le tri (ligne 7). L'idée est ensuite de trouver l'objet qui vient remplir le volume. La stratégie présentée sur l'algorithme précédent est assimilable à un firstFit, ce qui veut dire qu'on prend le premier objet qui loge dans le volume sélectionné.

A la sélection du volume, on vient donc itérer sur tous les objets pour regarder lequel peut rentrer (ligne 10). Pour chaque objet de la liste items, on regarde si il rentre dans sa configuration de base, c'est à dire tel qu'il est lu dans le fichier d'instance. Si il rentre, on le prend et on le sauvegarde dans bestObject, avant de mettre fin à la boucle for par un break. Si il ne rentre pas, on essaie de lui faire une rotation sur l'axe X, si l'objet accepte cette rotation. Encore une fois, si il rentre dans cette configuration, on le place dans bestObject et on met fin à la boucle for, car l'objet a déjà été trouvé. Dans le cas où il ne rentre pas avec la rotation sur l'axe X, on essaie la rotation sur l'axe Z, puis sur l'axe Y. Si l'objet ne rentre pas dans le volume dans aucune de ces quatre configurations, on passe à l'objet d'après dans la liste items.

A la sortie de cette boucle for, deux cas se présentent à nous. Soit nous avons trouvé un objet qui rentre dans le volume sélectionné, et la variable found est égale à true, soit nous n'avons pas trouvé (ce qui veut dire qu'aucun objet ne loge dans le volume dans aucune de ses configurations), et found est égale à false. Dans le second cas, on vient simplement incrémenter la variable indiceVolume, ce qui aura pour effet de recommencer toute l'opération, mais le pour le volume qui se situe une case après celui actuellement étudié. Dans le premier cas, on arrête de parcourir les volumes et les objets, car on peut maintenant placer l'objet dans le volume. Pour ce faire, on passe la valeur de la variable boolVolume à true, qui est la condition d'arrêt de la boucle itérant sur les volumes (ligne 6).

Une fois que nous sommes sortis des boucles de sélection de volume et d'objet, deux cas sont possibles. Soit nous avons trouvé un volume et un objet, auquel cas boolVolume est donc true, soit nous n'avons trouvé aucun objet ne logeant dans aucun des sous volumes, auquel cas boolVolume est resté à false. Dans le premier cas (lignes 42-45), nous prenons soin de placer l'objet dans le camion, et d'appeler la procédure de fusion des espaces résiduels, que je détaillerais juste après. Dans le second cas, nous ne plaçons pas d'objet car nous ne passons pas dans le if de la ligne 42. Dans tous les cas, on affecte la valeur de boolVolume dans canStillFit, qui est je le rappelle une de nos conditions d'arrêt de la chaîne de récursivité. Puis, on rappelle la fonction compute(). Si boolVolume était à false, et qu'aucun objet ne logerait dans aucun des volumes, l'algorithme s'arrêterait. Sinon, on continuerai jusqu'à rencontrer ce cas.

On peut voir que cette procédure s'appuie sur quelques fonctions utiles, telles que doesObjectFit, placeObject ou encore mergeResidualSpaces. La première consiste à rapidement vérifier si un objet loge dans un volume. Voici à quoi elle ressemble :

```

input : object of type RectangleObject, volume of type ResidualVolume
output : answer of type boolean
1 begin
2   if object.depth ≤ volume.depth and object.length ≤ volume.length and object.height ≤ volume.height then
3     | answer ← true
4   else
5     | answer ← false
6   end
7 end

```

Algorithme 2 : doesObjectFit

Si toutes les dimensions de l'objet sont plus petites que celles du volume, alors il loge dans le volume. Au contraire, si une des dimensions est supérieure, il dépasse et donc ne loge pas.

Ensuite, jetons un œil à la fonction permettant placer un objet. Le bon fonctionnement de cette procédure est capital, car c'est le bon placement des objets dans l'espace en trois dimensions qui permet d'évaluer un chargement. Cela requiert un petit travail de coordonnées et de dimensions des objets. Pour rappel, les coordonnées d'un volume représentent son coin inférieur gauche :

```

Data : residualVolumes, list of residualVolumes
       items, list of items to be placed
       placedItems, list of items already placed
input : object of type RectangleObject, volume of type ResidualVolume, indiceObject of type Integer
1 begin
2   object.setPosition(volume.Xposition, volume.Yposition, volume.Zposition)
3   v1 ← newResidualVolume(volume.length - object.length, volume.height, object.depth)
4   v2 ← newResidualVolume(object.length, volume.height - object.height, object.depth)
5   v3 ← newResidualVolume(volume.length, volume.height, volume.depth - object.depth)
6   v1.setPosition(volume.Xposition + object.length, volume.Yposition, volume.Zposition)
7   v2.setPosition(volume.Xposition, volume.Yposition, volume.Zposition + object.height)
8   v3.setPosition(volume.Xposition, volume.Yposition + object.depth, volume.Zposition)
9   residualVolumes.remove(volume)
10  residualVolumes.add(v1)
11  residualVolumes.add(v3)
12  if object.isAcceptStack is false then
13    | v2.unusable ← true
14  end
15  residualVolumes.add(v2)
16  placedItems.add(object)
17  items.removeAt(indiceObject)
18 end

```

Algorithme 3 : placeObject

Le principe de la fonction de placement des objets est assez simple. On crée trois nouveaux sous volumes, inclus dans le volume passé en paramètre. On leur attribue des dimensions bien particulières, en jouant sur les dimensions du volume de base et celles de l'objet à placer. On prend également bien soin de marquer le volume qui se situera au dessus de l'objet. En effet, ce volume doit être marqué avec un flag booléen pour savoir s'il est utilisable ou non. On attribuera *true* à ce flag si l'objet placé n'accepte rien sur sa face supérieure, ce que nous reverrons dans la partie concernant les contraintes prises en compte. Une fois les volumes créés, on supprime le volume initial de la liste *residualVolumes*, et on y ajoute les trois nouveaux. Puis, on retire l'objet à placer de la liste *items*, et on le place dans la liste *placedItems*.

La dernière procédure que nous pouvons présenter est *mergeResidualSpaces*. Je n'ai pas jugé pertinent de réaliser l'algorithme dans ce document, mais je vais cependant l'expliquer. L'idée derrière cette procédure est de réussir à réduire les petits volumes inutilisables. En effet, au fur et à mesure de l'exécution de la procédure *compute*, des volumes vont se retrouver adjacents,

mais seront quand même considérés par deux objets distincts, et traités comme tel. On voudrais fusionner ces volumes adjacents pour les rendre plus gros, augmenter donc les chances pour qu'un objet puisse y loger, mais également réduire la taille de la liste `residualVolumes`, et donc augmenter de manière globale les performances de l'algorithme. On peut distinguer deux cas de fusion :

- quand les volumes ont la même hauteur, la même largeur, mais pas la même profondeur. Autrement dit, ils ont les mêmes dimensions sauf pour l'axe Y, celui partant du fond du camion vers l'arrière. Ils doivent également être adjacents, ce qui signifie que leur position sur les axes Y et Z doivent être identiques. Cette configuration garantit un alignement des deux coins inférieurs gauche sur l'axe Y.
- quand les volumes ont la même hauteur, la même profondeur, mais pas la même largeur. Autrement dit, ils ont les mêmes dimensions sauf pour l'axe X, celui partant de la gauche du camion vers la droite sur camion. Ils doivent également être adjacents, ce qui signifie que leur position doivent être identiques cette fois sur les axes Y et X. Cette configuration garantit donc un alignement des deux coins inférieurs gauche sur l'axe X.

La procédure de fusion des espaces est également récursive. La condition d'arrêt est une valeur booléenne, disant si il reste des espaces potentiellement fusionnables. On parcourt alors la liste `residualVolumes` deux fois, en comparant chaque sous volume avec tous les autres. Dès que l'on trouve un couple qui remplit l'une des deux conditions citées plus haut, on crée un nouveau sous-volume qui est la fusion en trois dimensions des deux sous volumes correspondants. On supprime les deux anciens, et on place le nouveau dans `residualVolumes`. On arrête alors l'itération, et on rappelle la fonction. Si on parcourt tous les volumes et qu'aucun couple ne présente des possibilités de fusion, on passe la valeur booléenne à false, ce qui aura pour effet de stopper la chaîne d'appels récursifs. Comme on l'a vu dans la fonction `compute`, cette procédure de fusion est appelée à chaque placement d'objet, donc à chaque itération.

Nous avons donc vu les composantes principales permettant le fonctionnement de l'algorithme : la sélection des volumes, des objets, le placement des objets, ainsi que la fusion des espaces résiduels. Cependant, bien d'autres points et concepts complémentaires restent à aborder, ce que nous verrons dans les parties suivantes.

14 Technologies et instances de données

J'aimerais faire un point rapide sur les technologies utilisées lors de la phase de développement. L'objectif principal du projet étant de développer un algorithme, le choix du langage n'avait pas vraiment d'importance. En revanche, l'entreprise KeenSaas implémente la majeure partie de ses solutions en Ruby avec le framework RubyOnRails. Par souci de temps, j'ai préféré ne pas développer en Ruby, pensant que la phase d'apprentissage pourrait me ralentir. Ainsi, j'ai choisi de coder en Java, car c'est le langage que je maîtrise le mieux, et celui avec lequel je suis le plus à l'aise. L'algorithme et l'ensemble des classes créés dans ce projet peuvent toujours être retranscrits en Ruby si nécessaire, ce langage étant également orienté objet. J'ai développé le projet sur l'environnement de développement IntelliJ Idea.

Comme présenté en annexe, j'ai utilisé un visualisateur 3D développé par KeenSaas pour visualiser mes résultats. Cet outil prend en paramètre un fichier texte formaté de manière spécifique. Afin de fournir ce fichier à l'outil, il faut utiliser un logiciel de connexion SSH pour se connecter au serveur distant. Durant ma phase de développement, j'ai utilisé FileZilla.

Afin de pouvoir tester le code que je produisais, j'ai eu besoin d'instances de données. Ces instances, dans la mesure du possible, devaient être proches de la réalité, et pourquoi pas même des instances réelles, telles qu'elles peuvent être directement chez les potentiels utilisateurs finaux, comme par exemple des entreprises de livraison(But). J'ai, à plusieurs reprises, échangé avec KeenSaas pour qu'ils parlent à leurs partenaires et obtiennent de telles instances. Malheureusement, les échanges n'ont pas été fructueux, et j'ai dû me débrouiller avec des instances que je construisais moi-même. Cependant, comme on l'a vu dans la partie Etat de l'art, il existe beaucoup de types d'instances différents : fortement homogène, faiblement homogène, fortement hétérogène, etc. Ainsi, je ne garantis pas que les instances que j'ai créées soient parfaitement cohérentes. En revanche, elles me permettaient de faire les tests dont j'avais besoin pour garantir le bon fonctionnement de mon algorithme.

Pour ce faire, j'ai donc créé, dans ma classe Data, une fonction de lecture de fichier texte permettant de générer toutes les données (camion, objets) nécessaires à une exécution du programme. Le format de ces fichiers textes est le suivant :

```
1 100
2 2320 3000 11900
3 150 150 150
4 150 150 150
5 150 150 150
6 150 150 150
7 150 150 150
8 150 150 150
9 150 150 150
10 150 150 150
11 150 150 150
12 150 150 150
13 580 137 628
14 580 137 628
15 580 137 628
16 580 137 628
17 580 137 628
18 580 137 628
19 580 137 628
20 580 137 628
```

Figure 9 – *Format de fichier d'instance*

La fonction de lecture fonctionne très simplement. La première ligne est la taille de l'instance, donc le nombre d'objets à charger. La seconde représente les dimensions du camion. Plus exactement, la première valeur est la largeur, la seconde la hauteur, et la troisième la profondeur. Ensuite, pour chaque ligne suivante, nous avons les dimensions des objets. Une ligne représente un objet, il y a donc autant de lignes que la valeur lue à la première ligne. Pareillement au camion, la première valeur est la largeur de l'objet, la seconde sa hauteur et la dernière sa profondeur.

J'ai essayé de créer un ensemble d'instances un peu différentes. Typiquement, j'avais sous la main des instances qui remplitaient le camion presque totalement, et à l'inverse de petites instances qui logeaient largement. Aussi, j'ai de grandes instances hétérogènes, de grandes instances homogènes, ainsi que leur homologue respectif de petite taille. Cette variété de taille et de nature d'instance m'a permis de tester la robustesse de mon algorithme, et de repérer dans quels cas il rencontre des difficultés, et dans quels cas il est très performant.

15 Contraintes prises en compte

Dans cette partie, j'aimerais maintenant lister toutes les contraintes que mon algorithme prend en compte, par rapport aux contraintes et au besoin initialement émis par l'entreprise. Pour chaque contrainte, j'essaierai d'expliquer la logique et le fonctionnement.

- degrés de liberté sur trois axes : pour rappel, cette contrainte permet de pouvoir effectuer des rotations des objets selon les trois axes de l'espace X, Y et Z. L'idée est que certains objets peuvent parfois être tournés dans tout les sens, mais d'autres non, comme par exemple des frigos. Ainsi, il fallait pouvoir, pour chacun des objets, dire s'il acceptait ou non des rotations sur les différents axes. Dans ma modélisation, c'est ce que je fais grâce aux trois attributs de la classe `RectangleObject` qui sont `rotateXAxis`, `rotateYAxis` et `rotateZAxis`. Ces attributs sont booléens, et si leur valeur est à `true`, la rotation est possible, et elle ne l'est pas dans le cas contraire. La valeur de cet attribut est bien spécifique à chaque objet. Actuellement, la lecture dans le fichier d'instance ne permet pas de savoir si tel ou tel objet possède telle ou telle rotation. Cependant, à posteriori, il faut imaginer que chaque objet saisi par l'utilisateur possèdera dans ses données ces contraintes.

Si on fait le lien avec la procédure `compute` présentée précédemment, on remarque que les rotations que l'on tente de faire sur chaque objet ne sont faites que si la valeur de l'attribut `rotate` correspondant à l'axe en question est à `true`. Ce fonctionnement a pour conséquence d'éviter tout placement d'objet dans une ou des configurations indésirables.

Pour finir, voici comment sont traduites les rotations dans mon code, en termes de dimensions des objets :

- la rotation sur l'axe X : la profondeur devient la hauteur, et la hauteur devient la profondeur. On ne touche pas à la largeur.
 - la rotation sur l'axe Y : la profondeur devient la largeur, et la largeur devient la profondeur. On ne touche pas à la hauteur.
 - la rotation sur l'axe Z : la hauteur devient la largeur, et la largeur devient la hauteur. On ne touche pas à la profondeur.
- gerbabilité : cette contrainte nous dit si un objet accepte ou non d'autres objets sur sa face supérieure. Typiquement, on peut imaginer des objets d'une fragilité extrême, pour lesquels on ne veut rien poser dessus. Cette fois, c'est grâce à l'attribut `acceptStack` de la classe `RectangleObject` que l'on peut savoir. Cet attribut est booléen, et il est à `true` si l'objet peut accepter d'autres objets sur sa face supérieure, et à `false` dans le cas contraire.

En termes de code, cela se traduit d'une façon bien particulière. L'idée est que, lors de l'appel à la fonction `placeObject` présentée plus haut avec un objet possédant un `acceptStack` à `false`, on vient créer le sous volume situé au dessus de l'objet placé, mais on le rend inutilisable. En effet, chaque volume possède un attribut `unusable`, disant si ce volume peut être utilisé pour placer des objets ou non. En d'autres termes, si on passe cet attribut à `true`, on condamne le volume à ne jamais être utilisé lors du placement d'un objet ou lors de la fusion d'espaces libres. C'est en quelques sortes un volume perdu, qui ne sera jamais utilisé. Cette notion permet de condamner du volume au dessus des objets n'acceptant rien sur leur face supérieure, et de respecter la contrainte de gerbabilité.

- l'ordre de livraison des clients : cette contrainte nous dit que l'on souhaiterait placer les objets dans le camion en respectant une logique de déchargement. On veut pouvoir avoir à portée de main les objets du premier client que l'on va livrer d'abord, puis ceux du second client, etc. En gros, le chargement doit se diviser en un nombre de 'blocs', chaque bloc contenant les objets d'un client. J'ai réalisé cette contrainte en raisonnant sur l'ensemble items, représentant les objets à charger dans le camion.

Pour être tout à fait exact, l'algorithme ne s'exécute pas sur items, mais sur un autre ensemble appelé itemsParClient. Ce dernier est une liste de liste, chaque sous liste étant la liste des objets d'un client particulier. En d'autres termes, on divise l'ensemble items en autant de sous ensembles qu'il y a de client. Cette division s'effectue juste après le chargement de données. En termes d'exécution, on vient appeler la fonction compute non plus une fois sur items, mais sur chaque sous ensemble de itemsParClient de manière consécutive. Ce fonctionnement particulier a pour effet de placer d'abord les objets du premier client, puis ceux du second, etc. Au final, comme nous le verrons dans la partie de visualisation des résultats, on peut clairement distinguer des blocs de chargement, chacun relatif à un client.

- hauteur de chargement maximale : cette contrainte impose de fixer une limite de hauteur pour le chargement. On veut parfois que le chargement soit étalé au maximum pour garantir une certaine stabilité. La prise en compte de cette contrainte est directement en relation avec les indicateurs de qualité de solution présentés dans la partie suivante. Néanmoins, je vais expliquer ici les principes utilisés.

D'abord, la prise en compte d'une telle contrainte peut paraître assez simple : nous n'avons qu'à remplacer la hauteur initiale du camion par la valeur de hauteur souhaitée. Ainsi, le chargement ne dépassera jamais cette hauteur, et l'implémentation reste basique. Cependant, nous avons choisi de pousser la notion plus loin, afin de permettre une certaine flexibilité. L'idée est la suivante : laisser à l'utilisateur le choix du "degré d'étalement" du chargement. Même si la notion de hauteur n'est pas directement apparente, elle est très étroitement liée à la notion d'étalement du chargement. Plus le chargement est étalé, plus sa hauteur est basse, et plus le chargement est condensé en profondeur, plus sa hauteur est élevée.

Ainsi, on vient lier trois indicateurs : le volume du chargement, la distance entre la hauteur du chargement et le toit du camion, et la distance entre la limite du chargement en profondeur et la porte de chargement. Pour être plus clair, voici à quoi cela ressemble :

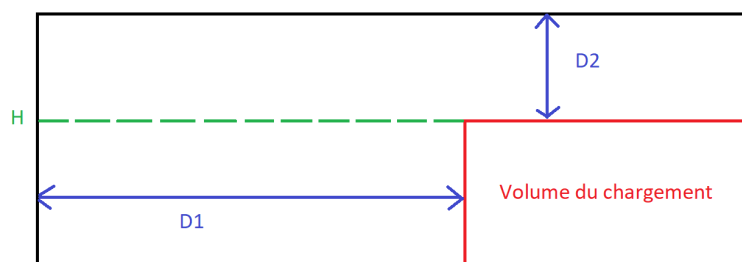


Figure 10 – Calcul de la hauteur de chargement

Au début du programme, on demande à l'utilisateur de saisir une valeur 'alpha' entre 0 et 1. Une valeur à 0 représente un chargement étalé au maximum. À l'inverse, une valeur à 1 représente un chargement étalé le moins possible. Grâce au schéma précédent, on peut voir que la hauteur H est directement liée aux distances D1 et D2. L'idée est de déterminer, grâce au volume du chargement, une valeur de H maximale, notée Hmax, qui ne sera théoriquement jamais franchissable. Typiquement, pour de très petits chargements, il est possible que la hauteur Hmax ne soit pas égale à la hauteur initiale du camion, étant donné que même en empilant tous les objets, on n'atteint pas cette hauteur initiale. La hauteur Hmax est donc directement dépendante du volume du chargement. Avec ces informations, on peut poser plusieurs choses :

- $D1 = (volumeCamion - volumeChargement) / (hauteurCamion * largeurCamion)$
- $Hmax = volumeChargement / (largeurCamion * (profondeurCamion - (alpha * D1)))$

Grâce à ces formules géométriques, on peut réussir à déterminer une hauteur maximale à ne pas dépasser, directement liée au degré de chargement alpha saisi par l'utilisateur en début de programme, et également au volume du chargement lu dans l'instance de donnée. On calcule donc d'abord la valeur de D1 en fonction du volume du chargement. Puis, on en déduit la valeur de Hmax en liant D1 à alpha, le degré d'étalement. Dans la partie sur les résultats, nous aurons l'occasion de voir quels effets ce paramètre alpha peut avoir sur la solution calculée, et nous verrons clairement les différences de chargement en fonction de la valeur saisie.

16 Les solutions et les indicateurs de qualité

16.1 Multiples solutions

L'une des contraintes émises par KeenSaas était de pouvoir générer plusieurs solutions, afin de permettre à l'utilisateur de choisir, si l'une des solutions ne lui convenait pas. Dans mon cas, j'ai développé les deux stratégies. En effet, il est actuellement possible de choisir si on veut générer une unique solution, ou plusieurs. Avant de rentrer dans ces détails, expliquons comment il est possible de générer plusieurs solutions.

Pour ce faire, il suffit de changer la séquence des objets dans l'ensemble de base items. En effet, les choix de l'algorithme avec une séquence particulière sont différents avec une séquence différente. Typiquement, la méthode qui a été utilisée est celle des swaps. Un swap consiste à échanger la place de deux éléments dans la liste des objets. L'idéal est de couvrir toutes les combinaisons possibles, en échangeant tous les éléments avec tous les autres. Pour chaque séquence, on exécute l'algorithme, et on a une solution différente.

Par contre, tout cela n'est valable que si nous exécutons une stratégie de sélection d'objets particulière dans la fonction compute. En effet, considérons le cas où nous adoptons une stratégie de firstFit. Alors, les swaps auront un intérêt, car il est possible que le premier objet trouvé logé dans le volume soit différent d'une séquence à l'autre. En revanche, si nous adoptons une stratégie de bestFit, les swaps perdent tout leur intérêt. Étant donné qu'on va chercher le meilleur objet selon un certain critère, on va, toujours, dans la sélection de l'objet

dans la fonction compute, parcourir tous les objets pour chaque sous volume. Ainsi, peu importe où se trouve l'objet dans la séquence affectée par les swaps, on le sélectionnera toujours de toute manière, car c'est toujours le meilleur, où qu'il soit. Il est donc nécessaire d'être vigilant : les swaps et donc les multiples solutions sont impossibles si on raisonne avec un bestFit, mais ils le sont si on applique un firstFit.

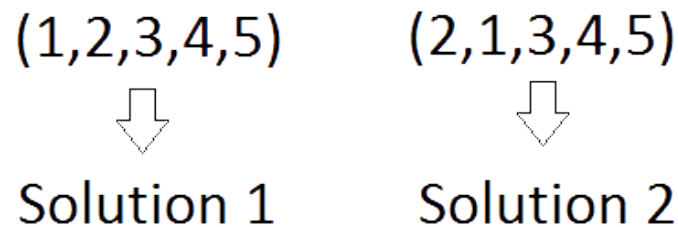


Figure 11 – Stratégie de swap

Comme je l'ai dit auparavant, les deux options sont possibles dans le code actuel. J'ai en fait un fichier de paramètres intégré au projet. Il existe un paramètre dans ce fichier appelé 'swap'. Si la valeur de ce paramètre est à true, alors le programme agit en fonction et exécute l'algorithme avec une stratégie de swap sur l'ensemble des objets à charger, une stratégie de firstFit sur la sélection des objets, en adaptant les portions de code appelées en fonction. Au contraire, si le paramètre est à false, il passe en mode bestFit, et ne générera qu'une seule solution.

Pour appuyer ces différentes configurations, nous avons la classe executionModes. Dans cette dernière, on retrouve des fonctions distinctes d'exécutions du programme pour une stratégie avec ou sans swaps. Si on ne veut pas de swaps, la fonction compute n'est appelée qu'une fois par sous ensemble de clients, comme dit précédemment. En revanche, dans le cas d'une stratégie avec swaps, on appelle la fonction compute autant de fois que nous avons de séquences. Et c'est également ici qu'il faut faire très attention. En effet, le fonctionnement habituel des swaps dirait de parcourir la liste des objets items, et de faire des swaps avec tous les éléments, ce qui génèrerait une complexité en n^2 .

Cependant, rappelons nous que nous avons plusieurs clients, et que nous ne raisonnons plus sur notre liste items pour exécuter l'algorithme, mais sur la liste de listes itemsParClient. Ainsi, on ne veut surtout pas effectuer un swap de tous les éléments avec tous les autres. On ne veut pas, par exemple, swap un objet appartenant au premier client avec un objet appartenant au dernier, sous peine de placer cet objet dans un bloc de client qui ne lui appartient pas. La stratégie de swap s'applique ici sur chaque sous liste de clients. On swap tous les éléments d'une sous liste clients avec tous les autres, toujours de la même sous liste clients. Alors, les swaps sont effectifs et on teste toujours toutes les combinaisons possibles, simplement de manière locale aux clients. De plus, la complexité est réduite à du n^2/p , p étant le nombre de clients.

Voici donc comment notre programme peut gérer une exécution avec génération d'une unique solution, ou bien plusieurs. Maintenant que nous pouvons avoir plusieurs solutions, nous devons être capables de les comparer. La section suivante nous dit comment comparer une solution avec une autre, et par quels critères de qualité nous pouvons les distinguer.

Nous souhaitons maintenant être capables de distinguer deux solutions, et de dire si une solution est bonne ou non, selon certains critères. Autrement dit, nous voulons juger de la qualité d'une solution. Pour ce faire, nous sommes passés par deux étapes au cours du temps, que je vais présenter.

Les premiers indicateurs de qualités que j'ai implémenté étaient en relation directe avec deux contraintes émises par KeenSaas : la contrainte de hauteur maximale, et la contrainte de centrage de la charge dans le camion. A ce stade de développement, la contrainte de hauteur maximale n'était pas encore prise en compte comme présentée dans la partie précédente. La contrainte de centrage de la charge nous dit que le chargement doit être au mieux équilibré dans le camion, en termes de poids. Par exemple, on ne veut pas tous les objets très lourds d'un côté, et tous les objets légers de l'autre, sous peine d'avoir un camion qui penche et des soucis de stabilité lors de la livraison. Ainsi, nous avons voulu lier ces deux contraintes pour en dégager trois critères :

- la distance entre la hauteur du chargement et le toit du camion : tout comme on peut se référer à la [Figure 10](#), ce critère est en relation avec la hauteur maximale de chargement.
- la distance entre les portes du camion et la profondeur maximale du chargement. Sur la [Figure 10](#), cette distance correspond à D1. Tout comme le critère précédent, il est lié à la hauteur du chargement.
- la distance entre le centre de gravité du camion et le barycentre des masses des objets chargés. Le centre de gravité du camion représente le point central du parallélépipède. Le barycentre des masses est le point central de l'ensemble des centres de gravité des objets, pondérés par leur poids. Cette distance représente un écart de répartition de charge. Si la distance est grande, la charge n'est pas assez bien répartie. En revanche si elle est courte, la charge globale du chargement est correctement répartie.

Une fois ces différents indicateurs fixés, il fallait attribuer une unique valeur de qualité à chaque solution. Pour ce faire, nous avons décidé de pondérer chacun des ces trois critères dans une seule et unique expression. Si on nomme $Z1$ le premier critère, $Z2$ le second et $Z3$ le dernier, la qualité d'une solution est équivalente à $\alpha Z1 + \beta Z2 + \gamma Z3$, où α, β et γ sont des coefficient tels que $\alpha + \beta + \gamma = 1$.

Grâce à ces trois coefficients, on peut pondérer les trois critères à souhait. Si on souhaite uniquement donner beaucoup d'importance au centrage de la charge, on peut initialiser γ à 1, et les deux autres à 0, par exemple. Alors la qualité des solutions sera uniquement exprimée en fonction du centrage de la charge.

Toutes les fonctions de calcul mathématique utiles se trouvent dans la classe MathUtil. Grâce à ces indicateurs, on pouvait déjà donner une valeur de qualité de solution sur les séquences produites par l'algorithme.

Dans un second temps, nous avons décidé de changer ces indicateurs de qualité. En effet, nous prenons maintenant en compte la contrainte de hauteur maximale de chargement. Comme nous l'avons vu dans la partie précédente, elle consiste à fixer une hauteur en début de programme, par le biais d'un coefficient α compris entre 0 et 1, définissant le degré d'étalement du chargement. Ainsi, la distinction des solutions se fait maintenant par un nombre de dépassements de cette hauteur.

Si nous considérons l'algorithme paramétré avec des swaps, alors, pour chaque combinaison d'objets, l'algorithme va dorénavant déterminer pour quelle hauteur il réussit à placer tout les objets de cette combinaison. Nous commençons par fixer notre H_{\max} , qui est une hauteur maximale théorique. Si on ne trouve pas de solutions pour une combinaison donnée et pour cette hauteur H_{\max} , on incrémente H_{\max} d'un certain pas. Puis, on recommence avec la même séquence, jusqu'à ce que l'algorithme arrive à la placer. Alors, la qualité de la solution sera relative à son nombre de dépassement de la hauteur H_{\max} initialement déterminée. Nous avons défini ces dépassements de hauteur comme des pénalités. Chaque solution est alors associée à un nombre de pénalités. Logiquement, la meilleure solution sera donc, en termes de qualité et de respect du critère de hauteur maximale, celle qui aura le moins de points de pénalités, car c'est la solution pour laquelle la hauteur de chargement sera la plus proche de H_{\max} .

17 Tests et étude des résultats

Dans cette partie, nous allons maintenant visualiser quelques résultats de l'algorithme. Notamment, il sera intéressant de voir les chargements générés pour des instances de type différent, ou encore des valeurs de α différentes. Comme nous l'avons déjà dit, j'ai utilisé un outil de visualisation 3D développé par l'entreprise KeenSaas. Cet outil m'a permis de tester la validité des solutions, et de savoir si le code que j'implémentais fonctionnait ou non. C'est mon principal outil de test, même s'il reste très visuel.

Commençons par visualiser l'impact du coefficient α saisi en début de programme, représentant le degré d'étalement du chargement. Voici les résultats pour une instance de 400 petits objets, relativement homogène, avec un α à 1 :

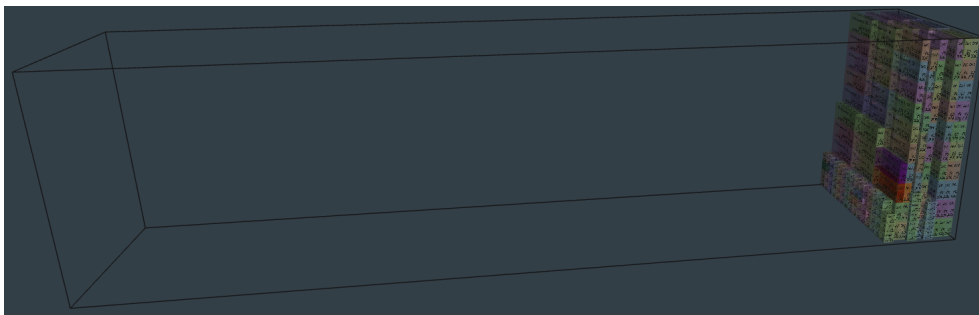


Figure 12 – 400 objets avec α à 1

Voici maintenant la même instance avec un α à 0.9 :

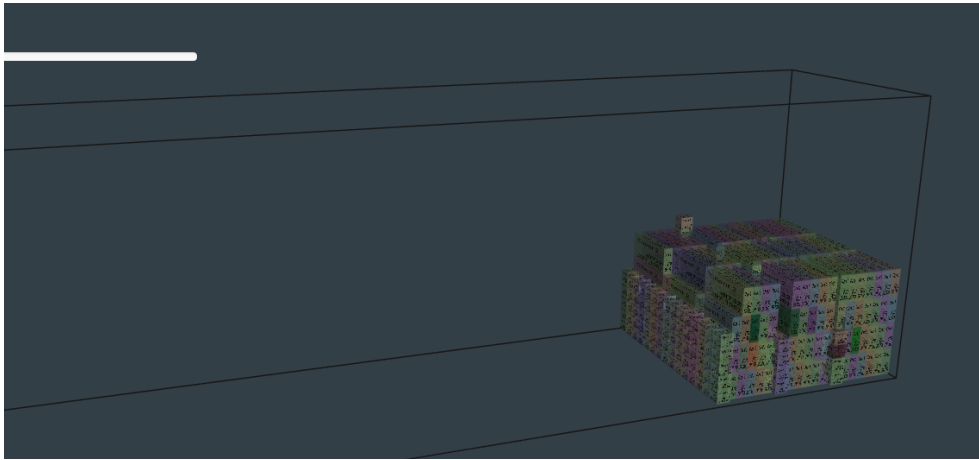


Figure 13 – 400 objets avec α à 0.9

Enfin, avec un α à 0.6 :

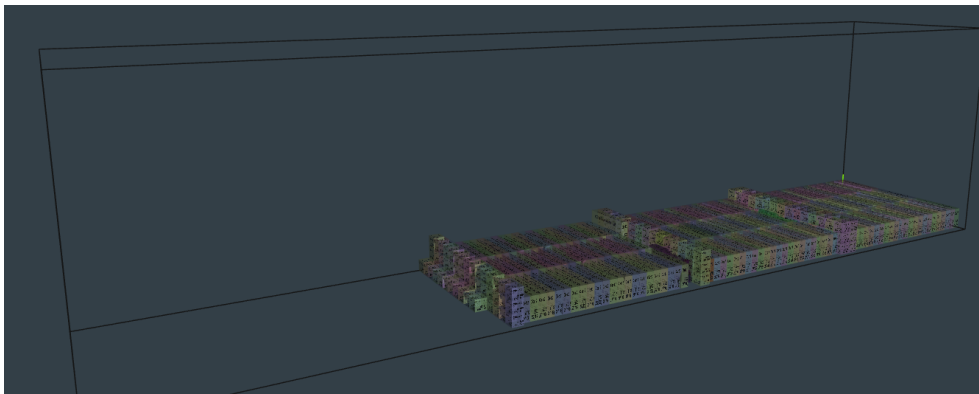


Figure 14 – 400 objets avec α à 0.6

Ces trois captures d'écran nous montrent parfaitement l'impact du paramètre α . Clairement, plus ce dernier sera petit, plus le chargement sera étalé. Il est important de noter les plages d'impact de ce fameux paramètre. En d'autres termes, les valeurs présentant de grosses différences d'étalement du chargement dépendent fortement de la nature de l'instance, tout comme la valeur de H_{max} calculée est dépendante du volume global du chargement. On peut, sur notre exemple, voir qu'avec un α à 0.6, le chargement est déjà étalé presque au maximum. Donc les valeurs inférieures à 0.6 seront presque sans importance, car ne présenteront pas de différences. Cela est dû au fait que le chargement possède un volume très faible. Donc, une petite variation de α affectera grandement la valeur de H_{max} calculée.

Si on raisonne de manière inverse, et qu'on imagine un chargement occupant presque tout l'espace du camion, l'impact du paramètre α sera plus difficilement visible. C'est ce qu'on peut voir sur l'exemple suivant, montrant une instance homogène de 278 objets, mais dont le volume est bien supérieur. Voici les résultats pour un α de 1 :

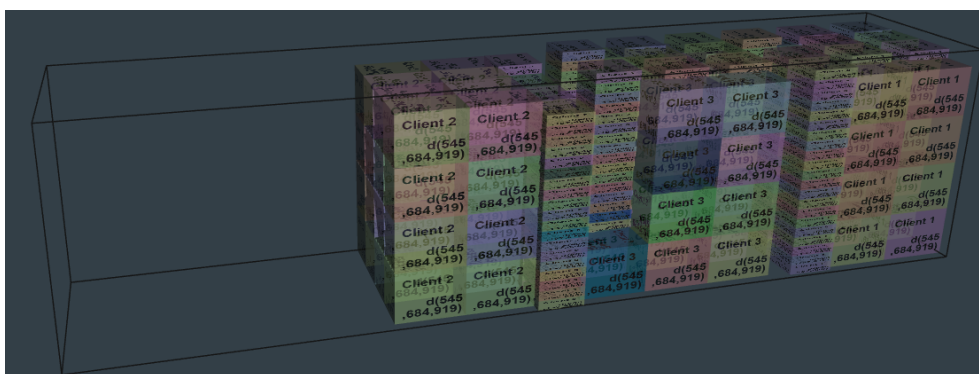


Figure 15 – 278 objets avec α à 1

Voici maintenant le même chargement avec un α à 0.3 :

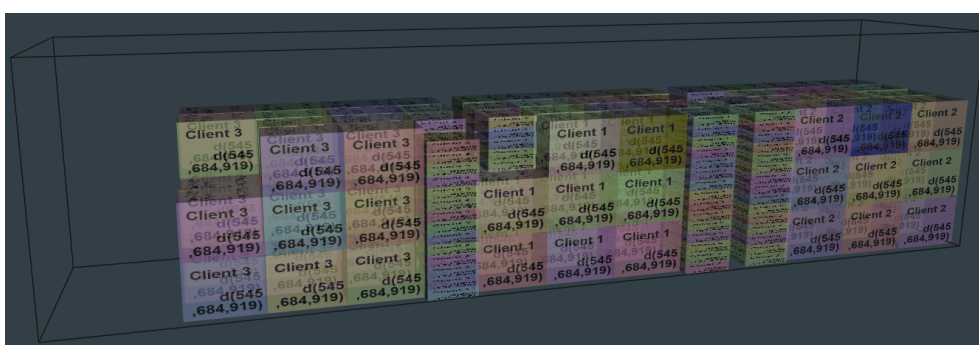


Figure 16 – 278 objets avec α à 0.3

Encore une fois, l'impact du paramètre α est clairement visible. Cependant, pour une variation de α de 1 vers 0.3, on remarque quand même que l'aspect du chargement est assez similaire. Sur la seconde photo, la hauteur de chargement est clairement plus petite, mais la différence de hauteur entre les chargements de la première et la seconde photo n'est pas si grande. Ainsi, à l'inverse de notre premier exemple, du fait que le volume de chargement global soit beaucoup plus important, l'impact du paramètre α demande une plus grande plage de valeurs pour être bien visible.

Il est juste important de noter que l'impact du paramètre α dépend du volume de chargement, tout comme H_{max} . Sur l'exemple précédent, on peut également visualiser la prise en compte de la contrainte de l'ordre des clients. Comme on peut le voir, le chargement global respecte un ordre dans les chargements de chaque client, et aucun objet d'un client n'est mélangé avec ceux d'un autre.

18 Limites de l'algorithme

Finalement, dans cette dernière sous partie, j'aimerais qu'on discute des limites de l'algorithme implémenté. Comme tout problème d'une telle complexité, la problématique de temps de calcul se présente rapidement. Dans mon cas, lorsqu'on j'ai implémenté la multiplicité des solutions avec la gestion des swaps, cette problématique a prit du sens.

Une exécution de l'algorithme reste instantanée. Cependant, lorsqu'on souhaite l'exécuter un nombre important de fois, notamment pour gérer les swaps, l'ajout de ces petites durées devient

vite un temps important. Je ne peux pas fixer de durée d'exécution exacte, mais elle est, encore une fois, totalement dépendant de l'instance lue.

D'abord, et naturellement, la taille de l'instance est un premier facteur de rapidité. En effet, plus l'instance est petite en taille, plus le nombre de swaps sera petit, et donc le nombre total de passage dans l'algorithme réduit. Ensuite, la valeur du α saisie est un second facteur. Il faut bien avoir en tête que plus le α saisi est petit, plus la hauteur maximale autorisée est réduite. Cette hauteur, H_{\max} , est comme un second "toit" du camion, que l'on rabaisse. Ainsi, une valeur de α à 1 ne rabaissera pas ou presque pas la hauteur originale du camion. Donc, pour tout type de chargement, avec des volumes ou tailles variables à condition que le chargement total loge dans le camion, l'algorithme arrivera presque toujours du premier coup à placer tout les objets. Ainsi, aucun point de pénalité ne sera attribué aux solutions, et l'algorithme n'aura pas besoin de se ré-exécuter avec une valeur de H_{\max} différente.

En revanche, considérons maintenant une valeur de α proche de 0 ou égale à 0. Dans ce cas, on réduit fortement la valeur de hauteur maximale, et donc la valeur du H_{\max} originale est très petite. On peut s'imaginer le toit du camion qui se rabaisse de manière importante, et qui au final vient créer un volume disponible initial fortement réduit, écrasé. Ainsi, pour la majorité des chargements, l'algorithme n'arrivera pas à placer tous les objets dans ce volume fortement réduit du premier coup. La hauteur étant trop faible, il attribuera des points de pénalités à la solution, et s'exécutera de nouveau avec une nouvelle valeur de H_{\max} . Cette nouvelle valeur augmente le volume initial disponible, elle "remonte le toit" du camion, et donc favorise cette fois le placement de tous les objets. On continue ainsi jusqu'au succès de l'algorithme, lorsque tous les objets sont placés, et ce, pour chaque séquence de swaps.

Par conséquent, plus la valeur de α saisie est faible, plus l'algorithme va itérer pour placer les objets, pour chaque séquence de swaps. Plus l'algorithme itère, plus la rapidité de l'exécution globale en est affectée.

Un dernier facteur concernant la rapidité de l'exécution est le nombre de clients qu'il y a à livrer. Rappelons nous que notre algorithme va placer les objets de chaque client dans l'ordre, travaillant sur une liste de sous listes d'objets, chacune des sous listes étant propre aux objets d'un seul et même client. Cependant, il faut maintenant lier cela à notre stratégie de swaps. Comme on l'a déjà mentionné, on ne veut surtout pas faire de swap entre les objets de différents clients, donc deux sous listes différentes. Naturellement donc, les swaps sont effectués de manière locale sur les sous listes, les unes après les autres. Donc, pour un nombre d'objets fixe, si on augmente le nombre de clients, la nombre de sous listes augmente, mais leur taille respective diminue. Si leur taille diminue, le nombre de swaps pour chaque sous liste diminue également. Dans ce cas et comme expliqué dans la partie sur la multiplicité des solutions, on réduit la complexité générale du programme.

Un exemple avec un ensemble de 12 objets, considérant chaque client avec le même nombre d'objets :

- pour un client, on effectue 12^2 swaps, donc 144.
- pour deux clients, chacun ayant 6 objets, on effectue 6^2 swaps pour le premier, et 6^2 swaps pour le second, un total de 72 swaps.
- pour trois clients, chacun ayant 4 objets, on effectue 4^2 swaps pour le premier, 4^2 swaps

pour le second et 4^2 swaps pour le dernier, donc un total de 48 swaps.

Pour contrer ce problème de rapidité d'exécution dans certains cas, j'ai choisi de demander à l'utilisateur quelques informations permettant de brider l'algorithme. D'abord, on fixe un temps maximal d'exécution en secondes. Au début du programme, l'utilisateur saisit un temps en seconde, et l'algorithme s'aide d'un timer pour se stopper au moment où ce temps arrive à expiration.

De plus, on peut aussi réduire directement le nombre de swaps. On demande à l'utilisateur en début de programme le nombre d'éléments sur lesquels on veut effectuer des swaps, par rapport à la taille totale de l'instance. Par exemple, pour une instance de 300 objets, il peut saisir 150, et uniquement les 150 premiers objets seront swap avec tous les autres de leur sous liste de clients respective. En d'autres termes, si on imagine ma double boucle itérant sur les objets, cette valeur vient limiter les indices étudiés sur la première boucle.

Sixième partie

Bilan et conclusion

19 Perspectives

Dans cette partie, nous allons discuter des points que j'aurais souhaité aborder si j'avais eu plus de temps. Que ce soit des nouvelles fonctionnalités ou des améliorations du fonctionnement existant, nous pouvons en distinguer quelques unes.

Le premier point concerne une contrainte qui n'a pas été prise en compte : le poids maximal sur la face supérieure d'un objet. Si un objet est gerbable, il ne peut accepter qu'un certain poids sur sa colonne supérieure. La prise en compte de cette contrainte peut paraître banale, mais elle est en réalité assez complexe. En effet, cela nécessiterait de savoir quel objet est sur quel ou quels objets. Un objet peut tout à fait être placé sur deux objets différents, grâce à notre fonctionnalité de fusion des espaces résiduels. Actuellement, aucun lien n'existe entre les objets. Nous ne pouvons pas, à part parcourir la liste des objets pour déterminer leur position et éventuellement détecter des objets adjacents, savoir si deux objets sont collés dans le chargement. Une possibilité serait de garder en mémoire, pour chaque objet placé, l'objet qui est dessous. Mais si il y en a plusieurs, cela devient tout de suite compliqué.

Le second point est une amélioration du comportement actuel de l'algorithme. Pour garantir un chargement compacté vers le fond, sans espaces résiduels, beaucoup d'algorithmes effectuent une "poussée" vers le fond. En d'autres termes, pour chaque objet placé, à intervalles réguliers, on essaie de les pousser vers le fond, jusqu'à rencontrer le mur du fond du camion, ou un autre objet. Même si le concept serait, je pense, plus adapté pour des algorithmes construisant le chargement par planchers, l'idée pourrait permettre de réduire le nombre de sous espaces résiduels. Cependant, la mise en place d'un tel fonctionnement serait extrêmement complexe dans notre cas. En effet, pousser un objet vers le fond signifierait une chose : l'objet que l'on pousse, initialement contenu dans un sous volume de l'espace, doit sortir de ce sous volume et entrer dans un autre sous volume. Cela implique la modification de l'état du découpage de l'espace à un instant t , ce qui serait sans doute beaucoup trop difficile à gérer. Même si le concept est intéressant, je le considère compliqué à appliquer sur notre algorithme.

Ensuite, nous avons également pensé à une autre amélioration, concernant la fixation de la hauteur H_{\max} . Actuellement, après fixation du paramètre α et le calcul de H_{\max} , on exécute l'algorithme avec une séquence donnée tant que les objets ne rentrent pas dans le camion, en augmentant H_{\max} à chaque itération d'un certain pas. Cette approche peut parfois prendre beaucoup d'itérations pour trouver une hauteur idéale telle que le chargement loge. Nous avons pensé à accélérer ce processus, en appliquant une dichotomie sur la fixation de H_{\max} . Typiquement, au lieu d'augmenter H_{\max} d'un certain pas, on garde en mémoire deux bornes, qui sont initialement le toit du camion et le premier H_{\max} . Ensuite, on applique une dichotomie classique. Cette approche pourrait permettre de gagner un nombre d'itérations important, au cas où l'estimation initiale de H_{\max} soit vraiment beaucoup trop faible par rapport à la hauteur nécessaire au chargement. Après avoir tenté de l'implémenter, j'ai rencontré un problème qui est que je n'ai pas réussi à définir une condition d'arrêt correcte pour cette dichotomie. Pour moi, on itère sans jamais s'arrêter car on peut toujours définir une hauteur plus exacte où le chargement loge. Cependant, on aurait pu imaginé la condition d'arrêt suivante : on stoppe quand la distance entre la hauteur trouvée et la précédente hauteur pour laquelle le chargement logeait est inférieure à une certaine valeur. Je n'ai pas eu le temps d'explorer cette possibilité, mais la condition d'arrêt n'est clairement pas évidente à trouver.

Enfin, j'aurais pu prendre en compte une dernière amélioration. En effet, pour des raisons de stabilité globale du chargement, on souhaiterait à chaque placement d'objet favoriser sa position pour laquelle la majorité de surface est en contact avec le sol ou l'objet du dessous. Cependant, actuellement, aucune distinction n'est faite entre les différentes rotations de l'objet au placement. D'ailleurs, dès que l'algorithme trouve une position de l'objet qui le fait rentrer dans le volume, il le place sans essayer les autres. Ici, l'idée serait de tester chaque position, celle de départ et celles qui suivent les trois différentes rotations, et les comparer, afin de sélectionner la meilleure, selon le critère précédemment cité. Le résultat d'une telle gestion des rotations assurerait des objets avec une surface de contact et d'appui maximale, pour garantir la stabilité du chargement.

Le nombre de contraintes qu'il reste ensuite à prendre en compte est très important. Les objets de différentes formes, les chargements par différentes portes, la charge répartie sur les essieux...les possibilités sont nombreuses et c'est ce qui fait la complexité du problème.

20 Bilan personnel et conclusion

Au final, l'algorithme développé respecte une partie des contraintes imposées par KeenSaas, mais certaines restent en suspend. Les choix et les directions prises ne sont pas les seules solutions à ce problème, et il existe une quantité d'implémentations différentes qui peuvent y répondre.

Mon implémentation est loin d'être parfaite, mais je reste néanmoins très satisfait du travail accompli. Non seulement grâce au fait que l'algorithme est fonctionnel et propose des résultats parfois très bons, mais également car le projet dans sa globalité m'a plu. Je suis content d'avoir pu répondre à un réel problème. Celui-ci étant un problème assez complexe, il a demandé des heures de réflexion sur différents aspects pas faciles à appréhender, qu'ils soient mathématiques ou informatiques. J'ai particulièrement apprécié cette complexité dans la réflexion, qui procure un sentiment de satisfaction supplémentaire, selon moi, lorsque la difficulté est surmontée. L'aide de M. Billaut sur certains aspects a été indispensable pour les comprendre.

Le fait de prendre à part à un projet réel d'une entreprise a également été pour moi une source de motivation supplémentaire. Ne souhaitant pas décevoir, je me suis tenu à utiliser mes capacités à leur maximum pour mener à bien ce projet. Le résultat final reste grandement améliorable, mais j'espère qu'il servira l'entreprise autant qu'il le pourra.

Annexes

A

Reste du cahier de spécifications

1 Interfaces

1.1 Interfaces matériel/logiciel

L'outil proposé sera, comme dit précédemment, accessible via une URL dans un navigateur Web. Ainsi, que ce soit ordinateur, tablette, ou même téléphone portable, le périphérique matériel sera unique.

Les seules influences que le changement de matériel puisse avoir peuvent se manifester en changeant de navigateur Web. L'interopérabilité sur l'ensemble des navigateurs relève ainsi de la responsabilité de KeenSaas.

1.2 Interfaces logiciel/logiciel

Aucune particularité au niveau des interfaces logiciel/logiciel n'est à spécifier. Le système reste très simple en interne, nous n'aurons qu'un algorithme qui prend des données en entrée et ressort un résultat. Il n'a pas besoin d'accéder à une base de données, ni d'échanger aucune informations avec d'autres applications.

La seule petite restriction pourrait apparaître dans le cas de l'utilisation d'une bibliothèque logicielle dont la licence est payante. Typiquement, des tests ont été faits avec la librairie LocalSolver. Dans un cas similaire, il faut alors s'assurer que nous possédons constamment une licence valide.

1.3 Interfaces homme/machine

L'IHM de l'outil possède déjà une première version développée par KeenSaas. En voici un aperçu :

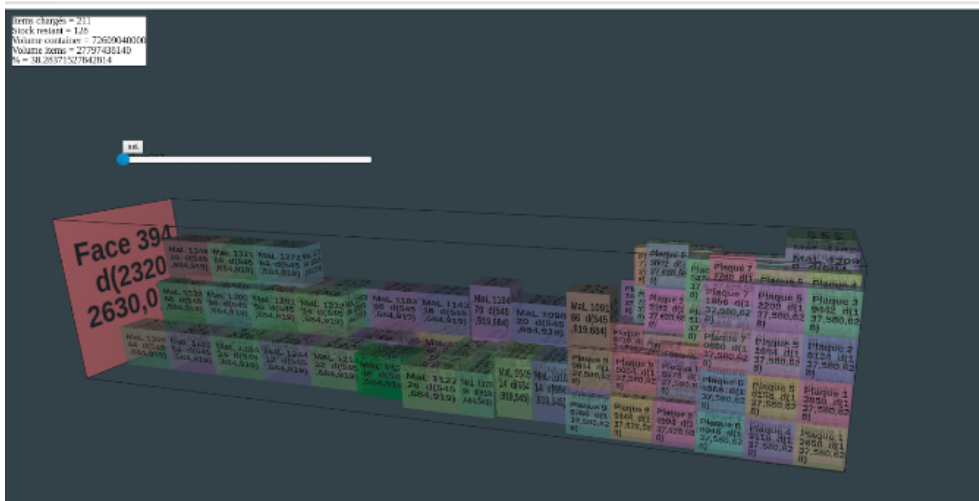


Figure 1 – Visualisateur 3D développé par KeenSaas

C'est une vue en trois dimensions du plan de chargement proposé par l'algorithme. On peut naviguer dans l'image et la pivoter avec la souris et le zoom. Nous avons une visualisation du contenant parallélépipédique et de tous les items stockés dedans. Pour chaque item, on peut voir une couleur de boîte, un label ainsi que ses dimensions.

En haut à gauche de l'écran, quelques informations sont présentes pour l'utilisateur : le nombre d'items chargés, le volume total du contenant, le volume total des items et le taux d'occupation du contenant

Cette vue permet donc de facilement visualiser le plan de chargement en trois dimensions. En plus d'être un prototype du produit fini, cet sera très utile pour afficher les résultats des tests effectués.

L'accès et l'utilisation de ce visualisateur sont un peu particuliers. En effet, c'est un outil qui est présent sur les serveurs de KeenSaas, à Orléans. On y accède via une url, et la vue est affichée sur un navigateur web. Cependant, l'outil va venir lire un fichier texte représentant le plan de chargement, pour pouvoir l'afficher à l'écran. C'est ce fichier qu'on veut venir éditer si on souhaite visualiser un plan de chargement particulier.

La structure de ce fichier doit respecter un format bien particulier, qui est celui-ci :

5	5	5				
3	3	1	2	1	2#d2acb2	Objet 1
0	0	2	1	1	1#acb9d2	Objet 2
0	3	0	1	1	2#b0d2ac	Objet 3
0	3	3	1	1	2#c7acd2	Objet 4
1	0	3	1	2	1#d2d0ac	Objet 5

Figure 2 – Format de fichier d'entrée pour le visualisateur

La première ligne indique trois chiffres, représentant la largeur, hauteur et longueur du conteneur. Ensuite, on vient lister les objets à placer. Chaque ligne représente un objet. Nous avons, à

la suite et pour chaque ligne :

- La position en X de l'objet.
- La position en Y de l'objet.
- La position en Z de l'objet.
- La largeur de l'objet.
- La longueur de l'objet.
- La hauteur de l'objet.
- La couleur de l'objet.
- La label de l'objet.

Seuls les fichiers texte adoptant ce format peuvent être lus par le visualisateur.

Une fois le fichier créé, il faut maintenant aller le donner en entrée de l'outil. Pour cela, nous devons accéder au serveur de KeenSaas par une connexion SSH à distance. La méthode qui a été utilisée lors des essais a consisté en l'utilisation du logiciel FileZilla, permettant de configurer et exécuter ce genre de connexions. Grâce à des identifiants fournis par KeenSaas, il est possible de créer cette connexion SSH, active sur le port 22 250.

Une fois connectés au serveur, il suffit d'aller à l'emplacement du fichier lu par le visualisateur, faire un clic droit, puis cliquer sur "Editer". On remplace le contenu du fichier par le contenu qu'on souhaite visualiser, et on enregistre. Alors, les modifications sont répercutées sur le serveur. Il suffit juste d'actualiser son navigateur pour visualiser les objets relatifs au fichier que nous venons de modifier.

2 Spécifications fonctionnelles

Comme il a déjà été expliqué, le périmètre du projet ne va pas plus loin que la réalisation d'un algorithme permettant de résoudre le problème. Ainsi, cet algorithme représente une grosse fonction à lui tout seul. Aucun cas d'utilisation particulier n'est à citer, l'algorithme prend ses entrées, tourne, et c'est tout.

L'algorithme prendra en entrée la liste d'objets saisis par l'utilisateur, et ressortira un plan de chargement, équivalent à une liste de positions dans le camion associées à tous les objets d'entrée. Chaque objet en entrée est associé à une longueur, largeur et hauteur.

Au sein de l'algorithme, on peut peut-être imaginer quelques fonctions essentielles, telles que :

- `loadData()` : fonction permettant de charger les objets saisis par l'utilisateur dans l'algorithme, et de les transformer en structure de données ou objet décrivant leur dimension. La fonction fait de même pour les dimensions du conteneur. En fonction des objets saisis, d'autres contraintes et caractéristiques seront potentiellement à lier à ces objets, telles que

la gerbabilité ou encore des degrés de libertés spécifiques.

- `process()` : c'est la fonction d'exécution de l'algorithme. En entrée, elle prend ce que la fonction `loadData()` a créé. En sortie, elle renvoie la liste des objets placés dans tel ou tel camion, ce qui correspond au plan de chargement.

Suivant l'implémentation de l'algorithme qu'on choisira, on pourra peut-être retrouver d'autres fonctions. Typiquement, dans les algorithmes de la littérature, on retrouve souvent une fonction type `fillVolume(volumes, objects)`, où `volumes` représente un ensemble de volumes restants à combler, et `objects` est l'ensemble des objets encore non placés. On fait tourner cette fonction tant qu'il reste des objets à placer, où tant qu'il y a des volumes disponibles, en mettant à chaque fois à jour les ensembles. Ceci n'est qu'un exemple, et peut différer totalement selon les choix d'algorithmes que nous ferons.

3 Spécifications non fonctionnelles

3.1 Contraintes de développement et conception

Comme il a déjà été dit, l'algorithme à développer n'est qu'une partie qui s'intègre dans le projet Ks-Fulltruck. Pour développer un algorithme, le langage de programmation n'est pas une barrière, même si certains langages peuvent s'avérer plus performants que d'autres. Néanmoins, KeenSaas souhaiterait que l'algorithme proposé puisse facilement être traduit en Ruby, afin qu'il puisse rapidement être inscrit dans le projet plus global lui-même codé en Ruby. Cependant, des tests ont été effectués en Java, avec l'API Java de LocalSolver. Il a été nécessaire de demander une licence mensuelle, gratuite pour les étudiants. Puis, de télécharger la librairie LocalSolver et de l'intégrer dans un projet Java.

De plus, les étudiants de quatrième année travaillent actuellement sur un algorithme en Python. Pour visualiser leur résultats, ils ont été amenés à utiliser une librairie VPython, qui permet l'affichage d'objets en trois dimensions.

Enfin, comme dit dans la partie interface, l'utilisation du visualisateur de KeenSaas à distance ne peut être faite qu'en utilisant certains logiciels. Typiquement, il faut se connecter en SSH sur leur serveur grâce à un outil qui le permet, comme FileZilla par exemple. Cette connexion s'effectue sur le port 22 250, port qui est filtré par le réseau de l'école. Il faut donc se connecter sur un autre réseau pour avoir un accès garanti, type 4G.

3.2 Contraintes de fonctionnement et d'exploitation

3.2.1 Performances

Il est important de noter que le "Container Loading Problem" est un problème complexe à résoudre. Ainsi, l'algorithme que je vais développer prendra du temps à le résoudre. Suivant l'instance d'entrée, représentative de la complexité du problème, il pourra mettre énormément de temps à le résoudre. Ainsi, nous allons probablement fixer des conditions d'arrêt de l'algorithme.

Typiquement, une première solution serait de fixer une simple limite de temps. Lorsque cette limite est atteinte, l'algorithme nous transmet sa meilleure solution, s'il en a une. Une seconde solution serait de dire à l'algorithme de continuer à faire des calculs tant qu'un certain pourcentage du volume du camion est encore libre. La meilleure solution serait en fait de mixer les deux, pour assurer à l'utilisateur d'avoir une solution convenable à chaque utilisation.

3.2.2 Capacités et contrôlabilité

En termes de données, aucune limite n'est imposée. L'algorithme doit être capable de prendre n'importe quel contenant, parallélépipédique dans un premier temps, de n'importe quelles dimensions, et des objets, également parallélépipédiques, de n'importe quelles dimensions, et de produire une solution optimale ou non.

Cependant, on peut quand même noter une spécificité. Nous aimerions que l'algorithme soit en mesure de proposer plusieurs solutions. En effet, il est possible qu'une des solution proposée par l'outil soit bonne mais ne convienne pas, pour une certaine raison, au chargeur. Alors, il pourrait demander à l'algorithme de réitérer, et donc de sortir une solution avec des positionnements d'objets différents.

Ou alors, nous demanderons directement à l'algorithme de sauvegarder différentes solutions, chacune ayant une granularité de difficulté de résolution différente, afin de toujours proposer des solutions aux chargeurs.

B

Gestion de projet

Dans cette annexe, j'aimerais apporter quelques précisions sur l'aspect gestion de projet.

1 Phase de recherche

1.1 Gantt et plan de développement

D'abord, nous allons détailler chaque tâche apparente sur le diagramme de Gantt que vous pouvez trouver à la fin de l'annexe.

- **Tâche 1 : Recherche**

Cela représente la première phase du projet dans sa globalité. Elle englobe toutes les tâches relatives à la recherche, et c'est pour cela qu'elle apparaît en rouge sur le diagramme.

- **Tâche 2 : Développement**

Cela représente la seconde phase du projet dans sa globalité. Elle englobe toutes les tâches relatives au développement, et c'est pour cela qu'elle apparaît en rouge sur le diagramme.

- **Tâche 3 : Etude et compréhension de la littérature**

Tâche assez importante, qui a pris un temps relativement conséquent. Il était important de bien étudier la littérature pour savoir comment se situe notre problème, et quelles solutions existent déjà. Cette tâche comprend la lecture, prise de notes et compréhension d'une petite dizaine d'articles scientifiques. Même si cette tâche a une durée exprimée sur le diagramme, il a fallu constamment revenir lire les articles, notamment lors de la rédaction de l'état de l'art.

- **Tâche 4 : Premier RDV avec KeenSaas à Polytech**

Cette tâche représente un évènement, elle est colorée en vert sur le diagramme. Ce fut la première rencontre avec l'entreprise KeenSaas, qui s'était déplacée à Polytech pour nous présenter le projet plus en détails. La réunion a duré entre une heure et une heure et

dem. L'entreprise nous a rapidement présenté le concept, et nous avons enchaîné par une série de questions / réponses, notamment pour demander plus de précisions sur certaines contraintes.

— **Tâche 5 : Rédaction du cahier de spécifications**

L'une des tâches les plus conséquentes de la phase de recherche. L'un des livrables attendus était le cahier de spécifications relatif au projet, et cette tâche représente son temps de rédaction. Même si on exprime une durée sur le Gantt, la rédaction du cahier s'est faite tout le long du semestre, au fur et à mesure que nous en apprenions d'avantage sur le projet. De plus, je pense qu'un cahier de spécifications à mi-stade n'est pas complet, et qu'il sera encore à enrichir dans la seconde partie.

— **Tâche 6 : Implémentation d'un algorithme Java permettant d'écrire des instances de données dans un fichier texte**

Suite aux premières semaines, M. Billaut a demandé de créer des instances de données fixes, que nous utiliserons sur l'ensemble des essais d'algorithmes que nous entreprendrons. Cela permet de comparer, par exemple, les performances des algorithmes. Cette tâche a donc consisté en la création de ces instances, grâce à un petit programme Java qui automatise le processus.

— **Tâche 7 : Rédaction de l'état de l'art**

L'état de l'art est le second livrable important de la phase de recherche. Il consiste à présenter les différentes implémentations existantes d'algorithmes permettant de résoudre notre problème. Ce fut l'une des tâches les plus conséquentes du premier semestre.

— **Tâche 8 : Mise en place de l'accès au visualisateur de KeenSaas**

Environ au milieu de la phase de recherche, et après les premiers essais effectués, j'avais grand besoin de visualiser mes résultats. KeenSaas a donc mis en place un accès à distance à leur visualisateur 3D, déjà existant. La configuration a pris un peu de temps, et beaucoup d'échanges par mail, car nous avons dû résoudre le problème du blocage de port par l'école.

— **Tâche 9 : Implémentation d'un modèle mathématique avec LocalSolver**

Suite à l'étude des articles scientifiques, j'ai tenté de mettre en place le modèle mathématique simpliste résolvant un problème de Bin Packing 3D décrit dans l'article [6]. L'acquisition de la licence LocalSolver, et la rédaction ont pris un certain temps.

— **Tâche 10 : Rédaction du rapport**

C'est sans doute la tâche la plus conséquente de la phase de recherche. La fusion du cahier de spécifications et de l'état de l'art, afin de créer un document rapportant toute la démarche et les recherches du premier semestre. C'est le livrable principal de la première phase, qui a pris une grande partie du temps total.

— **Tâche 11 : Première soutenance**

Cette tâche est un événement représenté en vert sur le diagramme. C'est la première

soutenance du projet, censée rapporter l'avancement et les spécifications du projet, en Décembre 2017.

— **Tâche 12 : Second RDV avec KeenSaas à Polytech**

A la fin de l'année 2017, un second rendez-vous avec KeenSaas est prévu. C'est un événement représenté en vert sur le diagramme. Cette réunion aura pour but de présenter l'algorithme créé par les deux étudiants de quatrième année. Dans un second temps, elle sera un point de transition entre la première et la seconde partie du projet. Nous allons potentiellement décider des méthodes de développement que nous allons mettre en place.

— **Tâche 13 : Développement d'un algorithme simple gérant le Bin Packing 3D**

Cette tâche est une des tâches principales de la phase de développement. Avant de pouvoir gérer le problème dans son intégralité, il est important de créer un algorithme permettant de gérer des cas de Bin Packing 3D simples. Il est possible que nous réalisons plusieurs algorithmes, impliquant plusieurs méthodes de résolution différentes. Ce ou ces algorithmes doivent être une solide base sur laquelle nous pourrions ajouter des contraintes au fur et à mesure.

— **Tâche 14 : Tests de l'algorithme simple avec des instances de données réelles**

Après la réalisation de ce ou ces algorithmes simples, nous devrions effectuer des tests. Cette tâche devrait être assez rapide, et sera faite sur des jeux de données réelles, sans doutes fournis par KeenSaas.

— **Tâche 15 : Ajout progressif des contraintes imposées par KeenSaas**

La seconde tâche conséquente de la phase de développement. Après avoir développé un bon algorithme de base, il faudra progressivement ajouter les contraintes du projet. L'idée est de réaliser cet ajout au fur et à mesure, en testant l'algorithme entre chaque contrainte ajoutée. A chaque ajout, il faudra aussi vérifier que les nouvelles prises en comptes de contraintes ne sont pas en conflit avec d'autres contraintes.

— **Tâche 16 : Rédaction du rapport final**

A la fin du projet et donc de la seconde phase, il faudra rédiger le rapport final. Il reprendra le premier rapport, et complètera les parties manquantes, notamment celle sur la mise en œuvre. Ce sera aussi un tâche qui prendra du temps.

— **Tâche 17 : Tests de l'algorithme avec des instances de données réelles**

Pendant la phase d'ajout des contraintes à l'algorithme, il faudra tester à chaque fois sur des instances de données réelles, encore une fois probablement fournies par KeenSaas, pour garantir sa validité.

— **Tâche 18 : Seconde soutenance**

Cette dernière tâche est un événement, représenté en vert sur le diagramme. C'est la soutenance finale du projet, présentant l'avancement final, ainsi que les implémentations effectuées, qui se déroulera sans doutes aux alentours de début Avril 2018.

Voici le diagramme de Gantt :

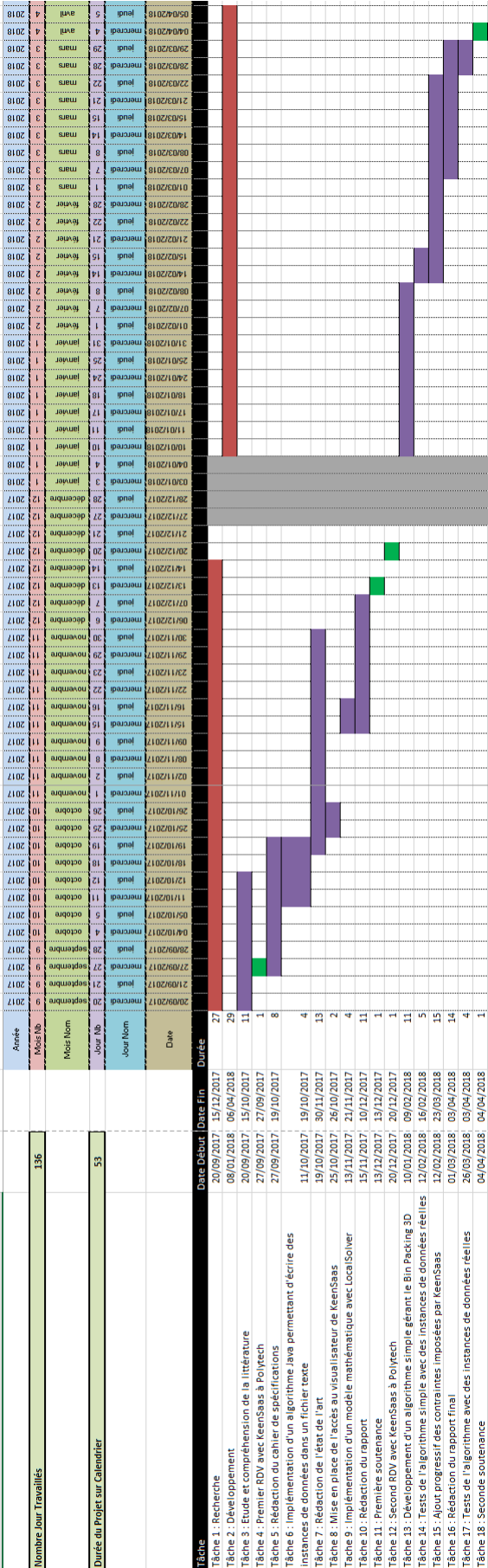


Figure 1 – Diagramme de Gantt

1.2 Cycles et outils

Faisons un petit point sur les outils de gestion de projets utilisés. Tout d'abord, il est difficile d'établir un cycle de développement pour la phase de recherche. Cependant, le but était de rendre compte de mon travail toutes les deux semaines auprès de M. Billaut, et de manière un peu plus éparse aux professionnels. Ces comptes-rendus étaient souvent sous formes de mail, et ils seront inclus dans ce rapport. Il y en a moins que de couples de semaines sur toute la période, mais durant les semaines de rédaction pures, je n'ai pas trouvé d'utilité à faire de compte-rendus détaillés, typiquement à partir du mois de Novembre.

Cependant, j'ai également créé des comptes-rendus par semaines, que j'ai gardés pour moi. En voici un aperçu :

Semaine	04-05/10/2017
Essayer de faire	Continuer cahier de spécif Continuer rapport Voir M.Billaut pour discuter des différents algorithmes de la littérature. Voir M.Ragot pour voir si le cahier de spécif ne peut pas être adapté
Fait	Cahier de spécif Révision des algorithmes pour les présenter demain Use case diagram Rdv M.Billaut : discussion des approches de remplissages, explication des différents types d'algorithmes (borne inférieure, heuristique, exact, solveur, prog contrainte) On a décrété que j'essaierai de résoudre avec un algorithme tabou, et avec de la PPC avec localsolver. Il faut donc commencer à réfléchir au modèle mathématique, variables, contraintes, fonction obj. Et commencer à l'implémenter sur un bin packing très simple. Aussi, faire une génération de données dans des fichiers txt. Il faut reprendre la license localsolver!
Prévu pour la semaine prochaine	License LocalSolver Génération données Commencer à rédiger le modèle mathématique Continuer cahier de spécif

Figure 2 – CRAH : Compte-rendu d'activité hebdomadaire

J'ai présenté sous forme de tableau, que je dupliquais chaque semaine. Chaque semaine, je prenais le contenu "Prévu pour la semaine prochaine" de la semaine précédente, et je le mettais dans la partie "Essayer de faire". Je décrivais en détails ce que j'avais fait durant les deux jours de la semaine dans la partie "Fait". Ce document me permet de me rappeler des tâches restantes à accomplir, et est un bon remplacement d'un outil type GitLab, pas très adapté pour une phase de recherche qui comprend surtout de la rédaction.

Concernant la phase de développement maintenant, je pense établir des cycles de développement. En effet, j'aimerais proposer à l'entreprise un fonctionnement orienté agile. Typiquement, un rapport toutes les deux semaines, ainsi que, pourquoi pas, une démonstration à M.Billaut. Une démonstration toutes les deux semaines aux professionnels semble compliqué, étant donné

la distance qui nous sépare. Ainsi, j'aimerais découper la phase de développement en sprints de deux semaines. A la fin de chaque sprint, un compte-rendu est fait, et si on trouve un désaccord ou des choses à revoir, les modifications sont apportées. Cette méthode permettra de garantir le fait que le code proposé soit constamment en accord avec les attentes de KeenSaas, et de prévenir toute potentielle déviation qui nous écarterait des objectifs.

Je pense qu'une gestion de projet agile est beaucoup plus sûre qu'un cycle classique, type cycle en V.

Pour terminer, je mettrai en place un Gitlab pour la partie développement. Comme déjà décrit dans la partie Bases Méthodologiques de ce rapport, cet outil permet à la fois de versionner, et de faire de la gestion de projet : liste de tâches, liste des développeurs, problèmes, backlog, etc. Pour l'avoir déjà utilisé sur de gros projets de développement, je pense pouvoir dire que Gitlab est un très bon outil de versionning, et je souhaite l'utiliser pour ce projet.

2 Phase de développement

2.1 Ecart des plannings réel et prévisionnel

Jetons maintenant un oeil à l'écart des plannings réel et prévisionnel. Le second a été réalisé en fin de premier semestre, pour essayer de déterminer les durées que les différentes tâches me prendraient. Le premier a été fait à la fin de la phase de développement, avec les durées que les tâches m'ont effectivement prises. La figure suivant présente les deux diagrammes de Gantt, le plus haut étant le prévisionnel et le plus bas le réel.

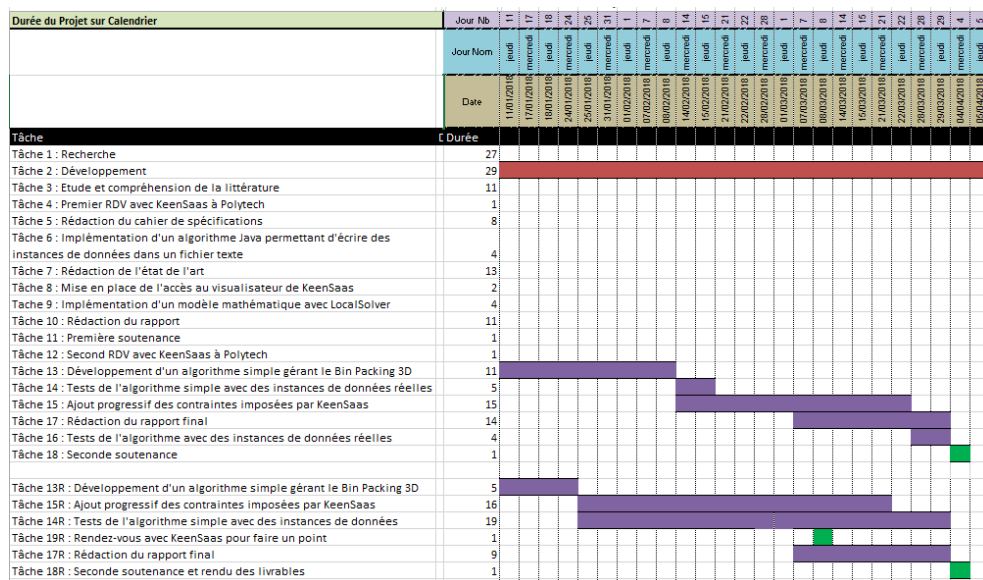


Figure 3 – Ecart des plannings

Quelques différences sont clairement notables. D'abord, j'avais estimé la première tâche consistant à développer la base d'un algorithme de bin packing 3D, pouvant placer des objets dans un repère 3D, avec une durée un peu trop longue. J'ai mis deux fois moins de temps à l'effectuer. Ensuite, on pouvait noter sur le planning prévisionnel que j'avais prévu deux phases de tests différentes. L'une venant après le développement de l'algorithme de base, et l'autre venant à la fin de la phase de développement approfondi (prise en compte progressive des contraintes). Cependant, il s'avère que la phase de tests a été continue pour moi. Je ne pouvais pas continuer

à développer sans savoir si ce que je venais de coder fonctionnait correctement. C'est pour cela qu'on peut voir sur le planning réel une seule et même tâche de test, qui aura finalement pris presque toute la durée du semestre.

On remarque aussi que la seconde tâche de développement des contraintes a été plus longue que prévue. En effet, la première tâche de développement ayant été terminée plus tôt que prévue, j'ai pu commencer le développement approfondi plus tôt. En ce qui concerne la rédaction du rapport, l'estimation de durée et des dates a été en accord avec ce qui s'est réellement passé. Enfin, on note également la présence d'un rendez-vous(en vert) supplémentaire, à peu près un mois avec la date de la soutenance. Ce rendez-vous n'était pas anticipé, et a servi à faire le point avec l'entreprise.

2.2 Gitlab

Toute la gestion de projet ainsi que le versionning ont été réalisés sur Gitlab. L'aspect versionning n'est plus à décrire. Cependant, Gitlab propose des outils très pratiques pour bien découper son projet en tâches, et suivre leur avancement. J'aime beaucoup utiliser Gitlab pour sa fonctionnalité des issues. Une issue est en quelques sortes une tâche. Nous pouvons en créer, et les faire naviguer entre différents tableaux, comme des tableaux "A faire", "Fait", etc. Chaque issue est associée à un ou plusieurs labels, ces labels pouvant être par exemple "Fonctionnalité", "Bug", "Documentation", et d'autres. Une issue a une description, un nom, des collaborateurs associés, et aussi pourquoi une date butoir.



Figure 4 – Gitlab

Je me suis servi de cette fonctionnalité de manière journalière. Dès que j'avais une nouvelle tâche à accomplir, je la créais sur mon Gitlab. Chaque issue possède une zone de commentaires. Je me servais de cette zone pour commenter mes issues, où je décrivais tout mon processus de réflexion pour la réalisation de cette tâche. Ainsi, pour chaque fonctionnalité, j'ai à plat toute ma réflexion, les idées et les problèmes que j'ai eu.

Egalement, il est possible de lier des commits à des issues. Ainsi, dès que je travaillais sur une issue en particulier, je la liais dans mes commits associés (il faut spécifier l’ID de l’issue qui est copiable sur Gitlab). Puis, si on va voir le descriptif de l’issue dans Gitlab, nous avons un historique de tous les commits qui sont associés à cette issue, ainsi que leur descriptif et leur date. Ce détail est très pratique pour garder un historique de travail complet sur chaque fonctionnalité. Une fois une issue considérée terminée, on la ferme. Voici un exemple d’une issue sur mon Gitlab :

The screenshot displays a GitLab issue page. At the top, the issue is marked as 'Closed' and was opened a month ago by Jean Cellier. It includes buttons for 'Reopen issue' and 'New issue'. The title is 'Faire varier les solutions'. The description states: 'On veut pouvoir troubler l'ordre des objets de base de manière à générer plusieurs solutions, et choisir la meilleure en s'appuyant sur les indicateurs de qualité de solutions. Ce "perturbement" s'appuiera sur une méthode de swap.' Below the description are reaction buttons (thumbs up, thumbs down, smiley) and a 'New branch unavailable' warning. The comment history shows two comments by Jean Cellier, both mentioning commit hashes (8bf07840 and 8cd56ef7). At the bottom is a 'Write' section for adding a new comment. On the right, a sidebar contains settings for the issue: Assignee (None), Milestone (None), Time tracking (No estimate or time spent), Due date (None), Labels (enhancement), Weight (None), Confidentiality (Not confidential), Lock issue (Unlocked), 1 participant, Notifications (checked), and Reference (JeanCellier/PRD#11).

Figure 5 – Une issue sur Gitlab



Comptes rendus hebdomadaires

Compte rendu n°1 du 20/09/2017

Bonsoir Monsieur,

Voici un résumé des tâches effectuées durant les deux premiers créneaux de PRD :

- Lecture des 5 papiers scientifiques fournis. Pour chacun, j'ai essayé de déterminer quel genre d'algorithme était utilisé, et quels étaient les besoins. Egalement, quels étaient les besoins non pris en compte par rapport à notre problème, pour essayer de voir quelles techniques nous pourrions être amenés à réutiliser.
- Catégorisation des différents types de loading problem.
- Exercices sur le bin packing et knapsack.
- Vidéos sur différents : first fit, first fit decreasing, full bin packing.
- Classification des techniques de résolution du problème : algorithmes et modèles.(block building approach, wall building approach)
- Réunion avec KeenSaas : émission plus claire des besoins, transmission de coordonnées.
- Tutoriels sur le Ruby, et téléchargement d'un IDE Ruby.(JetBrains)
- Rédaction du début du cahier de spécifications : le contexte, les besoins, le problème et son histoire dans la littérature. Schéma représentant le périmètre

du projet, et son intégration dans le projet plus global de KeenSaas.

Je rencontre quelques difficultés à remplir certaines parties du modèle de cahier de spécifications fourni sur Celène. Mon projet consistant uniquement en l'élaboration d'un algorithme, il y a certaines parties qui n'ont, selon moi, pas lieu d'être. J'irai voir M. Ragot pour plus de précisions à ce sujet.

Cordialement,

Jean Cellier

Compte rendu n°2 du 04/10/2017

Bonjour,

Je vous transmet l'avancement des deux dernières semaines.

- Suite à notre RDV de la semaine dernière, j'ai généré les données et les ai transmises aux étudiants de 4A. J'ai conservé le programme de génération au cas ou.
- Rédaction du cahier de spécif, notamment diagrammes de CU et schéma général du système (intégration de l'algorithme dans le projet KSFullTruck.
- J'ai commencé à rédiger un modèle mathématique avec un cas simple et des contraintes limitées. (Données, variables, contraintes)
- Après avoir renouvelé ma licence LocalSolver, j'ai implémenté le modèle en utilisant l'API Java de LocalSolver. Le solveur tourne donc, et me sort des résultats.

Je me suis arrêté là, car j'ai rencontré une difficulté : le solveur me donne des positions (x,y,z) pour chacun des objets. En m'appuyant sur les dimensions, j'aimerais pouvoir vérifier que toutes les contraintes sont satisfaites en dessinant les objets, ce qui est vite fastidieux en 3D. Je n'ai donc pas encore réussi à vérifier efficacement les solutions données par le solveur.

Avez-vous des retours de KeenSaas concernant l'accès à leur outil de visualisation des solutions ? Cela pourrait vraiment m'aider pour tester mes résultats.

Egalement, je pense que nous devrions peut-être discuter du modèle et des contraintes. Je ne suis pas totalement sûr de ce que j'ai fait et ai probablement omis des contraintes importantes.

Cordialement,

Jean Cellier

Compte rendu n°3 du 18/10/2017

Bonjour Monsieur,

Je vous transmet un résumé de mon activité sur les deux dernières semaines en PRD, un peu en retard et je m'en excuse.

- finalisation de la V1 du cahier de spécifications.
- début de rédaction de l'état de l'art.
- après notre dernière réunion, j'ai implémenté le modèle du pdf "A linear programming approach..." sur LocalSolver. Les résultats sont beaucoup plus satisfaisants. Cependant, une contrainte majeure est manquante dans leur modèle : le fait que les objets doivent avoir en dessous d'eux soit un autre objet, soit le sol du camion. Ainsi, deux cas se présentent. Le premier, lorsqu'on propose un volume de camion très grand, les objets sont tous bien séparés, mais il en placera en l'air sans appui. Le second, lorsqu'on impose un volume un peu limite en hauteur, où il négligera alors la contrainte de chevauchement pour réussir à caser tous les objets, mais certains se chevauchent du coup. Il faut donc réussir à exprimer une contrainte de "gravité" avec les données/contraintes que j'ai à ma disposition.
- j'ai obtenu l'accès au visualisateur de KeenSaas. J'ai donc ajouté une URL dans mon fichier système hosts pour accéder à l'URL sur un navigateur. La partie un peu plus délicate, est celle où je dois me connecter en SSH sur leur serveur pour aller remplacer le fichier lu par le visualisateur. Non pas que c'est compliqué, mais le port utilisé (22250) est filtré par le réseau de l'école...je dois donc utiliser, pour l'instant, mon téléphone et ma 4G pour accéder à leur serveur. Le service info ne veut pas me configurer une exception car je travaille sur ma machine perso.
- la journée du 25, j'ai eu un entretien avec LocalSolver. Je n'ai donc pas travaillé la matinée, et des réunions sur la rédaction du cahier de spécif ont pris place l'après midi jusqu'à 16h. L'entretien s'est bien passé, et je passe un second entretien, plus technique, le Mercredi 08 Novembre dans la matinée.

Encore désolé pour le retard, et bonne journée.

Cordialement,

Jean Cellier

Bibliographie

- [1] MT ALONSO, R ALVAREZ-VALDES, F PARRENO et JM TAMARIT. « Pallet building and truck loading strategies for an inter-depot transportation problem ». In : ().
- [2] Andreas BORTFELDT, Gerhard WÄSCHER et al. *Container loading problems : A state-of-the-art review*. Univ., Faculty of Economics et Management, 2012.
- [3] Tobias FANSLAU et Andreas BORTFELDT. « A tree search algorithm for solving the container loading problem ». In : *INFORMS Journal on Computing* 22.2 (2010), p. 222–235.
- [4] Guenther FUELLERER, Karl F DOERNER, Richard F HARTL et Manuel IORI. « Metaheuristics for vehicle routing problems with three-dimensional loading constraints ». In : *European Journal of Operational Research* 201.3 (2010), p. 751–759.
- [5] Michel GENDREAU, Manuel IORI, Gilbert LAPORTE et Silvano MARTELLO. « A tabu search algorithm for a routing and container loading problem ». In : *Transportation Science* 40.3 (2006), p. 342–350.
- [6] Mhand HIFI, Imed KACEM, Stéphane NÈGRE et Lei WU. « A linear programming approach for the three-dimensional bin-packing problem ». In : *Electronic Notes in Discrete Mathematics* 36 (2010), p. 993–1000.
- [7] Manuel IORI et Silvano MARTELLO. « Routing problems with loading constraints ». In : *Top* 18.1 (2010), p. 4–27.
- [8] D Alvarez MARTINEZ, R ALVAREZ-VALDES et F PARREÑO. « A grasp algorithm for the container loading problem with multi-drop constraints ». In : *Pesquisa Operacional* 35.1 (2015), p. 1–24.
- [9] David PISINGER. « Heuristics for the container loading problem ». In : *European journal of operational research* 141.2 (2002), p. 382–392.
- [10] Pierre SCHAUS et al. « Solving balancing and bin-packing problems with constraint programming ». In : *These de doctorat, Université catholique de Louvain* (2009).

Optimisation du chargement d'un camion

Jean Cellier

Encadrement : Jean-Charles Billaut



En collaboration avec KeenSaaS

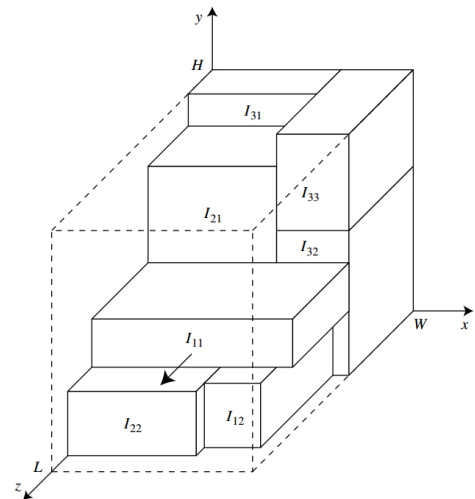
Le "Container Loading Problem"

On souhaite placer un ensemble d'éléments rectangulaires dans des camions de dimensions rectangulaires. La résolution de ce problème consiste à trouver une solution plaçant l'ensemble des objets de la manière la plus optimisée possible, en minimisant l'espace résiduel, et donc en minimisant le nombre de camions nécessaires.



Les objectifs

- Proposer des plans de chargement optimisés.
- Obtenir un chargement permettant de réduire au maximum le nombre de camions utilisés.
- Respecter les contraintes imposées : répartition de la charge, priorités de livraison, empilement, etc.



Résultats attendus

Au final, on souhaite développer un algorithme performant, proposant une ou plusieurs solutions au problème. Ces solutions sont visualisables par les logisticiens responsables de charger les camions de livraison, en 3D sur un écran.



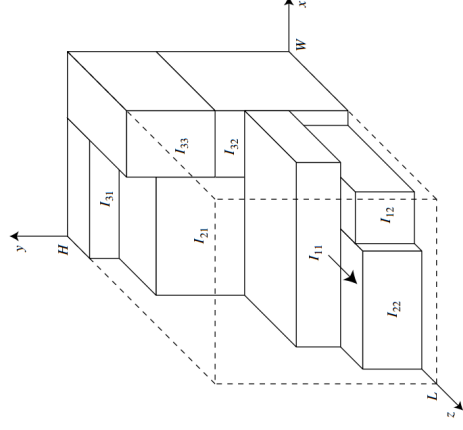
Optimisation du chargement d'un camion

Jean Cellier

Encadrement : Jean-Charles Billaut

Le "Container Loading Problem"

On souhaite placer un ensemble d'éléments rectangulaires dans des camions de dimensions rectangulaires. La résolution de ce problème consiste à trouver une solution plaçant l'ensemble des objets de la manière la plus optimisée possible, en minimisant l'espace résiduel, et donc en minimisant le nombre de camions nécessaires.



- Proposer des plans de chargement optimisés.
- Obtenir un chargement permettant de réduire au maximum le nombre de camions utilisés.
- Respecter les contraintes imposées : répartition de la charge, priorités de livraison, empilement, etc.

Résultats attendus

Au final, on souhaite développer un algorithme performant, proposant une ou plusieurs solutions au problème. Ces solutions sont visualisables par les logisticiens responsables de charger les camions de livraison, en 3D sur un écran.



En collaboration avec KeenSaas



ECOLE POLYTECHNIQUE DE L'UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH
TOURS

Informatique

Optimisation du chargement d'un camion

Résumé

Le problème du Bin Packing en trois dimensions est un problème très courant dans le domaine de l'optimisation. Plus spécifiquement, l'optimisation de chargement d'un conteneurs est une problématique récurrente de la littérature, et, selon le contexte, peut être extrêmement difficile. C'est ce que souhaite aborder KeenSaas, société basée à Orléans, en proposant un outil permettant le calcul et la visualisation de plans de chargements de camions. Un Projet de Recherche et Développement a donc été initié, avec pour but de développer l'algorithme qui optimise le chargement des objets dans les camions, sous un ensemble de contraintes très complexes, telles que la répartition de la charge, ou encore la prise en compte d'un ordre de livraison. Ce document présente toute la démarche de travail appliquée lors de ce projet, de la phase de recherche jusqu'à la description technique de la réalisation effectuée. L'outil souhaité par KeenSaas doit prendre en compte beaucoup de contraintes différentes, ce qui en a fait un projet proposant de réels challenges algorithmiques.

Mots-clés

Recherche opérationnelle, Algorithmique, Bin Packing 3D

Abstract

The three dimensional Bin Packing problem is very common in the optimization field. More specifically, the container loading optimization is a recurrent issue in the literature, and, given the context, can be extremely difficult. That is what KeenSaas, a company based in Orléans, wishes to try out, by developing a tool capable of computing and displaying trucks loading plans. Thus, a Research and Development Project has been launched, aiming to develop the algorithm optimizing the objects loading into the trucks, by respecting a set of very complex constraints, such as the weight distribution or the presence of a delivery arrangement. This document expresses the whole working process applied during this project, from the research phase to the technical description of the adopted solution. The tool KeenSaas wishes to build must take in account lots of different constraints, which made it a project with real algorithmic challenges.

Keywords

Operational Research, Algorithmic, 3D Bin Packing

Entreprise

KeenSaas



KeenSaaS

Tuteurs entreprise

Frédéric PERRIN

Bruno BOUDINSKI

Étudiant

Jean CELLIER (DI5)

Tuteur académique

Jean-Charles BILLAUT