MÉMOIRE

présenté en vue d'obtenir

LE DIPLÔME D'INGÉNIEUR I.I.E.

Rapport Intermédiaire

Thomas COFFIN

Optimisation de Production : Étude de cas

Minimisation de la consommation électrique d'une fonderie

Directeur du stage : M. Philippe MORIGNOT

Conservatoire National des Arts et Métiers

INSTITUT D'INFORMATIQUE D'ENTREPRISE

FICHE SIGNALÉTIQUE

Mémoire d'Ingénieur I.I.E.

Optimisation de Production : Étude de cas

Minimisation de la consommation électrique d'une fonderie

Auteur : Thomas COFFIN

Directeur de recherche : M. Philippe MORIGNOT, Ingénieur, chef de projet, consultant à Pacte Novation

Descriptif : une étude théorique sur la représentation d'un problème d'ordonnancement au moyen de la programmation par contraintes a permis la réalisation d'une application d'optimisation dédié à la résolution d'un problème de gestion de production.

RESUME

L'objectif de ce stage est la réalisation d'un outil d'optimisation adapté à la réduction de la consommation électrique d'une fonderie. Cet outil est fondé sur un modèle mathématique du problème de gestion de production.

Une étude théorique sur l'ordonnancement avec affectation des ressources a été menée, et l'utilisation de la programmation par contraintes, au moyen de la bibliothèque C++ dédiée à l'ordonnancement par contraintes Ilog Scheduler, a été décidée.

L'étape suivante a été de définir les principes du modèle pour que celui-ci soit à la fois représentatif de la réalité et performant. La modélisation a porté sur la représentation des différentes activités constituant le processus de production, et sur la fonction de coût utilisée.

Plusieurs critères de qualité pour le modèle ont été définis, des recherches pratiques ont été effectuées pour augmenter progressivement le modèle et optimiser sa résolution, à travers l'amélioration de la modélisation par contraintes du problème, et l'optimisation de la procédure de résolution. Pour cela nous avons exploité au mieux les spécificités du problème réel, ainsi que le fonctionnement de la programmation par contraintes.

Les résultats obtenus sont encourageants, les performances de la résolution du problème ayant été considérablement améliorées au fil du développement. De nombreuses évolutions futures du moteur d'optimisation sont prévues. La réalisation d'heuristiques adaptées au problème pourra encore augmenter les performances de sa résolution.

Le travail de recherche effectué et la maquette réalisée seront utilisés lors du développement d'un projet plus global de gestion des fonderies.

Les principaux ouvrages qui ont permis d'effectuer ces recherches sont :

- LE PAPE Claude. « Utilisation de la programmation par contraintes en planification et ordonnancement : principes et applications ». In Université Paris I. *Site des Journées Franciliennes de Recherche Opérationnelle*, [En ligne].
  panoramix.univ-paris1.fr/CERMSEM/soutif/JFRO/fichiers/c_le_pape.pdf
  (Page consultée le 18 janvier 2002)
- BAPTISTE Philippe, « Une étude théorique et expérimentale de la propagation des contraintes de ressources ». In Université de Technologie de Compiègne. Site de Philippe Baptiste, [En ligne]
  http://www.hds.utc.fr/~baptiste/phd.pdf (Page consultée en février 2002)

REMERCIEMENTS

Je tiens à remercier l'ensemble du personnel de Pacte Novation, qui m'a permis d'effectuer mon stage et de suivre parallèlement mon DEA, dans de très bonnes conditions.

Je remercie tout particulièrement :

- Philippe Morignot, pour m'avoir permis d'intégrer son équipe et m'avoir encadré tout au long de ces sept mois,
- Sylviane Roux, ingénieur système, et Mérida Girondin, pour leur disponibilité,
- Laure, Louis, Mohamed et Philippe, pour leur bonne humeur face à l'adversité,
- Vanessa Doucy, des relations humaines, pour son accueil,
- Christian Tora, PDG de Pacte Novation, pour nous avoir permis de regarder les défaites de l'équipe de France.

Table des matières

Chapitre 1 : Objet de la Recherche

Chapitre 2 : Recherches Théoriques

Chapitre 3 Recherches Pratiques - Implémentation du Modèle

Chapitre 4 Conclusions sur le Travail de Recherche

CHAPITRE 1
OBJET DE LA RECHERCHE

1.1 Encadrement et environnement

1.1.1 Établissement d'accueil

1.1.1.1 Description générale

Le stage se déroule au sein de Pacte Novation. Fondée en 1994 par Bruno Gaudinat et Christian Tora, deux ingénieurs de formation, Pacte Novation est une société d'ingénierie logicielle pluridisciplinaire, intervenant dans des secteurs d'activités très variés comme le transport, la banque et la finance, les télécommunications, l'énergie, l'industrie et le tertiaire. La société Pacte Novation compte aujourd'hui une centaine d'employés.

Les activités de Pacte Novation sont fortement centrées sur les technologies orientées objets, les I.H.M. (Interfaces Homme-Machine) avec une spécialisation en ergonomie du logiciel, et enfin les outils d'aide à la décision, domaine dans lequel s'inscrit le projet qui fait l'objet de ce stage.

Les systèmes d'aide à la décision ont pour objectif de mettre en valeur un savoir-faire pointu possédé par le client. Il s'agit d'une véritable coopération entre l'utilisateur, qui conserve la décision finale, une interface intuitive et des processus complexes producteurs de forte valeur ajoutée. Cela peut mettre en œuvre des moyens relatifs au traitement combinatoire tels que l'optimisation combinatoire, la programmation par contraintes, l'optimisation locale, l'I.A. (Intelligence Artificielle), mais aussi relatifs à l'expertise du domaine : le recueil de connaissances, l'utilisation d'heuristiques, le management de connaissances, les systèmes experts. Le raisonnement combinatoire permet notamment de résoudre des problèmes d'ordonnancement, de planification ou d'optimisation de ressources.

1.1.1.2 Informations pratiques

- Établissement d'accueil :
Pacte Novation
2 rue du docteur Lombard
92441 Issy-les-Moulineaux Cedex
Tél. : 01.45.29.06.06
Fax : 01.45.29.25.00
Web : http://www.pactenovation.fr

- Directeur de stage :
Philippe Morignot
philippe.morignot@pactenovation.fr
Tél. 01.45.29.06.06

1.1.2 Environnement technique du stage

Les solutions au problème posé sont implémentées en langage C++, et au moyen des bibliothèques de programmation par contraintes d'Ilog : Ilog Solver et Ilog Scheduler, regroupées dans l'environnement Ilog Concert.

Pacte Novation travaille dans un environnement Windows, et utilise largement les produits Microsoft : on utilise donc l'environnement de développement MS Visual C++, ainsi que la suite Office.

1.1.3 Intervenants

L'élève suivant, en parallèle avec le stage de fin d'études de l'I.I.E., le D.E.A. « Informatique et Recherche Opérationnelle » de l'école doctorale ÉDITE, le stage a été effectué à temps partiel, jusqu'à la fin des cours du D.E.A., à la fin du mois d'avril. Le stage a commencé le 7 janvier 2002, et se terminera le 31 juillet, afin de compenser ces journées de cours.

Il ne s'agit pas d'un travail en équipe, le stagiaire est seul sur ce projet, mais bénéficie de l'assistance de Philippe Morignot, qui possède une grande expérience dans ce domaine, aussi bien pour la théorie que pour la mise en pratique des solutions au moyen des produits développés par Ilog. De plus Ilog fournit une assistance technique par téléphone, ainsi qu'une liste de diffusion par courrier électronique permettant l'entraide entre développeurs usagers des technologies d'optimisation Ilog. Le projet est réalisé à partir d'un cahier des charges détaillé existant, et d'un exemple de données, mais il n'est pas exclu de contacter le client du projet pour obtenir des précisions.

1.2 Présentation du problème d'optimisation

1.2.1 Le problème réel

1.2.1.1 Objectif du stage

Pour réaliser des pièces moulées, une fonderie utilise des cuves qui chauffent du métal brut puis le déversent, liquéfié, dans des moules. Ces cuves sont électriques et leur consommation est donc soumise à tarification, variable au cours d'une même journée. La consommation électrique est la principale source de coût pour une fonderie, en plus du métal brut lui-même.

Le but du stage est la réalisation du prototype d'une application d'optimisation adapté à ce problème. Ce moteur sera un composant d'un outil générique qu'EDF souhaite développer pour répondre à ses besoins, notamment l'estimation des puissances minimales à souscrire et des réductions de factures, ainsi qu'à ceux de ses clients fondeurs : l'optimisation et l'aide à la conduite des fours. C'est cette dernière partie à laquelle nous nous sommes efforcés d'apporter une solution.

L'objectif est de minimiser la consommation électrique d'une fonderie, en utilisant au mieux les fours, tout en respectant absolument les contraintes sur le nombre de pièces à produire : on ne peut pas autoriser de retard, ni ne produire qu'une partie des pièces. Outre l'optimisation du temps de production, la source la plus importante de réduction des coûts est l'exploitation au mieux des variations de tarifs au cours de la journée proposées par EDF, et, depuis la déréglementation du marché de l'électricité, par ses concurrents. En effet il peut s'avérer moins coûteux de fondre le métal la nuit, lorsque l'énergie électrique est moins onéreuse, et maintenir en fusion le métal jusqu'à ce qu'une chaîne de moulage soit disponible.

1.2.1.2 Fonctionnement de la fonderie

Le coulage d'une série de pièces comporte plusieurs étapes (figure 1) : le métal brut est chargé dans les fours, fondu, puis transmis par une poche de prélèvement directement jusqu'à la chaîne de coulée, ou jusqu'à un four de stockage ou de coulée. Le processus doit être modélisé du chargement du four jusqu'au coulage des pièces ; on ne s'intéresse pas à ce qui se passe avant (approvisionnement des chargeuses en métal brut) ou après (conditionnement et livraison des pièces).

Chaque élément de la chaîne de production peut être présent en un ou plusieurs exemplaires, et être affecté précisément à un ensemble d'éléments qui sont la source ou la destination de son activité : par exemple une poche de prélèvement peut n'être utilisable qu'avec deux fours particuliers, et ne se vider que dans le four

de stockage ou les fours de coulée. Certains éléments sont facultatifs, et sont représentés en pointillé sur le schéma. Il n'existe que deux différents types de four à induction : moyennes ou basses fréquences. Les fours à basses fréquences ont une plus grande capacité, les fours à moyennes fréquences sont plus puissants. De plus, les fours à basses fréquences ont un niveau minimum : la quantité de métal dans ce four ne peut être en dessous de ce niveau que si le four est en train d'être vidé ou rempli.

Les fonderies peuvent utiliser d'autres types de fours, notamment des fours ne fonctionnant pas à l'électricité, mais ces configurations ne sont pas abordées, le cahier des charges (émis par EDF) ne les caractérisant pas.

La modélisation du problème doit être très souple : en effet, on ne cherche pas à réaliser une application d'optimisation pour une fonderie en particulier, mais une application qui puisse s'adapter facilement à toute configuration de fonderie. Or les degrés de liberté sont nombreux, le nombre d'équipements de chaque type ainsi que les contraintes de production variant d'une usine à l'autre.

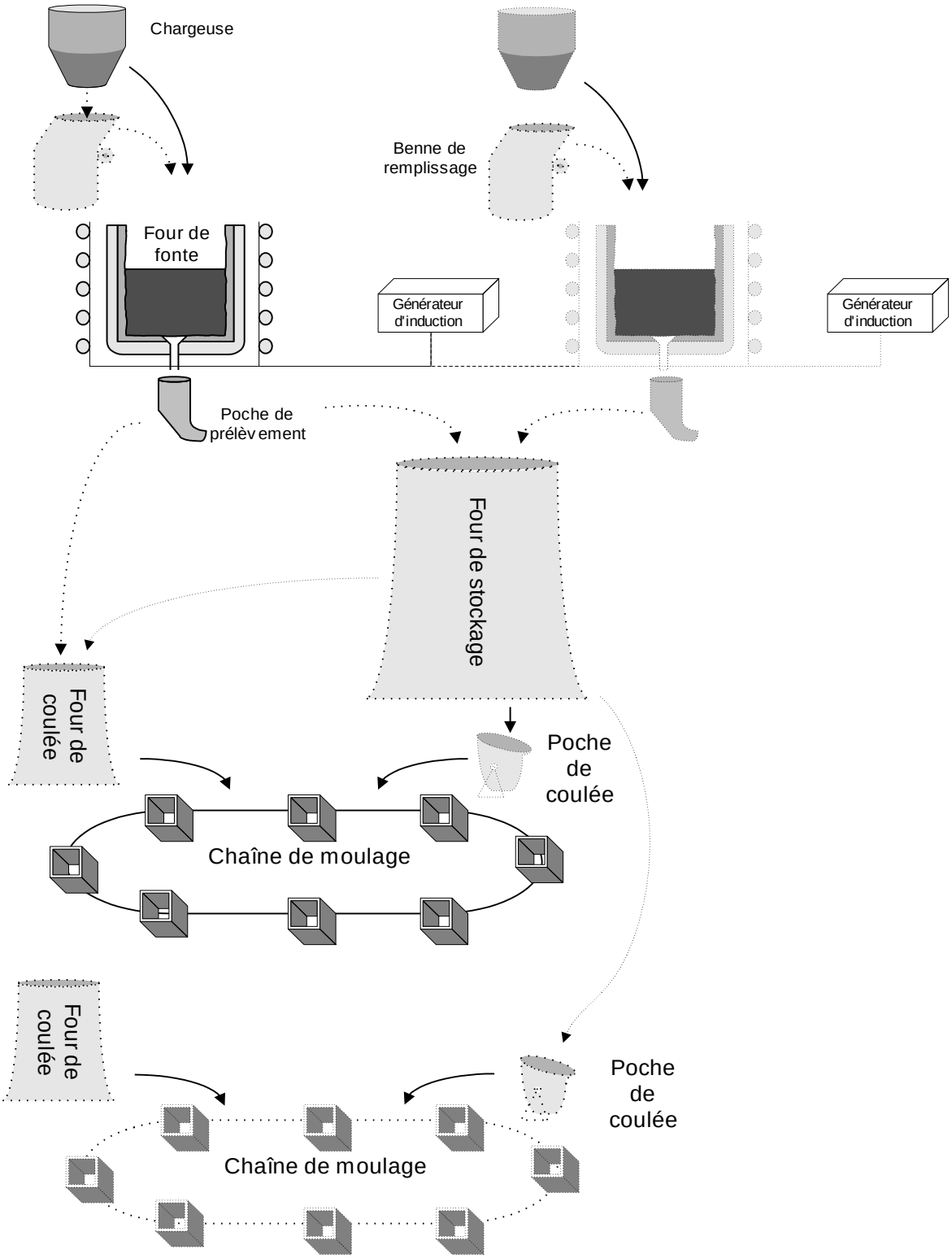

Figure 1 : processus de production

1.2.2 Le problème mathématique associé

1.2.2.1 Définition du problème d'ordonnancement

On trouve plusieurs définitions d'un problème d'ordonnancement dans la littérature :

« Un problème d'ordonnancement consiste à déterminer les dates d'exécution d'activités qui utilisent une ou des quantités connues d'un ensemble donné de ressources dont les capacités sont limitées » [Baptiste 98 p15]. Cette définition est celle d'un ordonnancement pur : il ne s'agit que de placer les activités sur un axe temporel. Les ressources demandées sont déjà connues, la quantité demandée aussi, même si elles peuvent varier au cours du temps. Un exemple typique de ce type de problème est le Job-Shop, décrit plus bas.

« L'ordonnancement concerne l'affectation de ressources limitées aux tâches dans le temps. C'est un processus de prise de décision dont le but est d'optimiser un ou plusieurs objectifs. » [Pinedo 95]. La définition de Michael Pinedo met plus l'accent sur l'allocation de ressources. Dans le problème d'allocation de ressources pur, les dates de début et de fin des activités sont déjà connues, le problème est de garantir que les ressources disponibles à chaque instant permettent de satisfaire la demande. Un exemple classique de problème d'allocation de ressources pur est l'affectation de personnel pour des avions ou des trains.

Jacques Carlier et Philippe Chrétienne proposent [Carlier 88] une définition plus générale : « Ordonnancer, c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution. [...] Les différentes données d'un problème d'ordonnancement sont les tâches, les contraintes potentielles, les ressources, et la fonction économique. » Dans le même ouvrage, Carlier et Chrétienne ajoutent qu'on dit qu'on a affaire à un problème d'ordonnancement lorsque : "on doit programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution [...]. Les tâches sont le dénominateur commun des problèmes d'ordonnancement, leur définition n'est ni toujours immédiate, ni toujours triviale".

Un problème d'ordonnancement peut être non-préemptif, si les activités sont exécutées d'un seul tenant, ou préemptif, si les activités peuvent être interrompues pour laisser s'exécuter d'autres activités.

On considère généralement que les ressources sont renouvelables (machine, fichier, processeur, homme, etc.), i.e. elles redeviennent disponibles après leur utilisation. Parmi les ressources renouvelables, on distingue les ressources disjonctives, qui ne peuvent exécuter qu'une opération à la fois et les ressources

cumulatives, qui peuvent exécuter un nombre limité d'opérations simultanément. Toutefois il est possible de définir des modèles de ressources plus complexes, ou non renouvelables.

Considérons deux problèmes types d'ordonnancement : le Job-Shop (JSSP), et le problème de gestion de projet à contraintes de ressources (RCPSP).

Les données du problème du Job-Shop sont *m* machines, *n* jobs, une date d'échéance *D*. Chaque job comporte une liste d'activités. Chaque activité possède un temps d'exécution entier, et une machine sur laquelle elle doit s'exécuter. La question est de savoir s'il existe un ordonnancement non-préemptif des activités qui respecte ces contraintes.

Le problème de gestion de projet à contraintes de ressources considère un ensemble de ressources de capacités, un ensemble d'activités non-interruptibles, un graphe de précédence entre ces contraintes, une date d'échéance, et les quantités de chaque ressource utilisées par chaque activité. Là aussi on recherche un ordonnancement non-préemptif qui satisfasse ces contraintes.

Le problème du Job-Shop et le problème de gestion de projet à contraintes de ressources sont NP-Complets au sens fort [Garey 79].

1.2.2.2 Spécificité du problème considéré

Les données de sortie de l'outil d'optimisation, spécifiées dans l'exemple de données fourni par EDF, sont « la liste des ordres qui doivent être préconisés aux opérateurs pour couler les pièces souhaitées en consommant le minimum d'électricité, à savoir :

- Les instants auxquels les fours doivent être chargés de matière et les quantités de matière à charger,
- Les instants auxquels les plots de puissance doivent être enclenchés,
- Les instants auxquels la matière peut être prélevée dans chacun des fours car arrivée à la température de 1470°C. »

Il s'agit donc bien d'un ordonnancement : les variables de décision du problème sont des dates de début d'activité et des quantités de ressources à allouer. À ces variables s'ajoutent, ce qui n'est pas précisé ci-dessus, le choix des ressources à affecter à chaque étape du processus : dans quel four effectuer chaque fonte, quand utiliser le four de stockage, vers quelle chaîne acheminer le métal fondu.

Ce problème est un problème d'ordonnancement mixte : il s'agit à la fois de placer différentes tâches à des dates précises, de choisir et d'allouer les ressources nécessaires à chaque activité. Chaque activité peut éventuellement utiliser plusieurs ressources équivalentes, le degré de liberté existe donc pour décider quelles activités

exécuter, à quelle date les exécuter, et pour décider quelles ressources rendre disponible pour chaque activité.

Le problème est globalement non-préemptif, seules les activités de fonte pourront éventuellement être interrompues.

Le problème cumule les difficultés du Job-Shop et du RCPSP, il est donc aussi NP-Complet : on peut représenter toute instance du Job-Shop ou du RCPSP comme une instance de notre problème.

Dans la fonderie, les activités principales sont la fonte du métal et le coulage des pièces, mais on doit créer les activités intermédiaires participant au processus général, comme le chargement des fours, ou le transfert du métal d'un point à un autre de la chaîne de production. Les ressources sont plus délicates à définir.

Notre problème n'est pas un problème de décision, mais un problème d'optimisation : la difficulté n'est pas de trouver une solution admissible, mais de trouver une bonne solution admissible, sinon la meilleure. Le but n'est pas de couler un maximum de pièces dans une plage de temps donnée, mais de placer au mieux les différentes tâches participant au processus de coulage des pièces, afin de minimiser le coût. Le nombre de pièces, et donc le nombre approximatif de tâches qui seront nécessaires, est déterminé par l'utilisateur, responsable de la fonderie, et qui possède une expertise sur la faisabilité du problème. Toutefois, s'il n'y a pas de solution admissible, le moteur d'optimisation le signalera, mais n'essaiera pas de tronquer, de lui-même, le nombre de pièces à couler par série, et ne supprimera pas de série de pièces.

Le problème étant un problème d'optimisation, il faut définir la fonction de coût qu'il faut minimiser. L'ordonnancement recherché définit un processus de production, c'est donc le coût réel de la production que l'on cherche à minimiser. Il n'est toutefois pas question de prendre en compte tous les coûts de l'usine, et nous nous en tiendrons au principal poste de dépense : l'énergie électrique consommée lors du processus, et plus particulièrement l'énergie consommée par les fours, la consommation du reste de la production étant négligeable par rapport à cette dépense d'énergie. L'autre principale source de coût pour une fonderie est comme nous l'avons dit le métal brut, mais il n'est pas possible d'optimiser ce critère.

Le coût de la consommation électrique d'une fonte est égal à l'énergie fournie par la fonte multipliée par le tarif en vigueur à cet instant. Si la puissance est constante au cours de la fonte, le produit de la durée de la fonte par le tarif restera proportionnel au coût. Le tarif étant variable au cours de la journée, et les fontes pouvant durer plusieurs heures, on ne peut se contenter d'utiliser le tarif en vigueur à un instant donné de la fonte, il faut que la variation de tarif au cours d'une même fonte soit pris en compte. Pour cela, on considère la fonction tarifaire *tarif(t)*, qui

donne le tarif valable à chaque instant. Le coût d'une fonte est alors l'intégrale de cette fonction sur la période d'exécution de la fonte :

$$coût(fonte_{(i)}) = \int_{début_{(i)}}^{fin_{(i)}} tarif(t).dt$$

Un ordonnancement comporte plusieurs fontes, le coût total du processus est donc la somme des coûts occasionnés par chaque fonte :

$$coût(processus) = \sum_{i \in fontes} coût(fonte_i)$$

1.3 État de l'art en programmation par contraintes

1.3.1 Définition

Un problème de satisfaction de contraintes (Constraint Satisfaction Problem, ou CSP) peut être défini ainsi : étant donné d'une part un ensemble de variables, chaque variable possédant un ensemble de valeurs possibles, et d'autre part un ensemble de contraintes entre ces variables, on cherche à savoir s'il existe une affectation de valeurs pour chaque variable qui satisfasse toutes les contraintes. Les contraintes peuvent être explicites, si les tuples de valeurs satisfaisant la contrainte sont enregistrés dans une base de donnée, ou implicite, si un calcul sur les variables est nécessaire pour déterminer si la contrainte est vérifiée.

1.3.2 La programmation par contraintes

1.3.2.1 Généralités

La programmation par contraintes, née il y a une trentaine d'années, est un domaine créé au croisement de la recherche opérationnelle (et notamment de la programmation linéaire), de la programmation logique, et de l'intelligence artificielle. Cette technologie permet de dissocier la représentation du problème, c'est-à-dire l'ensemble des contraintes spécifiées par l'utilisateur, de sa résolution, qui est à la charge du système. De plus, elle permet d'exprimer des problèmes combinatoires à l'aide d'un formalisme unique. Enfin, elle permet d'utiliser des fonctions de coût non linéaires, et non linéarisables.

La programmation par contraintes connaît un succès important depuis quelques années auprès de nombreux industriels qui cherchent à augmenter leur efficacité et leur compétitivité. De nombreuses applications sont aujourd'hui opérationnelles, aussi bien dans le domaine de la gestion du personnel, de la planification de tâches, de l'allocation de ressources, de la gestion de production, des réseaux de transport, qu'en finance.

Initialement, la programmation par contrainte était réservée à la résolution de problèmes de satisfaction, mais elle a été étendue afin de pouvoir s'appliquer aux problèmes mixtes, de satisfaction et d'optimisation, qui correspondent mieux aux réalités industrielles. De plus, on est aujourd'hui capable de résoudre des CSP dynamiques : l'objectif est de résoudre un problème, résultant de quelques modifications sur un problème précédent, sans refaire tout le raisonnement et en modifiant au minimum la solution précédemment trouvée. On utilise pour cela des méthodes à base de réutilisation de solution et de raisonnement.

Néanmoins, la programmation par contraintes connaît encore quelques limites : la détection des erreurs, l'interrogation de ce qui se passe à l'exécution, ne sont pas des problèmes bien résolus aujourd'hui. Les principaux axes de recherche concernent aujourd'hui l'explication des décisions prises [Jussien 01], la décomposition de systèmes de contraintes, la résolution distribuée d'un problème de satisfaction de contraintes, et l'extension de mécanismes d'apprentissage aux problèmes d'optimisation sous contraintes.

1.3.2.2 Fonctionnement de la programmation par contraintes

De manière générale, la programmation par contrainte s'attaque à la résolution de CSP. Les contraintes du CSP y sont utilisées dans un processus déductif, la propagation, qui permet de réduire les domaines des variables, et d'accélérer le traitement du problème, en évitant des énumérations longues de valeurs interdites. Par exemple, si l'on a les contraintes $a < b$ et $a > 10$, la phase de propagation des contraintes permet de déduire la contrainte supplémentaire $b > 11$. On évite ainsi de tester les valeurs 0..10 pour b, qui aboutiraient toutes à une incohérence.

La propagation des contraintes se fait au moyen de nombreuses techniques, la plus répandue étant la cohérence d'arc (arc-consistency). Étant donné une contrainte portant sur n variables $X_1 \ldots X_n$, et un domaine $D(X_i)$ pour chaque variable X_i, la contrainte est arc-cohérente si et seulement si pour toute variable X_i et pour toute valeur VAL_i de $D(X_i)$, il existe des valeurs $VAL_1, \ldots, VAL_{i-1}, VAL_{i+1}, \ldots VAL_n$, chaque VAL_k appartenant au domaine $D(X_k)$, telles que la contrainte soit vérifiée lorsque pour $j=1..n$, $X_j=VAL_j$. Il existe de nombreux algorithmes de cohérence d'arc, les plus couramment utilisés sont AC-3 et AC-4 [Pacte 01_2]

On travaille souvent avec une représentation des domaines plus compacte, sous la forme d'un intervalle, ce qui permet de manipuler un plus grand nombre de variables. La propagation de contraintes sur de telles variables consiste souvent à maintenir l'arc-B-cohérence, c'est-à-dire l'arc-cohérence restreinte aux bornes des domaines.

La complexité d'un CSP étant évidemment très importante, on ne propage pas complètement les contraintes : on ne peut calculer en temps limité toutes les conséquences des contraintes sur les domaines des variables. On utilise donc une recherche arborescente pour déterminer l'existence d'une affectation valide des variables pour le problème. Cette recherche est caractérisée par :

- Son comportement en cas d'échec, c'est-à-dire lorsqu'il a été prouvé qu'aucune affectation faisable ne peut être dérivée de l'état courant. Les outils de programmation par contraintes utilisent majoritairement une recherche en profondeur d'abord, où l'on revient à la dernière variable instanciée. Mais il existe maintenant d'autres méthodes de retour arrière plus

performantes, capables d'examiner si la dernière variable a une influence sur l'échec ou non, et de remonter plus haut dans l'arbre si ce n'est pas le cas.

- L'heuristique utilisée pour choisir l'ordre des variables à instancier, et l'ordre des valeurs à essayer pour les variables. On peut par exemple s'intéresser en priorité aux variables dont le domaine est le plus restreint.

1.3.3 Spécificité de l'ordonnancement par contraintes

Pour traiter un ordonnancement non-préemptif, on associe à chaque activité deux variables, représentant la date de début et la date de fin de l'activité. De plus on crée une autre variable représentant le temps d'exécution de l'activité, et l'on ajoute la contrainte spécifiant que

[date de fin] – [date de début] = [temps d'exécution]

Le plus souvent, le temps d'exécution est connu, fixé, et la variable correspondante est instanciée.

Il existe principalement deux types de contraintes [Baptiste 98 p17] : les contraintes temporelles, qui lient le début ou la fin de deux activités, comme par exemple les contraintes de précédence, et les contraintes de ressources, qui spécifient qu'une activité utilise une certaine quantité de ressource pendant son exécution. De nombreuses règles ont été trouvées pour propager les contraintes temporelles [Le Pape 01 pp39-48].

1.3.4 Utilisation des bibliothèques de programmation par contraintes

1.3.4.1 Avantages de l'utilisation d'Ilog Scheduler

Ilog fournit un environnement de résolution de problèmes d'optimisation combinatoire, dont les composantes Solver et Scheduler seront utilisées pour la réalisation de l'outil demandé. Ilog Solver est une bibliothèque de programmes permettant de modéliser et de résoudre des problèmes de satisfaction de contraintes. Ilog Solver permet de séparer clairement la définition du problème, la propagation des contraintes et la prise de décision. Ilog Scheduler se présente comme une sur-couche de Solver, dédiée à la résolution de problèmes d'ordonnancement.

L'utilisation de bibliothèques dédiées à la programmation par contraintes augmente énormément l'efficacité des CSP, ces bibliothèques étant capables d'avoir un point de vue global sur certains ensembles de contraintes. Il existe par exemple des algorithmes puissants pour propager une contrainte spécifiant qu'un ensemble de *n* variables doivent prendre des valeurs deux à deux distinctes [Régin 94].

Scheduler permet de créer facilement différents types d'activités et de ressources, et implémente les contraintes fondamentales qui peuvent exister dans un problème d'ordonnancement, notamment différentes contraintes de précédence et d'utilisation de ressource. La préexistence de ces contraintes facilite l'écriture du modèle, évitant l'ajout d'une multitude de contraintes pour décrire un comportement simple. Mais cela permet aussi d'accélérer la recherche de la solution, Ilog ayant optimisé la propagation de ces contraintes. Scheduler permet aussi d'orienter la recherche d'un ordonnancement réalisable en écrivant ses propres heuristiques.

1.3.4.2 Différents types de ressources

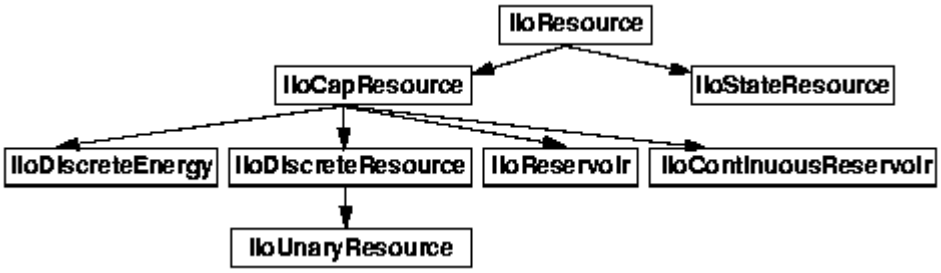

Figure 2 : hiérarchie des classes de ressource pour Ilog Scheduler

Les ressources de capacité (IloCapResource) regroupent toutes les ressources qui ont une capacité entière, limitée, au cours de l'ordonnancement. Il en existe plusieurs types :

- Les ressources d'énergie (IloDiscreteEnergy), qui permettent de représenter la quantité d'énergie, au sens large (par exemple en watt.heure ou en homme.mois) disponible par intervalle de temps donné (par exemple par jour).
- Les ressources discrètes (IloDiscreteResource), qui représentent des quantités entières d'instances de ressources disponibles, à chaque instant. Par exemple, une ressource discrète peut être utilisée pour représenter une équipe d'ouvriers interchangeables : chaque activité utilisera un certain nombre d'ouvriers, et le nombre total d'ouvriers occupés à chaque instant ne pourra dépasser la capacité de la ressource.
- Les ressources unaires (IloUnaryResource), qui sont un cas particulier de ressources discrètes, pour lesquelles la capacité est fixée à 1. Elles représentent typiquement l'utilisation d'une entité indivisible particulière (machine, ouvrier spécialisé). Elles permettent aussi de spécifier des coûts de transition entre deux utilisations de la ressource.
- Les réservoirs (IloReservoir), qui sont des ressources que les activités n'occupent pas, mais produisent ou consomment : la quantité de ressource réservoir nécessaire à l'exécution d'une activité ne sera pas restituée en fin

d'activité. De même, une ressource peut augmenter la quantité disponible d'une ressource réservoir à la fin de son exécution. Ces évolutions sont bornées par la capacité minimale et maximale du réservoir.

- Les réservoirs continus (IloContinuousReservoir), qui sont des ressources que les activités peuvent remplir ou vider, de manière continue et linéaire, entre le début et la fin de l'activité. Le réservoir ne peut être vidé s'il est déjà vide, ni rempli s'il a atteint sa capacité maximale.

Les ressources d'état (IloStateResource), elles, représentent des ressources de capacité infinie, dont l'état peut varier au fil du temps. Chaque activité peut, au cours de son exécution, requérir un état particulier de cette ressource (ou admettre un ensemble d'états possibles). Deux activités ne peuvent donc pas s'exécuter simultanément si elles requièrent des états incompatibles pour une même ressource d'état.

1.3.4.3 Programmation Ilog Solver

La programmation par contraintes au moyen d'Ilog Solver (et donc Scheduler) suit la structure suivante :

1- Création d'un manager : c'est l'entité qui va englober le modèle
2- Définition des variables
3- Pose des contraintes
4- Création des buts : c'est l'ajout de buts qui permet de définir les heuristiques sur les variables et les valeurs, et d'optimiser la recherche.
5- Recherche d'une ou des solutions : on peut itérer la recherche sur le modèle pour trouver toutes les solutions optimales.
6- Éventuellement, optimisation de la solution : Ilog Scheduler permet de partir d'un modèle et d'une solution admissible pour rechercher une meilleure solution, en utilisant l'optimisation locale.
7- Affichage de la ou des solutions
8- Fermeture du manager

1.4 Limites du sujet

1.4.1 Inexactitude de la méthode

Étant donné la modélisation choisie, nous ne chercherons pas à trouver l'optimum exact du problème, mais une bonne solution pour le problème d'ordonnancement, au sens de la fonction de coût qui sera définie. On pourra trouver facilement des bornes pour le coût de l'ordonnancement optimal, par exemple en considérant que tout le métal est fondu au tarif le plus bas, mais on n'aura pas de garantie a priori sur la qualité de la solution trouvée.

1.4.2 Limites du problème lui-même

Par rapport au problème réel, le problème modélisé est limité : la durée du stage ne permet pas de traiter le problème exact dans son intégralité, et il a donc fallu faire des choix, ne sélectionner qu'une partie des contraintes du problème. Ces limites étaient prévues dès l'établissement du sujet du stage, mais leur spécification a fait l'objet d'une partie du stage.

CHAPITRE 2
RECHERCHES THEORIQUES

2.1 Choix de la programmation par contraintes

Le choix de la programmation par contraintes est dû à la nature du problème : la diversité des configurations d'équipements possibles, et des contraintes qui régissent la production, rendrait très difficile la modélisation du problème général en programmation linéaire, par exemple. De plus, l'objet du stage ne recouvre qu'une petite partie du projet final, dont la fonction de coût est cubique.

De plus l'utilisation d'une bibliothèque dédiée pour la modélisation d'un problème en programmation par contraintes aboutit à du code largement réutilisable. En effet l'écriture de contraintes spécifiques à l'ordonnancement comme celles proposées par Ilog Scheduler donne une très grande lisibilité au modèle, ce que ne permet pas toujours la programmation mathématique. L'outil devant être réalisé étant une maquette d'un outil futur plus complet, cet aspect est primordial.

La programmation par contraintes nous permet de rechercher d'abord une bonne solution admissible, puis d'améliorer la solution générée par des techniques d'optimisation locale telles que l'algorithme tabou ou le recuit simulé. L'optimisation locale de ce problème particulier n'a pas encore été étudiée.

2.2 Modélisation du problème

2.2.1 Définition des différentes activités

Les activités de notre modèle seront les activités du processus réel de production de la fonderie : fontes, prélèvements, moulages. Chacune de ces activités sera représentée par une tâche dans le problème de satisfaction de contraintes.

Le nombre exact de tâches qu'il faut instancier n'est pas immédiatement déterminable. Il peut s'avérer moins coûteux, en fonction de l'évolution des tarifs de l'électricité au fil des heures, de couler une certaine quantité de métal en plusieurs fois, même si cette quantité est inférieure à la capacité d'un four, puisque la durée de la fonte est proportionnelle à la quantité de métal à fondre. Pour résoudre efficacement le problème, il faut évaluer le nombre de processus de fonte à instancier, et vraisemblablement essayer en priorité de fondre le maximum de métal à la fois. On ne peut pas non plus toujours grouper la fonte d'un grand nombre de pièces en un faible nombre de tâches de moulage, non plus pour des raisons de coût, qui ne prend pas en compte les moulage, mais à cause des différentes contraintes temporelles, et de l'occupation des ressources. L'idéal est que le nombre d'occurrences à accomplir de chaque tâche soit géré dynamiquement par Solver afin de trouver le processus optimal.

2.2.2 Choix des ressources

Une fois spécifiées les différentes tâches (ou activités) à ordonnancer pour modéliser le processus de fonte, il faut déterminer quelles seront les ressources nécessaires à l'exécution de chaque tâche.

Si l'on considère les cuves (ou tout autre contenant) comme des ressources discrètes, dont le niveau correspond à la quantité de métal présente, on n'est pas alors dans un modèle où une activité occupe une ressource, pour la libérer à la fin de l'exécution, mais où au contraire la ressource est consommée ou produite par chaque activité. Par exemple la poche de prélèvement consomme une certaine quantité d'une ressource « cuve de four », au début de son exécution, mais produit, lorsqu'elle s'achève, la même quantité de la ressource associée au contenant dans lequel elle déversera le métal. On choisit donc de modéliser chaque ressource correspondant à une quantité de métal présente dans un contenant comme un *réservoir*.

Par contre, si l'on a par exemple une poche de prélèvement pour plusieurs fours, la poche de prélèvement sera elle-même une ressource unitaire, que l'activité de prélèvement occupera pendant toute sa durée. On peut supposer de plus qu'un délai minimum est nécessaire entre deux prélèvements par la même poche. Les contraintes de circulation des poches de prélèvement ne sont pas explicitées, mais on

peut les approximer par une contrainte temporelle de transition entre deux utilisations, associée à chaque poche de prélèvement (ou tout autre élément de la chaîne de production ayant un comportement semblable.) Les équipements seront donc modélisés sous forme de *ressources unitaires*.

Le fait d'avoir plusieurs ressources qui peuvent être équivalentes à un certain stade du processus, si par exemple deux fours contiennent du métal de même qualité, n'est pas gênant : chaque activité peut « choisir » sa ressource dans un ensemble de ressources équivalentes.

Tout au long du processus, on ne représente que des masses de métal de qualité donnée, sans se soucier de leur état : du métal qui se trouve dans la poche de prélèvement est forcément liquide, du métal sur les chaînes de coulage est sous forme de pièces. L'état du métal dans le four n'a pas à être représenté : tant que la fonte n'est pas terminée, le métal n'est pas disponible, et n'est donc pas représenté. À l'issue de celle-ci, il est liquide, et le réservoir correspondant augmente en proportion de la quantité de métal, le rendant disponible pour les tâches de prélèvements.

2.2.3 Contraintes du système

Il faut faire la distinction entre les contraintes du problème qui sont fixes, et constituent des données numériques pour la modélisation, comme par exemple le nombre ou la capacité des fours, et les contraintes qui seront représentées effectivement par des contraintes dans la modélisation, comme le fait qu'on puisse ou non laisser du métal non fondu en attente dans le four. On peut faire le parallèle avec un problème résolu par programmation linéaire, où les premières contraintes correspondent aux matrices de données, et où les secondes spécifient la forme des équations générales du programme linéaire.

Si la durée des tâches de transport du métal est fixe, spécifiée dans le cahier des charges, la durée de chaque fonte ou moulage de métal dépend de la quantité de métal à fondre ou du nombre de pièces à réaliser. Le temps nécessaire pour que le métal atteigne la température voulue doit être calculé : en effet, dans la réalité de la production, la température peut être mesurée, mais chaque mesure détruit la sonde thermométrique et on ne peut donc pas mesurer la température en continu. D'autre part, l'ordonnancement est une simulation, et ne gère donc pas le processus de fonte de manière dynamique, en temps réel. On considère que les coefficients fournis par le cahier des charges pour estimer la durée en fonction de la quantité de métal à fondre sont exacts. On pourrait augmenter ces durées d'un certain pourcentage de sécurité, sans rien changer fondamentalement au problème et à sa résolution, les durées restant proportionnelles aux quantités de métal à fondre.

2.2.4 Fonction de coût

2.2.4.1 Modélisation du coût

La représentation du coût est un point délicat de la modélisation de ce problème. En effet on souhaite minimiser un coût total, qui sera la somme des coûts dus à chaque fonte (la fonte de métal, et éventuellement son maintien en température, sont seuls pris en compte dans les charges énergétiques, le reste de la consommation électrique étant négligeable). Le coût d'une fonte dépend de sa durée, mais aussi de la date de la fonte. De plus, le tarif ne peut pas être considéré comme fixe au long d'une tâche de fonte. Contrairement à un appel téléphonique, pour lequel le tarif retenu est celui en vigueur à l'instant où l'appel commence, l'électricité est consommée en permanence par l'usine et est facturée de manière continue.

Il faut donc sommer sur chaque intervalle de temps élémentaire la consommation au tarif en vigueur pendant cette durée. Ilog Scheduler propose des contraintes « intégrales », qui permettent de représenter ce type de contrainte, mais la fonction dont on calcule l'intégrale doit respecter une contrainte très forte : le maximum de cette fonction multiplié par la largeur de son intervalle de définition ne doit pas excéder l'entier le plus grand qui puisse être représenté sur la machine sur laquelle est compilée le programme.

2.2.4.2 Optimisation des coefficients tarifaires

Si l'on considère que l'entier le plus grand est 2^{15}, et si l'on cherche à représenter la fonction de coût sur une semaine exactement, définie par intervalles de temps de 10 minutes, valeurs qui semblent réalistes au vu du cahier des charges, la valeur maximale que pourra prendre la fonction de coût sera :

$$\text{Partie entière}(2^{15}/(7*24*6)) = 32.$$

Il en résulte que si l'on veut pouvoir calculer des ordonnancements suffisamment longs, et avec une atomicité temporelle satisfaisante, la fonction doit avoir un maximum très faible. EDF utilise actuellement 4 coefficients tarifaires différents (voir courbes tarifaires en annexe) : 1.00, 0.79, 0.59 et 0.28. La fonction devant être à valeurs entières, on multiplie ces coefficients par 100. Mais 100 est un maximum trop important, qui limite très fortement le nombre d'intervalles de temps élémentaires de l'ordonnancement. Nous avons donc recherché des coefficients équivalents plus petits.

59 et 79 sont des nombres premiers, on ne peut donc pas simplement diviser les coefficients par leur plus grand diviseur commun pour réduire les coefficients. Nous avons donc approximé 59 par 60 et 79 par 80. 28, 60, 80 et 100 sont tous divisibles par quatre, on peut donc les remplacer par 7, 15, 20 et 25 sans perdre davantage d'information. Une autre possibilité serait d'effectuer un décalage sur tous

les coefficients : si l'on diminue chacun des coefficients de la même quantité, et en considérant que les fontes ont une durée fixe, il y aura équivalence, puisqu'il suffirait d'ajouter la constante *(durée totale des fontes)*(quantité enlevée)* au coût total trouvé. On peut alors passer le plus petit des coefficients à zéro, pour simplifier au maximum le problème, et obtenir ainsi les coefficients équivalents (sous l'hypothèse ci-dessus) 0, 31, 51 et 72. Il suffit d'arrondir 31 à 30 pour que les coefficients soient divisibles par trois, et on obtient alors 0, 10, 17 et 24 comme valeurs possibles pour la fonction. On ne gagne qu'une unité sur le maximum précédent, en n'effectuant qu'un seul arrondi, mais plus important que les deux précédents ($1/31 > 1/59 > 1/79$), et surtout au prix d'une hypothèse qui ne sera pas vérifiable dans toutes les configurations de fonderie. On n'obtient pas de meilleur résultat en diminuant les coefficients d'autres quantités.

Nous avons donc initialement choisi d'utiliser les valeurs 7, 15, 20 et 25 dans nos calculs, lorsqu'il ne sera pas possible d'utiliser les valeurs exactes.

2.2.5 Contraintes particulières

2.2.5.1 Revêtement des fours

Tous les fours, quel que soit leur type, possèdent une couche de revêtement réfractaire qui s'use au fil des fontes. Cette usure provoque une légère augmentation de la capacité des fours, et peut donc être prise en compte. De plus, lorsque le réfractaire est trop usé, on ne peut plus fondre dans le four et on doit remplacer le revêtement. Le nombre de fontes entre deux remplacements étant nettement supérieur au nombre moyen de fontes que l'on aura à ordonnancer à la fois, il faudra être en mesure de spécifier au programme quel est l'état d'usure au début de l'ordonnancement. On peut considérer le revêtement réfractaire comme une ressource de type réservoir spécifique à chaque four, pour lequel chaque fonte va consommer une unité. Lorsque cette ressource est épuisée, on ne peut donc plus fondre dans ce four. On peut alors spécifier une activité « remplacer le réfractaire » qui s'exécute dès que le réfractaire d'un four est épuisé, et produit *n* unités de la ressource réfractaire, *n* étant le nombre de fontes possibles entre deux remplacements. L'augmentation de capacité peut se faire par un redimensionnement de la capacité maximale de la ressource réservoir associée au four, après chaque fonte. Néanmoins, et dans un premier temps, on pourra négliger cette faible augmentation.

2.2.5.2 Refroidissement du métal

Une fois que le métal est fondu, il faut l'utiliser avant qu'il ne refroidisse : on pourra fixer un délai maximal entre l'achèvement d'une activité « fonte » et le début de l'activité « prélèvement » suivante. La vitesse du refroidissement du métal n'est pas spécifiée dans le cahier des charges.

2.2.5.3 Nuance de métal

Chaque série de pièces nécessite une certaine qualité de métal, cette qualité étant définie d'une part par sa température et d'autre part par la nuance du métal, c'est-à-dire la proportion de chaque type de métal brut. Le nombre de séries à ordonnancer à la fois étant restreint, le nombre de couples (nuance, température) est très limité, et l'on peut choisir d'identifier chaque quantité de métal circulant dans la chaîne de production par un indice entier, sans spécifier explicitement la nuance ou la température du métal. Nous identifierons chaque quantité de métal tout au long du processus par cet indice. Si on utilise trois qualités de métal différentes, on peut créer pour chaque contenant trois ressources réservoir différentes, chacune étant affectée à une qualité de métal. Il suffit d'ajouter des contraintes spécifiant qu'un seul de ces trois réservoirs à la fois peut être non vide.

Dans la réalité, la nuance du métal peut, dans certaines fonderies, être ajustée en cours de processus. Cette modification nécessite une gestion plus complexe des quantités de métal liquide circulant, puisqu'il faut pouvoir déterminer comment passer d'une nuance à l'autre. Cela augmenterait donc considérablement la complexité du problème, et conformément aux recommandations d'EDF, nous écarterons cette possibilité.

2.2.5.4 Ordonnancement en cours de fonte

Un ordonnancement peut être demandé alors qu'un processus de fonte est en cours, mais l'initialisation des différentes variables permet de gérer ce type de problème de manière équivalente. Il suffit d'ajouter des contraintes fixant les dates et les variables de capacité des premiers événements en fonction des tâches déjà décidées.

2.2.5.5 Approvisionnement des fours

Suivant les us et coutumes de chaque industriel, on peut autoriser ou non d'approvisionner un four en métal solide alors que ce four contient déjà du métal fondu. Cette contrainte facultative s'exprime facilement en autorisant le chargement du four alors que la ressource réservoir associée n'est pas vide, et ne change pas la complexité du problème.

2.2.5.6 Alimentations alternées

Une même alimentation peut, dans certaines configurations, être affectée à plusieurs fours pseudo-simultanément, c'est à dire en alimentant en alternance, sur des durées relativement brèves (quelques minutes), les fours. Cela peut être modélisé

soit en considérant que la puissance est distribuée continuellement, mais autant de fois moins intensément qu'il y a de fours alimentés, soit en considérant les activités de fontes comme des activités interruptibles. La première option est plus complexe à mettre en œuvre, mais offre plus de souplesse, en éliminant le problème de la granularité temporelle du problème. De plus, considérer des activités interruptibles augmente considérablement l'espace de solutions, et donc la durée de la résolution.

2.2.5.7 Contraintes exceptionnelles

Certaines contraintes particulières peuvent apparaître, et seront modélisées au cas par cas. Ces contraintes seront typiquement des contraintes d'indisponibilité temporaire d'un équipement donné ou d'une partie du personnel, et se traduiront simplement par l'interdiction d'utiliser une ressource entre deux dates exactement fixées.

2.2.6 Flexibilité de la modélisation

Les équipements disponibles, ainsi que les contraintes réelles sur leur utilisation, varient énormément d'un site à l'autre, et il faut donc que la modélisation soit très souple. La programmation par contraintes rend l'ajout ou la suppression d'équipements ou de contraintes extrêmement simples : une nouvelle contrainte réelle ne nécessite souvent que l'ajout d'une seule contrainte dans le problème de satisfaction de contraintes.

La programmation par contraintes permet aussi de poser très simplement des contraintes de forme non linéaire.

CHAPITRE 3
RECHERCHES PRATIQUES
IMPLEMENTATION DU MODELE

3.1 Principe du processus de recherche

3.1.1 Entrées/Sorties du programme

3.1.1.1 Fichier « donnees.txt »

L'entrée du moteur d'optimisation est un fichier qui contient au format texte les données du problème d'ordonnancement. Ces données se répartissent en quatre catégories :

1. La configuration matérielle de la fonderie, qui comprend le nombre d'alimentations disponibles, le nombre de fours de chaque type et leurs caractéristiques (durée d'une fonte et capacité du four), le nombre de poches de coulée et leur capacité, la présence ou non d'un four de stockage, le nombre de poches de prélèvement et la durée de l'opération de prélèvement, le nombre de chaînes de coulée, et le délai maximum séparant la fonte du métal de son prélèvement.
2. La demande à satisfaire, en unités de métal, ainsi que les maxima des nombres de moulages et de fontes de chaque type autorisés pour satisfaire cette demande.
3. Le calendrier, qui comprend la saison en cours, les tarifs en vigueur, le nombre de jours de l'ordonnancement, la granularité temporelle utilisée (en nombre de plages temporelles élémentaires par heure), le jour de la semaine au début de l'ordonnancement.
4. Des contraintes spécifiques : la durée maximale autorisée pour l'ordonnancement (il ne s'agit pas de la date de fin maximale, mais de la durée séparant la première fonte du dernier moulage), le coût minimum recherché (si l'on connaît une meilleure borne inférieure du coût que celle calculée automatiquement), ainsi que des drapeaux imposant l'affectation statique ou non des fours aux fontes, la recherche d'une solution optimale ou simplement d'un ordonnancement admissible.

Un exemple de fichier de données est proposé en annexe de ce rapport.

3.1.1.2 Affichage de la solution

La sortie est aussi au format texte, et s'affiche directement sur la console. Le programme signale d'abord à l'utilisateur les éventuelles corrections ou optimisations effectuées sur les données d'entrée, puis rappelle les données du problème, et affiche un schéma du calendrier sur lequel peut être calculé l'ordonnancement. Une fois ces informations préalables affichées, la recherche est lancée.

Si la recherche montre qu'il n'existe pas de solution admissible, le message correspondant est affiché.

Si une solution est trouvée, le coût et la durée de l'ordonnancement proposés sont affichés, puis la solution est explicitée tâche par tâche, en indiquant les dates de début et de fin de chaque activité, les ressources affectées, les quantités requises de chaque ressource. Lorsqu'une plage de valeurs est acceptable de manière équivalente pour une variable, l'intervalle possible est affiché, plutôt que de choisir une valeur arbitrairement dans cet intervalle.

Dans les deux cas, des informations sur le processus de résolution sont finalement affichées.

Un exemple de sortie du programme est proposé en annexe de ce rapport.

3.1.2 Modalités des tests

Une fois les bases du modèle fixées (les activités, les ressources), il reste à définir exactement le modèle. Cela a été fait progressivement, en ajoutant les séries de contraintes une à une pour se rapprocher d'une représentation fidèle du processus de production.

À chaque modification ou ajout de contraintes, nous avons pratiqué de nombreux tests sur des exemples de données significatifs pour vérifier la qualité du modèle, et valider le choix effectué. Nous n'avons pas utilisé qu'un seul exemple de données pour tester le modèle. Premièrement, les différentes couches du processus de production de la fonderie ayant été ajoutées progressivement, il aurait fallu augmenter l'exemple de données au fur et à mesure, et la comparaison directe n'aurait pas eu de sens. Deuxièmement, les exemples ont à chaque fois été choisis en fonction de la modification faite, afin de mettre particulièrement en valeur le point concerné par la modification du modèle.

Pour pouvoir comparer deux versions du modèle, il nous faut d'abord définir la qualité d'un modèle.

3.2 Qualité d'un modèle

3.2.1 Compromis de qualité

La qualité d'un modèle de fonderie sous forme de problème d'optimisation par contraintes regroupe plusieurs aspects. On peut céder sur certains critères au profit d'un autre plus important, mais il est souvent possible, et c'est ce que nous nous sommes efforcés de faire, d'améliorer un aspect de la qualité du modèle sans que cela ait d'incidence négative sur les autres critères de qualité. Nous allons maintenant définir précisément les différentes facettes de la qualité d'un modèle.

3.2.2 Qualité d'écriture

Un des principaux intérêts de la programmation par contrainte réside dans la simplicité d'écriture de contraintes mathématiquement complexes. Cela apporte une grande facilité de relecture, de la concision, et par conséquent une facilité de débuggage. Pour que le modèle soit lisible, il faut notamment que les contraintes soient très proches d'une contrainte réelle. Les contraintes qui ne participent pas directement à la représentation du processus de fonderie, mais optimisent la recherche, doivent être aussi expressives que possible. La qualité d'écriture est importante, rend plus aisées les évolutions ultérieures du modèle, mais n'est pas un aspect essentiel de la qualité.

3.2.3 Commodité d'utilisation

Même si le programme d'optimisation que nous développons n'est qu'une maquette et n'est pas destiné à être utilisée directement par les ingénieurs d'exploitation des fonderies, la simplicité d'utilisation ne doit pas être négligée. À chaque étape du développement de l'outil, à chaque modification du modèle, de nombreux tests doivent être effectués, la réalisation de ces tests doit donc être pratique afin d'éviter d'importantes pertes de temps. La saisie des données d'entrée du problème doit être aussi simple que possible, et le résultat de la résolution doit être parfaitement lisible et donc facilement exploitable.

Néanmoins l'objet du stage est la réalisation d'une application d'optimisation, destiné à être incorporé à d'autres applications, il n'est donc pas envisagé de développer une interface homme-machine graphique spécifique au moteur.

3.2.4 Qualité de la solution

Le but du modèle est de fournir par résolution de celui-ci une solution au problème initial. La qualité du modèle est donc directement liée à la qualité de la solution. Lorsque la recherche donne une solution, il faut impérativement que celle-ci

soit effectivement admissible pour le problème réel. De plus, il faut éviter de passer à côté d'une solution.

Les heuristiques proposées par Ilog Solver ne sont pas toutes complètes et peuvent conduire à élaguer indûment certaines branches de l'arbre de recherche. Ainsi, un modèle parfaitement valable sur le papier peut ne pas fournir de solution sur certains cas extrêmes, alors que cette solution existe. Cet inconvénient n'apparaît qu'avec certaines contraintes particulières, notamment les contraintes temporelles comportant des délais négatifs, ou les contraintes fixant la durée d'une activité en fonction de sa date de début. De plus, on peut, dans un but d'optimisation de la vitesse de résolution, être amené à ajouter ou retirer certaines contraintes qui influeront sur l'ensemble des solutions admissibles du problème. Il faut limiter au maximum ce risque. Éliminer une solution admissible du problème peut faire perdre l'optimum si cette solution était la seule solution optimale, ou ce qui est plus grave, laisser croire à l'utilisateur qu'il n'existe aucune solution admissible, si le système de contraintes comporte un ensemble très réduit de solutions admissibles. Plus grave encore est l'acceptation d'une solution qui ne devrait pas être admise. Il faut impérativement que la solution proposée par le modèle soit effectivement admissible, l'utilisateur ne doit pas avoir à vérifier que la solution correspond effectivement à des ordres de production réalisables.

Notre problème est un problème de minimisation : cela signifie qu'à chaque solution est associée un coût, et que le coût de la solution trouvée doit être le plus bas possible. Donc la qualité du modèle réside aussi dans l'exactitude de la fonction de coût (il n'est pas toujours facile d'avoir un modèle de fonction de coût qui retranscrive fidèlement la réalité), ainsi que dans l'optimalité du coût trouvé. Il faut donc vérifier que le coût trouvé est le plus proche possible du coût optimal. Pour cela, il faut essayer de trouver les meilleures bornes inférieures possibles pour la solution optimale. De plus le calcul de bornes peut permettre d'accélérer la recherche d'une solution.

3.2.5 Performance de la recherche

3.2.5.1 Mesure de la performance

Pour qu'une implémentation d'un modèle soit réellement utile, il faut qu'elle soit capable d'apporter une solution au problème que le modèle représente le plus rapidement possible. Pour mesurer les performances d'un modèle, Ilog Solver fournit la méthode printInformation() qui, une fois la résolution effectuée, peut afficher plusieurs informations sur la recherche :

Number of fails	: 7487
Number of choice points	: 7519
Number of variables	: 184
Number of constraints	: 458
Reversible stack (bytes)	: 72384
Solver heap (bytes)	: 184944
Solver global heap (bytes)	: 337448
And stack (bytes)	: 4044
Or stack (bytes)	: 4044
Search Stack (bytes)	: 4044
Constraint queue (bytes)	: 11152
Total memory used (bytes)	: 618060
Elapsed time since creation	: 6.84

3.2.5.2 Quantité d'échecs

Les champs « Number of fails » et « Number of choice points » c'est-à-dire respectivement le nombre d'échecs dans la recherche et le nombre de nœuds de l'arbre de recherche, sont très importants pour comparer deux versions du modèle, et pour tester la qualité de l'heuristique employée. Ces deux valeurs sont généralement très proches, leur différence étant égale au nombre minimum de choix que le moteur de résolution doit faire pour instancier les variables du problème. Évidemment, plus ces valeurs sont faibles, plus la recherche est efficace. Un système de contraintes dont la résolution donne un très grand nombre d'échecs pour des données de petite taille est signe d'une mauvaise heuristique : le moteur essaie un très grand nombre de valeurs pour chaque variable avant de trouver la valeur correspondant à la solution optimale. Le problème d'ordonnancement auquel nous nous attaquons est NP-complet, sa résolution est donc par essence soumise à une explosion combinatoire, mais il faut s'attacher à limiter au maximum cet aspect combinatoire, ou ne rechercher qu'une solution approchée, pour pouvoir éventuellement l'éliminer complètement.

3.2.5.3 Nombre de variables

Le nombre de variables est l'un des principaux facteurs d'augmentation de la difficulté de la résolution, et est directement lié à la qualité d'écriture du modèle : il faut surtout éviter de multiplier inutilement les variables, ce qui augmente exponentiellement la taille de l'espace de recherche. À chaque activité est associée une variable marquant la date à laquelle débute l'activité, ainsi éventuellement qu'une variable représentant sa durée. De plus, d'autres variables peuvent être utilisées pour

les contraintes de ressources, ainsi que des variables qui ne font pas partie de la solution, mais participent aux calculs. Toutefois, les variables ne sont pas toutes aussi difficiles à instancier les unes que les autres, certaines (notamment celles qui ne font pas partie de la solution même mais participent à la recherche) sont directement calculées à partir d'autres. Aussi la variation du nombre brut de variables, d'une version du modèle à l'autre, est-elle moins significative que l'augmentation ou la diminution du nombre d'échecs dans la recherche.

3.2.5.4 Nombre de contraintes

Solver permet aussi d'obtenir le nombre de contraintes ajoutées au modèle. Si celles-ci restreignent l'espace de recherche, la propagation de contraintes effectuée à chaque nœud de l'arbre de recherche est d'autant plus coûteuse en temps de calcul que le nombre de contraintes est élevé.

3.2.5.5 Mémoire occupée

La quantité de mémoire vive utilisée par le modèle et pour sa résolution n'est pas un critère important : dans tous les cas, elle reste très faible par rapport à la quantité de mémoire disponible sur la machine.

3.2.5.6 Durée de l'exécution

Enfin le dernier critère, le plus important, est la durée totale qui s'est écoulée entre le début de la recherche même (l'instanciation et l'extraction du modèle n'est pas prise en compte) et l'obtention d'une solution ou d'un constat d'échec. La valeur donnée par Solver représente cette durée, soit en temps-machine, c'est-à-dire proportionnellement au nombre de cycles processeurs consacrés au processus de recherche, soit en temps réel, c'est-à-dire tel qu'elle est perçue directement par l'utilisateur, suivant le système utilisé. Nous avons effectué tous nos tests sur architecture NT, le temps qui apparaît dans ce champ est donc le temps machine écoulé pendant la recherche.

Cette durée n'est pas directement proportionnelle au nombre de noeuds parcourus, et donc au nombre d'échecs : ainsi, lors de nos tests, nous avons pu obtenir des ratios (nombre d'échecs)/(durée de la recherche) allant de 106 fails/s à près de 7000 fails/s. En effet, les contraintes sont plus ou moins complexes, la bibliothèque Scheduler en propose de très élaborées, et malgré les optimisations de propagation développées par Ilog, une contrainte d'égalité entre deux variables sera toujours plus rapide à traiter et donc à propager qu'une contrainte portant sur n variables. Si le rapport est important, c'est que le Solver passe beaucoup de temps à propager les contraintes du système. Or si la propagation des contraintes permet de réduire l'espace des solutions rapidement, il n'est pas toujours utile de passer trop de temps à propager certaines contraintes qui ont une influence réduite. On peut régler

pour chaque contrainte le degré d'effort que le solveur consacrera à sa propagation. On peut donc effectuer des tests sur cet aspect pour optimiser la vitesse de la recherche.

La durée de la recherche est le critère fondamental de la performance de la recherche : même si le nombre d’échecs en fonction de la taille du problème étudié est plus significatif de la qualité intrinsèque du modèle, le principal reste de trouver la solution rapidement. Le cahier des charges spécifie que le résultat du moteur d'optimisation doit être obtenu dans un temps très limité, de l'ordre de 10 secondes.

3.3 Modélisation du problème

3.3.1 Activités et ressources

Les activités du problème de satisfaction de contraintes sont les fontes, les prélèvements et les moulages. À ces séries de tâches s'ajoute une tâche unique de finalisation, qui va s'assurer que la demande a bien été satisfaite. Les durées des fontes et des prélèvements sont données par l'utilisateur.

Les variables sont toutes entières. L'axe des temps est divisé en intervalles de temps élémentaires, dont la taille est fixée par l'utilisateur. Les quantités de métal sont exprimées en unités correspondant à la capacité d'une poche de prélèvement.

Les fours et les poches de coulée sont représentés par des ressources de type réservoir. Le niveau du réservoir d'un four correspond à la quantité de métal disponible dans le four, c'est-à-dire de métal déjà fondu. De même, le niveau du réservoir d'une poche de coulée correspond à la quantité de métal disponible pour le moulage. Leur capacité est fixée par l'utilisateur. On ajoute un réservoir unique « pièces coulées » qui contient le résultat des opérations de moulage. L'unité est la même que pour les autres réservoirs du processus. Les niveaux de tous les réservoirs sont initialement nuls.

Les poches de prélèvement sont représentées par des ressources unaires. En effet le métal ne peut demeurer dans une poche de prélèvement, il est directement acheminé vers une poche de coulée. La ressource unaire permet d'interdire à deux prélèvements d'utiliser la même poche de prélèvement en même temps. De même, on ajoute des ressources unaires pour tous les réservoirs, afin d'éviter un usage simultané d'un four ou d'une poche de coulée par deux tâches différentes.

Les alimentations sont représentées par une ressource discrète renouvelable dont la capacité est égale au nombre d'alimentations de la configuration.

3.3.2 Processus de fonte

Les activités de fonte de chaque type produisent du métal fondu dans un four de même type, et se réservent l'utilisation de ce four pendant toute la durée de la fonte. À la fin de la fonte, le réservoir est à son niveau maximum. Chaque tâche de prélèvement peut alors consommer une unité de métal pour le reverser dans une poche de coulée. Chaque fonte requiert une alimentation, c'est-à-dire une quantité 1 de la ressource « alimentations », pendant toute son exécution. L'opération de prélèvement se réserve l'utilisation du four, de la poche de prélèvement et de la poche de coulée pendant toute sa durée. Un prélèvement ne peut donc utiliser un four sur lequel une fonte est en cours, ou une poche de coulée utilisée par un moulage.

Les tâches de moulage consomment le métal dans les poches de coulée, et augmentent d'autant le niveau du réservoir « pièces coulées. »

Chaque consommation de métal dans un réservoir se fait au début d'une activité, chaque production se fait à la fin d'une activité. Les tâches doivent choisir le réservoir qu'elles utiliseront parmi un ensemble de réservoirs : les fontes de type M.F. (respectivement B.F.) peuvent utiliser n'importe quel four M.F. (respectivement B.F.), les prélèvements peuvent utiliser indifféremment les fours B.F. ou M.F., et reverser le métal dans n'importe quelle poche de coulée disponible. Les tâches de moulages n'ont pas de poche de coulée prédéterminée.

À chaque activité de fonte est associée une variable qui correspond au coût de la fonte, et est égale à l'intégrale de la fonction tarifaire sur la période d'exécution de la fonte. Le coût total de l'ordonnancement est la somme des coûts de toutes les fontes.

3.4 Augmentation et optimisation du modèle

3.4.1 Utilisation d'ensemble de ressources

Une tâche nécessite souvent une ressource parmi un ensemble de ressources équivalentes : par exemple, les poches de prélèvements consomment du métal dans un four, sans que le type ou le numéro du four ne soit précisé lors de la déclaration du prélèvement. Nous avons donc créé des ensembles de ressources (classe IloAltResSet dans le source) qui permettent de représenter cette liberté de choix.

Lorsqu'une tâche de prélèvement consomme du métal dans un four, elle doit s'assurer qu'aucune autre activité n'utilise le même réservoir qu'elle au même moment. Or les ressources réservoir ne sont pas disjonctives, elles peuvent être utilisées simultanément par plusieurs activités. Nous avons donc associé à chaque entité nécessitant une ressource réservoir pour décrire le niveau de métal courant (fours, poche de coulée) une ressource unaire qui représente l'occupation de l'entité elle-même. La concordance des ressources unaires et des ressources réservoir affectées à la tâche est assurée par une contrainte « IloIfThen », spécifiant que si la tâche utilise le réservoir *i*, alors elle doit utiliser la ressource unaire *i* associée.

3.4.2 Optimisation de l'utilisation des coûts

Nous avons posé une contrainte sur l'utilisation de l'alimentation, fixant la « variable externe » associée à chaque activité utilisant une alimentation. Cette variable est le coût de l'activité en question. Seules les tâches de fontes utilisent les alimentations, cette contrainte permet donc, après avoir sommé toutes les variables de coût du problème, d'obtenir le coût total de l'ordonnancement. Le calcul du coût d'une fonte se fait au moyen d'une contrainte spécifique de Scheduler, qui calcule l'intégrale d'une fonction donnée sur la durée de l'activité, et la divise par la borne supérieure de la fonction. Dans notre cas la fonction utilisée est tarif(t), décrite au paragraphe 1.2.2.2. La borne supérieure de la fonction est fixée précisément au tarif « heures pleines » donné en entrée. La division de l'intégrale par cette valeur (cette division est imposée par Ilog) introduit une inexactitude dans la fonction de coût, qui ne peut renvoyer qu'une valeur entière : le résultat de la division est donc arrondi, inférieurement ou supérieurement, ce qui peut conduire le solveur à laisser une fonte empiéter inutilement sur les heures pleines.

Pour éviter que la résolution pâtisse de cette approximation, nous pouvons utiliser comme coût *0* en heures creuses et *1* en heures pleines : ainsi le coût d'une fonte sera exactement égal à la durée de la fonte effectuée en heures pleines. Toutefois ce choix de tarifs apporte un inconvénient majeur : en heures creuses, les fontes ne coûtent rien. Si la durée totale des fontes est fixe et connu, cela n'est pas gênant : le coût réel de l'ordonnancement ne sera pas proportionnel au coût fourni

par le programme, mais il peut être simplement déterminé. Mais si la durée totale des fontes est laissée à l'appréciation du moteur d'optimisation, rien ne l'empêche d'effectuer de très nombreuses fontes en heures creuses. Cela peut donc dans certains cas affecter largement l'optimalité de la solution. De plus, l'utilisation de coûts 0-1 à la place des coûts réels ralentit la résolution. Ces tarifs ne doivent donc être utilisés qu'après avoir fait une recherche avec les coûts réels, lorsque le nombre et les durées des fontes sont fixés, pour améliorer la solution.

3.4.3 Calcul des bornes

3.4.3.1 Avantages procurés par le calcul de bornes

Chaque variable est déclarée avec un intervalle initial de valeurs possibles. Plus cet intervalle est restreint, plus l'espace des solutions possibles sera réduit. De plus, si une variable doit être minimisée, si la valeur minimale de son intervalle de définition peut être atteinte, le solveur ne perdra pas de temps à chercher une solution inférieure. Aussi est-il très intéressant, pour accélérer la résolution du problème, de calculer des bornes pour les différentes variables du problème.

3.4.3.2 Bornes de coût

Le coût de chaque fonte peut être majoré par sa durée : si la fonte se déroule entièrement en heures pleines, le coût de la fonte sera

$$Sup(coût(\ fonte)) = \frac{\int_{début(i)}^{fin(i)} tarif\ \_haut.dt}{tarif\ \_haut} = fin(i) - début(i)$$

De même, le coût de chaque fonte peut être minoré par (tarif_bas/tarif_haut)*durée(fonte).

$$Inf\,(coût(\ fonte)) = \frac{tarif\ \_bas}{tarif\ \_haut} * durée(\ fonte)$$

Le nombre de fontes de chaque type étant déterminé par le solveur, dans les dernières versions du modèle, il est important de fournir à cette répartition de bonnes bornes. Nous avons donc calculé le nombre de fontes M.F. nécessaire pour satisfaire la demande si on n'effectue aucune fonte dans les fours basses fréquences, et de

même nous avons calculé le nombre de fontes B.F. nécessaire si l'on ne fond qu'en basses fréquences. Il est inutile d'instancier plus de fontes de chaque type. Ces bornes sont égales à la demande divisée par la capacité du type de four considéré.

On peut alors définir une borne inférieure du coût total de l'ordonnancement en prenant :

$$\underset{typedefonte}{Min}\left(Inf\left(coût(fonte)\right) * Inf\left(nombre(fontes)\right)\right)$$

Cette borne de coût est très grossière, mais pour certains exemples de données elle peut être atteinte, et nous avons donc pu vérifier l'amélioration considérable du temps de calcul lorsque l'on possède une « bonne » (pour l'exemple particulier considéré) borne du coût total. Nous sommes ainsi passé sur une instance du problème de 83000 échecs et 20 secondes à 69 échecs et 0.61 seconde. Ce résultat nous encourage à rechercher une meilleure borne inférieure du coût total, nous verrons dans la conclusion comment y parvenir.

3.4.3.3 Bornes de durée

Au début du développement du modèle, quand le nombre de fontes de chaque type était directement défini par l'utilisateur, nous avions introduit une borne inférieure de la durée totale de l'ordonnancement. Cette borne avait peu d'effet sur la vitesse de la recherche (nous devons optimiser le coût de l'ordonnancement, pas sa durée), mais permettait d'emblée de déclarer l'absence de solution si la borne trouvée était supérieure à la durée totale disponible pour l'ordonnancement. L'augmentation du modèle n'a pas permis de conserver cette borne, toutefois nous verrons en conclusion comment en calculer simplement quel que soit le modèle.

3.4.4 Optimisation des données fournies

Le meilleur moyen pour réduire la durée de la résolution d'un problème est de résoudre un problème plus facile. Nous avons donc cherché à optimiser les données fournies par l'utilisateur pour réduire la taille et la complexité du problème de satisfaction de contraintes, tout en conservant l'équivalence avec le problème réel correspondant aux données fournies. Parallèlement, nous corrigeons les valeurs incohérentes que peuvent comporter les données. L'utilisateur est mis au courant de ces modifications avant le lancement de la recherche.

3.4.4.1 Corrections effectuées

- On vérifie que la demande spécifiée par l'utilisateur est réalisable avec les maxima imposés au nombre de fonte de chaque type. Si ce n'est pas le cas, la demande est ramenée à la quantité de métal maximum qu'il est possible de fondre avec ces maxima.
- Un four de stockage au maximum peut être défini. Cette correction est là à titre préventif puisque pour l'instant le modèle n'utilise pas le four de stockage.
- Si l'on autorise au solveur d'ordonnancer des fontes d'un type donné, il est nécessaire qu'au moins un four de ce type soit créé.
- La quantité de métal qui peut être coulée lors d'une tâche de moulage est maximisée par la capacité d'une poche de coulée. Aussi un nombre de moulages minimum est-il nécessaire pour pouvoir satisfaire la demande. Le nombre de moulages est donc éventuellement corrigé, le solveur pouvant effectuer une longue recherche avant de se rendre compte que la demande ne peut être satisfaite.
- Comme nous l'avons vu dans le format des données du problème, l'utilisateur peut fixer une limite au makespan, c'est-à-dire à la durée totale de l'ordonnancement, limite que le solveur ne peut en aucun cas dépasser. La limite prise en compte est corrigée si elle est supérieure à la durée totale du calendrier, c'est à dire à la valeur de l'horizon du modèle. L'horizon n'étant pas directement fourni par l'utilisateur, l'utilisateur peut proposer une valeur « infinie » s'il ne souhaite pas contraindre la durée de l'ordonnancement, cette valeur sera corrigée.
- De même lorsqu'une borne inférieure de la durée de l'ordonnancement était utilisée, le maximum autorisé pour le makespan était corrigé pour éviter de lancer une longue recherche alors qu'il n'existe pas de solution.

3.4.4.2 Optimisation des données corrigées

- Il est inutile d'instancier trop de fontes de chaque type. Nous utilisons les bornes calculées précédemment pour définir le nombre de fontes qui seront instanciées (mais pas forcément effectuées). Si n fontes d'un même type suffisent à satisfaire la demande, il est inutile d'instancier $n+1$ fontes de ce type. Cette modification optimise la taille du problème : le nombre de tâches est réduit, et donc le nombre de contraintes et de variables l'est aussi.
- De même on peut réduire le nombre de moulages à la valeur de la demande : le métal étant traité par unités indivisibles, chaque moulage effectué coulera au minimum une unité de métal, et il est inutile d'instancier plus de moulages qu'il n'y a d'unités de métal demandées.
- Si n fontes au maximum d'un type donné sont autorisées, il est inutile de créer plus de n fours de ce type, même si la fonderie comporte effectivement plus de fours : les fours surnuméraires ne seraient de toutes façons pas

utilisés. Le nombre de fours d'un type de fonte est donc éventuellement réduit au nombre de fontes maximum de ce type.

- Si l'utilisateur ne connaît pas de borne de coût pour le problème qu'il propose, ou si la borne de coût qu'il donne est moins bonne que celle calculée, le modèle choisit la valeur la plus élevée. L'utilisateur peut donc spécifier une borne de coût inférieur nul sans perte de performance.

3.4.5 Répartition des fontes entre les fours

Dans les premières versions du modèle, le nombre de fontes de chaque type était directement et exactement défini par l'utilisateur. Nous avons ensuite laissé le choix du nombre de fontes à réaliser au solveur. L'utilisateur ne spécifie plus que les maxima des nombres de fontes de chaque type, ce qui peut être utile pour des tests. Une tâche ne peut pas être instanciée dynamiquement dans l'environnement Ilog Scheduler. Il faut donc que toutes les fontes potentiellement réalisables soient instanciées. Pour qu'une fonte ne soit pas « réellement effectuée » il faut que la tâche correspondante n'ait pas d'incidence sur le reste du système. Une activité n'a pas d'influence sur les ordonnancements possibles si sa durée est nulle et qu'elle ne consomme pas ou ne produit pas de réservoir.

Nous avons donc attribué à chaque fonte une variable entière valant 0 ou 1 qui spécifie si une fonte est réellement effectuée. La durée de la fonte est donc égale à la durée d'une fonte réalisée multipliée par cette variable. Les quantités intervenant dans les contraintes sur les réservoirs sont elles-aussi multipliées par cette variable.

Pour éviter que le solveur teste des solutions équivalentes, par exemple essaie de réaliser la première fonte mais pas la deuxième, puis la deuxième mais pas la première, nous avons considéré que le choix à faire n'était pas « quelles fontes doivent être effectuées » mais « combien de fontes doivent être effectuées. » Ainsi nous avons ajouté deux variables correspondant au nombre de fontes de chaque type qui seront réellement effectuées. Les variables booléennes sont directement déduites de ces variables, seules les « nombre de fontes à réaliser » premières fontes étant effectuées.

Enfin pour que le solveur évite de tester des ordonnancements ne comportant pas suffisamment de fontes pour satisfaire la demande, nous avons ajouté une contrainte imposant que les nombres de fontes réalisés produisent suffisamment de métal. Cette optimisation de la gestion des variables booléennes a permis de passer de 211000 échecs en 43 secondes à 4000 échecs en 7 secondes, sur une instance du problème laissant beaucoup de liberté sur le choix du nombre de chaque type de fonte.

3.4.6 Déclaration des variables de coûts des fontes

Nous avons remarqué en effectuant des tests sur la symétrie du modèle que l'heuristique utilisée par le solveur avait tendance à effectuer prioritairement les tâches de fonte dont les coûts ont été instanciés en dernier. Dans le cas ou un type de fonte a un rendement très supérieur à l'autre, cela augmente considérablement le nombre d'échecs de la recherche : si le type de fonte le moins performant voit ses coûts déclarés en dernier, le solveur va essayer d'abord de satisfaire la demande en fondant le métal uniquement dans ces fours, avant de se rendre compte que l'autre type de fonte permet de satisfaire la demande à moindre coût. Nous avons donc modifié la déclaration des coûts pour que ceux-ci soient ajoutés alternativement au tableau des variables que le solveur cherche à instancier. Cette optimisation a permis sur une instance du problème de passer de 275000 échecs et 135 secondes à 129 échecs et 1.6 secondes.

3.4.7 Gestion des moulages

Les moulages peuvent eux aussi être « nuls » dans le modèle, ce qui signifie qu'ils ne seront pas effectués dans la réalité de la production. De plus, les activités de moulage peuvent s'opérer sur des poches de coulée qui ne sont pas pleines. Nous avons donc affecté une variable qui correspond à la quantité de métal coulée (et qui peut être nul). Cette variable est représente aussi la durée du moulage, celle-ci étant directement proportionnelle à la quantité de métal fondu. Il suffira de multiplier cette durée par un coefficient pour pouvoir régler cette proportion, sans augmentation de la complexité du problème.

Nous avons observé que la résolution était plus performante si nous ajoutions une contrainte imposant que les moulages nuls soient ceux de plus faibles indices. Nous avons enfin contraint la somme des quantités de métal utilisées par chaque moulage à être égale à la demande.

3.4.8 Contraintes temporelles évitant les symétries

On remarque que si deux activités a1 et a2 sont strictement équivalentes, par exemple deux fontes de même type, on peut fixer une contrainte de précédence de la forme *début(a1)* $\leq$ *début (a2)* sans perdre de sens, et en restreignant l'espace des solutions, et donc en accélérant la recherche.

Cette optimisation a été effectuée pour chaque classe d'activités. Ainsi, les fontes B.F., les fontes M.F., les prélèvements et les moulages sont ordonnancés dans l'ordre de leur indice. Cette contrainte ne fixe que les dates de début des tâches, et rien n'empêche d'effectuer les prélèvements numéro 2 et 3 de s'effectuer simultanément ou la fonte B.F. numéro 4 de s'achever avant la fonte numéro 2.

3.4.9 Optimisation des buts guidant la recherche

Après avoir ajouté à l'environnement de recherches les activités, les variables et les contraintes, il faut fournir au Solver un but, qui lui indique quelles sont les recherches à effectuer, et comment effectuer ces recherches, quelles heuristiques employer.

Le but que nous utilisons est composé de quatre buts élémentaires :

1. Génération des variables de coût de chaque fonte.
2. Affectation des ressources nécessaires à chaque activité, pour les contraintes laissant le choix de la ressource.
3. Classement des contraintes de ressource sur l'axe temporel.
4. Fixation des dates de début et de fin de chaque activité.

L'ordre influe énormément sur la rapidité de la recherche, mais aussi sur l'obtention d'une solution : certains ordres conduisent à des coupures erronées dans l'arbre de recherche, qui éliminent les seules solutions admissibles. Certains ordres au contraire ne vont pas couper suffisamment de branches de l'arbre de recherche, et vont donc le parcourir largement avant de trouver une solution.

L'optimisation de l'ordre des buts a fait l'objet de nombreux tests. L'ordre proposé ci-dessus est optimal. Sur une même instance du problème d'ordonnancement, les durées de résolution varient de 1.4 à 16.6 secondes, et certaines positions des buts n'ont pas permis de trouver de solution en moins de 30 minutes. Le nombre d'échecs a varié de 320 à 31000.

Il s'est avéré inutile d'ajouter des buts de génération sur d'autres variables, par exemple le nombre de fontes à effectuer, ou les quantités de métal coulées par chaque moulage. Séparer le but d'affectation de ressources général (portant sur toutes les contraintes possédant une alternative) en plusieurs buts d'affectation (dédiés à chaque série de contrainte) n'a pas eu d'effet positif sur la performance du modèle.

Toutefois, il n'est pas exclu que par la suite il faille ajouter d'autres buts aux quatre existant actuellement. Plutôt que de chercher simplement l'ordre optimal pour ces quatre buts parmi les vingt-quatre possibles, nous avons cherché à dégager des règles sur les positions relatives des buts. Ainsi cette recherche sera réutilisable si l'augmentation du modèle nécessite l'ajout de nouveaux buts.

Les tests effectués ont permis de conclure que :

- Le but d'affectation doit être placé avant le classement temporel des contraintes, sans quoi on ne trouve pas de solution dans un délai raisonnable.
- La génération des coûts placée avant le classement temporel des contraintes réduit la durée de la recherche :

- L’affectation des dates placée après le classement temporel des contraintes améliore très nettement les performances de la recherche.

Une fois ces règles réunies, il ne restait qu’à choisir de placer la génération des coûts avant ou après l’affectation des ressources. Sur les problèmes résolus, le placement de la génération des coûts en premier accélérait légèrement la recherche. Cette différence n’est sensible que si les autres buts sont placés de manière optimale, c’est-à-dire en dernière position pour la détermination des dates, et en avant-dernière position pour le classement des contraintes sur l’axe temporel.

CHAPITRE 4
CONCLUSIONS SUR LE TRAVAIL DE RECHERCHE

4.1 Difficultés rencontrées

4.1.1 Indisponibilité de la licence Ilog Optimisation

La principale difficulté rencontrée au cours du début du stage a été l'impossibilité de tester les programmes réalisés, Pacte Novation n'ayant pas de licence Ilog Optimisation disponible jusqu'au mois d'avril. Le programme pouvait être compilé sans erreur, mais l'absence de licence empêchait son exécution. Il y a eu à nouveau quelques jours sans licence, après que la licence d'évaluation est arrivée à terme, et avant que la licence définitive ne soit disponible.

4.1.2 Utilisation d'Ilog Solver et Scheduler

Une fois ce problème réglé, les déficiences de la documentation ont constitué un frein à l'avancement du projet. De nombreuses classes et méthodes ne sont pas documentées, voire citées, dans l'ensemble de la documentation disponible. Certaines ne sont présentes que dans le code source des exemples fournis par Ilog, sans explication sur leur fonctionnement et leur utilisation.

De plus, Ilog a rajouté dans les versions récentes de sa suite d'optimisation une surcouche de modélisation. Le système de contraintes n'est pas directement traité, on procède d'abord à son *extraction* qui permet de le rendre utilisable par le solveur. Cette surcouche, si elle permet de modifier dynamiquement le modèle sans perturber une recherche en cours, a dans notre cas compliqué les choses : les méthodes de plus bas niveau ne sont plus directement accessibles, il faut les *envelopper* afin de les utiliser dans la surcouche du modèle, ce qui affecte la lisibilité du code source.

4.1.3 Refroidissement du métal

La difficulté majeure rencontrée à la fin du stage concerne l'implémentation d'une contrainte empêchant le refroidissement du métal dans les fours. Le métal, une fois fondu, doit être prélevé tant qu'il est à température optimale. Cette contrainte est fondamentale pour la finalisation du projet, car c'est elle qui va induire l'utilisation du four de stockage.

Pour interdire au métal de rester trop longtemps dans la cuve du four, il faut forcer les prélèvements à s'effectuer après un certain délai, une fois la fonte terminée. On ne peut pas poser de contrainte directement sur chaque prélèvement, car un prélèvement n'est pas a priori affecté à une fonte en particulier, le nombre de fontes étant variable.

Ilog Scheduler permet de poser des contraintes entre deux types d'activités pour l'utilisation d'une ressource, les types étant à déterminer par le développeur. On peut donc interdire à une activité de type *a2* d'utiliser une ressource *R* moins de *n* unités de temps après qu'une activité de type *a1* a utilisé cette même ressource. Cela peut se traduire dans notre cas par l'interdiction aux prélèvements d'utiliser un four pendant un certain délai après l'achèvement d'une fonte. Mais il ne semble pas possible de renverser cette contrainte pour interdire l'utilisation d'une ressource *après* un certain délai. Si aucune autre solution n'est trouvée, nous soumettrons ce problème au service d'assistance d'Ilog.

Une autre approche consiste à vérifier que la ressource réservoir correspondant à la cuve d'un four est effectivement vide (à une date correspondant à la fin d'une fonte s'étant effectuée sur ce four, augmentée du délai maximum autorisant le prélèvement du métal), mais on ne peut pas poser de contrainte directement sur le niveau d'un réservoir.

On peut forcer la capacité maximale d'une ressource réservoir à un certain niveau, éventuellement nul, pendant un intervalle de temps défini, ce qui permet de s'assurer qu'un réservoir est vidé à un moment donné. Malheureusement, ce type de contrainte ne peut s'effectuer que sur un intervalle de temps défini par deux constantes : on ne peut donc pas utiliser la date de la fin d'une fonte, qui est une variable.

Finalement, pour vérifier que le réservoir est vide à une certaine date, nous avons introduit des tâches de « vérification » associées à chaque fonte, s'effectuant exactement *délaiMaxi* unités de temps après la fin de la fonte. Ces tâches de vérification produisent *capacitéDuRéservoir* dans la ressource associée, ce qui assure que le réservoir était vide au début de l'activité : sans cela, il serait impossible d'augmenter le niveau d'autant. Ensuite, les activités de vérification vident le même réservoir, pour qu'il retrouve son niveau initial. Cette contrainte reste pour l'instant la seule envisageable sans modification profonde du modèle, mais nécessite des modifications : en l'état, des erreurs pour l'instant inexpliquées de produisent.

4.2 Suite des travaux

4.2.1 Position par rapport à l'échéancier

En raison des difficultés énoncées au paragraphe précédent, le planning établi lors du rapport préliminaire n'a pu être parfaitement respecté :

1- Assimiler le besoin à partir d'un problème réel de réduction des coûts dans une fonderie.
2- Acquérir les connaissances nécessaires à la résolution de ce problème, faire une étude bibliographique des techniques actuelles de planification et d'optimisation.
3- Délimiter le périmètre de l'étude à réaliser
4- Modéliser le problème sous forme de programmation par contraintes
5- Implémenter et valider une maquette de cette modélisation au moyen d'une bibliothèque de programmation par contraintes dédiée (fin avril).
6- Intégrer à la maquette différents algorithmes d'optimisation locale (fin mai)
7- Tester sur des exemples le résultat de l'optimisation ainsi que les temps de résolution (mi-juin).
8- Éventuellement, donner un aperçu de la modélisation des contraintes écartées à l'étape 2.

Les quatre premières étapes ont été effectivement achevées, la cinquième, dont l'achèvement a été repoussé dans le rapport intermédiaire, est toujours en cours. L'optimisation locale a été provisoirement abandonnée, l'heuristique de recherche actuelle donnant directement la solution optimale.

Au cours du dernier mois de stage, nous allons nous concentrer sur le problème du stockage du métal, et l'accélération de la résolution.

4.2.2 Problème du stockage

L'introduction de systèmes de stockage est cruciale dans le projet, c'est potentiellement une bonne source d'optimisation. Pour cela, nous devons résoudre le problème concernant les contraintes vérifiant que le réservoir d'un four a été entièrement vidé moins de *délaiMaxi* intervalles de temps après la fin de la fonte.

En effet, l'introduction elle-même du four de stockage ne pose pas a priori de problèmes et a déjà été prévue : en effet le four de stockage se comporte globalement comme un four de fonte. Sa consommation est nettement inférieure, ce qui permet l'optimisation des dépenses énergétiques : maintenir du métal en fusion à la température requise nécessite beaucoup moins de puissance que porter des lingots de

métal froid à fusion. Il suffira d'ajouter un coefficient à sa variable de coût. Le calcul du coût total n'en sera pas fortement affecté.

Pour pouvoir utiliser le stockage dans notre représentation du processus de production de la fonderie, il faut pouvoir interdire au métal (une fois fondu) d'attendre indéfiniment dans le four d'être prélevé. En effet, une fois la fonte terminée, que le métal soit prélevé ou non, l'alimentation n'est plus utilisée, et donc le coût n'augmente pas. Le métal pourrait alors rester disponible pour les prélèvements, sans dépenser d'énergie. Il n'y aurait par conséquent aucune raison pour le modèle (si ce n'est pour libérer un four) de chercher à utiliser le dispositif de stockage de métal fondu, qui lui, est pris en compte dans le coût.

Les contraintes actuellement posées (voir le paragraphe 4.1.3) ne fonctionnent pour l'instant pas correctement. Nous allons essayer de les débugger, et parallèlement, nous allons réfléchir à d'autres moyens pour implémenter cette contrainte. En effet, même si cette contrainte est corrigée, elle présente un défaut majeur pour la qualité d'écriture du modèle : les tâches de vérification ajoutées et leur comportement par rapport aux cuves ne correspondent à rien de réel.

Nous allons aussi nous intéresser à une autre piste, qui consiste à modifier la modélisation des prélèvements pour les regrouper sous une seule activité par fonte, sur laquelle on pourra poser les contraintes de délai voulues.

4.2.3 Développement d'heuristiques spécifiques

De plus, même si la vitesse de la résolution a été considérablement améliorée au cours du stage, elle est encore insuffisante pour une utilisation sur des instances du problème de taille réelle approchant la rapidité spécifiée dans le cahier des charges. Nous allons donc nous intéresser au développement de notre propre heuristique de recherche, exploitant au mieux les connaissances que nous avons du problème réel. En effet, l'heuristique fournie par Ilog d'affectation d'une ressource parmi un ensemble de ressources équivalentes cherche prioritairement à saturer l'utilisation d'une ressource. Dans notre cas, il vaut mieux au contraire répartir équitablement l'usage de chaque ressource : en pratique, l'heuristique d'Ilog appliquée à l'affectation des fours conduit à utiliser au maximum le temps disponible pour fondre tout le métal sur un seul four (ou un minimum de fours), ce qui maximise la durée totale de l'ordonnancement. Dans la réalité, il vaut mieux prioritairement paralléliser les fontes, pour exploiter au mieux les heures creuses, synonymes de moindre coût.

Pour résoudre ce problème sur les fours, nous avons forcé l'affectation des fours dans le modèle, mais cela n'est pas possible à tous les niveaux (pour l'affectation des poches de coulée notamment) sans perdre des solutions possibles, cette affectation « manuelle » ne pouvant pas être outrepassée. Il faudra donc écrire

une heuristique d'affectation reproduisant en priorité ce comportement, mais sans le forcer absolument.

4.2.4 Amélioration de la qualité

Parallèlement à l'optimisation du modèle mathématique et de la procédure de résolution, nous allons nous attacher pendant les dernières semaines du stage à améliorer la simplicité d'utilisation de l'outil d'optimisation. La sortie devra être plus lisible, le format du fichier d'entrée plus pratique, notamment dans la définition du calendrier qui impose pour l'instant un nombre entier de jours.

Le code source de l'application mérite lui aussi quelques améliorations. Nous allons devoir séparer l'unique fichier source en plusieurs classes, supprimer les variables globales. L'utilisation de classes permettra de séparer nettement la lecture des données, la modélisation elle-même, le noyau de l'application, et l'affichage du résultat. Le fichier de données pourra éventuellement être séparé en plusieurs fichiers, correspondant respectivement aux caractéristiques matérielles d'une fonderie, au calendrier sur lequel doit être effectué l'ordonnancement, et à la représentation de la demande.

Nous pourrons enfin, s'il reste du temps, implémenter quelques contraintes spécifiques de production, afin d'améliorer la qualité du modèle comme représentation d'un problème de gestion de production.

4.2.5 Calcul de bornes

Pour trouver des bornes meilleures que les bornes relativement naïves exploitées lors de la recherche, on peut utiliser la résolution du modèle, mais en relaxant certaines contraintes. Ainsi, en supprimant la fonction de coût, on obtiendra facilement une contrainte de durée minimale. De même, on peut utiliser une fonction de coût constante, de valeur égale au tarif « heures creuses », pour trouver une solution qui minimisera la durée totale des fontes, et donnera une meilleure borne inférieure du coût.

4.3 Conclusion sur les résultats de la recherche

4.3.1 Qualité du modèle

Nous avons réussi à construire un modèle représentant fidèlement les principales caractéristiques de la production de pièces dans une fonderie, au moyen de la programmation par contraintes. Ce modèle fournit une solution de bonne qualité : la recherche n'a jamais donné de solution qui n'aurait pas dû être admissible. La solution optimale est pour l'instant la seule recherchée, mais on pourra lors du développement final du projet s'intéresser à la recherche d'une « bonne » solution, sans chercher à obtenir l'optimum exact. Le système de contraintes n'a pas à être retouché, seule la stratégie de recherche devra être modifiée.

4.3.2 Efficacité de la résolution

On a vu que les problèmes résolus lors des tests nécessitaient souvent plus de 10 secondes de recherche, durée indiquée par EDF comme objectif de performance, et ce, alors que les instances sur lesquelles ces tests portaient étaient souvent de taille relativement modeste.

Néanmoins, les résultats restent encourageants : en utilisant uniquement les heuristiques de recherche proposées par Ilog, les optimisations successives du système ont souvent permis de réduire considérablement la durée de la recherche. De nombreuses pistes n'ont pas encore pu être explorées, qui pourraient, elles aussi, permettre de telles améliorations. Surtout, la réalisation d'une heuristique personnalisée, plus adaptée à la réalité du processus de production, laisse espérer une réduction considérable du nombre d'échecs rencontrés lors de la procédure de recherche de la solution optimale. Enfin, la machine sur laquelle nous avons procédé aux tests est déjà relativement ancienne, et ne comporte qu'un seul processeur. Or l'outil que nous avons développé n'est qu'une maquette, le projet global est très complexe, et ne rentrera pas en production avant longtemps. Non seulement la célèbre loi de Moore fait que les performances des processeurs auront considérablement augmenté, mais Ilog Solver peut être utilisé sur une architecture multiprocesseur, afin d'augmenter encore les performances du système. L'augmentation des capacités des ordinateurs ne peut bien évidemment diminuer la durée de la recherche que d'un facteur constant quel que soit la taille de l'instance donnée, aussi la réalisation d'heuristiques de recherche spécifiques reste la principale source de diminution du temps de calcul.

4.3.3 Avenir du travail effectué

Les différents travaux restant à effectuer énoncés au paragraphe 4.2 peuvent paraître ambitieux par rapport au nombre de semaines de stage restant, mais il ne faut pas perdre de vue que le projet réalisé n'est qu'une maquette, destinée à être reprise et développée. Le modèle que nous avons conçu est évolutif, et les pistes de recherches qui ne pourront pas être exploitées pendant le stage pourront l'être lors du développement final de l'application.

BIBLIOGRAPHIE

(la liste des ouvrages est donnée par ordre alphabétique des auteurs)

- [Baptiste 01] BAPTISTE Philippe. « Overconstrained scheduling problems ». In Université Paris I. *Site des Journées Franciliennes de Recherche Opérationnelle*, [En ligne]
  http://panoramix.univ-paris1.fr/CEMSEM/soutif/JFRO/fichiers/baptiste.pdf
  (Page consultée le 18 janvier 2002)

- [Baptiste 98] BAPTISTE Philippe, « Une étude théorique et expérimentale de la propagation des contraintes de ressources ». In Université de Technologie de Compiègne. Site de Philippe Baptiste, [En ligne]
  http://www.hds.utc.fr/~baptiste/phd.pdf (Page consultée en février 2002)

- [Bistarelli 96] BISTARELLI Stefano, FARGIER Hélène… et al, 1996. *Semiring-based CSPs and Valued CSPs : Basic Properties and Comparison.* Berlin, Allemagne : Springer.

- [Cabon 96] CABON Bertrand, VERFAILLIE Gérard… et al, 1996. « Using Mean Field Methods for Boosting Backtrack Search in Constraint Satisfaction Problems ». In Proc. of ECAI-96, pages 165-169, Budapest, Hongrie.

- [Carlier 88] CARLIER Jacques, CHRÉTIENNE Philippe, 1988. *Problèmes d'ordonnancement (modélisation / complexité / algorithmes)*. Paris, France : Masson.

- [Dago 96] DAGO Pierre et VERFAILLIE Gérard, 1996. « Nogood Recording for Valued Constraint Satisfaction Problems ». In *Proc. of ICTAI-96*, Toulouse, France.

- [Garey 79] GAREY Michael et JOHNSON David, 1979. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company.

- [Ilog 98] ILOG S.A., 1998. *Ilog Solver 4.3 User's Manual*. Paris, France : Ilog. 360p.

- [Ilog 01_1] ILOG S.A., 2001. *Ilog Concert Technology 1.2 User's Manual*. Paris, France : Ilog. 160p.

- [Ilog 01_2] ILOG S.A., 2001. *Ilog Scheduler 5.2 User's Manual*. Paris, France : Ilog. 526p.

- [Ilog 01_2] ILOG S.A., 2001. *Ilog Scheduler 5.2 Reference Manual*. Paris, France : Ilog. 884p.

- [Jussien 01] JUSSIEN Narendra. « Programmation par contraintes avec explications ». In Narendra Jussien. *Site du Dr Narendra Jussien*, [En ligne]. http://www.emn.fr/jussien/publications/jussien-AFPLC01.pdf (Page consultée le 18 janvier 2002)

- [Le Pape 01] LE PAPE Claude. « Utilisation de la programmation par contraintes en planification et ordonnancement : principes et applications ». In Université Paris I. *Site des Journées Franciliennes de Recherche Opérationnelle*, [En ligne]. http://panoramix.univ-paris1.fr/CERMSEM/soutif/JFRO/fichiers/c_le_pape.pdf (Page consultée le 18 janvier 2002)

- [Lesaint 95] LESAINT David. *Calcul d'ensembles de solutions pour problèmes de satisfaction de contraintes*. Thèse de doctorat, ENSAE, Toulouse, France, 1995.

- [Nuijten 01] NUIJTEN Wim, LE PAPE Claude et BAPTISTE Philippe, 2001. *Constraint-based scheduling : applying constraint programming to scheduling problems*. Boston, E.-U. : Kluwer Academic. 198p.

- [Pacte 01_1] PACTE NOVATION. « Dossier Technique : Optimisation Locale ». In Pacte Novation, *Site de Pacte Novation*, [En ligne]. http://www.pactenovation.fr/images/ImagesDivers/optimisation.pdf (Page consultée le 18 janvier 2002)

- [Pacte 01_2] PACTE NOVATION, 2001. « Dossier Technique : Programmation par Contraintes »

.

- [Pinedo 95] PINEDO Michael, 1995. *Scheduling, theory, algorithms, and systems*. Englewood Cliffs, New Jersey, E.-U. : Prentice Hall.

- [Régin 94] REGIN Jean-Charles, 1994. « A Filtering Algorithm for Constraints of Difference in CSPs ». In *Proc. of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, E.-U.

ANNEXES

Nous proposons en annexe :
Un graphique représentant l'évolution du tarif de l'électricité, utilisant notre modélisation (indice EDF multiplié par 25 et arrondi), sur une semaine type, c'est-à-dire ne comportant pas de jours fériés. Les tarifs des jours fériés sont les mêmes que ceux des samedi et dimanche. Une journée tarifaire commence à 2h du matin.

Un exemple de fichier de données, *donnees.txt*

Un exemple de sortie du programme, *sortie.txt*

Le code source commenté du programme *perf.cpp* réalisé.