école supérieure d'informatique
67 Rue Royale
B-1000 Bruxelles
Tél. +32 (0)22191546
Fax +32 (0)22194847
Email : esi@heb.be

Stage effectué dans le cadre d'un échange

ERASMUS

Intégration d'un solveur de problèmes mathématiques dans un logiciel d'aide à la décision

Rapport de stage effectué dans la firme

PSI Metals

PSI

DUT Informatique de gestion à l'IUT A de Lille 1

Janvier 2015

Mustapha Nezzari

PÔLE UNIVERSITAIRE EUROPÉEN DE BRUXELLES WALLONIE

Remerciements

Je tiens à remercier Monsieur Luc Van Nerom pour m'avoir donné l'opportunité d'effectuer mon stage au sein de PSI Metals et de m'avoir mis en confiance dès mon premier jour de stage.

Je tiens aussi et surtout à remercier mon maître de stage Monsieur Vivian De Smedt, sans qui ce stage n'aurait pas été aussi enrichissant tant sur le côté technique que le côté humain. La patience qu'il m'a accordé et son encadrement tout au long du stage m'a permis d'adopter les bonnes méthodes de travail ce qui m'a rendu autonome sur des outils que je ne connaissais pas avant d'arriver. Merci de m'avoir fait confiance tout au long cette expérience valorisante et intéressante.

Cet aide précieuse a bien sûre été confortée grâce à Julie, Wafa, Nicolas et Simone qui m'ont accordé de leur temps pour m'aider à comprendre les outils dont j'avais besoin pour avancer.

Je remercie également Madame De Mesmaeker pour avoir préparé au mieux mon accueil à l'ESI ainsi que Monsieur Nabet et Monsieur Vansteenkiste pour leurs précieux conseils lors de la rédaction de ce rapport. Je ne saurais oublier mon responsable de stage Monsieur Lebègue et tout le corps enseignant qui m'a formé tout au long de ces deux années d'étude au sein de l'IUT A de Lille 1.

Enfin je tiens à adresser mes remerciements à toute l'équipe PSI Metals pour avoir maintenu cette ambiance chaleureuse dont j'ai eu la chance de faire partie durant ces trois mois.

Sommaire

Introduction

À la fin de mon DUT Informatique de gestion à l'IUT A de Lille 1, j'ai choisi d'effectuer un stage Erasmus à Bruxelles suite à une première expérience motivante à Menin. Mon but était d'acquérir de l'autonomie tout en découvrant le monde professionnel à l'étranger.

Mon choix s'est tourné vers l'entreprise PSI Metals, une entreprise spécialisée dans le management de la production de métal. Cela est justifié par le fait que je voulais absolument faire mon stage dans une entreprise qui développe des logiciels dans un langage que je ne connaissais pas et comme le monde du web ne m'est pas inconnu, j'ai préféré m'orienter vers les entreprises créatrice de progiciels. PSI Metals correspondait parfaitement à mon profil car elle me proposait un stage où les langages utilisés serait le C++ et le Python.

La mission que mon tuteur m'a confiée était d'intégrer un nouveau solveur de problèmes mathématiques dans une solution d'aide à la décision appelée MPlanner. Ce solveur permet de générer un plan optimal qui permet d'organiser la production de métal par semaine. Grâce à cette organisation appelée planification, les entreprises de production de métal peuvent répondre aux commandes de leurs clients dans les meilleurs délais tout en prenant en compte les contraintes de machines et de coûts.

Je présenterai donc dans ce rapport l'entreprise PSI Metals, pour ensuite expliquer à quelles problématiques MPlanner répond et quelles améliorations j'ai apporté sur ce dernier. Je terminerai enfin sur les apports personnels que ce stage Erasmus m'a offerts.

Abstract

In order to complete my informatics studies in Lille, I've chosen to realize my three-month long work placement in Belgium. This choice was done thanks to a previous TV recycling job I had in Menin and which motivated me to retry the experience within the framework of an Erasmus program.

Instead of searching a work placement in Web development like the majority of my friends, I preferred to look for a management company which could give me the knowledge I didn't had yet.

PSI Metals is a provider for software solutions in production management for the metal industry. As I didn't know anything about production management and the programming languages used to develop these software, I immediately accepted PSI Metals's work placement offer.

The mission that was given to me was to integrate a new mathematical problem solver in one of these software in order to replace an old and slow solver by a new one and test its performances. So I was able to use very helpful tools like Visual studio which helped me during my C++ development. Its debugging tool helped me to understand how to implement the solver in the software.

This project made me acquire a lot of autonomy and technique and made me realize how complex the production management is.

1. Présentation de l'entreprise Psi Metals

1.1.Historique

L'entreprise PSI Metals Gmbh a été fondée en 2000 grâce à l'association entre le Steel Institute VDEh et le groupe PSI AG, qui étaient jusqu'en 1999, les principaux fournisseurs de logiciels pour l'industrie Allemande de métal. Devenue alors une société à responsabilité limitée en 2005, PSI Metals est aujourd'hui une entreprise internationale basée en Allemagne et répartie dans le monde entier. En effet elle possède plusieurs entreprises dans le monde notamment en Russie, en Chine, en Belgique ou encore à Rio de Janeiro.

Figure 1 Répartition de PSI Metals dans le monde

Le groupe PSI AG compte environ 1600 employés dans le monde, tandis que la branche PSI Metals en compte environ 350.

1.2.PSI Metals Bruxelles

L'entreprise PSI Metals, anciennement nommée AIS, est une PME (Petite et Moyenne Entreprise) située à Bruxelles. Elle compte 25 employés.

Figure 2 Logo d'AIS

AIS est une entreprise fondée en 1986. En 2000 elle est devenue une filiale autonome du groupe autrichien VATECH dont la division VAI (Voest Alpine Industrieanlagenbau) est un leader mondial en ingénierie et fourniture d'équipement pour la production de métal.

L'entreprise AIS fut rachetée par le groupe PSI AG en 2009 et est maintenant connu sous le nom de PSI Metals Bruxelles.

1.3. Domaine d'activité

Le groupe PSI AG fournit différents services pour aider les entreprises à gérer leur énergie, leur production et leur infrastructure. Son principal atout est qu'elle reste très proche de ses clients ce qui lui permet de développer des logiciels répondant exactement à leurs attentes. Elle s'occupe aussi d'intégrer ces solutions complètes au sein de l'entreprise concernée. Ayant des clients dans le monde entier, elle n'hésite pas à se déplacer chez ses clients afin de les conseiller sur place.

PSI Metals est spécialisée dans la création de solution visant la planification de la production de métal de ses clients. Autrement dit ces solutions permettent d'optimiser la production de métal et ainsi de répondre au carnet des commandes dans les délais tout en prenant en compte les ressources disponibles et en limitant les coûts de production, de maintenance et de stockage. Nous reparlerons de la planification dans le prochain chapitre.

1.4. Accessibilité

PSI Metals Bruxelles est située à Anderlecht à 10 minutes de l'arrêt de métro Erasme.

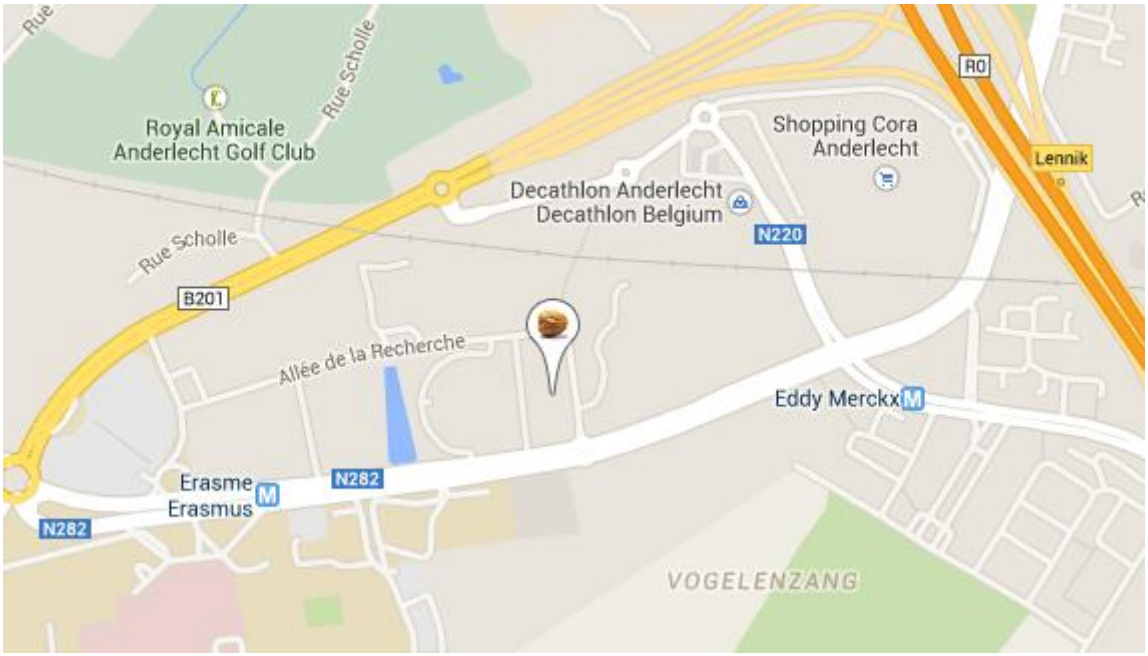


Figure 3 Plan d'accès à PSI Metals Bruxelles

2. Planification de la production de métal

Comme nous l'avons dit précédemment, PSI Metals offre un panel de logiciels afin de faciliter la planification de la production de métal. Pour être conscient de l'importance de planifier cette production, nous allons décrire en quoi elle consiste et à quelles problématiques la planification répond.

2.1.Chaine de production

Prenons l'exemple d'une entreprise classique qui produit du métal. Au début de la chaine de production nous avons la matière première qui est l'acier liquide. Cet acier liquide va passer à travers plusieurs machines et va donc subir plusieurs transformations. Chaque machine correspond à une ligne de produit, c'est-à-dire qu'après traitement du métal par la machine, nous avons un produit semi-fini que l'on peut soit vendre, soit laisser sur la chaine de production pour qu'il subisse ainsi la prochaine transformation. Chaque produit a donc une route à suivre et à la fin de celle-ci, le produit est vendu.

Par exemple l'acier liquide peut, sur la ligne suivante qui est le Caster, être transformé en une longue barre de métal appelée brame.

Figure 4 Brames

Cette brame, mesurant 15 mètres de long, 2 mètres de large et 20 centimètres de hauteur peut ensuite passer dans un laminoir à chaud qui va chauffer la brame afin de l'aplatir pour lui donner une hauteur de 8 mm et ainsi la rouler en forme de bobine chaude.

Figure 5 Bobine chaude

Après cela, cette bobine peut être décapée dans un bain d'acide et devenir très brillante. On appelle cela du pickling.

Figure 6 Bobine décapée

Elle peut ensuite repasser dans un laminoir à froid afin de diminuer sa hauteur. On a donc une bobine froide de 0.3 millimètres.

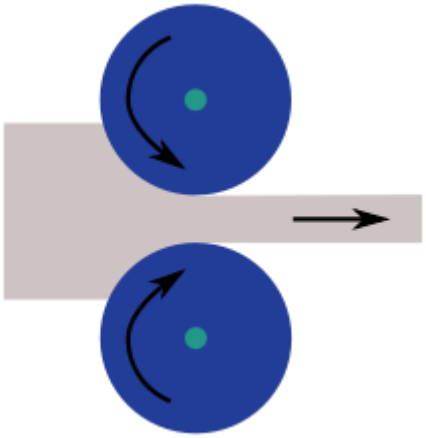


Figure 7 Laminage à froid

Enfin cette bobine peut subir différents traitements selon la route qu'elle empreinte. Par exemple on peut la galvaniser, c'est-à-dire lui ajouter une couche de zinc afin de la protéger contre la corrosion. On peut aussi, au lieu de la galvaniser, en faire une bobine de fil de métal. Cette bobine de fil peut même revenir sur la ligne de galvanisation et ainsi être transformée en une bobine de fil de métal galvanisée.

Cependant par ligne de production, nous ne sommes pas obligés de produire qu'un seul type de produit. Par exemple à l'étape de galvanisation, on peut avoir plusieurs façons de galvaniser. On aurait par exemple un produit galvanisé avec du Galvana et un autre avec du Galvanum. On parle alors de campagne (ou famille) de produit. Un exemple un peu plus parlant serait de dire que je pourrais avoir une ligne ou le métal est peint et que j'aurais une campagne de produit rouge et une autre de produit bleu.

Ce qu'il faut donc retenir c'est que la chaine de production est une succession de lignes de production desquelles peuvent sortir plusieurs familles de produit destinés à la vente.

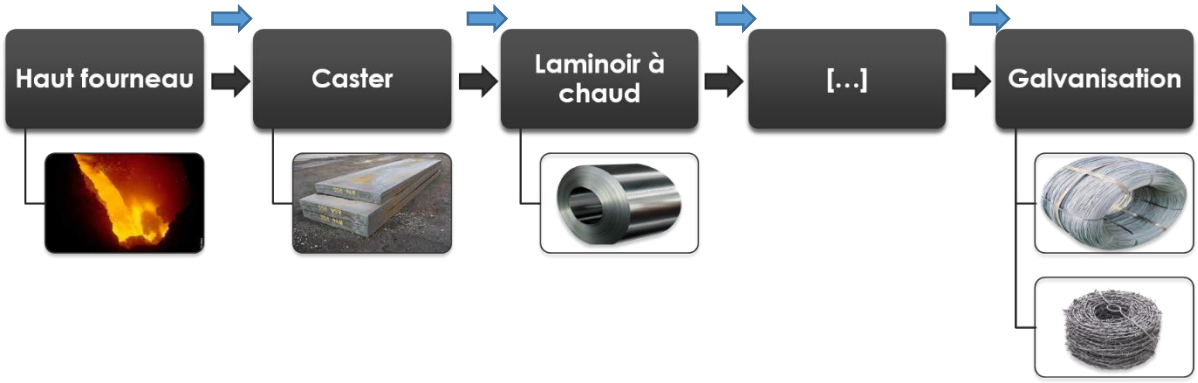


Figure 8 Exemple de chaîne de production

2.2.Problématiques

2.2.1. Carnet de commandes

Comme nous l'avons dit juste avant, nous pouvons avoir plusieurs familles de produits. Ces produits peuvent continuer leur route sur la chaine de production, ou être vendus à des clients. Nous allons donc avoir un carnet de commandes à satisfaire et donc des délais à respecter. Il faut donc que l'on sache à quel moment nous allons traiter chaque commande. Je vais donc établir un calendrier.

2.2.2. Calendrier de décisions

Chaque ligne de la chaine de production correspond à une machine qui fabrique plusieurs familles de produit. Et pour justement avoir plusieurs familles de produit je vais devoir agir sur mes machines pour qu'elles produisent ces familles. Reprenons l'exemple de la brame. Nous pourrions avoir des brames de couleur rouge et bleu. Il y aurait donc une ligne de production qui s'occupe de colorer mes brames. Lorsque je veux passer de la peinture rouge à la peinture bleu, il faut que j'arrête ma machine, que je la nettoie et que je change la couleur pour du bleu. Sauf que cela a un coût en argent et en temps. Je vais donc chercher à satisfaire le maximum de commande de brame rouge pour ensuite satisfaire celle en bleu. Je n'aurai donc besoin de changer de couleur qu'une seule fois. Je vais donc devoir tenir un calendrier de décisions qui contiendra les moments où je change de campagne. Cela me permettra aussi de dire à mon commercial, qui prend les commandes, d'essayer de vendre dans les bons mois la couleur qui convient et donc de ne pas promettre du rouge, lorsque je vais faire une campagne de bleu.

Ce calendrier de décisions va servir aussi à organiser l'arrêt des machines pour la maintenance de celles-ci. Si je demande à un sous-traitant de venir les nettoyer, il faut que je puisse lui dire à l'avance, c'est pour cela que je dois prévoir ce calendrier à l'avance.

Je vais aussi vouloir organiser l'arrêt de certaines lignes car je n'ai pas assez de commandes. Je dois négocier les heures de travail avec les syndicats et je dois pouvoir leurs communiquer les dates d'arrêt des lignes longtemps à l'avance.

Ce calendrier est donc important pour prendre des décisions qui prennent plus ou moins de temps à se mettre en place.

2.2.3. Contraintes des lignes

En plus de devoir prendre des décisions, je dois prendre en compte les contraintes de mes machines. En effet, toutes les machines ont des capacités de production et de stockage différentes. Je vais donc devoir veiller à organiser les tonnages passant par ces machines afin de ne pas faire exploser les stocks. Si j'ai par exemple une machine A et B dans cet ordre, et que A est plus rapide que B, cette dernière va cumuler du stock jusqu'à peut-être dépasser le stock maximum qu'elle peut contenir.

Le but est aussi de réduire les stocks pour ne pas avoir de coût de stockage trop élevés. De plus, le stock correspond à de l'argent dormant et donc à un investissement non rentabilisé.

2.3.Planifier la production

Afin de répondre à ces problématiques, nous allons avoir recours à la planification qui consiste à organiser la capacité des machines dans le temps. C'est ce que l'on appelle planifier la production. C'est dire quelle machine fait quoi et quand.

2.3.1. Pourquoi planifier la production ?

La planification permet de réduire le stress en se projetant dans le futur et ainsi voir si l'on saura respecter les objectifs qui sont ici de respecter les délais de nos commandes et prendre en compte les contraintes de machines et de stockage. Ainsi, on peut prendre immédiatement des décisions qui ont un certain délai de mise en œuvre et qui serviront à atteindre nos objectifs. On va pouvoir organiser l'arrêt des machines longtemps à l'avance pour la maintenance et le changement de campagne afin de satisfaire tout type de commande.

2.3.2. Comment planifier la production ?

Pour planifier la production, on va avoir recourt à un plan. Ce plan est une grille qui va décrire par semaine, les flux de matières passant par chaque ligne pour chaque famille de produit. Bien sur ces flux doivent respecter les contraintes des machines (stock minimum, stock maximum, …). Ce plan va permettre de voir si l'on pourra respecter les délais des commandes selon les flux que l'on y a inscrits.

On peut faire plusieurs plans en jouant sur les flux mais aussi sur les contraintes. Je peux par exemple jouer sur les contraintes dites douces, c'est-à-dire les contraintes qu'il est possible de modifier comme un stock minimum. Je peux ensuite voir les impacts sur les autres contraintes et éventuellement changer de plan.

Un plan peut donc être changé en cours d'application selon si l'on voit que l'on ne va pas respecter les délais et les contraintes. Ceci peut être dû à des imprévus qui ont ralenti la production.

2.3.3. Le Scheduling

La planification permet de faire des plan afin d'avoir une vision à moyen terme (Semaine). Il est aussi possible de faire un plan à plus court terme en déterminant par heures, le nombre de produits fabriqués. Cependant ce type de plan ne sera pas détaillé dans ce rapport car mon travail ne s'arrête qu'à la planification.

2.3.4. Outil d'aide à la décision

En général, une entreprise possède plusieurs usines avec donc plusieurs machines différentes ayant beaucoup de contraintes à prendre en compte. De plus chaque usine à son propre carnet de commandes et donc son propre calendrier de décisions. Faire un plan pour une seule usine est déjà un très gros travail d'analyse pour l'humain et le faire à la main n'est donc pas chose aisée. On utilise plutôt des logiciels qui permettent d'enregistrer ces informations et de nous faciliter le travail. MasterPlanner (ou MPlanner) est un outil puissant créé par PSI Metals et permettant de s'adapter à la configuration de n'importe qu'elle entreprise de production, et ainsi faciliter ces prises de décision. Il sera décrit dans les chapitres suivants.

Figure 9 Splash Screen d'MPlanner

3. Environnement de travail

Psi Metals dispose d'un étage, dans le bâtiment de Bisnode, où est concentrée toute son activité. Les locaux sont découpés en plusieurs bureaux où chacun peut travailler dans le calme et à son rythme.

3.1.Mon Bureau

À mon arrivée, j'ai pu m'installer dans le bureau de Robert ou j'ai trouvé trois grands bureaux chacun équipé d'un grand écran ainsi que d'un ordinateur portable sur lequel était branché un clavier français. Nous étions donc prêts à nous mettre au travail dès la première heure.

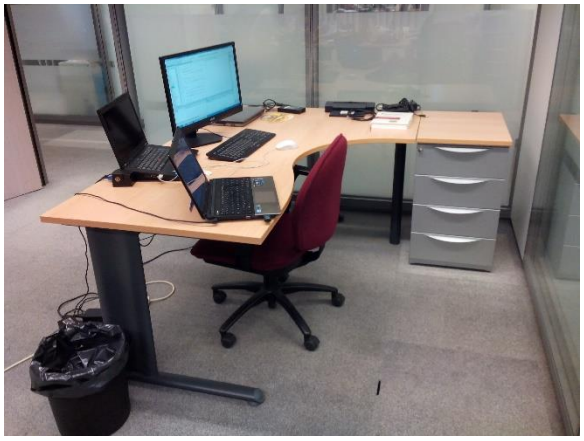


Figure 10 Mon bureau à PSI Metals Bruxelles

Le PC auquel j'avais accès était équipé de Windows 7 64 bits, d'un processeur Intel® Core™ i7-2820QM avec quatre cœurs cadencés à 2.30 Ghz (pouvant aller jusque 3.30 Ghz en mode turbo) et de 8 GO de RAM. J'ai donc pu travailler sans problème de latence durant toute la durée de mon stage. De plus, tous les PCs sont reliés grâce au réseau interne de PSI Metals, ce qui m'a facilité l'échange de gros fichiers.

3.2.Langage de programmation

Durant mes développements, j'ai pu me familiariser avec le langage C++ qui est un langage orienté objet. Ayant étudié le Java durant ces deux années à l'IUT, il n'a pas été très difficile de passer de l'un à l'autre car les notions de classes et d'objets sont à peu près les mêmes. Cependant, le C++ étant un langage de plus bas niveau que le Java, c'est-à-dire plus proche du langage machine, il était donc plus facile de faire des erreurs de gestion de mémoire. En contrepartie, il permet de mieux optimiser son logiciel contrairement au Java qui lui utilise une machine virtuelle pour lancer les logiciels et gérer la mémoire.

Les logiciels auquels j'ai dû apporter mes modifications utilisent aussi un langage de script qui est le Python. Ce dernier n'étant pas un langage compilé mais plutôt interprété, il est très facile d'ajouter ou de modifier des fonctions aux logiciels. Ceci est d'autant plus vrai lorsque l'on veut rapidement voir le résultat d'une

petite modification car il n'est pas nécessaire de recompiler tout le logiciel pour voir ces modifications car les scripts python sont dans des fichiers .py à part qui sont interprétés par Python.

3.3.Logiciels utilisés

3.3.1. Visual Studio 2008

L'environnement de développement utilisé par l'entreprise est Visual Studio 2008. Il permet de découper un projet en plusieurs morceaux afin de travailler sur chacune des parties de notre logiciel. Il est entièrement configurable et est doté de la fonctionnalité IntelliSense qui augmente la productivité grâce à l'auto-complétion qu'elle propose. Ainsi il suffit de taper un début de nom de variable ou de méthode, d'effectuer le raccourci « Ctrl + espace » pour qu'ensuite nous ayons une liste de fonctions et variables similaires à notre saisie. Il devient alors très rapide de développer une application lorsque l'on maitrise l'outil.

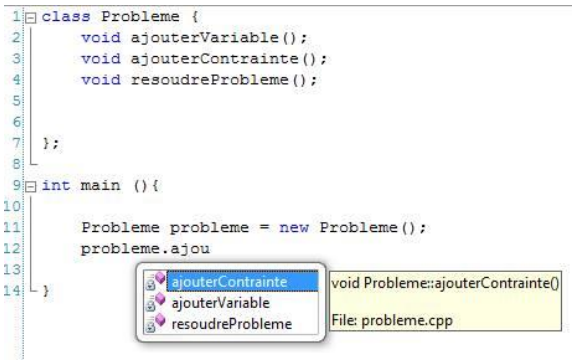


Figure 11 Exemple de la fonction Intellisense

3.3.2. Notepad ++

Pour l'édition des fichiers textes et quelques autres types de fichiers, notamment les fichiers sources python, j'ai utilisé Notepad++ qui est un d'éditeur de source qui possède un outil de coloration syntaxique. Selon le langage de programmation configuré, il met en valeur les mots clé de celui-ci afin de faciliter la lecture du code source. Il est aussi possible de définir nos propres mots clé. De plus, il propose une fonction d'édition du texte en colonne. Il est donc possible d'écrire sur plusieurs lignes en même temps ce qui est pratique lorsque l'on veut modifier des lignes qui se ressemblent.

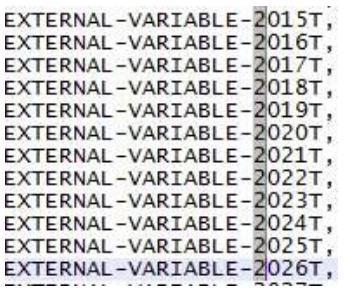


Figure 12 Mode colonne de Notepad++

3.3.3. Beyond Compare

J'ai dû aussi très souvent comparer les différences entre deux fichiers, par exemple avant et après modification. J'ai donc utilisé le logiciel Beyond Compare s'intègre au menu contextuel de Windows et qui permet ainsi de sélectionner facilement deux fichiers, en général textuel, et de comparer dans la même fenêtre les différences. On peut alors choisir les parties de texte que l'on souhaite garder et fusionner deux fichiers en un seul.

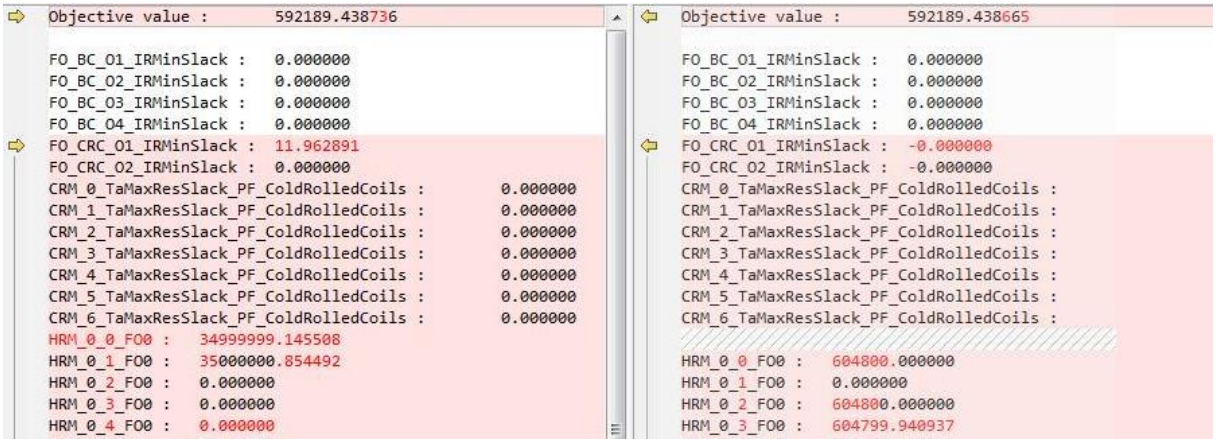


Figure 13 Exemple de comparaison dans Beyond Compare

3.3.4. Tortoise SVN

Lorsque l'on développe un logiciel, il est important de conserver les anciennes versions fonctionnelles. Pour cela on utilise un gestionnaire de versions comme Tortoise SVN. Il permet de mettre à disposition de tous les utilisateurs qui y sont connectés, de voir les différentes versions du logiciel en cours de développement, de les télécharger ou de modifier la dernière version à partir de son éditeur. On peut alors travailler à plusieurs sur le même code et fusionner notre code avec celui des autres grâce à la fonction « commit » de cet outil. De plus dans l'explorateur Windows on peut voir quels fichiers n'ont pas encore été synchronisés avec le serveur SVN.

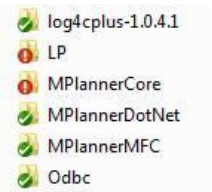
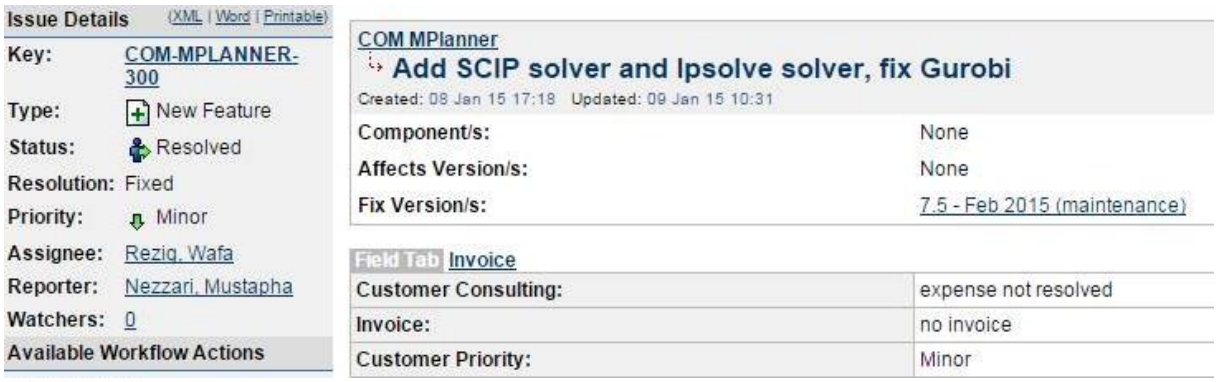


Figure 14 Exemple de fichiers synchronisés

3.3.5. Jira

Jira est un outil de gestion de projet permettant d'organiser les demandes de fonctionnalités sur un logiciel. Ainsi, on peut facilement voir la liste des demandes, l'assigner aux différents développeurs et suivre leur activité. Pour ma part, mon tuteur a créé une demande d'ajout de solveur de problèmes

mathématiques au logiciel MPlanner. Lorsque j'ai terminé ma mission, j'ai utilisé tortoise SVN pour envoyer mon travail sur le serveur en indiquant le numéro de la demande traitée. On sait maintenant que l'utilisateur « Nezzari Mustapha » a traité la demande numéro 300 sur le logiciel MPlanner qui est l'ajout du solveur SCIP et lpsolve.



Issue Details (XML | Word | Printable)

Key:	COM-MPLANNER-300
Type:	New Feature
Status:	Resolved
Resolution:	Fixed
Priority:	Minor
Assignee:	Rezig, Wafa
Reporter:	Nezzari, Mustapha
Watchers:	0

Available Workflow Actions

COM MPlanner

Add SCIP solver and lpsolve solver, fix Gurobi

Created: 08 Jan 15 17:18 Updated: 09 Jan 15 10:31

Component/s:	None
Affects Version/s:	None
Fix Version/s:	7.5 - Feb 2015 (maintenance)

Field Tab Invoice

Customer Consulting:	expense not resolved
Invoice:	no invoice
Customer Priority:	Minor

Figure 15 Demande à laquelle j'ai répondu

3.3.6. Ligne de commande Windows

La ligne de commande Windows m'a beaucoup servi lors de la compilation des différentes librairies nécessaires à l'avancement de mon projet. Le raccourci très utile « MAJ + Clic droit » permet de faire apparaître une commande caché dans le menu contextuel de Windows qui permet d'ouvrir ligne de commande dans le dossier actuel ou l'on a effectué le raccourci. Cela nous évite de se déplacer dans les dossiers en ligne de commande.

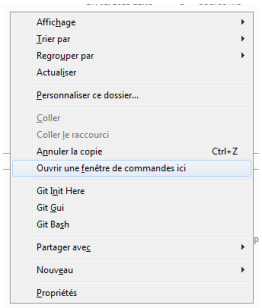


Figure 16 "Ouvrir une fenêtre de commande ici"

4. MPlanner : Solution de planification

Comme nous l'avions dit précédemment, MPlanner est un logiciel d'aide à la décision lors de la planification de la production de métal. Pour planifier cette production nous devons tenir compte de trois choses importantes : Le carnet de commandes, les contraintes de mes machines et de mon calendrier.

4.1.Les configurations

Chaque entreprise à son propre carnet de commande, son propre calendrier et ses propres contraintes. Dans MPlanner on parle de configuration. Une configuration correspond informatiquement à un dossier contenant toutes ces informations dans des fichiers textes. Le dossier racine a en général le nom de l'entreprise et l'arborescence de ce dossier est découpée en plusieurs sous-dossiers qui contiennent différents types d'information. Par exemple on peut avoir un dossier « Data » qui va contenir un fichier « constraints.txt » qui spécifie les contraintes des machines, ou encore le fichier « fos.txt » qui contient les commande. On peut aussi avoir un dossier « Routing » qui contient les différentes lignes de production et les routes de chaque produit.

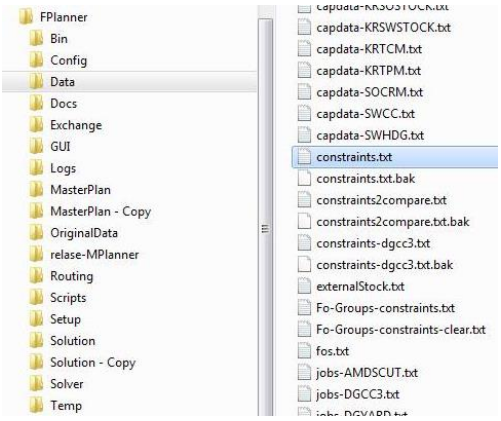


Figure 17 Exemple d'arborescence d'une configuration

Il est possible de changer toutes ces informations grâce à un fichier lu au lancement de MPlanner qui se trouve en général dans le dossier « setup ». Par défaut il est nommé « mfc.setup ». Il contient les chemins vers les différents fichiers de la configuration et permet ainsi de modéliser notre propre arborescence.

```
# routing files
Lines-File                          "..\Routing\lines.txt"
Material-Kinds-File                 "..\Routing\mks.txt"
Product-Families-Format-File        "..\Exchange\pfs.fmt"
Product-Families-Data-File          "..\Exchange\pfs.txt"
Technical-Elaborations-Format-File  "..\Exchange\tes.fmt"
Technical-Elaborations-Data-File    "..\Exchange\tes.txt"
Nodes-Format-File                   "..\Exchange\nodes.fmt"
Nodes-Data-File                     "..\Exchange\nodes.txt"
Arcs-Format-File                    "..\Exchange\arcs.fmt"
Arcs-Data-File                      "..\Exchange\arcs.txt"
Line-Groups-File                    "..\Routing\linegroups.txt"
Network-Contexts-File               "..\Config\NetworkContexts.txt"

# time horizon
Time-Buckets-File                   "..\Exchange\Buckets.txt"

# scripts
Scripts-File                        "..\Config\Scripts.cfg"

# calendar
stops-definition-file               "..\Config\stopsdefinition.txt"
stops-periods-file                  "..\Data\stops.txt"

# order book
Orders-Format-File                  "..\Config\fos.fmt"
Orders-Data-File                    "..\Data\fos.txt"
Orders-Category-File                "..\Config\foxategories.txt"
Orders-Group-File                   "..\Config\fogroups.txt"
Final-Orders-Group-Constraints-File "..\Data\fo-Groups-constraints-clear.txt"

# cap gui (spreadsheet)
Cap-Spreadsheet-Contexts-File       "..\Config\capcontexts.txt"
Cap-Spreadsheet-Configurations-File "..\Config\CapConfigs.txt"
Cap-Spreadsheet-Colour-Preferences  "..\Config\CapColours.txt"
```

Figure 18 Exemple de fichier mfc.setup

Il suffit donc d'ouvrir une configuration dans MPlanner en lui passant en paramètre le chemin vers le dossier contenant le fichier « mfc.setup » et il s'occupe de charger toutes les données de la configuration dans sa mémoire.

MPlanner va même encore plus loin en proposant de configurer complètement son interface selon les préférences de l'utilisateur. Lors de son lancement il lit les fichiers définissant son interface graphique et selon les données, MPlanner peut avoir une forme très différente d'une configuration à l'autre.

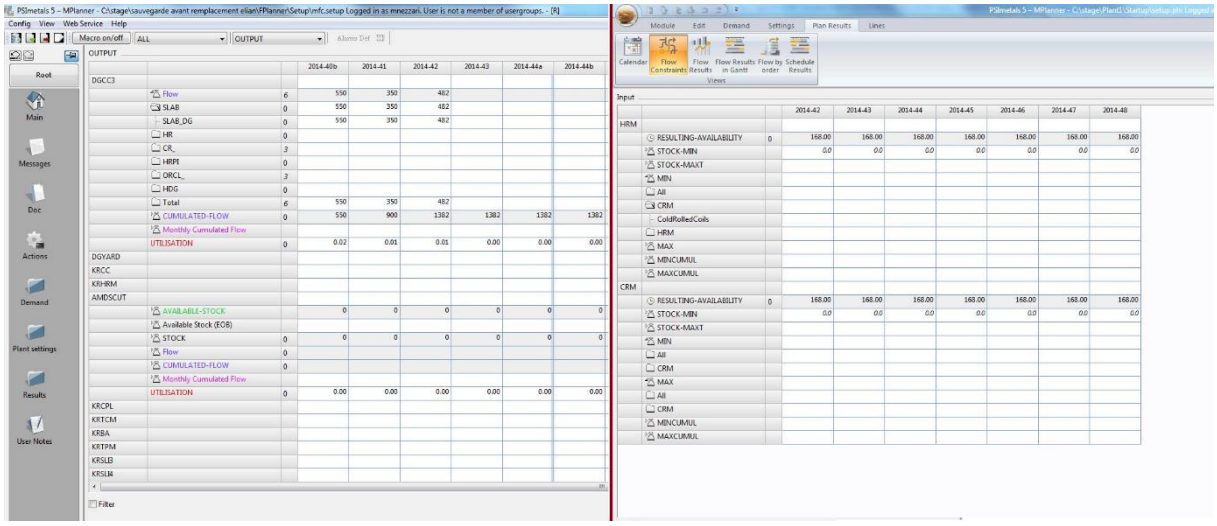


Figure 19 Exemple de deux configurations ayant une interface graphique différente

On peut ainsi cacher des menus jugés non nécessaire et n'afficher que ce qui sont pertinents pour l'utilisateur.

4.2.Les actions

Afin de manipuler MPlanner, les utilisateurs ont accès à une liste d'actions qu'il est possible d'effectuer. Il existe deux types d'action : Celles qui sont codées en dur dans MPlanner et celles codées dans des scripts python. On peut alors charger les contraintes des machines grâce à l'action « import-cap-constraints » ou exécuter un script python grâce à l'action « Run-python-script ». Ainsi il est possible d'ajouter de nouveau module à MPlanner grâce à cette dernière action.

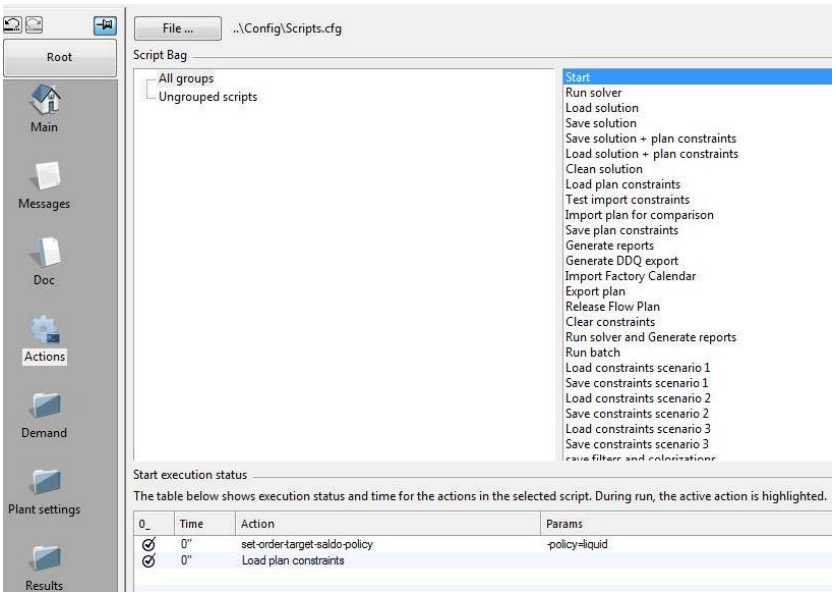


Figure 20 Exemple d'une liste d'action

Les actions sont définies dans un fichier appelé « script.cfg ». Pour définir un script on écrit « script » suivi du nom du script puis entre crochet on met les actions à effectuer. Ces actions peuvent être celles de MPlanner comme « import-cap-constraints » mais cela peut très bien aussi être un autre script existant dans le même fichier.

```
script "Start"
[
    set-order-target-saldo-policy "-policy=liquid"
    #Run-python-script "-script=..\Scripts\PreprocessInitialStocks"
    #Init-Dynamic-Data
    #Run-python-script "-script=..\Scripts\PreprocessOrderBook"
    #export-order-group-constraints "-path=..\data\Eo-Groups-constraints.txt"
    'Load plan constraints"
    #Init-Global-Solver "-setupfile=..\Solver\Gs.setup -capstocks -posstocks -min2target"
    #generate-constraints-from-irs "-gslpos"
]

script "Run solver"
[
    "Clean solution"
    Init-Global-Solver "-setupfile=..\Solver\Gs.setup -capstocks -posstocks -min2target"
    run-gslp-solver "-configfile=..\Solver\gslpsolver.txt"
    load-completion-info "-planning -computefullstats"
    save-gs-solution "-path=..\Solution\solution.txt"
    Run-python-script "-script=..\Scripts\CalcMatInTransition"
    Run-python-script "-script=..\Scripts\CalcStockEob"
    Run-python-script "-script=..\Scripts\CalcCleanedUpFlow"
    Run-python-script "-script=..\Scripts\CalcMonthlyCumulFlow"
]

script "Load solution"
[
    load-gs-solution "-path=..\Solution\solution.txt"
    load-completion-info "-planning -computefullstats"
    Run-python-script "-script=..\Scripts\CalcMatInTransition"
    Run-python-script "-script=..\Scripts\CalcStockEob"
]

script "Save solution"
[
    save-gs-solution "-path=..\Solution\solution.txt"
]

script "Save solution + plan constraints"
[
    "Save solution"
    "Save plan constraints"
]
```

Figure 21 Fichier script.cfg correspondant aux actions de la figure précédente

4.3.Le plan

Comme nous l'avons dit MPlanner de faire des plans. Nous avons pour cela deux vues importantes. La grille des inputs qui correspondent aux contraintes et les outputs qui correspondent au flux. Je peux donc consulter la grille de contraintes fixées par semaine et par famille de produit sur chaque ligne après avoir chargé les données grâce à l'action « import-cap-constraints ». Je vais par exemple avoir la ligne Caster qui contient une liste déroulante de contraintes dont le « Stock minimum » qui est aussi une liste déroulante contenant chaque type de produit qui eux contiennent toutes les familles de produit qui leurs sont associés

			2014-40	2014-41	2014-42	2014-43
Caster						
	Stock Minimum		700	700	700	700
	Brame		700	700	700	700
	Brame 15m		300	300	300	300
	Brame 20m		400	400	400	400
	Stock Maximum		2000	2000	2000	2000
Laminoir						
	Stock Minimum		800	800	800	800
	Brame		500	500	500	500
	Bobine chaude		300	300	300	300
	Stock Maximum		3000	3000	3000	3000
Peinture						

Figure 22 Exemple de grille d'Inputs

Ensuite je peux saisir mes flux dans la grille des flux et voir si je viole certaines contraintes grâce à une coloration rouge. Par exemple lorsque je saisie un flux d'approvisionnement de 200 tonnes alors qu'il faut au minimum un stock de 400 tonnes, mon flux apparaît en rouge pour me prévenir que je viole la contrainte du stock minimum. Je peux aussi avoir un flux cible à atteindre, et si je ne l'atteins pas cela me sera signalé.

			2014-40	2014-41	2014-42	2014-43
Caster						
	Flux		600			
	Brame		600			
	Brame 15m		400			
	Brame 20m		200			
Laminoir						

Figure 23 Exemple d'erreur dans la grille d'Outputs

Comme nous l'avons aussi dit tout à l'heure, MPlanner est entièrement configurable et il est possible d'avoir un ensemble de vues « constraints violation » qui nous listent toutes les contraintes violées. Nous pouvons avoir des violations de stock, des violations de flux, des saturations de lignes, etc. De même qu'il est possible de créer une vue avec les inputs et les outputs sur une seule grille. Cela nous permet de corriger plus rapidement nos flux sans devoir naviguer entre les deux vues.

			2014-40	2014-41	2014-42	2014-43
Caster						
	Flux		600			
	Brame		600			
	Brame 15m		400			
	Brame 20m		200			
	Stock Minimum		700	700	700	700
	Brame		700	700	700	700
	Brame 15m		300	300	300	300
	Brame 20m		400	400	400	400
	Stock Maximum		2000	2000	2000	2000
Laminoir						
	Flux		1000			
	Stock Minimum		800	800	800	800
	Brame		500	500	500	500
	Bobine chaude		300	300	300	300
	Stock Maximum		3000	3000	3000	3000
Peinture						

Figure 24 Exemple de nouvelle vue permettant de voir les flux ainsi que les contraintes

4.4.Le solveur

Nous avons vu que lorsque l'on planifie la production de métal, nous devons prendre en compte plusieurs informations et il est difficile de créer un bon plan du premier coup. En général on va vouloir savoir quel est le meilleur plan que je peux obtenir et qui respecte au mieux les contraintes. Autrement dit, on aimerait avoir un plan où l'on a les flux optimaux par ligne et par semaine et qui respectent quasiment toutes les contraintes.

Ceci peut être fait grâce à un solveur qui est un outil permettant de générer un problème mathématique à partir de variables et de contraintes. À partir de ces informations le solveur va nous générer une solution optimale. Dans notre cas, les variables correspondent aux flux de matière passant par ligne et par semaine. Quant aux contraintes ce sont celles des machines et du calendrier. Enfin la solution optimale sera notre plan optimisé afin d'utiliser au mieux la capacité de nos machines tout en respectant les contraintes imposées.

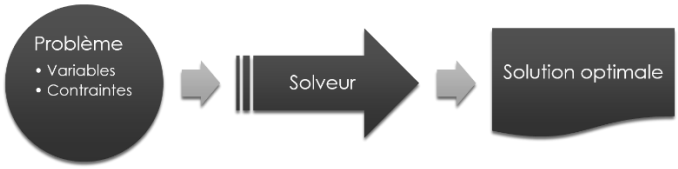


Figure 25 Schéma d'une résolution d'un problème d'optimisation

Il est possible de générer le fichier problème afin de voir si les bonnes variables et contraintes sont bien prises en compte lors de la résolution. De même qu'après une résolution on peut générer le fichier contenant la solution.

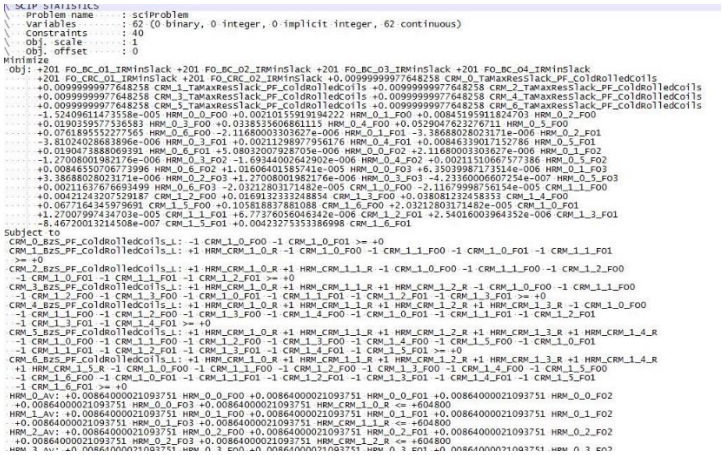


Figure 26 Exemple de fichier de problème

Cependant il est possible qu'un problème n'ait pas de solutions et donc que ce que l'on veut soit impossible avec les contraintes actuelles. C'est pourquoi il est possible de configurer le solveur afin qu'il génère une solution s'approchant de la solution optimale en lui permettant de violer les contraintes dites douces. Par exemple, on lui permettra d'aller en dessous d'un stock minimal même si une contrainte avait été définie sur ce stock.

Dans MPlanner, trois solveurs étaient déjà intégrés avant mon arrivée. Deux d'entre eux sont des solveurs commerciaux nommés Cplex et Gurobi et le dernier du nom de Clp, est gratuit et open source. Chacun de ces solveurs possède plusieurs fonctions afin de modéliser un problème composé de variables et contraintes comme c'est le cas ici. Les quatre fonctions principales de chaque solveur sont : Créer un nouveau problème, ajouter une variable à ce problème, ajouter une contrainte à ce problème et résoudre ce problème.

Figure 27 Les trois solveurs de MPlanner

Dans une configuration, nous avons un fichier qui s'appelle « GsLpSolver.cfg » qui permet d'indiquer le solveur que l'on veut utiliser. Et selon le solveur actif, MPlanner va appeler les fonctions de ce solveur afin de résoudre un problème.

```
# Decide which solver will be used.
# It could be:
# - clp
# - cplex
# - gurobi
# - scip
# - lpsolve
# Remark: The name of the solver is case sensitive
Solver ..... "cplex"
```

Figure 28 Partie du fichier Gslpsolver.cfg dédiée à la sélection du solveur

Après résolution, MPlanner va appeler la fonction de ce solveur afin d'enregistrer la solution dans un fichier texte. Rappelons que cette solution est un nouveau plan qui correspond au plan optimisé contenant les meilleurs flux de matières afin de satisfaire les commandes dans les délais.

lpsolution.txt - Notepad

File Edit Format View Help

```
Objective value :       592189.438724

FO_BC_O1_IRMinSlack :   0.000000
FO_BC_O2_IRMinSlack :   0.000000
FO_BC_O3_IRMinSlack :   0.000000
FO_BC_O4_IRMinSlack :   0.000000
FO_CRC_O1_IRMinSlack :  11.962891
FO_CRC_O2_IRMinSlack :  0.000000
CRM_0_TaMaxResSlack_PF_ColdRolledCoils :        0.000000
CRM_1_TaMaxResSlack_PF_ColdRolledCoils :        0.000000
CRM_2_TaMaxResSlack_PF_ColdRolledCoils :        0.000000
CRM_3_TaMaxResSlack_PF_ColdRolledCoils :        0.000000
CRM_4_TaMaxResSlack_PF_ColdRolledCoils :        0.000000
CRM_5_TaMaxResSlack_PF_ColdRolledCoils :        0.000000
CRM_6_TaMaxResSlack_PF_ColdRolledCoils :        0.000000
HRM_0_0_FO0 :   34999999.145508
HRM_0_1_FO0 :   35000000.854492
HRM_0_2_FO0 :   0.000000
HRM_0_3_FO0 :   0.000000
HRM_0_4_FO0 :   0.000000
HRM_0_5_FO0 :   0.000000
HRM_0_6_FO0 :   0.000000
HRM_0_0_FO1 :   0.000000
HRM_0_1_FO1 :   1.708984
HRM_0_2_FO1 :   34999999.145508
HRM_0_3_FO1 :   34999999.145508
HRM_0_4_FO1 :   0.000000
HRM_0_5_FO1 :   0.000000
HRM_0_6_FO1 :   0.000000
HRM_0_0_FO2 :   0.000000
HRM_0_1_FO2 :   0.000000
HRM_0_2_FO2 :   0.000000
HRM_0_3_FO2 :   0.000000
HRM_0_4_FO2 :   34999999.145508
HRM_0_5_FO2 :   35000000.854492
```

Figure 29 Exemple de fichier solution

5. Présentation des projets

Selon la taille d'une entreprise et de son carnet de commande, la taille du problème mathématique peut varier. Et en général, plus un problème est grand en terme de variables à trouver et de contraintes à respecter, plus il est long à résoudre. Cplex, développé par IBM, et Gurobi étant des solveurs dont la licence pour chacun peut aller jusque 120000 €, une certaine garantie de rapidité de résolution est exigée contrairement à Clp. Ce dernier étant assez lent, ma mission fut de le remplacer.

5.1.Intégration de SCIP à MPlanner

5.1.1. Présentation de SCIP

SCIP est actuellement l'un des plus rapides solveurs non commerciaux de problèmes mathématiques de notre type. Il est développé en C par le Zuse Institute à Berlin.

Figure 30 Logo de SCIP

Il se présente sous forme de wrapper C++, c'est-à-dire que SCIP est une bibliothèque de classes C++ faisant appel à des fonctions écrites en C ce qui permet d'utiliser des fonctions à la base écrite dans un langage non orienté objet, dans un langage orienté objet comme le C++.

5.1.2. Objectifs

La mission qui m'a été confiée par mon tuteur a été d'intégrer SCIP et de tester ses performances en le comparant à Clp et voir s'il peut le remplacer. Le but est aussi d'utiliser le moins possible les solveurs commerciaux.

Afin de commencer il fallait aussi que je télécharge la dernière version de MPlanner depuis le SVN et que je configure Visual Studio afin de pouvoir compiler MPlanner et commencer à implémenter SCIP.

5.1.3. Implémentation

Comme dit précédemment, selon le solveur actif, MPlanner va appeler les fonctions de ce dernier. Ceci est possible grâce à l'utilisation d'une interface dans le logiciel qui s'appelle CLPBase. Pour implémenter SCIP, il suffit d'implémenter les fonctions de cette interface dans une nouvelle classe CLPScip.

Les fonctions à implémenter sont principalement le constructeur qui créer un problème vide, le destructeur qui supprime toute les variables et contraintes ainsi que le problème, la fonction setFuntion() qui créer les variables, la fonction addConstraint() pour ajouter une contrainte (et addConstraints() pour ajouter toutes les contraintes d'un coup), la fonction solve() qui résout le problème et enfin la fonction writeSolution() qui écrit la solution dans un fichier texte. (Cf Annexe 1.1)

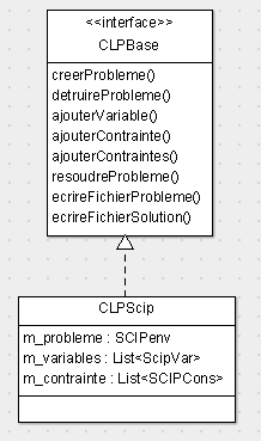


Figure 31 Diagramme UML de l'implémentation de CLPBase dans CLPScip

Pour tester SCIP, j'ai reçu de mon tuteur quelques configurations d'entreprises plus ou moins grandes. Il ne me restait plus qu'à créer une action qui importe les contraintes et qui lance la résolution du problème en utilisant le fichier GsLpSolverScip.cfg que j'ai créé et qui contient la ligne qui définit le solveur par défaut sur « scip ».

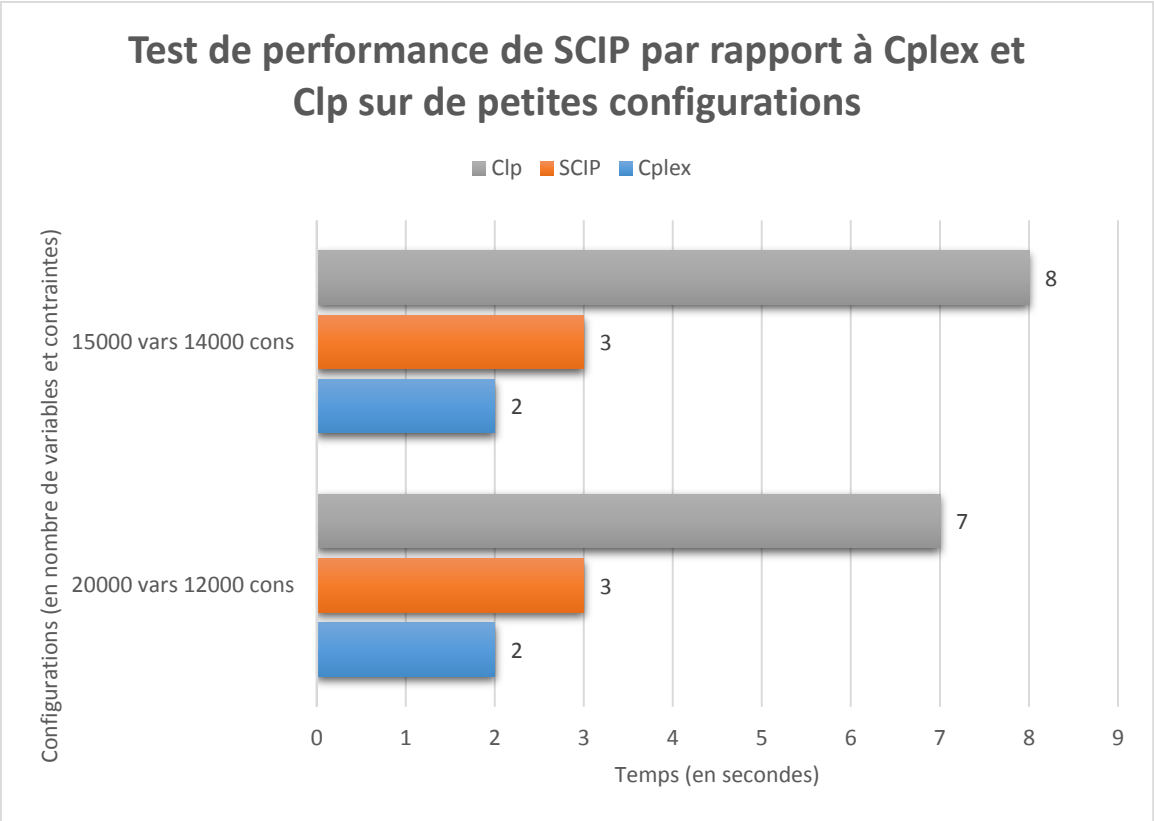
```
SCRIPT→"START"
[

··INIT-DYNAMIC-DATA
··"Init Solver"
··load-category-file·····→"-path=../../StaticConf/Groups/CapCategories.txt"

····#carga los archivos de inputs generados desde MDW
··import-cap-constraints···→"-line=all -path=..\..\data\Inputs\Availability.txt"
··import-cap-constraints···→"-line=all -path=..\..\data\Inputs\Modifiers.txt"
··import-cap-constraints···→"-line=all -path=..\..\data\Inputs\ExternalStock.txt"
··import-cap-constraints···→"-line=all -path=..\..\data\Inputs\FlowConstraints.txt"
··import-cap-constraints···→"-line=all -path=..\..\data\Inputs\StockConstraints.txt"
··import-cap-constraints···→"-linegroup=all -path=..\..\data\Inputs\FlowConstraintsLG.txt"
··import-cap-constraints···→"-linegroup=all -path=..\..\data\Inputs\StockConstraintsLG.txt"

··#import-cap-constraints···→"-line=all -path=..\..\data\Inputs\StockTargetFactor.txt"

··#carga la restriccion de 0 stock para las lineas ficticias, generada en la configuracion estatica
··import-cap-constraints···→"-line=all -path=..\..\StaticConf\SCM\Routes\StockConstraintsFI.txt"

··import-cap-constraints···→"-line=all -path=..\..\data\Inputs\StockConstraintsXLS.txt"

··
··run-system-command·"-cmd=echo %time%: Fin Script START >> ..\logs\TotalTime.log"

]

SCRIPT·"RunScip"
[
——→"START"
——→"RunSolverScip"
——→"Export solution"
——→
]

SCRIPT→"RunSolverScip"
[
run-gslp-solver·——→"-configfile=..\..\StaticConf\Tuning\gslpsolver-scip.cfg" # -linegroup=Todo_PLAN"]
]

SCRIPT·"Export solution"
[

save-gs-solution——→——→"-path=..\..\data\Outputs\gssolution.txt"

export-cap-constraints·"-line=all·-path=..\Outputs\Total_tons.txt·-export=PvtTns·-reloadExportFormat"
export-cap-constraints·"-line=all·-path=..\Outputs\Total_hours.txt·-export=PvtHrs"
export-cap-constraints·"-linegroup=all·-path=..\Outputs\LineGroup_tons.txt·-export=PvtTns"

export-cap-constraints·"-line=all·-path=..\Outputs\stock_flow.txt·-export=stockflow"
```

Figure 32 Exemple de fichier script.cfg pour tester SCIP

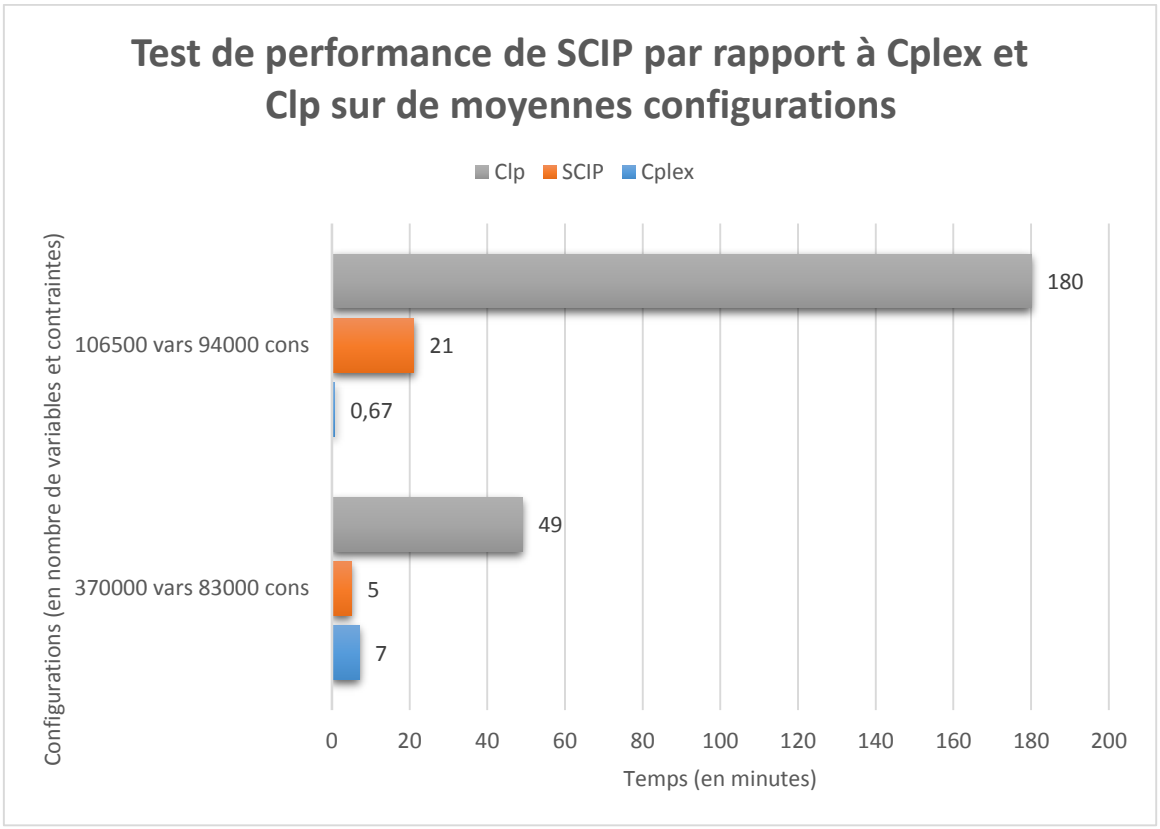
Pour vérifier si les fichiers problèmes et solutions générés sont bons, je les ai comparés à ceux générés par Cplex grâce à l'outil Beyond Compare. Cela m'a permis de savoir aussi si j'utilisais les bonnes fonctions de SCIP.

5.1.4. Test de performance

Afin de tester SCIP, j'ai reçu de mon tuteur, plusieurs configurations représentées ici en nombres de variables à trouver (vars) et contraintes à respecter (cons). Voici quelques résultats :

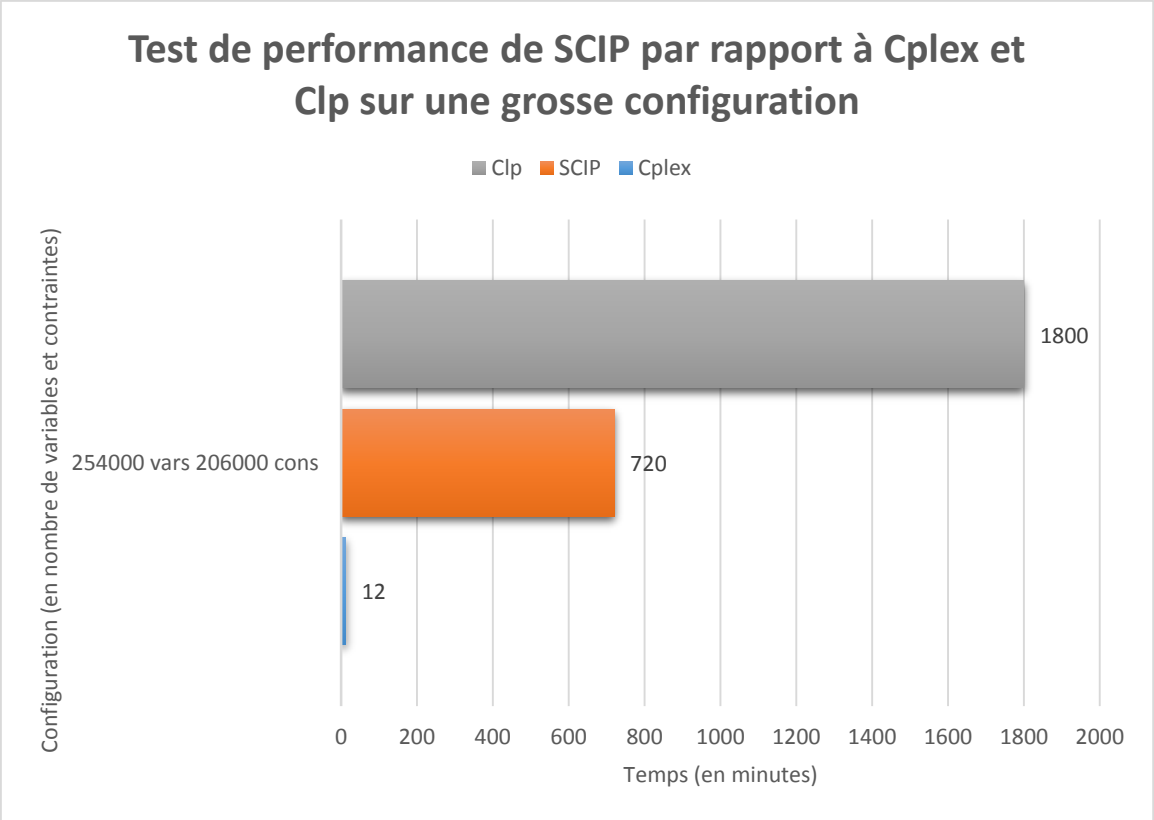


On voit ici que SCIP reste assez proche de Cplex et que Clp met déjà plus de temps à résoudre un petit problème. De plus, la solution de Clp ne ressemble pas du tout à celle de Cplex. Par contre celle de SCIP a juste quelques différences négligeables. Les différences entres les solutions de SCIP et Cplex sont très minimes (Cf annexe 1.2).



Ici nous avons deux résultats différents entre SCIP et Cplex. Dans un premier problème nous avons SCIP qui met 21 minutes tandis que Cplex met 40 secondes. Cependant dans le deuxième SCIP arrive à battre Cplex pour la première fois avec 5 minutes contre 7 alors que Clp met 49 minutes. Ce deuxième test a été effectué deux fois pour savoir s'il ne s'agissait pas d'un ralentissement dû à l'ordinateur. Les résultats ont été quasiment pareils à quelques secondes près.

SCIP est encore une fois plus rapide que Clp et de loin. Il est environ 9 fois plus rapide sur les tests moyens.



Sur ce dernier test, nous voyons qu'il y a un trou énorme entre Cplex et SCIP mais un autre encore plus grand entre SCIP et Clp. Cplex a mis 12 minutes tandis que SCIP a mis 12h ce qui est 60 fois plus lent. Cependant, ce qui nous intéresse est de savoir si SCIP est plus rapide que Clp et nous voyons bien que sur ce gros test il est au moins 2 fois plus rapide que lui. D'autres tests ont été effectués mais Clp a planté en pleine résolution car il gérait mal la mémoire.

Voici un tableau récapitulatif des résultats de chaque graphique :

	Cplex	SCIP	Clp
20 000 vars 12 000, cons	2 secondes	3 secondes	7 secondes
15 000 vars 14 000 cons	2 secondes	3 secondes	8 secondes
370 000 vars 83 000 cons	7 minutes	5 minutes	49 minutes
106 500 vars 94 000 cons	40 secondes	21 minutes	180 minutes
148 000 vars 102 000 cons	7 minutes	1h40	OUT OF MEMORY
183 000 vars 125 000 cons	19 minutes	9h	OUT OF MEMORY
254 000 vars 206 000 cons	12 minutes	12h	>26h

Tableau 1 Tableau récapitulatif des temps de résolution de Cplex, SCIP et Clp

Deux lignes ont été rajoutées car elles ne pouvaient pas être représentées dans le graphique car Clp a planté en plein résolution à cause d'une mauvaise gestion de mémoire (Out of memory).

5.1.5. Commit du projet

À la fin de mon développement, j'ai dû envoyer mon code sur le serveur SVN afin que mon travail soit pris en compte dans MPlanner. Pour cela j'ai dû comparer mes fichiers avec ceux sur le serveur avec Beyond Compare et choisir les bouts de code que je voulais garder. Cela m'a permis donc de fusionner mon code avec la dernière version de MPlanner et j'ai ainsi pu répondre à la demande d'ajout de solveur.

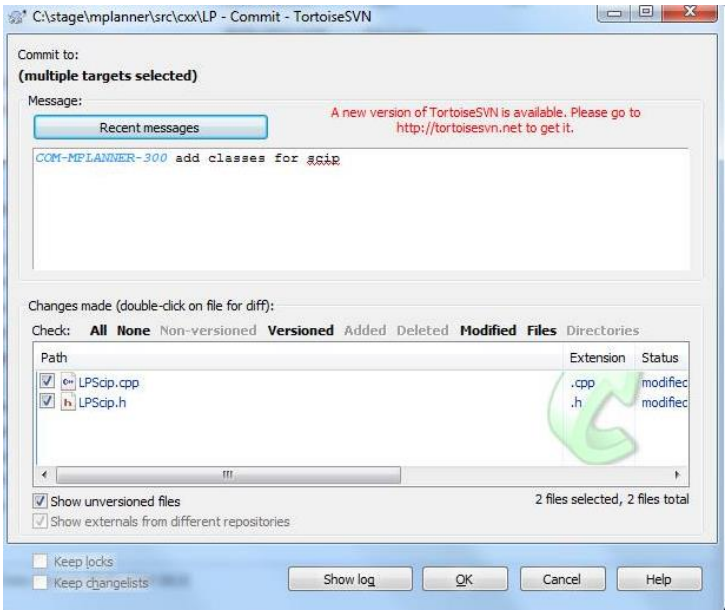


Figure 33 Commit du code de la classe du solveur SCIP

5.1.6. Bilan

Comme l'on a pu le voir, SCIP bat Clp sur tous les tests effectués et selon la taille du problème, il se rapproche plus ou moins de Cplex. De plus sur certains des tests du Tableau 1, SCIP a été jusqu'au bout des résolutions tandis que Clp s'arrêtait en plein milieu faute de mémoire. SCIP gère donc mieux la mémoire que Clp et est plus rapide que lui.

La version que j'ai implémenté était une version en cours de développement et contient encore des bugs. A l'heure où j'écris ce rapport une nouvelle version est sortie corrigeant beaucoup d'entre eux. La mettre à jour et la tester pourraient constituer une mission pour de futures améliorations.

5.2.Intégration de lpsolve à MPlanner

5.2.1. Présentation de lpsolve

Lpsolve est lui aussi un solveur gratuit et open source supportant plusieurs langages tels que le C, C++, Java, Python et beaucoup d'autres langages. Ce n'est pas un wrapper comme SCIP mais cela n'empêche pas de l'utiliser car le C++ reste avant tout une « mise à jour » du C et donc ces deux langages peuvent être combinés.

5.2.2. Implémentation de lpsolve

Contrairement à SCIP, il ne m'a pas été demandé de tester les performances de lpsolve mais juste de l'implémenter. Je l'ai donc fait de la même manière que SCIP en créant une nouvelle classe CLPLpsolve et en implémentant les fonctions de l'interface LPBase (Cf annexe 2.1). Le problème c'est que je n'ai eu le temps de le tester que sur une seule configuration dont il générait le bon fichier problème mais pas la bonne solution. Mon tuteur m'a donc conseillé de ne pas m'en occuper mais plutôt de me concentrer sur le benchmarking de SCIP.

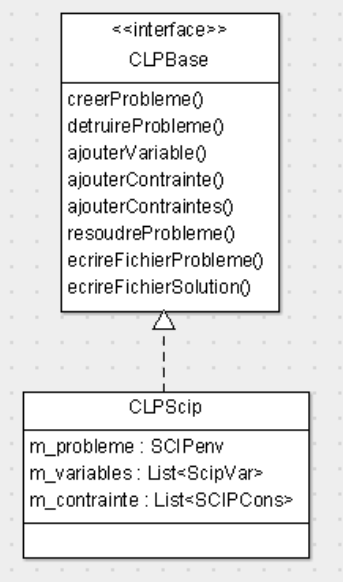


Figure 34 Diagramme UML de l'implémentation de CLPBase dans CLPLpsolve

5.2.3. Bilan

Implémenter lpsolve m'a permis de découvrir comment utiliser une librairie écrite en C dans du code C++ sans l'intermédiaire d'un wrapper. N'ayant presque jamais fais de C++ avant cela, j'ai pu découvrir différentes façons d'utiliser une librairie avec un langage orienté objet.

De plus il me reste 15 jours pour voir s'il est plus rapide que Clp et donc s'il est nécessaire de le garder dans MPlanner ou si l'on peut juste utiliser SCIP.

6. Problèmes rencontrés

Au cours de mon stage, j'ai rencontré quelques difficultés lors de mes développements. Devoir les surmonter a été pour moi un réel plaisir car j'ai un grand sens du défi et je pense que c'est en essayant de comprendre nos erreurs que l'on apprend mieux.

6.1. Problèmes techniques

6.1.1. Apprentissage du C++

Cela faisait très longtemps que je voulais me mettre au C++ mais que je n'en n'avais pas le courage car je pensais qu'il était « difficile » à utiliser à cause du fait qu'il est assez bas niveau. A mon arrivée je ne connaissais que les bases du C mais je n'avais jamais osé toucher au C++. Cependant, lors de ma formation j'ai appris à programmer en Java ce qui m'a permis d'acquérir de solide connaissance en programmation orientée objet. Cela m'a permis de passer très facilement du Java au C++ grâce à l'aide précieuse de mon tuteur et aux cours que j'ai pu lire et visionner sur internet (cf. bibliographie).

6.1.2. Configuration de Visual Studio

N'ayant jamais travaillé en équipe sur de gros projets tel que MPlanner, je ne savais pas à quoi cela ressemblait. Après avoir synchronisé mon ordinateur avec le SVN et téléchargé le code source de la dernière version de MPlanner, j'ai ouvert le projet dans Visual Studio. J'ai été surpris de voir comment était découpé le projet. Chaque partie du logiciel avait son propre projet comme par exemple l'intelligence du programme qui se trouvait dans le projet MPlannerCore, ou encore ma partie sur les solveurs qui se trouvait dans le projet LP. Le tout fait partie de ce que l'on appelle une solution qui représente le projet MPlanner.

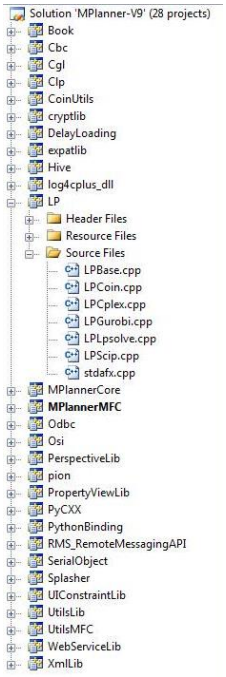


Figure 35 Arborescence de la solution MPlanner

La difficulté a été de compiler MPlanner car l'ordinateur que j'avais reçu était vierge et donc il a fallu installer Python, Boost (une bibliothèque mathématique), OpenSSL (un outil de cryptographie) et surtout configurer Visual studio pour qu'il pointe vers les différentes librairies de chaque solveur. Il fallait donc aller dans les propriétés de LP et définir le chemin vers le code source de SCIP, celui vers le fichier scip.lib qui contient les liens vers les fonctions de la bibliothèque de SCIP, et enfin le chemin vers ladite bibliothèque au format .dll. Pour ce faire, j'ai reçu l'aide de mon tuteur puis j'ai pu reproduire ces actions pour lpsolve et j'ai même pu aider un collègue de l'entreprise à le faire.

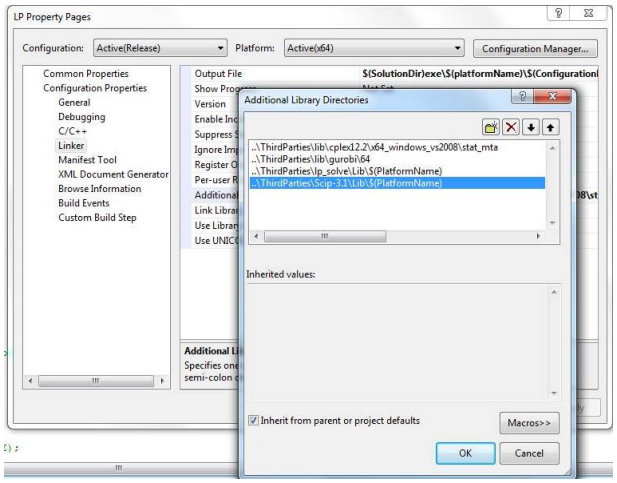


Figure 36 Ajout des chemins vers la bibliothèque de SCIP

6.1.3. Comprendre les rouages de MPlanner

Comme l'équipe de PSI Metals est stable, il a été jugé non nécessaire de maintenir une documentation de MPlanner pour se concentrer sur l'amélioration de ce dernier. Cependant pour un stagiaire n'ayant aucune connaissance en matière de production de métal et ne connaissant pas le logiciel, il est difficile de comprendre comment le logiciel fonctionne. Après avoir compris ce qu'était un solveur et comment il fonctionnait dans MPlanner, je me suis demandé comment intégrer le mien. J'ai pris comme exemple Cplex et j'ai commencé par faire une recherche dans toute la solution du mot « cplex » car le fichier GsLPSolver.cfg qui contient le nom du solveur à utiliser est lu par MPlanner et selon le nom du solveur spécifié, la classe du solveur sélectionné est utilisée (cf. Annexe 1.1.2).

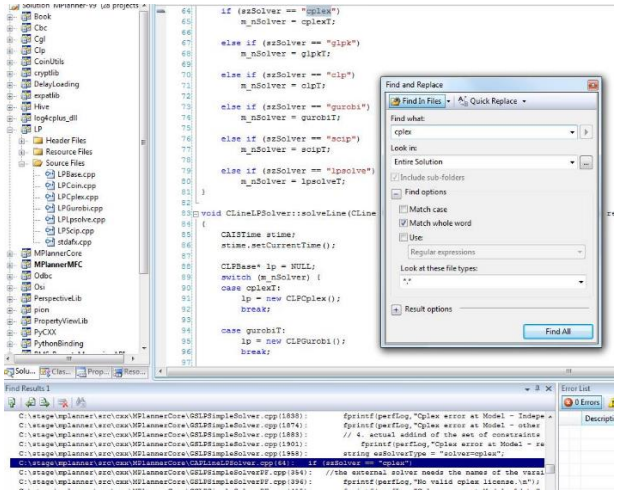


Figure 37 Exemple de recherche

Grâce à cette recherche j'ai pu voir où les classes sont créées et j'ai ensuite compris que je devais implémenter l'interface CLPBase. J'ai dû comprendre dans un premier temps à quoi correspondait chaque fonction à implémenter, avec bien sûr l'aide de mon tuteur, puis lorsque je les ai implémentées, j'ai rencontré beaucoup de bug dont les deux desquels je vais parler dans les parties suivantes.

6.1.4. Trouver les bonnes fonctions de SCIP

La bibliothèque de SCIP est très riche en fonction, si bien que certaines fonctions ont l'air d'avoir le même fonctionnement. De plus comme je n'avais pas les connaissances nécessaires pour savoir si les valeurs renvoyées par ces fonctions étaient celles attendues, j'ai du bien lire la documentation. Et pour être sûr que les valeurs de la solution générée étaient les bonnes j'ai utilisé le logiciel Beyond Compare pour comparer les solutions générées par Cplex et celles générées par SCIP. J'ai aussi reçu l'aide de mon tuteur pour trouver certaines fonctions.

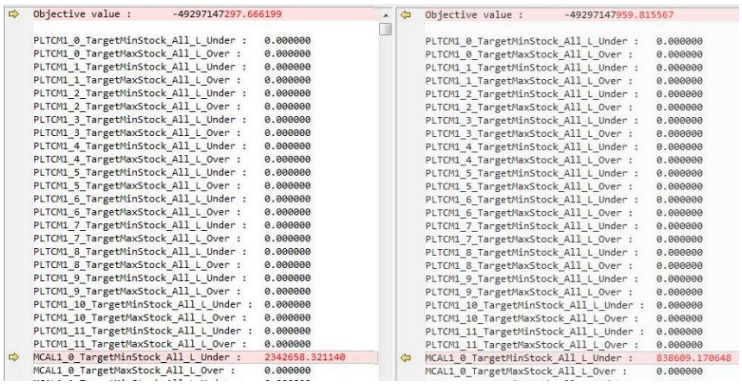


Figure 38 Exemple de comparaison avec Beyond Compare

6.2.Crash et bugs

Lors de mon développement, j'ai rencontré deux bugs majeurs qui m'ont pris un certain temps avant de réussir à les corriger.

Le premier était constitué de plusieurs bugs de chacun des solveurs (sauf Cplex) mais qui étaient le résultat d'un seul. Gurobi et SCIP crashaient en pleine résolution du problème tandis que lpsolve n'associait pas les bonnes valeurs au bonnes variables dans la solution ce que j'ai su en comparant avec la solution générée par Cplex. Tous ces bugs étaient dus au fait que la fonction qui ajoute toutes les contraintes en même temps (addConstraints() avec un « s ») et qui est utilisée le plus souvent par tous les solveurs avait certains paramètres qui provoquaient ces bugs. Les paramètres, sous forme de tableaux contenant les valeurs pour créer les contraintes, contenaient parfois des « trous » et donc les solveurs Gurobi et SCIP crashaient lors de la création du problème tandis que lpsolve ignorait ce problème mais générait une solution fausse. Pour corriger ces problèmes, j'ai dû créer de nouveaux tableaux et copier les anciens en « bouchant les trous ». Cela a eu pour effet de corriger les trois problèmes en même temps.

Le deuxième bug était que seul SCIP ne générait pas de solution avec la fonction dumpSolution() propre à MPlanner mais seulement avec sa fonction SCIPwriteProblem(). Ce n'était pas très dérangeant au début car mon tuteur m'avait donné un script python afin de normaliser la solution de SCIP et je pouvais donc générer la solution avec la fonction de SCIP et normaliser cette solution pour pouvoir la comparer à celle de Cplex. Ce fut dans le dernier mois de mon stage que je commençais à comprendre la logique de MPlanner et donc que lorsque MPlanner détecte une erreur il renvoie un code d'erreur positif tandis que SCIP renvoie une valeur nulle. Lorsque SCIP termine la résolution d'un problème il renvoi la valeur 1 qui est interprétée par MPlanner comme une erreur. J'ai donc dû faire en sorte que lorsque que SCIP renvoi son code erreur, il renvoi l'inverse du code pour que MPlanner comprenne le bon code. Tout ceci a été rendu possible grâce au puissant outil de débogage de Visual Studio qui permet d'exécuter le code ligne par ligne et de voir quelles valeurs circulent dans les variables de MPlanner. J'ai pu donc voir qu'il faisait une vérification sur le code erreur renvoyé par le solveur.

7. Conclusion

L'opportunité d'effectuer mon stage de 3 mois à l'étranger a été très bénéfique pour moi tant le plan technique qu'humain. En effet, ayant eu une première expérience à Menin, dont j'ai gardé un bon souvenir, revivre l'aventure dans le cadre d'un programme Erasmus m'a permis de confirmer mon sentiment d'affection pour la Belgique.

7.1. Bilan technique

Grâce à mon tuteur et à ma formation à l'IUT, j'ai acquis beaucoup de connaissances techniques sur tous ce qui concerne le C++ et SVN. Ainsi, je sais maintenant utiliser Visual Studio qui est un outil indispensable lorsque l'on développe en C++ notamment pour son outil de débogage très puissant et son auto-complétion très utile. J'ai aussi découvert beaucoup d'outils qui simplifient la gestion de projets comme Tortoise SVN et Jira. Enfin le projet que j'ai pu mener à terme grâce à l'encadrement de mon tuteur m'a apporté beaucoup de connaissances en termes de gestion de production et de planification et j'en suis très heureux car à mon arrivée j'avais peur de ne jamais comprendre ces notions. Faire ce stage à PSI Metals était une façon de me forcer à découvrir le C++ et les logiciels de gestion ce que je ne regrette pas.

7.2. Bilan humain

Durant ce stage j'ai dû faire preuve de beaucoup de maturité et d'autonomie, qualités très importantes dans le monde professionnel. En effet, chaque jour j'ai pu rencontrer plusieurs personnes que je ne connaissais pas notamment des clients de PSI Metals et des personnes des autres entreprises du bâtiment. De plus, j'ai particulièrement apprécié l'ambiance de PSI Metals. Les personnes que j'ai côtoyées m'ont appris le respect et l'entraide et je les en remercie encore.

Une fois de plus, je n'ai pas été déçu par la Belgique. Les gens sont accueillants et très cultivés comme j'ai pu le constater lors des soirées avec mes collègues de PSI Metals et les rencontres avec mes professeurs de l'ESI. De plus les endroits que j'ai pu visiter étaient très beaux, notamment la Grande place qui est très lumineuse et vivante le soir. J'aime aussi la verdure très présente à Anderlecht. Les nombreux parcs qui s'y trouvent sont les meilleurs endroits pour se reposer et réfléchir.

Ce stage fut aussi l'opportunité de rencontrer des personnes de différentes cultures et parlant l'anglais. L'un de mes buts principaux était de travailler mon anglais ce que j'ai pu faire avec différents membres de l'équipe lors de mes déjeuners. J'ai aussi découvert le cinéma en VOSTFR à l'UGC de Brouckère ce qui ne m'a pas déplu.

Durant ces trois mois, j'ai eu aussi l'occasion de vivre en autonomie dans un appartement situé à Veeweyde. Ce fût pour moi tout nouveau que de devoir m'occuper tout seul de moi-même. Cela m'a forcé à sortir et découvrir les différents magasins et centres de loisirs de la ville, choses que je n'ai presque jamais faites à Lille. De plus, en vivant seul je me suis rendu compte que le contact social est très important dans la vie d'une personne, ce à quoi je ne donnai pas forcément d'importance avant mon stage car je restai presque tout le temps dans ma chambre à Lille, sans parler à personne. Heureusement, j'ai pu accueillir mon ami Elian ce qui m'a permis d'expérimenter la vie en collocation, ce que j'ai pas mal apprécié même si mon côté solitaire me rattrapait parfois.

7.3.Conclusion générale

Je tiens encore à remercier mon tuteur Vivian de m'avoir encadré tout au long de mon stage dans cette ambiance très agréable. J'ai pu mettre en application toutes mes connaissances et en avoir de nouvelles. Ce fut une expérience professionnelle très valorisante et un tremplin pour mon avenir professionnel. J'espère pouvoir retenter l'expérience dans les années à venir.

Index

Bibliographie

Berlin Zuse Institute SCIP documentation [En ligne] // SCIP. - http://scip.zib.de/doc/html_devel/.

Buckys C++ Programming Tutorials [En ligne] // The New Boston. - https://www.thenewboston.com/videos.php?cat=16.

C++ Tutorials for Beginners [En ligne] // Cave of Programming Youtube Channel. - https://www.youtube.com/watch?v=1MKhigIml3E&list=PLmpc3xvYSk4wDCP5zjt2QQXe8-JGHa4Kt.

Gurobi Optimizaition Gurobi documentation [En ligne] // Gurobi. - http://www.gurobi.com/resources/documentation.

IBM IBM Knowledge center [En ligne] // IBM. - http://www-01.ibm.com/support/knowledgecenter/SSSA5P_12.2.0/ilog.odms.cplex.help/Content/Optimization/Documentation/CPLEX/_pubskel/CPLEX.html.

lpsolve Lpsolve documentation [En ligne] // Lpsolve. - http://lpsolve.sourceforge.net/5.5/.

Metals, Psi Psi Metals Brochure [En ligne] // Psi Metals. - http://www.psimetals.de/fileadmin/files/downloads/PSI_BT/Brochures/PSI_Metals_brochure.2014.english.pdf.

Metals, Psi Psi Metals History [En ligne] // Psi Metals. - http://www.psimetals.de/en/met-company/met-history/.

Nebra Mathieu Apprendre à programmer en C++ [En ligne] // OpenClassrooms. - http://openclassrooms.com/courses/programmez-avec-le-langage-c.

Psi Psi History [En ligne] // Psi. - http://www.psi.de/en/psi-group/history/.

Table des matières

Annexes

1. Annexes du projet SCIP

1.1.Fonctions principales du code de SCIP

1.1.1. Attributs de la classe du solveur SCIP

```
private :
    SCIPEnv m_env;
    SCIP_RETCODE m_status;
    std::vector<SCIPVar> m_scipVars;
    std::vector<SCIPCons> m_scipCons;
    SCIPSol m_solution;

    int m_isMip;
    double m_tolerance;
```

Ces attributs sont notamment la liste des variables et contraintes du problème ainsi que la variable m_env représentant ce problème.

1.1.2. Sélection du solveur SCIP dans MPlanner après lecture du fichier GsLpSolve.cfg

```
CLPBase* pLP = NULL;
    switch (m_pGlobalSolver-
>m_GlobalSolverConfig.m_LPConfiguration.lpSolver) {
    case cplexT:
        pLP = new CLPCplex();
        break;

    case gurobiT:
        pLP = new CLPGurobi();
        break;

    case scipT:
        pLP = new CLPScip();
        break;

    case lpsolveT:
        pLP = new CLPLpsolve();
        break;

    default:
        pLP = new CLPCoin();
        break;
    }
```

1.1.3. Création d'un problème

```
CLPScip::CLPScip()
{
    m_rowCount = 0;
    m_rowsPreAllocated = false;

    //Création du problème
    m_status = SCIPcreate(&m_env);

    //Charge les outils nécessaires à la résolution du problème
    m_status = SCIPincludeDefaultPlugins(m_env);

    m_tolerance = 0;

    m_isMip = false;

    //Créé un fichier de log
    SCIPsetMessagehdlrLogfile(m_env, "scip.log");
}
```

On créé le problème et on charge les outils de SCIP pour qu'il puisse le résoudre.

1.1.4. Destruction du problème

```
CLPScip::~CLPScip()
{
    //Suppression de toute les variable
    for (int i = 0; i < m_scipVars.size(); i++) {
        SCIPreleaseVar(m_env, &m_scipVars[i]);
    }

    //Suppression de toutes les contraintes
    for (int i = 0; i < m_scipCons.size(); i++) {
        SCIPreleaseCons(m_env, &m_scipCons[i]);
    }

    //Suppression du problème
    m_scipVars.clear();
    m_scipCons.clear();

    m_status = SCIPfree(&m_env);
}
```

On supprime dans un premier temps les variables, puis les contraintes et enfin le problème qui les contenait.

1.1.5. Ajout des variables

```
void CLPScip::setFunction(CLPFunction* function)
{
    assert(m_scipVars.size() == 0);

    m_lpFunction = function;
```

```
    // On spécifie le type de problème ici basique avec variable et
contrainte
    m_status = SCIPcreateProbBasic(m_env, "sciProblem");

    //Fonction à maximiser ou minimiser ?
    switch (function->getType()) {
    case lpMinFunction:
        m_status = SCIPsetObjsense(m_env, SCIP_OBJSENSE_MINIMIZE);
        break;

    case lpMaxFunction:
        m_status = SCIPsetObjsense(m_env, SCIP_OBJSENSE_MAXIMIZE);
        break;
    }

    assert(m_status == SCIP_OKAY);

    if (function->getNumCoefficients() == 0)
        return;

#if 0
    double* lbs = &function->getLowerBounds().at(0);
    double* ubs = &function->getUpperBounds().at(0);
    double* coefs = &function->getCoefficients().at(0);
    char* types = &function->getIntegers().at(0);
#endif

    for (int i = 0; i < function->getNumCoefficients(); ++i)
    {

        SCIPVar scipVar;
        SCIP_VARTYPE type;

        //Determine le type de variable
        switch(function->getIntegers().at(i)) {
    case 'C' :
        type = SCIP_VARTYPE_CONTINUOUS;
        break;

    case 'B' :
        type = SCIP_VARTYPE_BINARY;
        break;

    case 'I' :
        type = SCIP_VARTYPE_INTEGER;
        break;

    default:
        assert(false);
        break;

#if 0
    case 'S' :
        type = SCIP_VARTYPE_IMPLINT;
        break;

    case 'N' :
        type = SCIP_VARTYPE_IMPLINT;
        break;
#endif
        }
```

```
        // Création de la variable SCIP
        if (function->getVarNames().at(i) != "")
        {
            char varName[SCIP_MAXSTRLEN];
            (void) SCIPsnprintf(varName, SCIP_MAXSTRLEN - 1, function-
>getVarNames().at(i).c_str(), i);
            m_status = SCIPcreateVarBasic(m_env, &scipVar, varName,
function->getLowerBounds().at(i), function->getUpperBounds().at(i),
function->getCoefficients().at(i), type);
        }
        else
        {
            char varName[SCIP_MAXSTRLEN];
            (void) SCIPsnprintf(varName, SCIP_MAXSTRLEN - 1, "X%d", i);
            m_status = SCIPcreateVarBasic(m_env, &scipVar, varName,
function->getLowerBounds().at(i), function->getUpperBounds().at(i),
function->getCoefficients().at(i), type);
        }

        if((m_status = SCIPaddVar(m_env, scipVar)) != SCIP_OKAY)
        {
            assert(false);
        }
        else
        {
            m_scipVars.push_back(scipVar);
        }
    }

}
```

La fonction reçoit en paramètre un modèle de données de MPlanner que nous allons traduire pour SCIP. Chaque variable a plusieurs caractéristiques qu'il faut faire correspondre à celles définies par SCIP.

1.1.6. Ajouter une contrainte

```
int CLPScip::addConstraint(CLPConstraint* c)
{

    if (!m_rowsPreAllocated) //false
    {
        if (c->getNumCoefficients() <= 0)
            return 0;

        SCIPVar *vars = new SCIPVar[c->getNumCoefficients()];
        //double coefs = new double[c->getNumCoefficients()];

        for (int i = 0; i < c->getNumCoefficients(); ++i) {
            vars[i] = m_scipVars[c->getCoefficientColumns()[i]];
        }

        double *coeffs = &c->getCoefficients().at(0);

        int index = m_rowCount;
        SCIPCons constraint = 0;

        //Determine le type de contrainte
        switch (c->getType()) {
```

```
                case lpLessThanConstraint:
                    m_status = SCIPcreateConsBasicLinear(m_env,
&constraint, c->getName().c_str(), c->getNumCoefficients(), vars, coeffs, -
(SCIPinfinity(m_env)), c->getRhs());
                    break;

                case lpGreaterThanConstraint:
                    m_status = SCIPcreateConsBasicLinear(m_env,
&constraint, c->getName().c_str(), c->getNumCoefficients(), vars, coeffs,
c->getRhs(), SCIPinfinity(m_env));
                    break;

                case lpEqualToConstraint:
                    m_status = SCIPcreateConsBasicLinear(m_env,
&constraint, c->getName().c_str(), c->getNumCoefficients(), vars, coeffs,
c->getRhs(), c->getRhs());
                    break;
        }

        if ((m_status = SCIPaddCons(m_env, constraint)) != 1)
        {
            assert(false);
        }
        else
        {
        //Ajout de la contrainte
            m_scipCons.push_back(constraint);
            m_rowCount++;
        }

        delete [] vars;
    }
    else
    {

    }
    return getStatus();
}
```

Les contraintes aussi ont plusieurs caractéristiques qu'il faut traduire en code SCIP.

1.1.7. Ajouter toute les contraintes en même temps

```
int CLPScip::addConstraints(
                            int rcnt,
                            const double* rhs,
                            const char* sense,
                            const double* rngval,
                            char** rowname,
                            int numcoefs,
                            const int* rowlist,
                            const int* collist,
                            const double* vallist
                            )
{

    /*
    int rcnt, //number of constraints
```

```
    const double* rhs, //constraints's bounds
    const char* sense, //constraints'sense
    const double* rngval,
    char** rowname, //constraints's names
    int numcoefs, //nonzeros variables number
    const int* rowlist, //x index of coeffs
    const int* collist, //y index of coeffs
    const double* vallist //coeffs's table
    */

    int rowindex = -1, nbVals, constraintCounter = 0;

    int *beglist = new int[rcnt];
    double *newRhs = new double[rcnt];
    char * newSense = new char[rcnt];
    char ** newRowname = new char*[rcnt];

    //Création des tableau "sans trou" à partir des paramètres
    for (int i = 0; i < numcoefs; ++i) {
        int newrowindex = rowlist[i];
        if (newrowindex == rowindex)
            continue;

        //assert(newrowindex == rowindex + 1);

        rowindex = newrowindex;

        beglist[constraintCounter] = i;
        newRhs[constraintCounter] = rhs[rowindex];
        newSense[constraintCounter] = sense[rowindex];

        newRowname[constraintCounter] = rowname[rowindex];
        ++constraintCounter;
    }

    for (int i = 0; i < constraintCounter; ++i)
    {
        if(i < constraintCounter - 1)
            nbVals = beglist[i+1] - beglist[i];
        else
            nbVals = numcoefs - beglist[i];

        if(nbVals <= 0)
            continue;

        SCIPCons constraint = 0;

        SCIPVar *vars = new SCIPVar[nbVals];

        for (int j = beglist[i]; j < beglist[i] + nbVals; ++j) {
            vars[j - beglist[i]] = m_scipVars.at(collist[j]);
        }

        switch(newSense[i])
        {
        case 'L' :
            m_status = SCIPcreateConsBasicLinear(m_env, &constraint,
newRowname[i], nbVals, vars, &(const_cast<double*>(vallist)[beglist[i]]), -
(SCIPinfinity(m_env)), newRhs[i]);
            break;
        case 'G' :
```

```
            m_status = SCIPcreateConsBasicLinear(m_env, &constraint,
newRowname[i], nbVals, vars, &(const_cast<double*>(vallist)[beglist[i]]),
newRhs[i], SCIPinfinity(m_env));
            break;
        case 'E' :
            m_status = SCIPcreateConsBasicLinear(m_env, &constraint,
newRowname[i], nbVals, vars, &(const_cast<double*>(vallist)[beglist[i]]),
newRhs[i], newRhs[i]);
            break;
        default:

            break;
        }

        if(m_status != SCIP_OKAY)
        {
            assert(false);
        }

        if ((m_status = SCIPaddCons(m_env, constraint)) != 1)
        {
            assert(false);
        }
        else
        {
            m_scipCons.push_back(constraint);
            m_rowCount++;
        }

        delete [] vars;


    }

    delete [] beglist;
    delete [] newRhs;
    delete [] newSense;
    delete [] newRowname;

    return getStatus();
}
```

Nous pouvons voir que j'ai bouché les trous du tableau en testant chaque contrainte pour voir si elle est incorrecte.

1.1.8. Résolution d'un problème

```
bool CLPScip::solve(
                    std::tstring logfile /*= ""*/,
                    std::tstring problemFile /*= ""*/,
                    std::tstring solutionFile /*= ""*/,
                    ESimplexSolverType initialSolverType /*= NoSimplexT*/,
                    ESimplexSolverType reSolverType /*= NoSimplexT*/,
                    bool doLpPresolveInInitialSolve /*= true*/,
                    bool doLpPresolveInReSolve /*= true*/,
                    int scaling /*= 1*/,
                    double timeLimit /*= -1*/,
                    int numberOfThread /* = 0*/,
```

```
                    int nCPX_PARAM_BARCOLNZ /*=-1*/,
                    int nCPX_PARAM_BARITLIM /*=-1*/,
                    int nCPX_PARAM_BARALG /*=1*/,
                    int nCPX_PARAM_BARSTATALG /*=1*/,
                    int nCPX_PARAM_DEPIND /*=1*/,
                    int nCPX_PARAM_BARORDER /*=1*/,
                    bool writeStatistics /*= false */
                    )
{


    //changes the tolerance
#if 0
    if (m_tolerance) {
        SCIPchgFeastol(m_env, m_tolerance);
        SCIPchgLpfeastol(m_env, m_tolerance, FALSE);
        SCIPchgDualfeastol(m_env, m_tolerance);
        SCIPchgBarrierconvtol(m_env, m_tolerance);

    }
#else
    if (m_tolerance) {
        SCIPsetRealParam(m_env, "numerics/feastol", m_tolerance);
        SCIPsetRealParam(m_env, "numerics/lpfeastol", m_tolerance);
        SCIPsetRealParam(m_env, "numerics/dualfeastol", m_tolerance);
        SCIPsetRealParam(m_env, "numerics/barrierconvtol", m_tolerance);
    }

#endif



    if(timeLimit >= 0)
    {
        SCIPsetRealParam(m_env, "limits/time", timeLimit);
    }

#if 1
    if(numberOfThread > 0)
    {
        SCIPsetIntParam(m_env, "lp/threads", numberOfThread);
    }
#endif

#if 1
    if(scaling == 1)
    {
        SCIPsetBoolParam(m_env, "lp/scaling", TRUE);
    }
    else
    {
        SCIPsetBoolParam(m_env, "lp/scaling", FALSE);
    }

#endif

#if 1
    switch(reSolverType) {
    case PrimalSimplexT:
        m_status = SCIPsetCharParam(m_env, "lp/initalgorithm", 'p');
        break;
```

```
    case DualSimplexT:
        m_status = SCIPsetCharParam(m_env, "lp/initalgorithm", 'd');
        break;

#if 1
    case BarrierT:
        //barrier not supported soplex yet so we use dual mode
        //m_status = SCIPsetCharParam(m_env, "lp/initalgorithm", 'b');
        m_status = SCIPsetCharParam(m_env, "lp/initalgorithm", 'd');
        break;
#endif

    case CrossoverPrimalT:
        m_status = SCIPsetCharParam(m_env, "lp/initalgorithm", 'c');
        break;

    default: //automatic simplex
        m_status = SCIPsetCharParam(m_env, "lp/initalgorithm", 's');
        break;
    }
    assert(getStatus() == 0);
#endif

#if 0

    if(doLpPresolveInInitialSolve)
    {

    }

#endif

#if 1
    if(problemFile != "")
    {
        writeProblem(problemFile.c_str(), NULL);
    }
#endif

    m_status = SCIPpresolve(m_env);


    m_status = SCIPsolve(m_env);
    m_solution = SCIPgetBestSol(m_env);

    //char varname[SCIP_MAXSTRLEN];
    //_snprintf(varname, SCIP_MAXSTRLEN - 1, "%s", "problemee.txt");

    //For now we don't care about args


    if (solutionFile != "") {
        writeSolution(solutionFile.c_str());
        //m_status = SCIPprintBestSol(m_env, NULL, FALSE);
        //m_status = SCIPprintStatistics(m_env, NULL);

    }

    //if(writeStatistics)
    //{
```

```
    //  SCIPprintStatistics(m_env, NULL);
    //}

    return getStatus();
}
```

Ce code vérifie si les paramètres spécifient une configuration spéciale du solveur, écrit le fichier du problème vers le chemin spécifié dans `problemFile`, puis résout le problème grâce à `m_status = SCIPsolve(m_env);` et écrit enfin le fichier solution vers le chemin spécifié en paramètre `solutionFile`.

1.1.9. Ecrire le fichier problème

```
void CLPScip::writeProblem(const char* pathname, const char* fileType)
{
    m_status = SCIPwriteOrigProblem(m_env, pathname, fileType, 0);

}
```

Ce code utilise la fonction de SCIP pour écrire le fichier du problème.

1.1.10. Ecrire le fichier solution

```
void CLPScip::writeSolution(const TCHAR *pathname)
{
    char buf[200];
    buf[sizeof(buf) - 1] = '\0';

    FILE *str = _tfopen(pathname, _T("w"));
    if (!str) {
        _snprintf(buf, sizeof(buf) - 1, "Can't create file: '%s'.",
t2a(pathname).c_str());
        GlobalMessage(buf);
        return;
    }


    _snprintf(buf, sizeof(buf) - 1, "Objective value :\t%f\n\n",
getPrimalObjectiveValue());
    fprintf(str, buf);
    for (int i = 0; i < m_lpFunction->getNumCoefficients(); i++) {
        _snprintf(buf, sizeof(buf) - 1, "%s : \t%f\n", m_lpFunction-
>getVarNames().at(i).c_str(), getPrimalVariableValue(i));
        fprintf(str, buf);
    }
    fclose(str);


}
```

Ce code récupère les valeurs de chaque variable et les écrits dans le fichier solution.

1.1.11. Code permettant à MPlanner de comprendre les code erreurs de SCIP

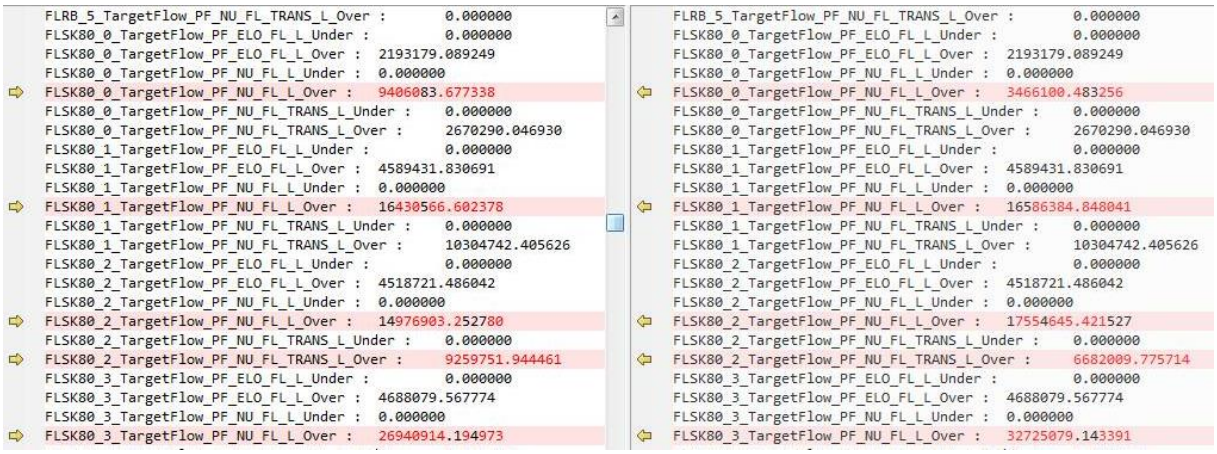
```
int CLPScip::getStatus()
{
    if(m_status == SCIP_OKAY) //SCIP_OKAY est égale à 1
    {
        return 0;
    }
    else
    {
        return 1;
    }
    //return 0;


}
```

Ce petit bout de code n'a été trouvé qu'à la fin du stage et suffisait à générer la solution avec la fonction dumpSolution() de MPlanner plutôt que writeSolution() de SCIP. Comme les deux fonctions sont utilisées, il était important pour moi de trouver l'erreur que j'ai pu corriger grâce à ce petit bout de code.

1.2.Comparaison de solution

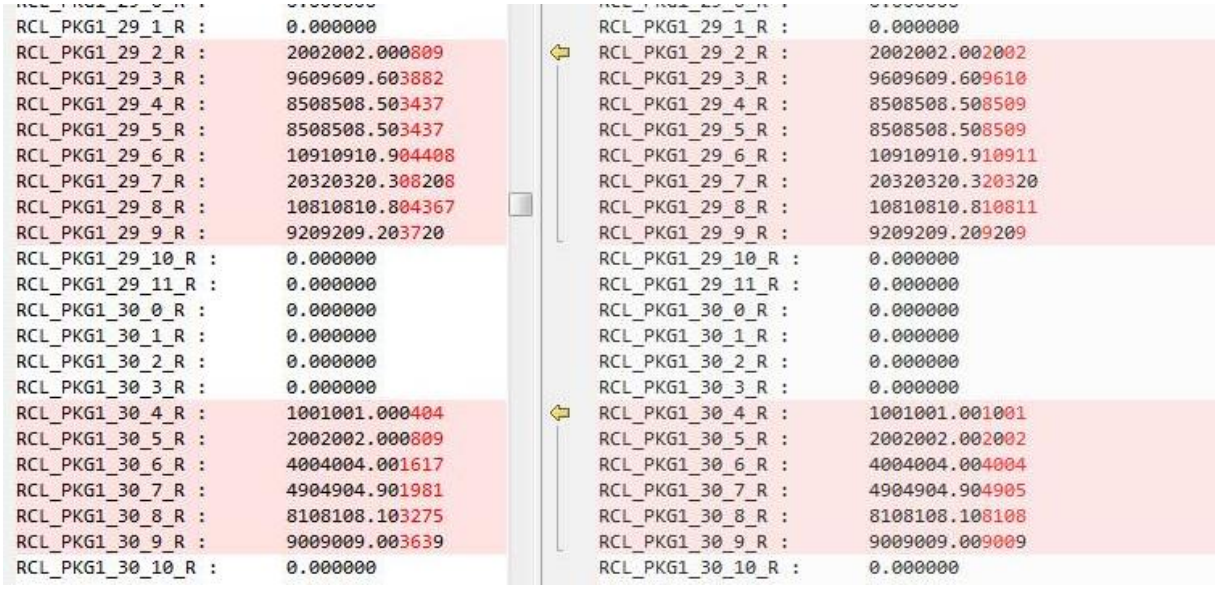
Lorsque l'on compare deux solutions, on a tendance à regarder le « score » de la solution. Plus ce score est haut, plus la solution est bonne. Et dans presque tous les tests effectués, le score de SCIP est très proche de celui de Cplex.

1.2.1. Problème à 15000 variables et 14000 contraintes



On voit qu'il y a certaines différences entre les solutions mais aussi des valeurs identiques. Mais même s'il y'a ces différences, le score de SCIP (à droite) et de Cplex (à gauche) se rapproche :

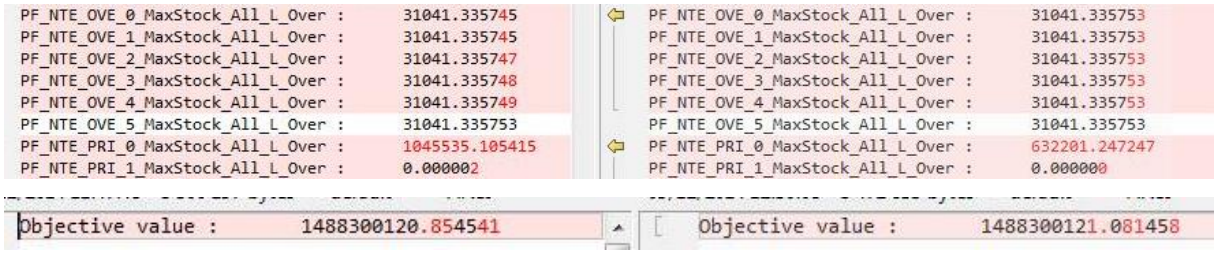
1.2.2. Problème à 20000 variables et 12000 contraintes



On voit que SCIP (à droite) et Cplex (à gauche) ont la même solution à quelques virgules près, ce qui est négligeable. Voici les scores :

Encore une fois ils sont très proches et cela prouve que SCIP a bien été implémenté sans erreur.

1.2.3. Problème à 106500 variables et 94000 contraintes

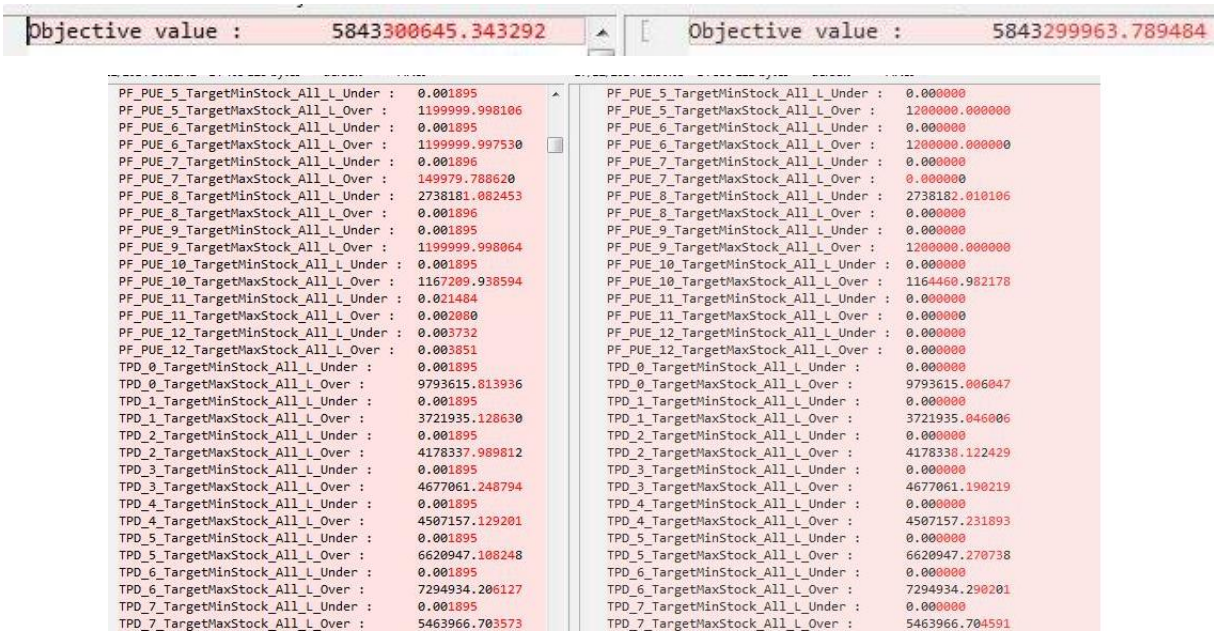


On constate encore une fois que les solutions et les scores sont quasiment les mêmes.

1.2.4. Problème à 370000 variables et 83000 contraintes

On constate que sur ce problème les scores sont quasiment les mêmes.

1.2.5. Problèmes à 254000 variables et 206000 contraintes



On remarque que SCIP (à droite) a tendance à arrondir au degré supérieur. Cependant c'est négligeable et l'on voit bien que même sur de gros problèmes, SCIP arrive à donner, certes après un certain temps, une solution identique à celle de Cplex.

2. Annexes du projet lpsolve

2.1.Fonctions principales de lpsolve

2.1.1. Attributs de la classe du solveur lpsolve

```
private:

    lprec *m_env;
    int m_status;
    double m_tolerance;
```

La variable `lprec *m_env;` représente le problème à créer.

2.1.2. Détruire le problème

```
CLPLpsolve::~CLPLpsolve()
{
    if(m_env != NULL)
    {
        delete_lp(m_env);
    }

}
```

On supprime simplement le problème avec la fonction delete_lp(m_env).

2.1.3. Créer le problème et ajouter une variable

```
void CLPLpsolve::setFunction(CLPFunction* function)
{
    m_status = 0;

    m_lpFunction = function;
    const int nVars = function->getNumCoefficients();

    // Creates an empty problem with getNumCoefficients variables
    m_env = make_lp(0, nVars);

    if(m_env == NULL)
    {
        m_status = 1;
    }

    REAL * row= new REAL[nVars + 1];

    //set variables names
    for (int i = 0; i < nVars; ++i)
    {
        int lpIndex = i + 1;
```

```
        row[lpIndex] = function->getCoefficients().at(i);



        //Determines the type
        switch(function->getIntegers().at(i)) {
            case 'C' :
                m_status = set_unbounded(m_env, lpIndex);
                break;

            case 'B' :
                m_status = set_binary(m_env, lpIndex, TRUE);
                break;

            case 'I' :
                m_status = set_int(m_env, lpIndex, TRUE);
                break;

            case 'S' :
                m_status = set_semicont(m_env, lpIndex, TRUE);
                break;

            default:
                assert(false);
                break;
        }

        //Sets upper bound and lower bound
        set_upbo(m_env, lpIndex, function->getUpperBounds().at(i));
        set_lowbo(m_env, lpIndex, function->getLowerBounds().at(i));
        set_col_name(m_env, lpIndex, const_cast<char*>(function-
>getVarNames().at(i).c_str()));

    }

    //set_obj_fnex(m_env, nVars, row, colno);
    set_obj_fn(m_env, row);

    // Set the type of the problem (Min or Max)
    switch (function->getType()) {
    case lpMinFunction:
        set_minim(m_env);
        break;

    case lpMaxFunction:
        set_maxim(m_env);
        break;
    }

    if(!set_add_rowmode(m_env, TRUE))
    {
        m_status = 1;
    }

    delete [] row;

}
```

2.1.4. Ajouter une contrainte

```
int CLPLpsolve::addConstraint(CLPConstraint* c)
{
    if (!m_rowsPreAllocated) //false
    {
        if (c->getNumCoefficients() <= 0)
            return 0;

        REAL * row = new REAL[c->getNumCoefficients()];
        int * colno = new int[c->getNumCoefficients()];
        //double coefs = new double[c->getNumCoefficients()];

        for (int i = 0; i < c->getNumCoefficients(); ++i) {
            row[i] = c->getCoefficients().at(i);
            colno[i] = (c->getCoefficientColumns().at(i)) + 1;
        }

        char sense;
        switch (c->getType()) {
        case lpLessThanConstraint:
            sense = LE;
            break;

        case lpGreaterThanConstraint:
            sense = GE;
            break;

        case lpEqualToConstraint:
            sense = EQ;
            break;
        }

        m_status = add_constraint(m_env, row, sense, c->getRhs());

        ++m_rowCount;
        m_status = set_row_name(m_env, m_rowCount, const_cast<char*>(c-
>getName().c_str()));

        delete [] row;
        delete [] colno;

    }

    return getStatus();
}
```

2.1.5. Ajouter toutes les contraintes en même temps

```
int CLPLpsolve::addConstraints(
                                int rcnt,
                                const double* rhs,
                                const char* sense,
                                const double* rngval,
                                char** rowname,
                                int numcoefs,
```

```
                                const int* rowlist,
                                const int* collist,
                                const double* vallist
                                )
{

    /*
    int rcnt, //number of constraints
    const double* rhs, //constraints's bounds
    const char* sense, //constraints'sense
    const double* rngval,
    char** rowname, //constraints's names
    int numcoefs, //nonzeros variables number
    const int* rowlist, //y index of coeffs
    const int* collist, //x index of coeffs
    const double* vallist //coeffs's table
    */

    int rowindex = -1, nbVals, constraintCounter = 0;
    int *beglist = new int[rcnt];
    int * cols = new int[numcoefs];
    double *newRhs = new double[rcnt];
    char * newSense = new char[rcnt];
    char ** newRowname = new char*[rcnt];

    for (int i = 0; i < numcoefs; ++i) {

        int newrowindex = rowlist[i];
        if (newrowindex == rowindex)
            continue;

        rowindex = newrowindex;

        beglist[constraintCounter] = i;
        newRhs[constraintCounter] = rhs[rowindex];
        newSense[constraintCounter] = sense[rowindex];

        newRowname[constraintCounter] = rowname[rowindex];
        ++constraintCounter;
    }

    for(int i = 0; i < constraintCounter; ++i)
    {
        if(i < constraintCounter - 1)
            nbVals = beglist[i+1] - beglist[i];
        else
            nbVals = numcoefs - beglist[i];

        if(nbVals <= 0)
            continue;

        for (int j = beglist[i]; j < beglist[i] + nbVals; ++j) {
            cols[j - beglist[i]] = collist[j] + 1;
        }

        char senseV;
        switch (newSense[i]) {
        case 'L' :
            senseV = LE;
            break;
```

```
        case 'G' :
            senseV = GE;
            break;

        case 'E' :
            senseV = EQ;
            break;
        }
        m_status = add_constraintex(m_env, nbVals,
&(const_cast<double*>(vallist)[beglist[i]]), cols, senseV ,newRhs[i]);

        m_status = set_row_name(m_env, i + 1, newRowname[i]);
    }

    delete [] beglist;
    delete [] cols;
    delete [] newRhs;
    delete [] newSense;
    delete [] newRowname;


    return getStatus();
}
```

2.1.6. Résoudre le problème

```
bool CLPLpsolve::solve(
                      std::tstring logfile /*= ""*/,
                      std::tstring problemFile /*= ""*/,
                      std::tstring solutionFile /*= ""*/,
                      ESimplexSolverType initialSolverType /*=
NoSimplexT*/,
                      ESimplexSolverType reSolverType /*= NoSimplexT*/,
                      bool doLpPresolveInInitialSolve /*= true*/,
                      bool doLpPresolveInReSolve /*= true*/,
                      int scaling /*= 1*/,
                      double timeLimit /*= -1*/,
                      int numberOfThread /* = 0*/,
                      int nCPX_PARAM_BARCOLNZ /*=-1*/,
                      int nCPX_PARAM_BARITLIM /*=-1*/,
                      int nCPX_PARAM_BARALG /*=1*/,
                      int nCPX_PARAM_BARSTATALG /*=1*/,
                      int nCPX_PARAM_DEPIND /*=1*/,
                      int nCPX_PARAM_BARORDER /*=1*/,
                      bool writeStatistics /*= false */
                      )
{
    if(problemFile != "")
    {
        writeProblem(problemFile.c_str(), NULL);
    }

    set_add_rowmode(m_env, FALSE);


#if 1
    switch(reSolverType) {
    case PrimalSimplexT:
        set_simplextype(m_env, SIMPLEX_PRIMAL_PRIMAL);
```

```
        break;

    case DualSimplexT:
        set_simplextype(m_env, SIMPLEX_DUAL_DUAL);
        break;

    default: //SIMPLEX_DUAL_PRIMAL
        set_simplextype(m_env, SIMPLEX_DEFAULT);
        break;
    }
#endif

    m_status = ::solve(m_env);

    m_status = getStatus();



    if (solutionFile != "") {
        writeSolution(solutionFile.c_str());
    }

    return true;
}
```

2.1.7. Ecrire le fichier du problème

```
void CLPLpsolve::writeProblem(const char* pathname, const char* fileType)
{
    char path[200];
    _snprintf(path, sizeof(path) - 1, pathname);
    write_lp(m_env, path);
}
```

2.1.8. Ecrire le fichier de la solution

```
void CLPLpsolve::writeSolution(const TCHAR *pathname)
{
    char buf[200];
    buf[sizeof(buf) - 1] = '\0';

    FILE *str = _tfopen(pathname, _T("w"));
    if (!str) {
        _snprintf(buf, sizeof(buf) - 1, "Can't create file: '%s'.",
t2a(pathname).c_str());
        GlobalMessage(buf);
        return;
    }


    _snprintf(buf, sizeof(buf) - 1, "Objective value :\t%f\n\n",
getPrimalObjectiveValue());
    fprintf(str, buf);
    for (int i = 0; i < m_lpFunction->getNumCoefficients(); i++) {
        _snprintf(buf, sizeof(buf) - 1, "%s : \t%f\n", m_lpFunction-
>getVarNames().at(i).c_str(), getPrimalVariableValue(i));
        fprintf(str, buf);
```

```
    }
    fclose(str);
}
```