MASTER EN SCIENCES INFORMATIQUES

Optimisation de l'ordonnancement des tâches d'un atelier robotisé de laboratoire

MARMORO, Massimo

Award date:
2000

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX, NAMUR

INSTITUT D'INFORMATIQUE

RUE GRANDGAGNAGE, 21, B-5000 NAMUR (BELGIUM)

Optimisation de l'ordonnancement des tâches d'un atelier robotisé de laboratoire

Massimo Marmoro

Mémoire présenté en vue de l'obtention du grade de

Licencié en Informatique

Année Académique 1999 - 2000

Résumé

Ce document traite du problème de l'optimisation de l'ordonnancement des tâches d'un atelier robotisé à partir de l'étude d'un cas réel d'un laboratoire d'analyses.

Nous avons suivi, dans ce travail, une démarche en quatre étapes. Au cours de la première étape, nous avons étudié le contexte du problème. Nous avons ensuite, dans la deuxième étape, mis en évidence les difficultés de modélisation de ce problème d'ordonnancement et des contraintes qui lui sont associées. Dans la troisième étape, nous avons exploré différentes méthodes de résolution adaptées à la difficulté du problème. Enfin, au cours de la quatrième étape, nous avons réalisé un prototype en utilisant la programmation logique sous contraintes sur des domaines finis et comparé les résultats obtenus avec ceux obtenus par l'algorithme actuel et une heuristique gloutonne.

Nous concluons par les perspectives offertes par la programmation logique sous contraintes pour résoudre des problèmes réels rencontrés dans les entreprises.

mots clés: Ordonnancement d'atelier, contraintes, méthodes approchées, heuristiques, algorithme glouton, recuit simulé, méthodes exactes, branch and bound, programmation logique sous contraintes sur des domaines finis

Abstract

This paper deals with a robotised jobshop scheduling optimisation problem on a concrete analytical laboratory case basis.

This work follows a four steps approach. The first step studies the problem context. The second step shows the difficulties to find the model descrbing such a jobshop schedule and its linked constraints. The third step explores different solving methods fitted to the problem. Finally, the fourth step implements a prototype using constraints logic programs in finite domains and compares results to those obtained thhrough both current algorithm and greedy heuristic.

We conclude on the future prospects offered by constraints logic programs in finite domaines to solve concrete undertaking problems.

keywords: Jobshop scheduling, constraints, approximation methods, heuristics, greedy algorithm, simulated annealing, exact methods, branch and bound ,Constraints Logic Programs in Finite Domains

Avant-propos

Je tiens à remercier:

Monsieur Jean-Paul Leclercq, promoteur de ce mémoire, pour sa disponibilité, ses critiques judicieuses et ses conseils durant toutes les étapes de ce travail;

Monsieur Jean-Marie Jacquet pour son aide et ses suggestions avisées;

Monsieur Patrick Heymans pour ses conseils et sa grande collaboration;

Ma famille et mes collègues de chez Solvay, qui par leurs encouragements et leur patience, m'ont permis d'aller jusqu'au terme de ces deux années d'études.

Toutes les personnes qui de près ou de loin m'ont aidé à réaliser ce travail, en particulier Bernard Dermine.

Je remercie les membres du Jury de l'intérêt qu'ils voudront bien porter à la lecture de ce travail.

Table des matières

Glossaire

Atelier robotisé (de laboratoire): un atelier robotisé est un ensemble de dispositifs assemblés dans le but de réaliser des travaux de laboratoire (préparations et analyses) de manière autonome

CLP(FD): Constraint Logic Programs in Finite Domains.

Echantillon: un échantillon est une matrice (liquide ou solide) prélevée dans un contenant (flacon ou sachet) à un point de prélèvement.

Fabrication: voir unité de production.

Fonction économique: la fonction économique est une fonction sur l'ensemble des solutions réalisables que l'on cherche à minimiser (ou maximiser) dans un problème d'optimisation. Par exemple, pour notre problème, cette fonction économique est le temps de fin de traitement de tous les échantillons, depuis la station d'entrée jusqu'à la station de sortie de l'atelier robotisé, que l'on cherche à minimiser.

Heuristique: une heuristique est un algorithme qui construit une solution réalisable (pour un problème) sans garantie d'optimalité.

Laboratoire: le laboratoire est une unité de services chargée de réaliser des analyses sur des échantillons prélevés dans les unités de production.

Matrice: la matrice d'un échantillon est son type de produit .

Points de prélèvement: un emplacement est un endroit identifié d'une installation d'une unité de production où un prélèvement d'échantillon est possible.

Ressources: les ressources sont des emplacements (on utilise également le terme de "position") dans une station appartenant à une même ressource type.

Ressources Type: une ressource type est un ensemble de ressources ayant en commun des caractéristiques équivalentes.

Robot: un robot est un bras articulé (sur plusieurs axes) dont l'extrémité , la "main", est une pince ou une seringue, fixe ou interchangeable automatiquement.

SICP: c'est le nom d'un atelier robotisé du laboratoire Mesures Physiques de l'unité de Services Laboratoire de Solvay à Jemeppe-sur-Sambre.

Station: une station est un accessoire en périphérie du robot, accessible par celui-ci, qui réalise une tâche donnée sur un échantillon. Elle a une ou plusieurs ressources types pour accueillir les échantillons.

Tâches: la tâche est l'opération élémentaire subie par un échantillon dans une station.

Traitement (d'un échantillon): le traitement d'un échantillon est une suite de tâches qui doivent être réalisées dans un ordre particulier avant que l'échantillon ne sorte de l'atelier robotisé.

Travail ou Job (d'un échantillon): voir traitement.

Unité de Production (chez Solvay): les unités de production (ou fabrication) sont des unités qui produisent un ou plusieurs produits finis vendables à l'extérieur de l'usine ou en interne entre unités de production.

Unité de Services (chez Solvay): les unités de services sont des unités de support technique aux unités de production.

Introduction

L'objectif de ce travail est de trouver une modélisation du problème d'ordonnancement des tâches d'un atelier robotisé de laboratoire et une méthode permettant de découvrir un ordonnancement optimal minimisant le temps de passage global de tous les échantillons.

Pour répondre à cet objectif, nous avons appliqué une démarche en plusieurs étapes:

1ère étape: Etude de l'existant
2ème étape: Définition du problème
3ème étape: Recherche de méthodes de résolution
4ème étape: Application d'une méthode à notre problème

La structure de ce document suit cette démarche.

Les chapitres 1 et 2 sont consacrés à l'étape de l'étude de l'existant:

Le chapitre 1 présente l'environnement et le contexte industriel du travail proposé. Il présente succinctement la diversité des travaux de laboratoire et l'organisation qui en découle. Enfin, ce chapitre met en évidence les motivations économiques de la robotisation dans les environnements professionnels actuels.

Le chapitre 2 est l'étude de l'existant proprement dite: il introduit, en toute généralité, ce qu'on entend par "atelier robotisé" dans un laboratoire. Dans cette étude, nous décrivons, de manière détaillée, l'atelier robotisé "SICP" du laboratoire Mesures Physiques du Service Laboratoire de Solvay à Jemeppe-sur-Sambre. Cette exemple réel servira de "fil rouge" tout au long de ce travail.

Le chapitre 3 est consacré à l'étape de la définition du problème: nous définissons le problème de l'optimisation de l'atelier robotisé du Laboratoire en mettant en évidence son appartenance à une classe particulière de problème: l'ordonnancement d'atelier de type JOB-SHOP. Nous spécifions, en toute généralité, les contraintes fonctionnelles rencontrées durant l'étude de l'existant du chapitre 2.

Le chapitre 4 est consacré à l'étape de la recherche de méthodes de résolutions: nous cherchons des méthodes pour résoudre notre problème, tel qu'il a été défini au chapitre 3. Nous déterminons d'abord que ce problème appartient à la classe des problèmes NP-difficiles. Pour résoudre un problème NP-difficile, nous avons le choix entre des méthodes approchées (heuristiques gloutonnes, recherche locale et recherche globale) et des méthodes exactes (méthodes arborescentes "branch and bounds", programmation logique sous contrainte,...). Nous envisageons brièvement quelques applications possibles à notre problème de certaines de ces méthodes.

Le chapitre 5 est consacré à l'étape de l'application d'une méthode à notre problème: nous avons essayé d'appliquer une méthode de programmation logique sous contraintes sur des domaines finis (Constraint Logic Programs in Finite Domains, en abrégé CLP(FD)) au problème de l'optimisation de l'atelier robotisé "SICP" du Laboratoire, afin de produire une planification réalisable et optimale.

Nous avons choisi CLP(FD) car il nous a semblé qu'une méthode CLP(FD) modélise avec beaucoup plus d'expressivité les problèmes du monde réel que les autre méthodes. De plus, de nombreux outils CLP(FD) sont maintenant disponibles sur le marché. Deux d'entre eux ont été essayés et ont montré leurs qualités et leurs limites: GNU Prolog et IF/Prolog. L'utilisation de ces outils a montré également la nécessité d'utiliser des prédicats optimisés pour la propagation efficace des contraintes.

Finalement, nous présentons une comparaison des résultats obtenus par l'algorithme actuel, par un algorithme "glouton" et par un programme logique écrit avec IF/Prolog sur un échantillonnage réel de petite taille de l'atelier robotisé SICP.

Nous concluons ce travail en envisageant quelques perspectives.

Chapitre 1: Contexte

Ce chapitre présente l'environnement et le contexte industriel du travail proposé. Il présente succinctement la diversité des travaux de laboratoire et l'organisation qui en découle. Enfin, ce chapitre met en évidence les motivations économiques de la robotisation dans les environnements professionnels actuels.

1.1. Présentation de l'entreprise

1.1.1. Le groupe Solvay

L'entreprise Solvay a été fondée en 1869 par un industriel Belge, Ernest Solvay.

C'est aujourd'hui un groupe multinational constitué de 327 usines et sites dans 47 pays, qui occupe 33104 employés dans le monde (au 1er janvier 1999).

Son chiffre d'affaires a été de 8,7 milliards de USD en 1998.
L'entreprise est divisée en quatre grands secteurs d'activités:

- Secteur Chimique
- Secteur Plastique
- Secteur Transformation
- Secteur Pharmaceutique

La groupe Solvay possède plusieurs usines en Belgique dont une à Jemeppe-sur-Sambre.

1.1.2. L'usine de Jemeppe-sur-Sambre.

L'usine de Jemeppe-sur-Sambre produit, à partir de sel et de produits pétroliers, les produits finis suivants: sel alimentaire, soude caustique, chlore et dérivés (eau de Javel,..), PVC et eau oxygénée. Elle emploie plus ou moins 1000 personnes.

L'usine est structurée en deux grands types d'entités: les unités de production et les unités de services:

- Les unités de production ont comme finalité un ou plusieurs produits finis vendables à l'extérieur de l'usine ou en interne entre unités de production: on dit dans ce cas que les unités sont intégrées, c'est à dire que le produit fini de l'une est la matière première de l'autre.

- Les unités de services sont des unités de support technique aux unités de production.

1.2. Le laboratoire

1.2.1. Rôle du laboratoire

Afin de garantir la qualité de leurs produits et le bon fonctionnement des installations, chaque unité de production convient d'un plan de contrôle avec le laboratoire, une unité de services de l'usine.

Pour une unité de production donnée, le plan de contrôle décrit, par emplacement de l'installation, les déterminations (c'est-à-dire les grandeurs à mesurer) que le responsable souhaite que le laboratoire contrôle afin de s'assurer du fonctionnement correct de son installation. On y trouve également d'autres informations comme la périodicité des mesures (la fréquence des contrôles), le délai de réponse, les spécifications des valeurs (c'est-à-dire l'intervalle dans lequel les valeurs mesurées sont considérées comme normales). Il est négocié annuellement par le laboratoire et chaque unité de production.

1.2.2. Travaux du laboratoire

Pour répondre à la demande des responsables des unités de production dans les délais demandés, les responsables du laboratoire ont divisé le travail en plusieurs étapes:

Le prélèvement

Cette étape consiste à aller prélever des échantillons dans les unités de production. Un échantillon est une matrice (liquide ou solide) prélevée dans un contenant (flacon ou sachet) à un emplacement identifié de l'installation d'une unité de production (le point de prélèvement): par exemple, à un robinet sur une tuyauterie, dans un réservoir....

La préparation

Lorsque l'échantillon a été réceptionné au laboratoire, une opération de préparation est nécessaire pour permettre la réalisation de l'étape d'analyse. Par exemple, on peut filtrer l'échantillon, le diluer,...

L'analyse

L'analyse consiste à obtenir les déterminations demandées par l'unité de production en appliquant sur l'échantillon diverses techniques analytiques à l'aide d'appareillages et méthodes plus ou moins sophistiqués. Exemple: la mesure de pH est la valeur (la réponse) obtenue par un appareil de mesure appelé pH-mètre dont l'électrode est plongée dans l'échantillon.

La validation et la transmission des résultats:

Cette dernière étape consiste à s'assurer de la validité des résultats obtenus par les analyses sur base de l'état de fonctionnement des appareils utilisés, des méthodes d'analyses utilisées et de la connaissance que le laboratoire a de l'état de marche de l'installation contrôlée. Si tel est le cas, les résultats des déterminations sont transmis alors aux responsables de l'unité de production concernée.

1.2.3. Organisation du laboratoire

Pour réaliser ces différentes phases, le personnel chimiste du laboratoire a été organisé en deux types de groupes: les groupes d'interface (GRI) et les groupes d'appui (GRA).

Les groupes d'interface

Les groupes d'interface sont des groupes en contact direct avec les unités de production.

- ils y prélèvent les échantillons;
- ils réceptionnent ces échantillons;
- ils réalisent des analyses "simples";
- ils réalisent les préparations pour les groupes d'appui;
- ils transmettent les résultats d'analyses aux unités de production;

Les groupes d'appui

Les groupes d'appui sont des groupes qui ne sont pas en contact avec les unités de production.

- ils réalisent des analyses techniquement "complexes";
- ils transmettent leurs résultats aux groupes d'interface demandeurs;

1.3. L'automatisation dans le laboratoire et sa motivation

Comme expliqué ci-dessus, le travail du laboratoire consiste à analyser des échantillons et d'en rendre les résultats dans un délai donné. Il faut donc que le laboratoire soit dimensionné en conséquence (personnel et appareillage) afin de tenir les délais demandés par les unités de production.

Les décideurs du laboratoire ont fait les constats suivants:

- les réalités économiques imposent aux entreprises la maîtrise des coûts à tous les niveaux afin de rester concurrentiel: elles cherchent donc à optimiser le travail avec, il est vrai, le moins de personnel possible.
- certaines analyses se font en grand nombre avec des préparations connues et relativement simples.

Sur base des ces constatations, les décideurs du laboratoire ont pris le parti d'automatiser certains travaux dans des ateliers robotiques (voir chapitre 2, "Etude de l'existant") .

Mais les ateliers robotisés sont chers et moins flexibles que des opérateurs: il faut donc les utiliser au mieux:

- on ne les utilise que pour des préparations et des analyses sur des échantillons connus, pour autant que l'automatisation soit techniquement réalisable et que la fréquence d'arrivée au laboratoire de l'échantillon soit significative.
- il faut que le travail à l'intérieur de ces ateliers robotisés soit optimisé afin d'augmenter leur productivité et d'assurer la remise des résultats dans les délais requis.

Chapitre 2: Etude de l'existant

L'introduction de ce chapitre présente en toute généralité ce qu'on entend par "atelier robotisé" dans un laboratoire. Puis, il décrit de manière détaillée l'atelier robotisé SICP du laboratoire Mesures Physiques du Service Laboratoire de Solvay à Jemeppe-sur-Sambre. Cette exemple réel servira de "fil rouge" tout au long de ce travail : dans le chapitre 3, il nous permettra d'introduire le problème de l'optimisation et les conditions de fonctionnement qui contraignent celui-ci.

2.1. Introduction

Un atelier robotisé est un ensemble de dispositifs assemblés dans le but de réaliser des travaux de laboratoire (préparations et analyses) de manière autonome.

Un atelier robotique se compose des éléments suivants:

- un superviseur (programme et hardware)
- une base de données
- un point d'entrée
- un ou plusieurs points de sortie
- un robot
- plusieurs stations

Examinons en détail ces différents éléments:

Le programme "superviseur" est une application dont le but principal est de gérer la succession des étapes (les tâches) du traitement des échantillons (voir paragraphe 2.2) dans l'atelier depuis leur entrée jusqu'à leur sortie. A tout moment, le programme superviseur a une connaissance de l'état de tous les autres éléments de l'atelier.

La base de données est l'ensemble des connaissances que le programme "superviseur" a à sa disposition pour réaliser les manipulations sur les échantillons: on y décrit exactement les tâches à réaliser pour chaque type d'échantillons. Un atelier robotisé ne peut donc traiter que des échantillons connus (= décrit dans la base de données) au moment de leur entrée dans le système. Cette base de données est le savoir-faire de l'atelier robotique.

Le point d'entrée est une zone (généralement un rack) à partir de laquelle les échantillons sont introduits dans le système : chaque échantillon y est connu par l'application "superviseur".

Le point de sortie est une zone (un bac, une poubelle,...) où finissent les échantillons dont le traitement est terminé.

Un robot est un bras articulé (sur plusieurs axes) dont l'extrémité , la "main", est une pince ou une seringue, fixe ou interchangeable automatiquement. Il est actionné à l'aide de commandes transmises par le programme superviseur à un contrôleur gérant tous les paramètres physiques des moteurs (un par axe) du robot. Il manipule les échantillons d'une station à l'autre.

Une station est un accessoire en périphérie du robot, accessible par celui-ci, qui réalise une tâche donnée sur un échantillon. La station est soit simple et totalement asservie au programme supervision, ou soit "intelligente", c'est-à-dire qu'une fois l'échantillon placé dans la station, celle-ci reçoit du programme supervision l'ordre de démarrage. Elle réalise alors, de manière autonome, une opération (la tâche). Lorsque celle-ci est terminée, la station renvoie un signal au programme superviseur qui prendra alors la suite du traitement de l'échantillon dans une autre station. Certaines stations peuvent accueillir plusieurs échantillons simultanément: elles disposent d'une zone tampon d'entrée-sortie d'échantillons, dont l'ordre de traitement est géré par le programme superviseur.

2.2. Echantillons traités par l'atelier robotisé.

2.2.1. Identification

Rappel: Un échantillon est un flacon contenant un fluide prélevé dans une unité de production.

On identifie tout échantillon par deux informations:

- le type, c'est-à-dire la classe générale à laquelle appartient l'échantillon; classe dont les caractéristiques suivantes sont prédéfinies à l'avance:
  - la matrice (le type de produit: acide, eau, saumure,...)
  - les attributs de traitements (temps de refroidissement...)
  - l'emplacement où il est prélevé dans l'unité de production
  - le type de contenant (dimensions et volume)
  - le degré d'urgence, c'est-à-dire un critère de rapidité de réalisation du traitement de l'échantillon (0,1 ou 2: 0= le plus urgent; 2= le moins urgent)
- les informations relatives à son prélèvement (date de prélèvement,...)

Cet atelier ne traite que des échantillons liquides contenus dans des flacons de 100, 250, 500 et 1000 ml. On peut voir ces flacons à la figure 2.1.

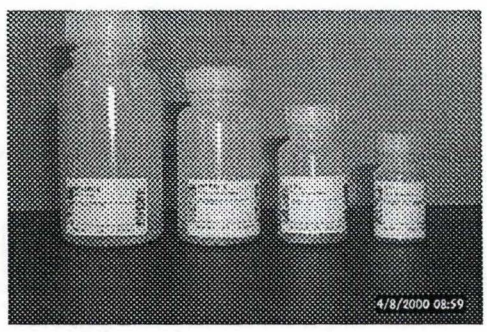


Figure 2.1 Flacons traités par l'atelier robotique SICP

2.2.2. Arrivée des échantillons au laboratoire

L'arrivée des échantillons au laboratoire se fait par flots en fonction des tournées de prélèvement correspondant à peu près aux degrés d'urgence (les échantillons les plus urgents sont prélevés en premier). L'arrivée du flot est aléatoire ainsi que son contenu exact.

2.2.3. Quantification

Le tableau 2.1. quantifie le nombre d'échantillons et le nombre de tâches traitées au maximum par l'atelier robotique SICP.

Tableau 2.1. Quantification des échantillons et des tâches par jour et urgence

Jour	Urgence	Nbre Ech.	Refroidissement	Dévissage	Densité	Pompage	Revissage	Terminus	Nbre Tâches
	0	15	5	13	6	11	13	15	63
LUNDI	1	16	16	12	1	11	12	16	68
	2	25	18	9	1	8	9	25	70
sous-total		56	39	34	8	30	34	56	201
	0	10	1	10	6	8	10	10	45
MARDI	1	8	4	5	1	4	5	8	27
	2	38	15	23	8	22	23	38	129
sous-total		56	20	38	15	34	38	56	201
	0	18	4	17	10	15	17	18	81
MERCREDI	1	6	6	2	1	1	2	6	18
	2	54	34	28	0	28	28	54	172
sous-total		78	44	47	11	44	47	78	271
	0	12	2	12	7	9	12	12	54
JEUDI	1	15	5	12	2	10	12	15	56
	2	26	19	11	2	11	11	26	80
sous-total		53	26	35	11	30	35	53	190
	0	14	4	13	6	11	13	14	61
VENDREDI	1	5	5	1	1	0	1	5	13
	2	22	12	9	0	9	9	22	61
sous-total		41	21	23	7	20	23	41	135

2.3. Description des stations

Cette section décrit toutes les stations disponibles dans l'atelier robotique SICP.
Le schéma de la disposition physique de l'atelier robotique SICP se trouve à la fin de cette section (figure 2.4).

2.3.1. La station d'entrée ("rack in")

La station d'entrée est une armoire de quatre étages contenant des emplacements identifiés et numérotés de 1 à 44.
Chaque emplacement est spécialisé pour un et un seul type de flacon:

- 14 emplacements pour des flacons de 100 ml (de n°1 à 14)
- 22 emplacements pour des flacons de 250 ml (de n°15 à 36)
- 4 emplacements pour des flacons de 500 ml (de n°37 à 40)
- 4 emplacements pour des flacons de 1000 ml (de n°41 à 44)

Un échantillon est introduit dans le système lorsque :

- l'échantillon est placé dans une position du "rack in"
- le système est informé par l'opérateur qu'à la position x se trouve un échantillon identifié dont le plan de travail est connu et décrit dans la table des traitements (voir 2.5. ci-dessous)

2.3.2. La station refroidisseur ("cooler")

Le refroidisseur est une armoire frigo où les échantillons sont placés par le robot dans des emplacements identifiés et numérotés de 1 à 24.

Chaque emplacement est spécialisé pour un et un seul type de flacon:

- 12 emplacements pour des flacons de 100 ml (de n°1 à 12)
- 8 emplacements pour des flacons de 250 ml (de n°13 à 20)
- 4 emplacements pour des flacons de 500 ml (de n°21 à 24)

Remarque: Il n'y a pas d'emplacement prévu pour des flacons de 1000 ml. Ceux-ci ne peuvent donc pas être refroidis.

Le temps de refroidissement varie de 1200 sec à 2400 sec. Il dépend du type de flacon et de la température initiale de l'échantillon. Par convention, il dépend du type d'échantillon.

2.3.3. Les stations de dévissage / revissage ("decap" 1 à 2)

Les stations de dévissage / revissage sont des stations qui réalisent deux opérations complémentaires: dévissage du bouchon du flacon d'un échantillon et le revissage du même bouchon sur le même flacon.

Chaque station de dévissage est spécialisée dans deux types de flacons:

- decap1: ne dévisse (revisse) que des flacons de 100 et 250 ml
- decap2: ne dévisse (revisse) que des flacons de 500 et 1000 ml

Une station de dévissage / revissage peut conserver au plus 6 bouchons (emplacements prévus à cet effet dans la station numérotés de 1 à 6).

Le temps de dévissage (ou de revissage) est une constante quelque soit le type de flacon: il est de 40 sec.

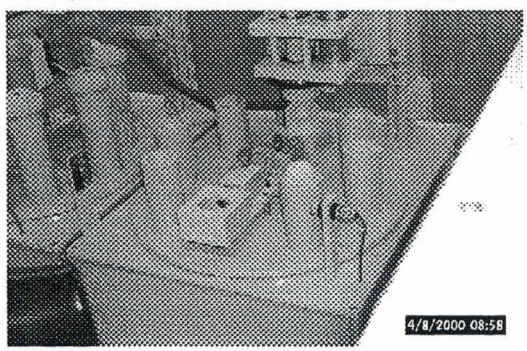


Figure 2.2 Station de dévissage

2.3.4. Les stations de pompages ("pomp" 1 à 6)

Une station de pompage est un dispositif automatique qui réalise une méthode de préparation en fonction de paramètres dépendant du type d'échantillon qui a été placé dans la station par le robot. Le flacon déposé est toujours débouché au préalable.

Chaque station de pompage est dédicacée pour une ou plusieurs matrices analytiques. Le but est d'éviter ainsi les contaminations chimiques inter-matrices incompatibles, ou d'éviter des problèmes dus à des limites techniques des équipements (pompe, tuyaux,...).

Actuellement:

- "Pomp1" accueille les échantillons dont la matrice est "ACIDE"
- "Pomp2" et "Pomp5" accueillent les échantillons dont la matrice est "EAU PROPRE" ou "EAU CONTAMINEE"
- "Pomp3" accueille les types d'échantillons dont la matrice est "EAU OXYGENEE"
- "Pomp4" accueille les échantillons dont la matrice est "SAUMURE"
- "Pomp6" accueille les échantillons à traiter pour détermination de la teneur en "MERCURE"

Une station de pompage peut accueillir tous les types de flacons, mais un seul à la fois.

Le temps de préparation dans une station de pompage varie de 360 sec à 600 sec. Il dépend de la méthode de préparation et d'autres critères propres à chaque type d'échantillon (viscosité,...). Par convention, ce temps dépend du type d'échantillon.

Remarque: bien qu'il ne soit pas nécessaire de dédicacer explicitement une station de pompage pour un échantillon donné, car la station de pompage pourrait se déduire de la matrice (unique) de l'échantillon, le système actuel ne le permet pas. On a donc imposé une station de pompage à chaque type d'échantillons nécessitant un pompage. L'attribut "numéro de la station utilisée" dépend donc du type d'échantillons

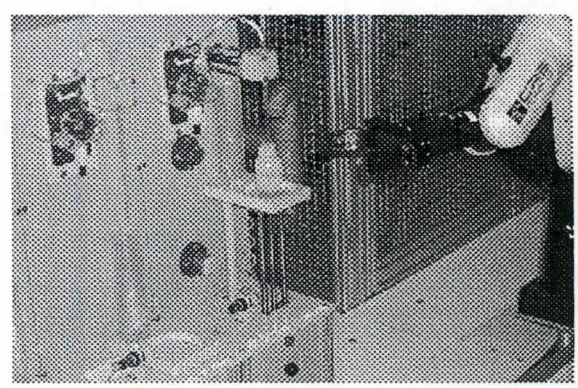


Figure 2.3 Station de pompage

2.3.5. La station "densimètre"

La station "densimètre" réalise la détermination de la densité de l'échantillon déposé par le robot. Le flacon déposé est toujours débouché au préalable.

Le temps de mesure d'une densité est, par convention, une constante quelque soit le type d'échantillon: il est de 300 sec.

2.3.6. La station de sortie

La station de sortie est une armoire de 4 tiroirs superposés ayant chacun 4 couloirs. Il permet aux opérateurs de récupérer les échantillons dont le traitement est terminé. Il y a donc 16 couloirs numérotés de 1 à 16.

Lorsque l'échantillon a subi toutes les tâches prévues dans son traitement, le robot place celui-ci dans un des couloirs de la station de sortie où l'opérateur viendra le reprendre au moment qu'il jugera opportun. Cette station est donc considérée comme ayant une capacité virtuelle illimitée.
Le couloir de la station de sortie dépend du type d'échantillon.

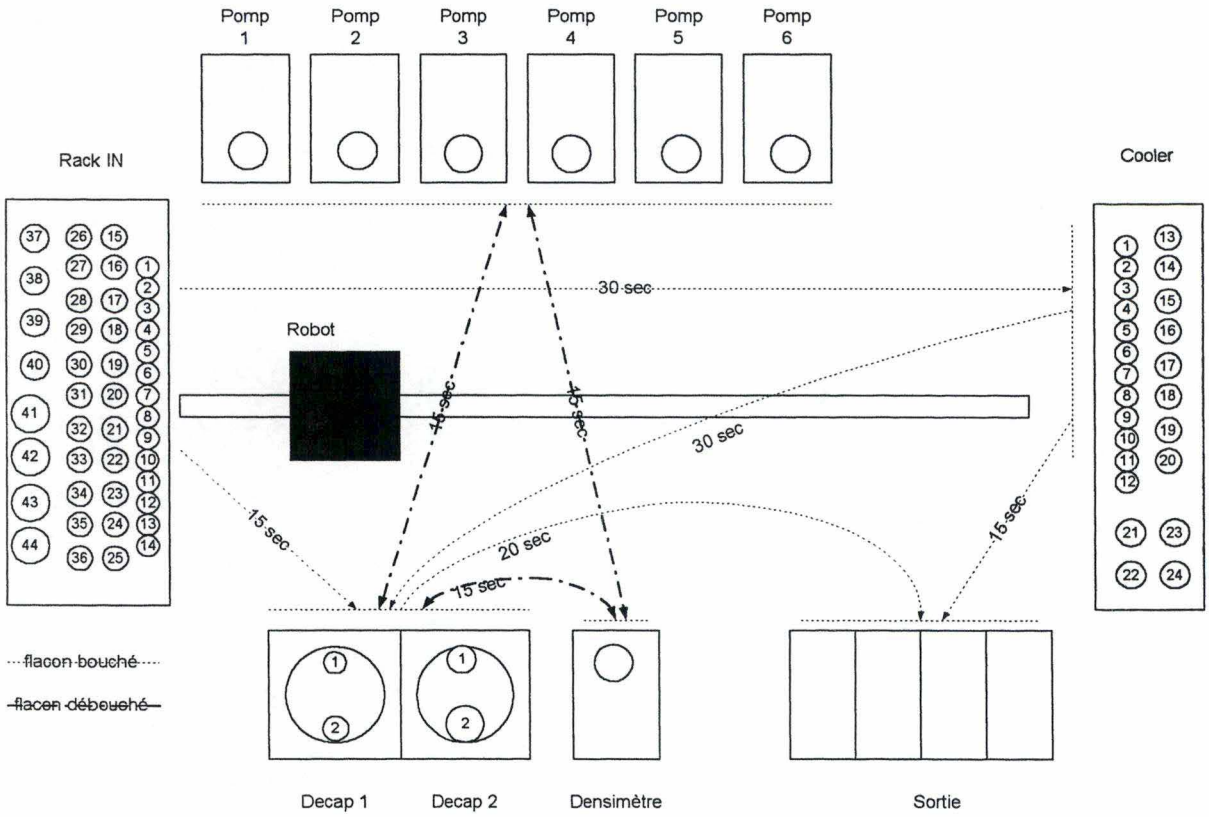


Figure 2.4 Schéma de l'atelier robotique SICP

2.4. Contraintes fonctionnelles

Sur base de l'existant, on peut énoncer les contraintes fonctionnelles suivantes:

2.4.1. tout échantillon est toujours introduit dans le système par le "rack in"
2.4.2. une tâche d'un échantillon n'est réalisée que dans une et une seule station
2.4.3. tout échantillon devant subir la tâche de refroidissement est placé de la station d'entrée ("rack in") dans la station refroidisseur ("cooler") avant tout autre tâche
2.4.4. un échantillon est toujours bouché lorsqu'il entre dans la station refroidisseur (également lorsqu'il en sort, forcément)
2.4.5. la station refroidisseur peut réaliser la tâche de refroidissement de plusieurs échantillons simultanément
2.4.6. un échantillon ne peut être placé dans la station refroidisseur que s'il y a encore un emplacement disponible pour son type de flacon
2.4.7. il n'y a pas d'emplacement prévu dans la station "cooler" pour des flacons de 1000 ml: on ne peut donc jamais refroidir un flacon de 1000 ml
2.4.8. un échantillon n'est placé qu'une et une seule fois dans le "cooler" au cours du travail prévu pour son traitement
2.4.9. un échantillon est toujours débouché dans la station de dévissage supportant son type de flacon
2.4.10. une station de dévissage ne dévisse qu'un seul échantillon à la fois
2.4.11. une station de dévissage ne peut conserver que 6 bouchons maximum.
2.4.12. un échantillon devant subir les tâches de pompage et de densité ne peut être rebouché entre la station de pompage et la station de densité
2.4.13. si un échantillon doit subir les tâches de pompage et de densité, le temps de succession entre celles-ci doit être le plus court possible ("safety requirement")
2.4.14. un échantillon doit être toujours débouché avant d'être placé dans la station "densimètre" ou dans une station de pompage "pompX".
2.4.15. un échantillon n'est placé que dans une station de pompage acceptant sa matrice analytique
2.4.16. une station de pompage ne prépare qu'un seul échantillon à la fois.
2.4.17. un échantillon n'est préparé qu'une seule fois dans une et une seule station de pompage au cours du travail prévu
2.4.18. un échantillon n'est placé qu'une et une seule fois dans la station "Densimètre" au cours du travail prévu
2.4.19. la station de densité ne réalise l'analyse que d'un seul échantillon à la fois
2.4.20. tout échantillon termine son parcours à travers le système dans un couloir du bac sortie déterminé par son type d'échantillon
2.4.21. tout échantillon doit être rebouché avant d'être mis dans un couloir de la station de sortie
2.4.22. les échantillons d'urgence 0 doivent être traités avant ceux d'urgence 1, eux-mêmes traités avant ceux urgence 2

Malgré toutes ces contraintes de fonctionnement, des degrés de libertés existent dans le système. Nous les aborderons dans la critique de l'existant (2.9.).

2.5. Table des traitements

Le traitement (= le travail) d'un échantillon est constitué d'une succession de tâches dans un ordre pré-établi ou aléatoire (suivant le cas), qui doivent obligatoirement être réalisées pour que l'échantillon puisse sortir de l'atelier robotique.

La table des traitements est un tableau qui décrit le traitement de chaque échantillon.(Le tableau 2.2. est un extrait de cette table. Voir annexe A pour la table complète).

Cette table présume certaines règles:

- chaque ligne de cette table représente le traitement d'un échantillon.
- la tâche "refroidissement" est réalisée dans la station "refroidisseur".
- la tâche "refroidissement" précède toujours les tâches de dévissage, de pompage , de "densité" et de revissage si elles existent.
- la tâche "pompage" est réalisée dans une station de pompage dont le numéro est donné
- la tache "densité" est réalisée dans la station "densimètre".
- une tâche implicite de dévissage dans la station de "dévissage / revissage" compatible avec le type de flacon de l'échantillon précède toujours une tâche de pompage et /ou de densité. Sa durée est de 40 secondes.
- une tâche implicite de revissage dans la station de "dévissage / revissage" compatible avec le type de flacon de l'échantillon précède toujours la tâche "aller au terminus" lorsqu'au moins une des deux tâches "pompage" ou "densité" existe. Sa durée est également de 40 secondes.
- le traitement se termine toujours par une tâche "aller au terminus" dans la station de sortie (terminus): c'est donc le déplacement du flacon de la station courante vers un couloir prédéfini de la station de sortie.

Les temps de travail exacts des tâches (c'est à dire la durée d'accomplissement de la tâche dans la station) sont repris également dans cette table.

Tableau 2.2. Table des traitements (extrait de l'annexe 1): journée du lundi urgence 0

Nom	Code	Urg	Flacon	Refroidir	Tps Refroid. (sec)	Densité	Tps densité (sec)	Matrice	Pompe	Tps Pomp (sec)	Couloir Sortie
A030/E2/1/1111	26	0	PLT500	FAUX		VRAI	360		0		5
C115/E1/1/1000	123	0	PLT500	VRAI	2400	FAUX			0		3
C115/E3_A/1/1111	124	0	PLT100	VRAI	1200	FAUX		SAUMURE	4	600	5
C115/E3_B/1/1111	698	0	PLT100	VRAI	1200	FAUX		SAUMURE	4	600	5
C425/E1/1/1111	146	0	PLT250	VRAI	1800	VRAI	360		0		3
G330/E1/1/1111	308	0	PLT100	FAUX		FAUX		ACIDE	1	460	8
G410/E1/1/1111	309	0	PLT100	FAUX		FAUX		ACIDE	1	460	8
G530/E.1/1/1111	312	0	PLT100	FAUX		FAUX		ACIDE	1	460	8
G751/E.1/1/1111	627	0	PLT250	FAUX		VRAI	360	ACIDE	1	460	8

2.6. Le robot

Nous appelons "robot" un bras articulé sur 6 degrés de liberté (6 axes) de la firme CRS Canada, voyageant sur un rail, et muni d'une pince à son extrémité (la "main"). On peut en trouver les spécifications techniques à l'annexe B.

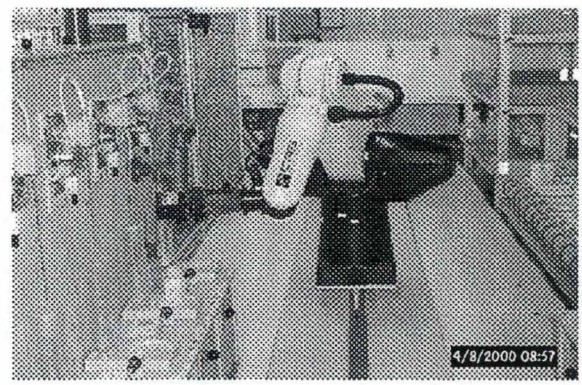


Figure 2.5 Le robot: un bras articulé CRS

Le robot peut être vu comme une ressource unique qui réalise des tâches de déplacement entre les stations. Cette tâche n'est pas représentée dans la table des traitements car elle est implicite. Le robot ne déplace qu'un échantillon à la fois.

Le tableau 2.3 reprend les déplacements autorisés par le robot entre les stations ainsi que leurs temps respectifs.

Tableau 2.3: Mouvements autorisés du robot				Temps
Station	à vide	avec échantillon	Station	(sec)
Rack In	⇔	⇒	Cooler	30
Rack In	⇔	⇒	Decap (1 ou 2)	15
Cooler	⇔	⇒	Decap (1 ou 2)	30
Cooler	⇔	⇒	Sortie	15
Decap (1 ou 2)	⇔	⇒	Sortie	20
Decap (1 ou 2)	⇔	⇔	Pomp (1 à 6)	15
Decap (1 ou 2)	⇔	⇔	Densité	15
Pomp (1 à 6)	⇔	⇔	Densité	15

Sens du mouvement:

⇔ signifie que le mouvement est autorisé dans les deux sens

⇒ signifie que le mouvement n'est autorisé que de la station de gauche vers la station de droite

Remarque: il n'y a pas de différence entre les durées des mouvements "à vide" et celles des mouvements avec échantillon.

2.7. Heuristique actuelle de sélection des échantillons

Actuellement, le programme "superviseur" utilise un algorithme pour choisir les échantillons qu'il va traiter. Cet algorithme est une heuristique dont la philosophie générale est de construire une solution réalisable sans rechercher un optimum global. Elle cherche, en choisissant parmi les tâches en attente de station, à optimiser la meilleure occupation possible des stations, en donnant la

priorité aux échantillons qui ont déjà au moins une tâche terminée. Cet heuristique fonctionne donc de manière "événementielle" guidée par les événements (statuts) qui lui sont retournés par les stations.

Nous avons reconstituée, en pseudo-code, cette heuristique par retro-ingénierie du fonctionnement actuel du système.

2.7.1. Introduction

Cette heuristique est constituée d'un corps principal et de procédures auxiliaires.
L'état d'un échantillon ou d'une position d'une ressource est connu grâce à une propriété: le "statut"

- le statut d'un échantillon peut prendre les valeurs :
  - "en attente" : l'échantillon n'a pas encore commencé son traitement
  - "en cours": la tâche courante est en cours de réalisation
  - "fini": une tâche courante du traitement de l'échantillon est terminée
  - "terminé": tout le traitement de l'échantillon est terminé
- le statut d'une ressource peut prendre les valeurs :
  - "occupée par" e: la position de la ressource est occupée par l'échantillon e
  - "libre": la position de la ressource est libre

Pour des raisons de lisibilité, les deux paragraphes suivants sont écrit en pseudo code:

- une expression du type "e.statut" signifie qu'on fait appel à la propriété "statut" de l'objet "e"
- ":=" entre deux termes signifie l'affectation du terme de droite vers le terme de gauche

2.7.2. Corps principal

soit E1 l'ens. des échantillons qui ont une tâche terminée = { e ∈ Echantillons : e.statut = fini }
soit E2 l'ens. des échantillons qui sont en attente = { e ∈ Echantillons : e.statut = en attente }

Au départ: E1 = ∅ et E2 = {l'ens. de tous les échantillons dans la station d'entrée }
tant que (E1 ∪ E2) ≠ ∅ faire
    ' on essaie d'abord de continuer les traitements d'échantillons déjà commencés
    Répéter
        ' "modifié" est un drapeau qui est mis à vrai si le traitement de 'e1' a pu être continué
        modifié := faux
        liste1 := E1
        tant que liste1 ≠ ∅ faire
            choisir au hasard e1 ∈ liste1
            modifié := Essayer(e1)
            liste1 := liste1 \ {e1}
            si e1.statut = terminé alors E1 := E1 \ {e1}
        fin
    tant que modifié = vrai *' on recommence tant qu'une modification a été possible*
    ' on essaie en suite de commencer le traitement des échantillons en attente
    liste2 := E2
    modifié := faux
    tant que liste2 ≠ ∅ et modifié = faux faire
        choisir au hasard e2 ∈ liste2
        modifié := Essayer(e2)
        ' si modifié = vrai, on a commencé le traitement de e2
        si modifié alors
            E2 := E2 \ {e2}
            si e2.statut ≠ terminé alors E1 := E1 ∪ {e2}
        liste2 := liste2 \ {e2}
        ' on arrête la boucle si le traitement d'un échantillon a commencé
    fin
fin

2.7.3. Fonction et procédures auxiliaires

La fonction Essayer a pour but d'essayer de faire "avancer" le traitement d'un échantillon. Elle retourne la valeur vrai si le traitement de l'échantillon a "avancé", sinon faux.

```
fonction Essayer (e: échantillon): booléen
début
  retour := faux
  si e.tâche_suivante = 'Refroidissement' alors
    ' s'il y a une position libre dans le refroidisseur
    si ∃ p ∈ positions du Refroidisseur telles que
                Refroidisseur(p).taille_flacon = e.taille_flacon ∧ Refroidisseur(p).statut = libre alors
            ' on réserve la position de destination dans le refroidisseur
            Refroidisseur(p).statut =  occupée par e
            e.statut := en cours
            Déplacer l'échantillon e vers Refroidisseur(p)
            ' on libère la position de départ de l'échantillon
            (e.position_actuelle).statut := libre
            ' on met à jour la position actuelle de l'échantillon
            e.position_actuelle := Refroidisseur(p)
            Démarrer la tâche refroidissement
            retour := vrai
    fin
  fin
  si e.tâche_suivante = 'Densité' ∧ Densimètre(1).statut = libre alors
            ' on réserve la station de destination,
            ' c'est à dire la position unique (1) de la station densimètre
            Densimètre(1).statut =  occupée par e
            e.statut := en cours
            ' un échantillon doit être dévissé avant d'être mis sur la station de densité
            si e.dévissé = faux alors
                Dévisser(e)
            fin
            Déplacer l'échantillon vers Densimètre
            ' on libère la position de départ de l'échantillon
            (e.position_actuelle).statut := libre
            ' on met à jour la position actuelle de l'échantillon
            e. position_actuelle := Densimètre(1)
            Démarrer la tâche de densité
            retour := vrai
fin
si e.tâche_suivante = 'Pompage' ∧ Pomp_X(1).statut = libre alors
            ' on réserve la station de destination, c'est à dire la position unique (1) de la station
            '  de pompage où X est le numéro de la station de pompage assignée à e
            Pomp_X(1).statut =  occupée par e
            e.statut := en cours
            ' un échantillon doit être dévissé avant d'être mis sur la station de pompage
            si e.dévissé = faux alors
                Dévisser(e)
            fin
            Déplacer l'échantillon vers Pomp_X(1)
            ' on libère la position de départ de l'échantillon
            (e.position_actuelle).statut := libre
            ' on met à jour la position actuelle de l'échantillon
            e. position_actuelle := Pomp_X(1)
            Démarrer la tâche de pompage
            retour := vrai
            fin
fin
```

```
si e.tâche_suivante = 'Terminus' alors
            e.statut := en cours
            ' un échantillon doit être revissé avant d'être mis dans la station de sortie
            si e.devisé = vrai alors
                  Revisser (e)
            fin
            Déplacer l'échantillon vers Sortie(e.couloir)
            ' on libère la position de départ de l'échantillon
            (e.position_actuelle).statut := libre
            ' on met à jour le statut de l'échantillon
            e. statut := terminé
            retour := vrai
            fin
 fin
 retourner la valeur de retour
fin
```

Les deux procédures suivantes décrivent en pseudocode le dévissage et le revissage d'un échantillon

```
procédure Dévisser (e: échantillon)
début
        ' il faut attendre que la station de dévissage / revissage soit libre
        attendre tant que Decap(e.taille_flacon).statut ≠ libre
        ' on réserve la station de dévissage / revissage
        Decap(e.taille_flacon).statut := occupée par e
        Déplacer l'échantillon vers Decap(e.taille_flacon)
        ' on libère la position de départ de l'échantillon
        (e.position_actuelle).statut := libre
        ' on met à jour la position actuelle de l'échantillon
        e. position_actuelle := Decap(e.taille_flacon)
        Démarrer la tâche de dévissage
        attendre tant que Decap(e.taille_flacon).statut ≠ fini
        e.dévissé = vrai

fin

procédure Revisser (e: échantillon)
début
        ' il faut attendre que la station de dévissage / revissage soit libre
        attendre tant que Decap(e.taille_flacon).statut ≠ libre
        ' on réserve la station de dévissage / revissage
        Decap(e.taille_flacon).statut := occupée par e
        Déplacer l'échantillon vers Decap(e.taille_flacon)
        ' on libère la position de départ de l'échantillon
        (e.position_actuelle).statut := libre
        ' on met à jour la position actuelle de l'échantillon
        e. position_actuelle := Decap(e.taille_flacon)
        Démarrer la tâche de revissage
        attendre tant que Decap(e.taille_flacon).statut ≠ fini
        e.dévissé = faux
fin
```

Remarque: lorsqu'une tâche d'un échantillon est démarrée sur une station, celle-ci devient autonome. Dès que la tâche est finie, c'est la station qui change le statut de l'échantillon (e.statut := fini)

2.8. Schéma conceptuel des données

Le schéma 2.6 est le schéma conceptuel des données selon le modèle Entité-Association de l'atelier robotisé SICP créé à partir de l'étude de l'existant décrit dans les chapitres précédents.

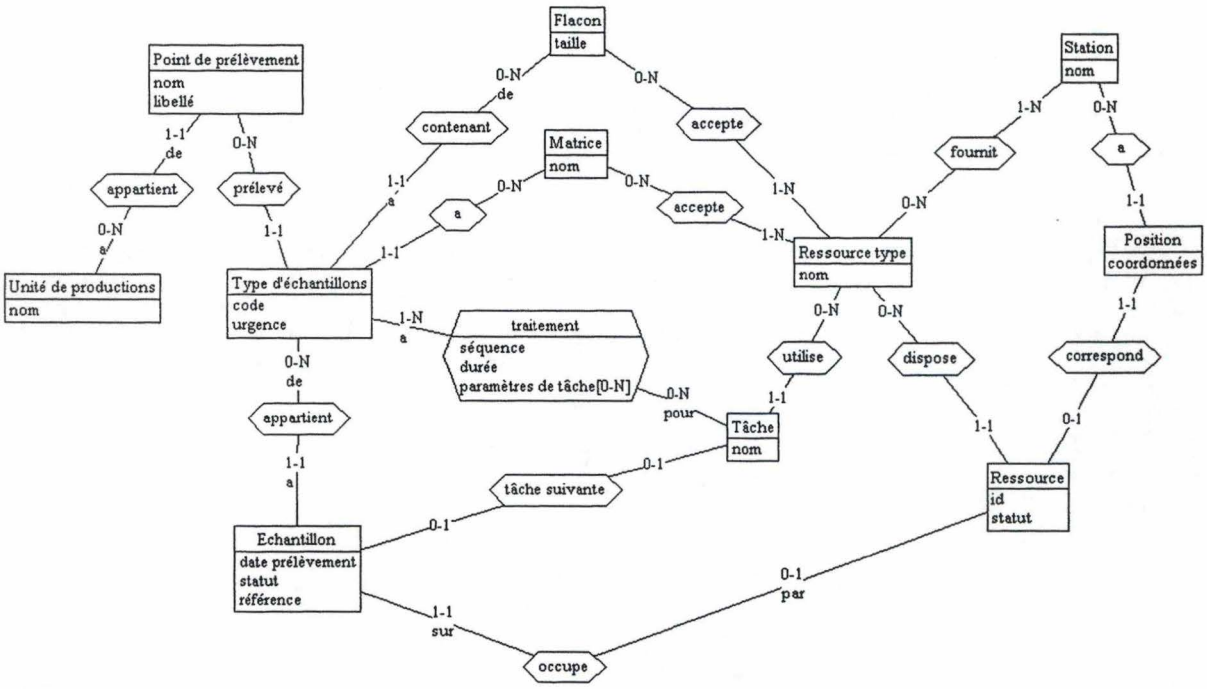


Figure 2.6 Schéma conceptuel des données

2.9. Critique de l'existant

Le programme "superviseur" de l'atelier robotisé SICP fonctionne bien pour la communication avec le robot et les stations, mais il a les lacunes suivantes:

- le choix de l'échantillon en entrée de l'atelier est réalisé :
  - au hasard en fonction de la disponibilité fortuite d'une ressource nécessaire
  - sans tenir compte des urgences
  - indépendamment d'une vue globale des échantillons en présence dans la station d'entrée.

- lorsque les tâches de pompage et de densité existent dans le traitement d'un échantillon, le système actuel fige l'ordre de déroulement de ces tâches à priori (densité d'abord, puis pompage). Or rien ne justifie cette rigueur: le système devrait pouvoir choisir, parmi les deux tâches, celle dont le choix est le plus judicieux pour que globalement le temps de passage de tous les échantillons soit minimum.

- le système actuel ne permet pas de choisir dynamiquement une des stations de pompage compatible avec la matrice de l'échantillon (car il y a confusion entre "ressource type" et "station" dans le système actuel): Une station de pompage a dû être assignée à chaque échantillon, ce qui enlève toute possibilité d'utilisation rentable des stations de pompage.

- le déroulement du traitement est figé à l'intérieur de procédures: toute évolution nécessitera une modification du code.

- il y a de nombreux "temps morts" (d'attente) sur les stations en attendant que la station suivante du traitement de l'échantillon se libère: un échantillon reste débouché trop longtemps.

D'où:

- l'utilisation des stations n'est pas toujours optimale
- les priorités des échantillons ne sont pas respectées
- aucune maîtrise des "temps morts" n'est possible
- le temps global de passage ne peut être amélioré dynamiquement sur base de la connaissance des échantillons présents dans la station d'entrée

Or, le système comporte les degrés de liberté suivants:

- les échantillons arrivent par flot: le début du traitement du premier échantillon du flot peut être retardé de 5 minutes pour permettre de calculer un ordonnancement "déterministe" (plutôt qu'événementiel) des tâches.
- les traitement sur les échantillons sont connus:
  - l'ordre entre les tâches et les durées de celles-ci sont connus.
  - l'ordre des tâches de densité et de pompage d'un même échantillon peut être choisi .
- le système peut choisir l'ordre des tâches qui utiliseront une ressource
- lorsqu'une tâche est terminée, l'échantillon peut rester sur la ressource le temps que la ressource suivante se libère.

2.10. Conclusion

L'objectif de ce travail est de trouver une modélisation du problème et une méthode permettant de découvrir un ordonnancement optimal des échantillons dans le but de minimiser le temps de passage global de tous les échantillons. La méthode utilisée doit respecter un temps de calcul compatible avec l'utilisation de l'atelier robotique.

Chapitre 3: Spécification du problème

Dans ce chapitre, nous spécifions le problème de l'optimisation de l'atelier robotisé du Laboratoire en mettant en évidence l'appartenance de celui-ci à une classe particulière de problème. Nous poursuivons en spécifiant les contraintes fonctionnelles rencontrées durant l'étude de l'existant du chapitre 2.

3.1. Définition générale du problème

Le problème de l'atelier robotisé du Laboratoire, comme décrit dans le chapitre précédent, consiste à trouver un agencement des tâches tel que le temps global de tous les traitements soit minimisé.

3.2. Classe du problème

Ce problème appartient à la classe des problèmes d'ordonnancement:

> Soient une ensemble de tâches à réaliser et un ensemble de ressources sur lesquelles les tâches doivent être réalisées: un problème d'ordonnancement consiste à trouver l'allocation des ressources aux tâches et de fixer les dates de début de celles-ci.

Cette classe est très vaste: il nous faut donc préciser certaines notions afin de bien cerner notre problème.

Ce paragraphe s'inspire très largement de [Le Pape, Baptiste,1997], un document qui présente le problème de l'ordonnancement d'un point de vue théorique, et [Pieret,1992] , un document qui présente le problème de l'ordonnancement d'un point de vue pratique en laboratoire analytique.

3.2.1. Type d'ordonnancements

Il existe plusieurs types d'ordonnancements:

- type d'ordonnancement sans prévision:
  - statique: on prend toujours les mêmes décisions d'ordonnancement quelque soit l'état du système. Ce cas concerne un environnement bien défini, sans surprise, toujours le même.
  - dynamique: on prend la "meilleure" décision d'ordonnancement en fonction de l'état du système au moment de la décision. Ce cas concerne un environnement bien défini, mais qui peut être différent d'un cas à l'autre.

- type d'ordonnancement avec prévision:

  - statique: on calcule une fois pour toutes un ordonnancement, appelé planification statique, sur une situation connue qui ne changera plus. Dès que la situation se présente, on exécute l'ordonnancement comme une partition de musique. Ce cas concerne un environnement bien défini et dont les situations sont prévisibles.

  - dynamique: on calcule l'ordonnancement d'une situation au moment où elle se produit: c'est la planification dynamique. Puis, on exécute l'ordonnancement comme une partition de musique. Ce cas concerne un environnement bien défini, mais où les situations sont aléatoires.

Le mode actuel de fonctionnement du système de l'atelier SICP (voir 2.8.) utilise un type d'ordonnancements sans prévision et dynamique ("événementiel"). L'inconvénient de ce type est qu'il ne permet pas d'optimiser globalement l'ordonnancement.

Or, les échantillons du Laboratoire arrivent par flots d'échantillons, ce qui n'est pas incompatible avec un type d'ordonnancement avec prévision puisqu'on a un certain nombre d'échantillons présents dans la station d'entrée. Mais le nombre et la nature des échantillons en présence est inconnu à l'avance: nous sommes dans le cas d'une situation aléatoire et la recherche d'un ordonnancement optimal devra se faire par une planification dynamique des traitements.

3.2.2. Types de tâches

Il existe deux types de tâches: les tâches préemptives et les tâches qui ne le sont pas.

Une tâche "préemptive" est une tâche morcelable qui autorise la préemption. La préemption d'une tâche 1, dite "préemptive", par une tâche 2 se traduit par l'éviction en cours de traitement de la tâche 1 de la ressource qui lui était allouée au profit de la tâche 2. La tâche 1 est donc partiellement achevée.

La préemption n'est pas autorisée sur les tâches de l'atelier SICP: elles doivent être exécutées dans leur entièreté en une seule fois.

3.2.3. Types de ressources

Deux types de ressources existent: les ressources renouvelables et les ressources consommables.

Une ressource est dite renouvelable si après utilisation elle peut redevenir disponible, contrairement à une ressource consommable qui n'existe plus après utilisation.

Sur l'atelier SICP, les ressources sont renouvelables: ce sont les positions dans les stations. Celles-ci sont libérées après l'exécution de la tâche.

Il faut remarquer également

- qu'il existe plusieurs ressources identiques mais en nombre limité (exemple: les positions du refroidisseur)
- qu'une ressource peut être utilisée plusieurs fois dans le traitement d'un échantillon (exemple: la position de la station "decap" sert à dévisser un bouchon et également à le revisser)

3.2.4. Ordonnancement d'atelier

Dans le cas des ateliers, une tâche est une opération et les ressources sont des machines. L'exécution de chaque opération se fait sur une machine.

Un problème d'ordonnancement d'atelier est classiquement composé des éléments suivants: on dispose de M machines devant servir à réaliser N travaux composés chacun d'un nombre éventuellement différent d'opérations. Un problème est dit "à ressources multiples" si on considère plusieurs machines différentes.

Une classification des problèmes d'ateliers a été réalisée en fonction de l'ordre dans lequel les machines doivent être utilisées pour exécuter les différentes opérations:

- si les opérations sont indépendantes, l'ordre d'utilisation est indifférent, l'atelier est de type "OPEN-SHOP"

- si les opérations doivent être réalisées dans un ordre pouvant être différent pour les différents travaux, il s'agit d'un "JOB-SHOP"

- dans le cas où les opérations doivent toujours être exécutées dans le même ordre quel que soit le travail, l'atelier est appelé "FLOW-SHOP"

Notre problème est typiquement le cas d'un ordonnancement d'atelier de type "JOB-SHOP" où un travail (job) est le traitement que doit subir un échantillon.

3.2.5. Nouvelle définition

Le problème de l'optimisation de l'atelier robotisé SICP du Laboratoire est de trouver la planification d'un problème d'ordonnancement d'atelier de type "JOB-SHOP" dont:

- les tâches sont non préemptives, de durée fixée,
- les ressources sont multiples et renouvelables,
- le but est de minimiser le temps global de tous les travaux tout en respectant un ensemble de contraintes.

Cette définition, bien que plus précise, est encore incomplète. En effet, la particularité d'un problème de ce type réside dans les contraintes rencontrées. Dans le paragraphe suivant, nous formalisons les contraintes telles qu'elles se présentent dans le cas réel de l'atelier SICP du Laboratoire.

3.3. Modélisation des contraintes

Les contraintes fonctionnelles du chapitre 2, paragraphe 2.4 peuvent être toutes exprimées par trois types de contraintes:

- contrainte de précédence
- contrainte cumulative
- contrainte disjonctive

Nous allons, dans les paragraphes suivants, spécifier ces différentes contraintes.

3.3.1. Contrainte de précédence

Cette contrainte exprime la relation de précédence qui existe entre deux tâches (voir [Diaz,1995], [Schulte,Smolka,2000] et [VanHentenryck,1989]):

Soient

$start(t_1)$, $start(t_2)$	le début de deux tâches
$d(t_1)$	la durée de la tâche t_1

t_1 précède t_2 ssi

$$start(t_2) \geq start(t_1) + d(t_1) \qquad (3.1)$$

Les contraintes $c \in C$, l'ensemble de contraintes de précédence, sont faites de paires de tâches $c=(t_1,t_2)$: on l'écrira par la suite ($t_1 << t_2$), ce qui signifie que t_2 ne peut commencer qu'après la fin de t_1.

Une tâche *t* est dite indépendante ssi $\forall a \in Taches, (a,t) \notin C \wedge (t,a) \notin C$.

3.3.2. Contrainte cumulative

Une contrainte cumulative exprime la capacité maximale de ressources d'un même type disponible. Dans notre cas, la station refroidisseur peut réaliser simultanément les tâches de refroidissement de plusieurs échantillons, mais dans un nombre limité par une valeur maximale. Elle s'applique donc à un groupe de tâches utilisant un même type de ressources.

On trouve, dans [Caseau,Laburthe,1996], la définition suivante de la contrainte cumulative:
soient

- un ensemble de ressources R,
- un ensemble de tâches T et
- un ensemble de contraintes de précédence C (voir 3.3.1.)

Pour une ressource $r \in R$, available(r) dénote le nombre d'unités identiques de type r disponibles.
Pour une tâche $t \in T$,

- res(t), minuse(t) et maxuse(t) dénotent respectivement la ressource, le nombre minimal et maximal d'unités de ressources nécessaires pour réaliser t sur res(t)
- size(t) dénote la quantité totale de ressources mesurée en (durée * unités de ressources) consommée par t durant son traitement

La solution de cette contrainte est l'ensemble des variables
"date de début", "date de fin" et "niveau de consommation"
{start(t), end(t), use(t) | t ∈ T} tels que:

$\forall$ t ∈ T, $minuse(t) \leq use(t) \leq maxuse(t)$

$\forall$ t ∈ T, $(end(t) - start(t)) * use(t) = size(t)$

$\forall$ r ∈ R et $\forall$ moment x, le besoin en ressources n'excède pas la quantité de ressources disponibles:

$$\sum_{\{t \in T | res(t)=r \wedge start(t) \leq x \leq end(t)\}} use(t) \leq available(r)$$

$\forall$ contraintes de précédence $(t_1 << t_2) \in C$, $start(t_2) \geq end(t_1)$ (3.2)

3.3.3. Contrainte disjonctive

Une contrainte disjonctive exprime l'exclusivité réciproque de réalisation d'une tâche par rapport à une autre. (voir [Caseau,Laburthe,1995], [Diaz,1995], [Schulte,Smolka,2000] et [VanHentenryck,1989]):

Soient
- $start(t_1)$, $start(t_2)$ les débuts de deux tâches, et
- $d(t_1)$, $d(t_2)$ les durées respectives des tâches t_1 et t_2

alors

$$start(t_1) + d(t_1) \leq start(t_2) \vee start(t_2) + d(t_2) \leq start(t_1) \quad (3.3)$$

Si la contrainte de disjonction s'applique sur deux tâches utilisant une même ressource, on peut l'exprimer plus précisément par:

Soient $t_1, t_2 \in T$,
- $start(t_1)$, $start(t_2)$ les débuts respectifs de t_1 et t_2,
- $d(t_1)$, $d(t_2)$ les durées respectives des tâches t_1 et t_2
- $use(t)$ la ressource utilisée par la tâche t

alors

$$use(t_1)=use(t_2) \Rightarrow start(t_1) + d(t_1) \leq start(t_2) \vee start(t_2) + d(t_2) \leq start(t_1) \quad (3.3bis)$$

Cette contrainte (3.3bis) est un cas particulier d'une contrainte cumulative sur une ressource type de capacité maximale égale à 1 : on l'appellera, par analogie, également contrainte cumulative.

3.3.4. Matrice contraintes fonctionnelles / contraintes modélisées

Le tableau ci-dessous met en correspondance les contraintes fonctionnelles et les contraintes modélisées ci-dessus.

Tableau 3.1. Matrice contraintes fonctionnelles / contraintes modélisées

n°	Contraintes fonctionnelles	Contraintes modélisées
2.4.1	tout échantillon est toujours introduit dans le système par le "rack in"	contrainte de précédence sur tâches d'un job
2.4.2	une tâche d'un échantillon n'est réalisée que dans une et une seule station	contrainte de précédence sur tâches d'un job
2.4.3	tout échantillon devant subir la tâche de refroidissement est placé de la station d'entrée ("rack in") dans la station refroidisseur ("cooler") avant tout autre tâche	contrainte de précédence sur tâches d'un job
2.4.4	un échantillon est toujours bouché lorsqu'il entre dans la station refroidisseur (également lorsqu'il en sort, forcément)	contrainte de précédence sur tâches d'un job
2.4.5	la station refroidisseur peut réaliser la tâche de refroidissement de plusieurs échantillons simultanément	contrainte cumulative
2.4.6	un échantillon ne peut être placé dans la station refroidisseur que s'il y a encore un emplacement disponible pour son type de flacon	contrainte cumulative sur une ressource par type de flacon
2.4.7	il n'y a pas d'emplacement prévu dans la station "cooler" pour des flacons de 1000 ml	contrainte de précédence sur tâches d'un job
2.4.8	un échantillon n'est placé qu'une et une seule fois dans le "cooler" au cours du travail prévu	contrainte de précédence sur tâches d'un job
2.4.9	un échantillon est toujours débouché dans la station de dévissage supportant son type de flacon	contrainte de précédence sur tâches d'un job
2.4.10	une station de dévissage ne dévisse qu'un seul échantillon à la fois	contrainte de précédence sur tâches d'un job
2.4.11	une station de dévissage ne peut conserver que 6 bouchons maximum.	contrainte cumulative de tâches fictives "du dévissage au revissage"
2.4.12	un échantillon devant subir les tâches de pompage et de densité ne peut être rebouché entre la station de pompage et la station de densité	contrainte de précédence sur tâches d'un job
2.4.13	si un échantillon doit subir les tâches de pompage et de densité, le temps de succession entre celles-ci doit être le plus court possible ("safety requirement")	contrainte de précédence sur tâches d'un job
2.4.14	un échantillon doit être toujours débouché avant d'être placé dans la station "densimètre" ou dans une station de pompage "pompX".	contrainte de précédence sur tâches d'un job
2.4.15	un échantillon n'est placé que dans une station de pompage acceptant la matrice analytique de son type d'échantillon	contrainte de précédence sur tâches d'un job et contrainte cumulative
2.4.16	une station de pompage "pompX" ne prépare qu'un seul échantillon à la fois.	contrainte disjonctive (ou cumulative)
2.4.17	un échantillon n'est préparé qu'une seule fois dans une et une seule station de pompage au cours du travail prévu	contrainte de précédence sur tâches d'un job
2.4.18	un échantillon n'est placé qu'une et une seule fois dans la station "Densimètre" au cours du travail prévu	contrainte de précédence sur tâches d'un job
2.4.19	la station de densité ne réalise la densité que d'un seul échantillon à la fois	contrainte disjonctive (ou cumulative)
2.4.20	tout échantillon termine son parcours à travers le système dans un couloir du bac sortie déterminé par son type d'échantillon	contrainte de précédence sur tâches d'un job
2.4.21	tout échantillon doit être rebouché avant d'être mis dans un couloir de la station de sortie	contrainte de précédence sur tâches d'un job
2.4.22	les échantillons d'urgence 0 doivent être traités avant ceux d'urgence 1, eux-mêmes traités avant ceux urgence 2	contrainte de précédence sur tâches d'un job

Remarque: la contrainte disjonctive peut modéliser également le choix entre la station de pompage et la station "densité" sans devoir figer un ordre entre ces deux tâches répondant ainsi à la critique exprimée dans l'étude de l'existant (voir 2.9.).

3.5. Contraintes sur les déplacements du robot

3.5.1. Contrainte disjonctive

Comme vu dans le paragraphe 2.6 du chapitre précédent, le robot est une ressource unique: les tâches utilisant le robot ne peuvent avoir lieu en même temps. Il y a donc une contrainte disjonctive sur toutes les tâches "déplacer le robot" et "aller au terminus" qui utilisent la ressource robot (voir la formule 3.3bis).

3.5.2. Contrainte de synchronisation

Le déplacement du flacon d'une station à une autre par le robot n'a de sens que si la station destination est libre (c'est-à-dire prête à recevoir le flacon). Il existe donc une contrainte de synchronisation stricte entre une tâche "déplacer le robot" et la tâche qui suit ce déplacement dans le traitement de l'échantillon.

Soient

$t_1, t_2 \in J_e$	où	J_e est l'ensemble des tâches du job de l'échantillon e
$start(t_1)$, $start(t_2)$		les débuts respectifs de t_1 et t_2,
$d(t_1), d(t_2)$		les durées respectives de t_1 et t_2,
$t_1 << t_2$		contrainte de précédence: t_1 précède t_2 (formule 3.1)
$use(t_1) = robot$		la tâche t_1 utilise la ressource "robot",
$use(t_2) \neq robot$		la tâche t_2 n'utilise pas la ressource "robot"

alors

$$start(t_1) + d(t_1) = start(t_2) \qquad (3.4)$$

3.6. Difficultés de modélisation

Supposons la situation suivante:
soient

J_1, J_2 les deux jobs de deux échantillons e_1, e_2
t_1, t_2 deux tâches $\in J_1 \wedge t_3, t_4$ deux tâches $\in J_2$
$d(t_1)$, $d(t_2)$, $d(t_3)$, $d(t_4)$ les durées respectives des tâches t_1, t_2, t_3, t_4
$(t_1 << t_2) \wedge (t_3 << t_4)$

$use(t_1) = use(t_4) =$ robot , de capacité = 1
$(use(t_2) = use(t_4)) \neq$ robot , dont la capacité = 1

La situation de la figure 3.1 répond correctement aux contraintes de précédence sur les tâches et de capacité sur les ressources. Cependant, cette situation est un blocage car pour que l'échantillon e_1 puisse aller sur la station $use(t_2)$, il faut que celle-ci puisse être libérée de l'échantillon e_2 par le robot. Or, le robot est occupé à faire son mouvement vers la station $use(t_2)$ en transportant l'échantillon e_1. Il ne peut donc pas prendre l'échantillon e_2 pour libérer la station $use(t_2)$!

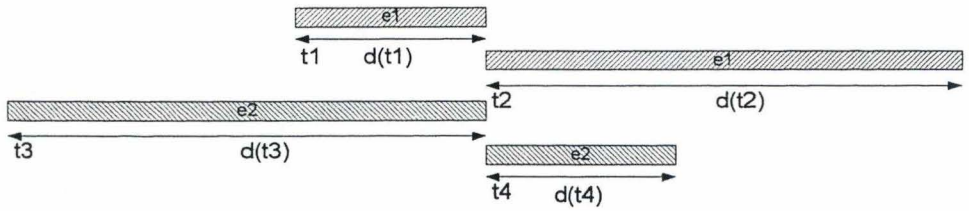


Figure 3.1 inter blocage

La modélisation de l'atelier robotique avec les contraintes vues précédemment doit donc être complétée par de nouvelles contraintes.

Pour présenter ces nouvelles contraintes, supposons les conditions suivantes:

$t_1, t_r, t_2 \in J_e$	où J_e est l'ensemble des tâches du job de échantillon e
$start(t_1), start(t_r), start(t_2)$	les débuts respectifs de t_1, t_r et t_2,
$d(t_1), d(t_r), d(t_2)$	les durées respectives de t_1, t_r et t_2,
$t_1 << t_r \wedge t_r << t_2$	t_1 précède t_r et t_r précède t_2 (voir formule 3.1)
$use(t_r) =$ robot	la tâche t_r utilise la ressource robot,
$use(t_1) \neq$ robot $\wedge$ $use(t_2) \neq$ robot	

3.6.1. Durée du déplacement du robot

La durée de la tâche "déplacer le robot" t_r est constituée en réalité de deux tâches de déplacement (voir figure 3.2) successives et indissociables:

- un déplacement "à vide" (t') pour se rendre vers la station origine où se trouve 'e'
- un déplacement "chargé" (t'') pour transporter 'e' vers la station destination

où

- d(t') est la durée de t' qui n'est pas connue à priori car sa valeur dépend de la position du robot avant son mouvement
- d(t'') est la durée de t'' qui elle est connue à l'avance

par conséquent, $d(t_r) = d(t') + d(t'')$ doit être calculée dynamiquement dans la contrainte de précédence ($t_r << t_2$) et dans la contrainte cumulative impliquant toutes les tâches de déplacement du robot . (3.5)

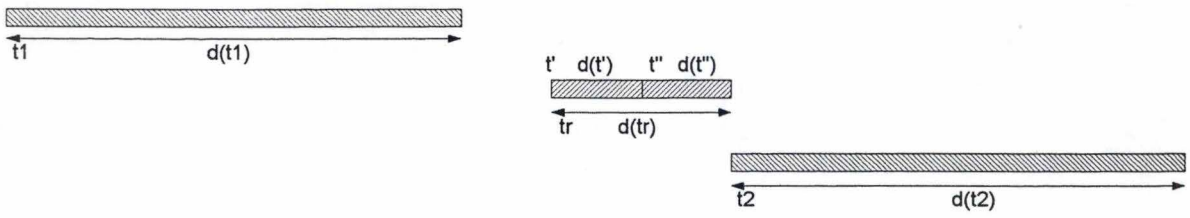


Figure 3.2 Problème de la durée du déplacement du robot

3.6.2. "Temps morts"

L'occupation d'une ressource ne prend pas fin lorsque la tâche se termine: l'échantillon 'e' est alors toujours présent sur la ressource, on ne peut donc pas affecter la ressource à une autre tâche tant que le robot n'est pas venu rechercher l'échantillon 'e' (voir figure 3.3):
$\Delta(t_1)$ est la durée du "temps mort" de l'échantillon 'e' sur $use(t_1)$:

$$\Delta(t_1) = start(t'') - (start(t_1) + d(t_1)) \qquad (3.6)$$

La contrainte cumulative impliquant t_1 et la ressource $use(t_1)$ doit être adaptée dynamiquement pour s'appliquer pendant une durée $d_1 = d(t_1) + \Delta(t_1)$ (3.7)

3.6.3. Réservation des ressources

Pour répondre au problème de l'inter-blocage (figure 3.1), il suffit de s'assurer que la ressource destination est libre avant même qu'un échantillon ne soit pris par le robot. Par extension, on peut considérer que la ressource est occupée dès que le robot prend l'échantillon pour aller vers celle-ci: le moment entre la prise de l'échantillon par le robot et sa présence effective dans la ressource est appelé la durée de réservation (voir figure 3.3):

> si $\Delta(t_2)$ est la durée de réservation de l'échantillon 'e' sur $use(t_2)$, alors la contrainte cumulative impliquant (t_2) et la ressource $use(t_2)$ doit être adaptée pour s'appliquer pendant une durée $d_2 = \Delta(t_2) + d(t_2)$.

Or $\Delta(t_2) = d(t'')$ où $d(t'')$ est connu à priori (voir 3.6.1), d'où $d_2 = d(t'') + d(t_2)$

Afin de simplifier les contraintes cumulatives, on peut donc systématiquement modifier les durées de toutes les tâches t_i où $use(t_i) \neq$ robot pour y inclure les $d(t_i'')$ correspondants: $d(t_i)' = d(t_i'') + d(t_i)$ (3.8)

La contrainte de synchronisation des tâches "déplacement du robot" (formule (3.4)) devient:

$$start(t_{ri}) + d(t_{ri}) - d(t_i'') = start(t_i) \quad (3.9)$$

Comme le robot est une ressource unique (contrainte 3.3bis) et que chaque tâche est précédée par une tâche de déplacement du robot, les modifications (3.8) et (3.9) assurent également qu'il n'y aura pas de situation de blocage.

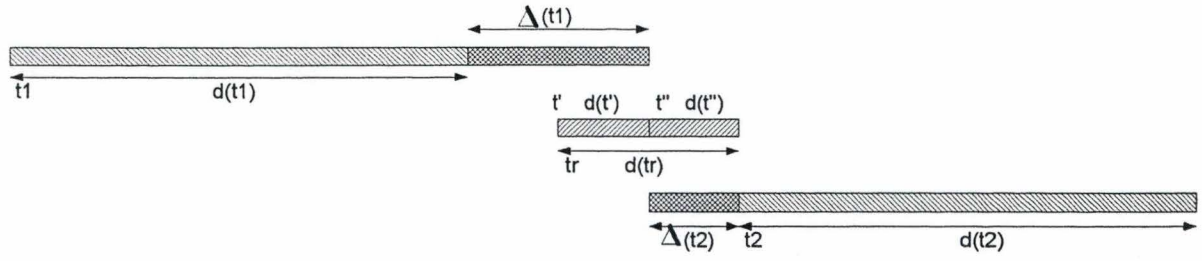


Figure 3.3 Temps morts et réservations des ressources

3.7. Conclusion

Nous avons, au cours de ce chapitre, cerné indépendamment de toute solution beaucoup mieux le problème de l'optimisation de l'atelier robotisé du Laboratoire SICP présenté lors de l'étude de l'existant du chapitre 2.

Nous savons également que la difficulté de ce problème réside dans ces contraintes: toutes celles-ci ont été modélisées. Néanmoins, la modélisation des déplacements du robot, des temps morts sur les ressources et des réservations de celles-ci révèle une difficulté assez grande.

Chapitre 4: Recherche de solutions

Dans ce chapitre, nous cherchons des méthodes pour résoudre notre problème, tel qu'il a été défini au chapitre 3.

Nous déterminons d'abord que ce problème appartient à la classe des problèmes NP-difficiles. Pour résoudre un problème NP-difficile, nous avons le choix entre des méthodes approchées (heuristiques gloutonnes, recherche locale et recherche globale) et des méthodes exactes (méthodes arborescentes "branch and bounds", programmation logique sous contrainte,...)

Nous envisageons brièvement quelques applications possibles de certaines de ces méthodes à notre problème.

4.1. Introduction.

La recherche d'une solution pour notre problème d'ordonnancement nous conduit d'abord à parler de la complexité des problèmes d'ordonnancement.

4.1.1. Notions sur la complexité

Les brèves notions de cette section sont empruntées au cours de Théorie des Graphes de Monsieur Leclercq, FUNDP. Le lecteur qui souhaite plus d'informations sur la Théorie de la Complexité peut consulter [Prins,1994] et [Aho, Ullman,1993].

En simplifiant fortement, la complexité théorique d'un algorithme mesure son comportement face aux échantillons de données qui lui sont le plus défavorables. Seule la complexité en temps offre un réel intérêt (plutôt que la complexité en place).

La complexité ne s'intéresse
- qu'à l'ordre de grandeur du temps de calcul mesuré non pas en secondes mais en nombre d'opérations élémentaires K : les opérations élémentaires font l'objet d'un simple comptage, sans distinction.
- qu'aux problèmes de grande taille, soit n tendant idéalement vers l'infini: les puissances de n inférieures à la puissance maximale seront ainsi systématiquement négligées.

Le nombre d'opérations élémentaires est généralement une fonction de la taille du problème de sorte que $K=K(n)$. La fonction $K(n)$ peut prendre toutes sortes de formes, mais pour beaucoup d'algorithmes, $K(n)$ a la forme ou est borné par un polynôme de degré r en n. Lorsque n tend vers l'infini, les termes d'ordres inférieurs à r sont alors négligeables: $K(n) < a \cdot n^r$ où a est une constante.

On dit alors que l'algorithme est en n^r , ce que l'on note par $O(n^r)$.

Si, dans un algorithme, $K(n)$ ne peut être borné $\forall\ n$ par aucune fonction polynomiale quel que soit son degré, celui-ci est dit de complexité exponentielle.

4.1.2. Complexité d'un problème d'ordonnancement

Comme nous l'avons vu dans la section 4.1.1., la complexité est liée à un algorithme et non à un problème. Cependant, par extension, on considère souvent la complexité d'un problème comme celle de l'algorithme le plus performant servant à le résoudre.

Dans [Cormen, et al.,1994], les auteurs classent les problèmes d'ordonnancement - mise en séquence et planification - dans la catégorie des problèmes NP-Complets *(Ils s'inspirent d'un catalogue de problèmes NP-Complet de Garey et Johnson - Computers and Interactability: A Guide to the Theory of NP-Completeness – W.H. Freeman, 1979)*

La classe des problèmes NP-Complets (appelés également NP-difficiles pour les problèmes d'optimisation combinatoire dans [Prins,1994]) sont des problèmes pour lesquels il n'existe pas d'algorithmes polynomials connus permettant de les résoudre (et il n'en existera sans doute jamais !)

4.1.3. Conclusion

Comme nous l'avons vu précédemment (4.1.2), il est illusoire de rechercher un algorithme qui résolve, dans un temps polynomial, un problème d'ordonnancement !

Pour résoudre des problèmes d'optimisation combinatoire NP-difficiles, il existe deux classes de méthodes ([Prins,1994]) :

- les méthodes approchées ou heuristiques,
- les méthodes exactes

Dans les deux prochains paragraphes, nous explorons ces deux classes de méthodes afin d'envisager des solutions pour notre problème.

4.2. Méthodes approchées

Les méthodes approchées ou heuristiques sont des algorithmes rapides qui ont pour but de trouver une solution réalisable en tenant compte de la fonction économique, mais sans garantie d'optimalité. Le principal problème est alors d'évaluer la qualité des résultats obtenus.

Il existe trois grands types d'heuristiques:

1. méthodes construisant une seule solution par une suite de choix partiels et définitifs c'est-à-dire sans retour en arrière. On les appelle méthodes gloutonnes quand elles cherchent, à chaque itération, à faire le choix le plus avantageux.
2. recherches locales aussi appelées recherches de voisinage. Pour une minimisation, on part d'une solution initiale et, par petites transformations successives, on construit une suite de solutions de coûts décroissants. Le

processus stoppe quand on ne peut plus améliorer localement la solution courante.

3. recherches globales: une recherche locale peut être piégée dans un minimum local. Les méthodes dites de recherche globale, appelées aussi méta-heuristiques, sortent de ces pièges en construisant aussi une suite de solutions, mais dans laquelle la fonction économique peut temporairement croître.

Dans les deux sections suivantes, nous envisageons ce que pourrait être une solution à notre problème par une heuristique du type 1 (4.3.1) et par une heuristique du type 3 (4.3.2).

4.2.1. Stratégies de sélection d'échantillons

L'idée de cette approche est de trouver une heuristique qui sélectionne, selon une stratégie donnée, les échantillons dans la station d'entrée tout en construisant une seule solution (une planification) réalisable (c'est-à-dire respectant l'ensemble des contraintes énoncées au chapitre 3) en espérant que globalement cette stratégie conduise à un "bon résultat". Il ne s'agit évidemment plus de rechercher l'optimum !

On peut ainsi "essayer" différentes stratégies sur le problème de l'atelier robotique et voir si globalement une stratégie ne donne pas de meilleurs résultats qu'une autre, en moyenne, sur un ensemble d'échantillons représentatifs (une sorte de "jeu de tests"). Le critère d'évaluation d'une stratégie par rapport à une autre étant le résultat de la fonction économique obtenu par la stratégie, c'est-à-dire le temps total nécessaire pour traiter tous les échantillons.

On pourrait ainsi envisager les stratégies suivantes:

- choix des échantillons de manière aléatoire,
- choix d'abord des échantillons dont la durée totale de traitement est la plus petite,
- choix d'abord des échantillons dont la durée totale de traitement est la plus grande,
- choix d'abord des échantillons utilisant le plus de ressources,
- choix d'abord des échantillons utilisant le moins de ressources,
- (...).

Ces stratégies restent des méthodes empiriques qui ne garantissent par de trouver un optimum. Mais elles ont l'avantage de donner rapidement une solution dans un temps de calcul acceptable pour le fonctionnement d'un atelier robotique.

A titre d'exemple, nous avons réalisé une heuristique planifiant une liste d'échantillons ordonnée selon une des stratégies ci-dessus. Cet algorithme utilise des listes de réservations:

soient E : un vecteur d'échantillons
R ={ensemble des ressources types}et n = #R ,
un ensemble de listes L_r telles que $\forall r \in R : \exists L_r$, où L_r contiendra les plages de temps réservées aux tâches pour l'utilisation de la ressource type *r* .

heuristique(E: vecteur d'échantillons)
début
e := le nombre d'éléments de E
$L_{r1} = L_{r2} = \ldots = L_m = \varnothing$
pour i:=1 jusqu'à e faire
soit T_i le vecteur des tâches de l'échantillon E[i] (c'est-à-dire que T_i est le travail de E[i]) telles que si n est le nombre d'éléments de T_i , $1 \leq j \leq n$, $t_j \in T_i$,
alors $t_j \neq$ "déplacement vers..."
$\wedge \exists$ une contrainte de précédence(<<) entre les éléments de T_i: $t_1 << t_j \ldots << t_n$
planifier(T_i,1,0)
fin pour
les listes L_r contiennent la planification réalisable de l'ensemble des tâches sur les différentes ressources
fin

Fonctions auxiliaires

Cette fonction planifie la $i^{ème}$ tâche du travail T d'un échantillon sur sa ressource après un moment donné en argument. Elle renvoie le moment où la ressource utilisée par la tâche est libérée:

Fonction planifier (T: pointeur du vecteur des tâches, i: indexe, m : moment de départ): le moment réserver
début
si i > nombre d'éléments de T alors
renvoyer m
sinon
t := T[i] /* *tâche courante*
d_t := durée(t) /* *durée de la tâche t*

réservé := faux
répéter
m' := rechercher(t, m) /* *recherche une plage libre sur la ressource de t*
m" := planifier(T,i+1,m'+ d_t) /* *appel récursif pour planifier les T[k] où k > i*
réservé := réservation(t, m', m")/* *réserve la place d'occupation de la ressource*
si non réservé alors /* *si la plage réservée n'est pas valide*
annuler_réservation(T, i+1) /* *on annule les réservations des T[k] où k > i*
m := m" /* *on réessaie à partir de m"*
fin si
tant que non réservé
renvoyer m"
fin si
fin

Cette fonction retourne le moment de début au plus tôt d'une tâche, à partir d'un moment de départ donné.

```
Fonction rechercher ( t: une tâche, m: moment de départ): le moment m'
début
        d_t := durée(t)                          /* durée de la tâche t
        r := use(t)                              /* la ressource type utilisée par t
        L := liste_réservation(r)                /* L est le pointeur vers la liste de réservation de r
        t_r := déplacement(t)                    /* tâche de déplacement du robot qui accompagne t
        d_r := durée(t_r)                        /* durée de déplacement du robot pour initier t

        Parcourir la liste L en recherchant un moment m' ≥ m tel que essayer(L, m', m'+ d_t, d_r) = vrai
        renvoyer m'
fin
```

Cette fonction réserve la plage de temps d'utilisation d'une ressource d'une tâche: elle renvoie vrai si elle y parvient, sinon faux.

```
Fonction réservation ( t: une tâche, m': moment de début, m'': moment de fin ): booléen
début
        r := use(t)                              /* la ressource type utilisée par t
        L:= liste_réservation(r)                 /* L est le pointeur vers la liste de réservation de r
        t_r:= déplacement(t)                     /* tâche de déplacement du robot qui accompagne t
        d_r := durée(t_r)                        /* durée de déplacement du robot pour initier t

        si essayer(L, m', m'', d_r) = vrai alors
            enregistrer(t, L, m', m'')           /* enregistre la réservation [m',m''] de t dans L
            enregistrer(t_r, L_robot, m', m'+ d_r) /* enregistre le déplacement t_r [m',m'+ d_r] dans L_robot
            renvoyer vrai
        sinon
            renvoyer faux
        fin si
fin
```

Cette fonction vérifie qu'une plage d'utilisation d'une ressource est libre et si sa réservation permet de respecter les contraintes cumulatives et de synchronisations. Elle renvoie vrai si la plage est libre, sinon faux.

```
Fonction essayer (L: liste réservation, m': moment de départ, m'': moment de fin, d_r: durée robot): booléen
début
        si la plage [m', m''] est libre dans L, c'est à dire vérifiant la contrainte cumulative sur r, tel que:
                ∀ t ∈ [m', m'+ d_t]: nombre de ressources utilisées au moment t ≤ capacité(r)
          ∧  la plage [m', m'+ d_r] est libre dans L_robot  alors
            renvoyer vrai
        sinon
            renvoyer faux
        fin si

fin
```

Cette procédure retire les réservations de tâches dans les listes des ressources.

```
Fonction annuler_réservation (T: pointeur du vecteur des tâches, i: indexe)
début
        si i > nombre d'éléments de T alors
            fin
        sinon
            t := T[i]                            /*  tâche courante
            r := use(t)                          /* la ressource type utilisée par t
            L:= liste_réservation(r)             /* L est le pointeur vers la liste de réservation de r

            effacer(L,t)
            annuler_réservation(T,i+1)
        fin si
fin
```

Remarques:

- on peut améliorer les performances de cette heuristique en optimisant certaines fonctions: par exemple, la fonction "rechercher" pourrait être plus rapide si les ajouts de réservations dans les listes sont ordonnés chronologiquement et si on calcul pour chaque insertion la valeur d'inactivité de la ressource par rapport à la réservation juste précédente.

- la contrainte disjonctive sur des tâches d'un même travail peut être implémentée de plusieurs manières différentes:

  - 1ère solution: lorsqu'on réalise le vecteur T, on peut tirer au sort les tâches pour les ordonnées ou imposer un ordre arbitraire entre les tâches
  - 2ème solution: on peut modifier l'algorithme pour qu'il choisisse les tâches de tel façon à minimiser le temps de fin du travail de l'échantillon.

- la contrainte cumulative vérifiée dans la fonction "essayer" impose malheureusement de parcourir systématiquement à la liste de réservation de la ressource type dans l'autre sens, ce qui évidemment va diminuer les performances de l'algorithme

4.2.2. Recuit simulé

Cette section s'inspire très largement de [Prins,1994] p50-52.

a) Explication de la méthode

Le "recuit simulé" est une méthode de type "recherche globale" inventée par les physiciens Kirkpatrick, Gelatt et Vecchi en 1983. Elle s'inspire d'un phénomène de thermodynamique en métallurgie.

Pour comprendre cette méthode, [Prins,1994] cite l'analogie suivante: Considérons l'ensemble des solutions d'un problème d'optimisation combinatoire comme un paysage de montagne (voir figure 4.1). Une pierre lâchée dans un tel paysage va glisser dans le premier creux venu: c'est l'analogue d'une recherche locale simple. Une balle élastique peut, par contre, rebondir et contourner les obstacles. L'énergie initiale de la balle va diminuer lentement à chaque rebond et la balle finira plus bas que la pierre, du moins en probabilité.

Le recuit simulé en optimisation combinatoire n'a plus qu'un lointain rapport avec la thermodynamique. L'énergie du système est représentée par un réel arbitraire: T, la température. A partir d'une recherche locale quelconque pour un problème, on obtient une méthode de recuit comme suit (on part toujours d'une solution réalisable initiale, s_0):

- tirer au sort une transformation (c'est-à-dire une solution s' de V(s) au lieu de chercher la meilleure ou la première solution voisine améliorante.
- construire la solution résultante s' et sa variation de coût: $\Delta f = f(s') - f(s)$.
  - si $\Delta f \leq 0$, le coût diminue et on effectue la transformation améliorante comme dans une recherche locale (s := s').
  - si $\Delta f > 0$, le coût remonte, c'est un rebond, qu'on va pénaliser d'autant plus que la température est basse et que Δf est grand. Une fonction exponentielle a les propriétés désirées: calculer une probabilité d'acceptation $a = e^{\frac{-\Delta f}{T}}$, puis tirer au sort p dans [0,1]:
    - si $p \leq a$, la transformation est déclarée acceptée (bien qu'elle dégrade le coût), et faire s := s'.
    - sinon, la transformation est rejetée: conserver s pour l'itération suivante.
- Pour assure la convergence (analogie de la balle qui rebondit de moins en moins), diminuer lentement T à chaque itération, par exemple $T := k.T$, k inférieur à 1 mais proche de 1. T peut aussi décroître par paliers.
- arrêter quand T atteint un seuil fixé ε, proche de 0.

Toute la difficulté de cette méthode réside
- dans le réglage des paramètres (valeur de T, facteur de diminution de T et nombre d'itération),
- dans le calcul rapide de la variation de la fonction objective.

Pour être efficace, un recuit doit diminuer T assez lentement (soit à chaque itération, ou soit à chaque palier de plusieurs dizaines d'itérations) et effectuer plusieurs milliers ou dizaines de milliers d'itérations au total !

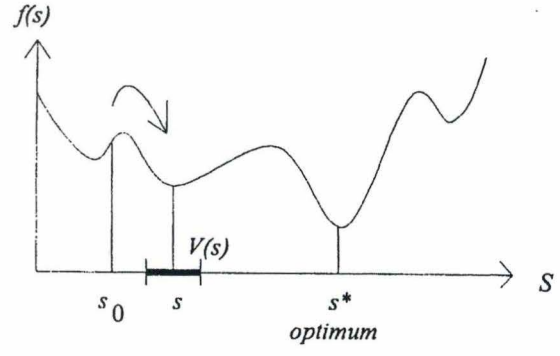


Figure 4.1 Piège d'une solution locale évité par le recuit simulé

b) Application de la méthode à notre problème

Nous nous inspirons du squelette d'algorithme proposé dans [Prins,1994].

On initialise les constantes:

T	: la Température (ce nombre est fixé par expérimentation)
MaxIter	: le nombre maximum d'itérations de l'algorithme (par exemple 10000)
ε	: la valeur d'arrêt de la Température (par exemple: 10^{-2})
MaxGel	: le nombre maximum d'itérations sans amélioration de f (par exemple 200)
k	: le facteur de diminution de la Température

Initialiser *l* comme la liste de tous les échantillons dans la station d'entrée suivant un ordre aléatoire. Produire, à partir de *l*, une solution réalisable s_0 avec une heuristique *h* "rapide" de type 1 (voir 4.3.1). Cette heuristique sélectionne les échantillons un à un dans l'ordre de la liste et planifie leurs tâches sur les ressources en respect avec les contraintes du problème énoncées au chapitre 3.

soient $s_0 := h(l).planification$

$f(s_0) := h(l).durée$ *' la fonction économique = le temps total de traitement des échantillons*

$z^* := f(s_0)$ ' z* est le meilleur coût = le meilleur temps total de traitements des échantillons

$s^* := s_0$ ' s* est la meilleure planification obtenue correspondant à z*

$NIter := 0$ ' compteur d'itération

$NGel := 0$ ' compteur d'itération sans amélioration du coût

Répéter

$NIter := NIter + 1$

choisir au hasard deux échantillons de *l* que l'on permute, soit *l'*

produire une nouvelle solution $s' := h(l').planification$ et $f(s') := h(l').durée$

calculer $\Delta f := f(s') - f(s)$

si $\Delta f < 0$ alors *' on fait une amélioration locale*

$Accepte := vrai$

sinon *' on n'a pas amélioré la fonction économique*

tirer au sort p dans [0,1]

$Accepte := \left(p \le e^{-\Delta f / T} \right)$

fin si

si Accepte alors

si $\Delta f = 0$ alors *' on n'a pas amélioré le coût*

$NGel := NGel + 1$

sinon

$NGel := 0$

fin si

si $f(s') < z^*$ alors *' si on a une meilleure solution*

$z^* := f(s')$

$s^* := s'$

fin si

$s := s'$

fin si

$T := k.T$ *' on diminue la Température*

jusqu'à $(T \le \varepsilon)$ ou $(NIter = MaxIter)$ ou $(NGel = MaxGel)$

c) Critique

Les performances de tout recuit simulé et en particulier de celui-ci dépendent de la rapidité du calcul de Δf . Or, cette tâche est rendue très difficile par le fait que dans un problème d'ordonnancement deux solutions "voisines" diffèrent fortement du point de vue du calcul de la planification et donc qu'il est difficile de calculer directement Δf sans passer par le calcul séparer de $f(s')$ et $f(s)$. Il faut donc trouver une heuristique très rapide h pour réaliser la planification à partir d'une liste d'échantillon et ainsi calculer la fonction économique $f(s)$.

4.3. Méthodes exactes

Au vu de la complexité de notre problème (4.1.2), la recherche d'une solution optimale par l'énumération complète de toutes les combinaisons d'ordonnancement possibles s'avère une tâche impossible.

L'idée des méthodes exactes est de diminuer l'aspect combinatoire du problème en faisant en quelque sorte une énumération intelligente de l'espace de solutions.

Cependant, les méthodes exactes sont des méthodes qui trouvent toujours l'optimum si on leur en laisse le temps. En effet, appliquées sur des problèmes NP-difficiles, elles restent des méthodes exponentielles qui peuvent devenir incontrôlables en temps de calcul dans le pire des cas !

Parmi d'autres, on peut citer
- les méthodes arborescentes "Branch and bounds"
- les méthodes de programmation logique sous contraintes

4.3.1. Méthodes arborescentes (Branch and bounds)

Les méthodes arborescentes "Branch and bounds" partagent l'espace des solutions en sous-ensembles de plus en plus petits, la plupart étant éliminés par des calculs de bornes avant d'être construits explicitement (voir [Prins,1994] p55).

Ces méthodes ont trois éléments en commun:

- une règle de séparation du domaine de solution
- une fonction d'évaluation (calcul de bornes) des sous-ensembles de solutions
- une stratégie d'exploration

Dans [Caseau,Laburthe,1995], les auteurs présentent une méthode de ce type appliquée à un problème d'ordonnancement de type job-shop avec uniquement des contraintes de précédence et des contraintes disjonctives. Sans rentrer dans les détails, l'idée de leur méthode est de prendre toutes les tâches utilisant une même ressource et de trouver un ordre entre elles (pour répondre évidemment à la contrainte disjonctive):

$$\text{soit } \{n \text{ taches} \mid use(t_i) = r,\ i = 1..n\},\ \text{on cherche } t1 << t2 << .. << tn$$

La méthode construit un arbre en prenant les tâches deux par deux. Un nœud de l'arbre (on l'appelle aussi nœud de décision) représente une paire de tâches (par exemple t1,t2). A chaque nœud, on a deux branches: une pour chacune des alternatives: t1 avant t2 ou t2 avant t1. C'est la règle de séparation.

Lors de l'exploration de l'arbre, à chaque nœud de décision, la méthode calcule une fonction prédictive ("entropic function") permettant de choisir la branche la plus intéressante (ils utilisent une heuristique qui cherche à minimiser les temps morts sur la ressource). C'est la fonction d'évaluation.

Enfin, la stratégie d'exploration de cette méthode consiste à choisir la paire dans chaque nœud: ils utilisent pour cela une heuristique, dans l'ensemble des tâches non encore planifiées sur la ressource, qui choisit une paire de tâches critiques (c'est-à-dire celles qui ont le plus de contraintes) et pouvant être premières ou dernières de cet ensemble.

L'efficacité de ce type de méthode dépend de l'information propagée de nœud en nœud. Un nœud mieux informé permet à la fonction d'évaluation de prendre des décisions plus rapidement (ou meilleures) diminuant ainsi l'arbre de solutions et augmentant les performances de la méthode.

4.3.2. Méthodes de Programmation Logiques sous contraintes

a) Généralités

Les notions sur la programmation logique sous contraintes de cette section sont empruntées au cours de Techniques d'Intelligence Artificielle de Monsieur Jacquet, FUNDP.

La programmation logique sous contraintes (Constraints Logic Programs - CLP) est une famille de langage de programmation qui permet :

- d'exprimer les objets d'un problème,
- d'exprimer les contraintes du problème par des règles.

Un langage CLP(X) est défini par:

- des contraintes sur un domaine de valeurs X
- un résolveur de domaine X
- un simplificateur de contraintes de domaine X

Sans rentrer dans les détails, le principe de résolution du résolveur est de partir d'un but initial et d'essayer de dériver le but courant (simplifié) par l'application d'une règle. Par dérivations successives, le résolveur construit un arbre de dérivation jusqu'à une solution ou un échec s'il n'y a pas de solutions.

b) CLP(FD)

Nous nous intéressons plus particulièrement aux méthodes recherchant la satisfaction de contraintes en programmation logique sur des domaines finis d'entiers, en abrégé CLP(FD).

Le principe de ces méthodes est de diminuer la taille de l'arbre de solutions en diminuant les valeurs possibles que peuvent prendre les variables du problème. Cela peut se faire puisque les domaines des variables sont finis.

On a toujours deux phases:

1ère Phase: Propagation de contraintes

Dans cette phase, on utilise les contraintes du problèmes pour diminuer autant que possible (par propagation) les domaines des variables.

Par exemple:

Soient x de domaine initial [1..10] et y de domaine initial [4..8]
et les contraintes suivantes: $x \geq 5$ et $x \leq y$
Les valeurs possibles pour x sont maintenant [5..8] et pour y [5..8]

2ème Phase: Génération de solutions

Dans cette phase, le résolveur énumère les solutions possibles (labelling) c'est-à-dire qu'il attribue à chaque variable une valeur appartenant à son domaine.

Par exemple: en reprenant le problème de l'exemple ci-dessus,
$x=5$ et $y=5$ est une solution possible

Remarque: Il existe plusieurs stratégies possibles dans l'ordre de choix des variables à énumérer ainsi que dans l'ordre d'énumération des valeurs possibles du domaine. Par exemple, les variables les plus contraintes d'abord, celles qui ont le plus petit domaine d'abord...

Si on souhaite une autre solution ou si on débouche sur un échec (c'est-à-dire que les valeurs attribuées aux variables ne satisfont pas les contraintes), le résolveur réagit en faisant un "retour en arrière" (backtracking). Il remet en cause les valeurs attribuées aux variables déjà "énumérées", attribue (si c'est possible) une nouvelle valeur et recommence.

Par exemple: en reprenant le problème de l'exemple ci-dessus,
$x=6$ et $y=5$ n'est pas une solution: c'est un échec
le résolveur énumère une nouvelle valeur pour y : $x=6$ et $y=6$

Dans cette étape, on réalise donc une énumération de valeurs pour chaque domaine jusqu'à ce qu'on ait une solution réalisable obtenue lorsqu'on a pu donner une valeur pour chaque variable ou jusqu'à ce que, après énumération complète des domaines, on ne puisse trouver aucune valeur satisfaisant les contraintes (car, par exemple, on a des contraintes contradictoires). Dans ce cas, on termine par un échec.

c) Optimisation en CLP(FD)

Dans tout problème d'optimisation, on cherche à minimiser ou maximiser une fonction économique. Par exemple, pour notre problème, cette fonction économique est le temps de fin de traitement de tous les échantillons, depuis la station d'entrée jusqu'à la station de sortie de l'atelier robotisé, que l'on cherche à minimiser.

La recherche d'une solution optimale consiste à trouver d'abord une solution réalisable: on obtient aussi une valeur pour la fonction économique. En utilisant l'algorithme du "branch and bounds", le résolveur recommence la recherche d'une solution en introduisant au préalable une contrainte supplémentaire forçant la solution recherchée à être meilleure que la solution actuelle.

d) Critique

L'énumération au cours de la deuxième étape peut être la partie la plus coûteuse en temps. Par conséquent, plus les domaines sont contraints et plus rapidement une solution (ou un échec) sera dégagée car on aura très peu de valeurs à énumérer.

4.4. Conclusion

Bien que les problèmes d'ordonnancement soient des problèmes NP-difficiles, nous avons trouvé dans ce chapitre des méthodes de résolutions applicables.

Cependant, aucune de ces méthodes n'est parfaite. Le choix dépend du type de problème et du temps qu'on est prêt à consentir pour trouver une solution optimale.

Chapitre 5: Application d'une méthode CLP(FD) au problème d'optimisation de l'atelier robotisé du Laboratoire SICP

Dans ce chapitre, nous appliquons une méthode de programmation logique sous contraintes sur des domaines finis CLP(FD) au problème d'optimisation de l'atelier robotisé du Laboratoire SICP afin de produire une planification réalisable et optimale.

Nous avons choisi une méthode CLP(FD) parce que la solution (le programme) est plus proche de l'énoncé du problème (contraintes).

De plus, de nombreux outils CLP(FD) sont maintenant disponibles sur le marché. Deux d'entre eux ont été essayés et ont montré leurs qualités et leur limites: GNU Prolog et IF/Prolog. L'utilisation de ces outils a montré la nécessité d'utiliser des prédicats optimisés pour la propagation efficace des contraintes.

Finalement, nous présentons les résultats obtenus par la version IF/Prolog sur l'exemple du tableau 2.2 du chapitre 2, comparés avec ceux obtenus par l'heuristique actuelle et par une heuristique "gloutonne".

5.1. Introduction

5.1.1. Généralités

Afin d'illustrer ce chapitre, nous allons reprendre l'exemple du paragraphe 2.5 (tableau 2.2) que nous allons utiliser pour modéliser notre problème en utilisant la syntaxe d'un résolveur CLP(FD).

La syntaxe d'un résolveur CLP(FD) est celle du langage Prolog, auquel viennent s'ajouter des prédicats propres à CLP(FD). Nous n'expliquons pas dans ce travail le langage Prolog . Nous renvoyons le lecteur vers des livres spécialisés (par exemple, *Sterling L., Shapiro E., The Art of Prolog*).

Pour que cette illustration soit la plus explicite même pour des personnes ne connaissant pas le langage Prolog, dans la mesure du possible, nous n'emploierons pas des structures récursives.

5.1.2. Simplification du problème

L'évaluation dynamique de la durée de déplacement "à vide" du robot telle qu'elle est décrite dans 3.6.1. n'est pas évidente[1] à réaliser (en tout cas, pas en utilisant le jeu d'instructions standard) avec les deux résolveurs utilisés (GNU Prolog et IF/Prolog). C'est pourquoi nous avons considéré que la durée de déplacement du robot "à vide" était une constante. Nous lui avons donné une valeur moyenne de 15 secondes (calculée sur une journée de travail).

Vu l'exemple choisi pour illustrer ce chapitre, les urgences des échantillons n'ont pas été envisagées ici. Cependant, la prise en compte des urgences dans la planification n'est pas une difficulté en soi. Une façon de faire consiste à séparer chaque groupe de tâches d'échantillons de même urgence par des tâches fictives, puis d'ajouter des contraintes de précédence entre les groupes de tâches et les tâches fictives:

Exemple:

- G_0, G_1, G_2 sont les groupes des tâches d'éch. d'urgence 0, 1, 2 respectivement
- tf_1 et tf_2 deux tâches fictives de durée = 0

alors on a $G_0 << tf_1 << G_1 << tf_2 << G_2$

[1] On peut modifier le compilateur pour l'enrichir de nouveaux prédicats spécifiques à notre problème: nous ne l'avons pas fait dans ce mémoire vu la difficulté qu'implique un tel travail.

5.1.3. Modification de la table des traitements

La table des traitements, telle qu'elle se présente dans le paragraphe 2.5. du chapitre 2, présume des conventions implicites qu'il vaut mieux faire apparaître explicitement pour pouvoir générer le programme logique.
Les règles de transformation peuvent être facilement automatisées:

1) Pour des raisons de manipulation, les noms des échantillons et le noms des tâches vont être codés.

   - le nom de l'échantillon est remplacé par E + le code de l'échantillon
   - on crée un nom de tâche par ressource type:
     - T02x : tâche "de refroidissement" de
       - 1 pour les flacons de 100 ml
       - 2 pour les flacons de 250 ml
       - 3 pour les flacons de 500 ml
     - T06x : tâche "de pompage" où x=
       - 1 pour un échantillon de type "ACIDE"
       - 2 pour un échantillon de type "EAU PROPRE" ou "EAU CONTAMINEE"
       - 3 pour un échantillon de type "EAU OXYGENEE"
       - 4 pour un échantillon de type "SAUMURE"
       - 5 pour un échantillon de type "MERCURE"
     - T08 : tâche "densité"
     - T11 : tâche "aller au terminus"

2) On ajoute les tâches "dévissage", "revissage" explicitement pour tout échantillon qui a une tâche "densité" ou "pompage"

   - T041 : tâche de dévissage pour des flacons de 100 et 250 ml
   - T042 : tâche de dévissage pour des flacons de 500 et 1000 ml
   - T101 : tâche de revissage pour des flacons de 100 et 250 ml
   - T102 : tâche de revissage pour des flacons de 500 et 1000 ml

3) On ajoute une tâche de "déplacement du robot" explicitement vers chaque station

   - T01 : tâche de déplacement du robot vers la station refroidisseur
   - T03 : tâche de déplacement du robot vers la station de dévissage / revissage
   - T05 : tâche de déplacement du robot vers la station de pompage
   - T07 : tâche de déplacement du robot vers la station de densité
   - T09 : tâche de déplacement du robot vers la station de dévissage / revissage

4) On ajoute les durées de chaque tâche / échantillon dans la table

   - les durées des tâches "déplacement robot" (2.6.) incluent également la durée moyenne de déplacement "à vide" du robot (voir 5.1.2)
   - une tâche / échantillon qui n'a pas de temps dans la table signifie que la tâche n'existe pas pour cet échantillon

5) On "allonge" la durée de toutes les tâches (≠ de celles utilisant le robot) avec les durées des tâches de déplacement respectives comme décrit dans 3.6.3. (formule 3.8).

Après transformation, on obtient une table comme représentée par le tableau 5.1 .

Tableau 5.1 Table des durées des traitements transformée

Ech.	T01	T021	T022	T023	T03	T041	T042	T05	T061	T062	T063	T064	T065	T07	T08	T09	T101	T102	T11
E026					30		70							30	390	30		70	35
E123	45			2445															30
E124	45	1245			45	85		30				630				30	70		35
E698	45	1245			45	85		30				630				30	70		35
E146	45		1845		45	85								30	390	30	70		35
E308					30	70		30	490							30	70		35
E309					30	70		30	490							30	70		35
E312					30	70		30	490							30	70		35
E627					30	70		30	490					30	390	30	70		35

Le tableau 5.2 donne la ressource type utilisée par chaque tâche (une seule ressource est utilisée par tâche, voir contrainte fonctionnelle 2.4.2.). Ce tableau donne également le nombre total de ressources disponibles dans l'atelier pour chaque type de ressources.

Tableau 5.2 Association tâches / ressources type

Tâche	Ressource type utilisée	Nombre total
T01	robot	1
T021	cooler 100 ml	12
T022	cooler 250 ml	8
T023	cooler 500 ml	4
T03	robot	1
T041	decap 100 ml & 250 ml	1
T042	decap 500 ml & 1000 ml	1
T05	robot	1
T061	pomp "ACIDE"	1
T062	pomp "EAU PURE" & "EAU IMPURE"	2
T063	pomp "EAU OXYGENEE"	1
T064	pomp "SAUMURE"	1
T065	pomp "MERCURE"	1
T07	robot	1
T08	dens	1
T09	robot	1
T101	decap 100 ml & 250 ml	1
T102	decap 500 ml & 1000 ml	1
T11	robot	1

5.1.4. Variables de domaine fini

Pour notre problème, on peut représenter chaque début de tâche par une variable de domaine fini : par exemple, un intervalle d'entiers compris entre 0 et 28800 représentant, en secondes, la journée de travail de l'atelier.

a) variables de tâche

Par convention, le nom d'une tâche d'un échantillon est composé de la concaténation du nom abrégé de l'échantillon et du non abrégé de la tâche: par exemple, la tâche de refroidissement de l'échantillon E123 s'appelle E123T023. On associe à chacune de ces tâches une variable de domaine.

b) variables de job

On crée également une variable par échantillon: cette variable est associée à une tâche fictive représentant la fin du traitement du travail (job) d'un échantillon. Par convention, elle portera le nom abrégé de l'échantillon. La syntaxe exacte et leur définition dépendent de l'outil CLP(FD) utilisé (chacun ayant son "dialecte"). On crée un variable par une commande ayant le format suivant : liste_de_variables in [0,28800]

c) variables de durée

On crée une variable de durée par tâche. Pour les nommées, nous utilisons la même convention que pour les variables de tâche mis à part que la lettre 'D' remplace la lettre 'T'

5.2. GNU Prolog

Dans ce paragraphe, nous présentons les essais de modélisation avec GNU Prolog.

GNU Prolog est un compilateur Prolog disposant d'un résolveur de contraintes sur des domaines finis. Il a été développé à l'origine à l'INRIA en France par Daniel Diaz (voir [Diaz, 1996] et [Diaz, 1999]) et mis par son auteur dans le domaine public par l'intermédiaire de la communauté GNU (http://www.gnu.org/software/prolog).

Le jeu de prédicats offerts par cet outil répond à la norme ISO Prolog, plus quelques méta-prédicats non standards pour la CLP(FD).

Dans les sections suivantes, nous présentons succinctement le codage dans GNU Prolog du problème et de ces contraintes vues au chapitre 3. Le programme logique complet écrit en GNU Prolog se trouve en annexe C.

5.2.1. Contraintes de synchronisation

Dans notre problème, les contraintes de synchronisation concernent les tâches qui doivent être synchronisées exactement (voir 3.6.3). Suite à la simplification 5.1.2., on fait correspondre exactement le début de la tâche "déplacement robot vers station x" et le début de tâche suivant le déplacement dans le traitement de l'échantillon (dont la durée a été "allongée" par la durée du déplacement du robot, dans la 5ème règle de transformation ci-dessus). Ceci nous assure la réservation de la ressource dès que le robot réalise la tâche de déplacement.

Par exemple: pour exprimer que la tâche de refroidissement de l'échantillon E146 doit commencer en même temps que la tâche de déplacement vers la station refroidisseur, on écrit `E146T021 #= E146T01`

5.2.2. Contraintes de précédence

Exprimer les contraintes de précédence consiste à écrire toutes les formules (voir 3.1.1., formule 3.1) des tâches successives d'un même échantillon.

Prenons, par exemple, le travail (Job) de l'échantillon E146. Les contraintes de précédence et de synchronisation s'expriment par le programme logique suivant:

```
prec([E146T01,E146T022,E146T03,E146T041,E146T07,E146T08,E146T09,
      E146T101, E146T11,E146]):-
            E146T022 #=  E146T01,
            E146T041 #>= E146T01   + 45,
            E146T041 #>= E146T022  + 1845,
            E146T041 #=  E146T03,
            E146T08  #>= E146T03   + 45,
            E146T08  #>= E146T041  + 85,
            E146T08  #=  E146T07,
            E146T101 #>= E146T07   + 30,
            E146T101 #>= E146T08   + 390,
            E146T101 #=  E146T09,
            E146T011 #>= E146T09   + 45,
            E146T011 #>= E146T101  + 85,
            E146     #>= E146T11   + 35.
```

Remarque: la variable E146 est le temps total de réalisation du job de l'échantillon E146.

5.2.3. Contraintes disjonctives

Pour représenter la contrainte décrite en 3.3.3, on va généraliser le problème. Soient: Aa, Ba des variables de domaine finis de deux tâches utilisant une même ressource, et Ad, Bd leur durée respective.

Aa et Ba ne peuvent avoir lieu en même temps car (1) elles utilisent une ressource unique (voir tableau 5.1: ex. toutes les tâches utilisant la ressource robot) ou (2) parce qu'elles appartiennent à un même échantillon et doivent se succéder dans un ordre indifférent (exemple, les tâches de densité et de pompage de l'échantillon E627 du tableau 5.2).

On peut modéliser la contrainte disjonctive par le programme logique suivant:

```
disjonctive(Aa,Ad,Ba,_Bd[2]):-
      Ba #>= Aa+Ad.

disjonctive(Aa,_Ad,Ba,Bd):-
      Aa #>= Ba+Bd.
```

Le résolveur essayera le premier prédicat puis l'autre (dans cet ordre puisque le résolveur suit l'ordre d'écriture des règles) et gardera la solution la plus intéressante lors de la minimisation.

Nous devons donc écrire cette contrainte pour toutes les paires de tâches concernées par le cas (1) ou par le cas (2).

5.2.4. Contraintes cumulatives

Malheureusement, la contrainte cumulative ne fait pas partie des méta-prédicats fournis en standard. Malgré toutes nos tentatives d'implémentation, aucune n'a abouti efficacement. Nous avons contacté Monsieur Diaz qui nous a confirmé l'extrême difficulté d'implémentation de cette contrainte qui doit être implémentée, d'après-lui, directement dans le corps du compilateur pour être efficace. Nous ne l'avons pas fait car, vu la difficulté de ce travail, le codage de cette contrainte aurait constitué l'équivalent d'un travail de mémoire à lui seul.

5.2.5. Enumération de valeurs

Comme nous l'avons vu au chapitre 4 (4.3.2.), la phase d'énumération consiste à donner une valeur parmi les valeurs possibles du domaine à toutes les variables.

On utilise le prédicat fd_labelling/1[3]

[2] le symbole '_' signifie que cette variable n'est pas utile dans le prédicat
[3] le symbole '/' suivi par un nombre exprime l'arité d'un prédicat, c'est-à-dire son nombre d'argument

```
Soit L la liste de toutes les variables de domaine:
[E146T01,E146T021,E146T03,E146T041,E146T07,E146T08,E146T09,E146T
101,E146T11,…,E026,E123,E124,E698,E146,E308,E309,E312,E627],

fd_labelling(L).
```

Nous n'avons pas essayé des stratégies particulières d'énumération.

5.2.6. Minimisation

La minimisation consiste à minimiser la fonction économique c'est-à-dire, pour notre problème, à trouver des valeurs pour les variables qui minimisent les temps de fin de tous les travaux des échantillons (rappel: un travail est le traitement de toutes les tâches d'un échantillon).

GNU Prolog n'a pas d'instructions permettant de minimiser la valeur maximale d'une liste de variables. On doit créer une variable fictive END (toujours de domaine [0,28800]) et de durée nulle: on ajoute les contraintes de précédence suivantes à toutes les autres:

```
END #>= E026,
END #>= E123,
END #>= E124,
END #>= E698,
END #>= E146,
END #>= E308,
END #>= E309,
END #>= E312,
END #>= E627
```

On utilise le prédicat fd_minimize/2 avec le prédicat fd_labeling/1:

```
Soit L la liste de toutes les variables de domaine:
[E146T01,E146T021,E146T03,E146T041,E146T07,E146T08,E146T09,E146T101,E1
46T11,…,E026,E123,E124,E698,E146,E308,E309,E312,E627,END],

fd_minimize(fd_labelling(L),END).
```

Le résolveur GNU Prolog cherche à énumérer des valeurs pour toutes les variables de L en utilisant l'algorithme "branch and bounds" avec redémarrage (voir le 4.3.) pour minimiser la valeur de la variable END .

5.2.7. Résultats des tests avec GNU Prolog

Puisque nous n'avons pas pu implémenter la contrainte cumulative, les essais n'ont pas atteint totalement leur objectif. Néanmoins, la facilité d'utilisation de GNU Prolog et l'expressivité du langage Prolog nous ont permis de coder facilement les contraintes de précédence et les contraintes disjonctives.

Cependant, les performances du programme créé décroissent assez rapidement dès que le nombre d'échantillons augmente. Cela s'explique par le fait que la

contrainte disjonctive, utilisée comme contrainte cumulative de capacité 1 (voir 3.3.3.), ne propage aucune information permettant au résolveur de prendre la bonne décision.

Cette implémentation par GNU Prolog n'est donc pas applicable à notre problème.

5.3. IF/Prolog

IF/Prolog est un outil professionnel développé par Siemens AG Autriche et vendu en Europe par IF Computer GmbH (http://www.ifcomputer.de). C'est aussi un compilateur Prolog dont le jeu de prédicats répond également à la norme ISO Prolog, et auquel s'ajoutent quelques méta-prédicats non standards (et différents de GNU Prolog !) pour la CLP(FD).

Nous avons obtenu une licence d'essai (3 mois) pour étudier IF/Prolog. La prise en main de l'outil s'est avérée un peu plus facile que pour GNU Prolog: l'environnement est plus convivial et mieux documenté. Néanmoins, les possibilités du système ont nécessité un temps d'apprentissage non négligeable. Le programme logique complet écrit en IF/Prolog se trouve en annexe D.

5.3.1. Contraintes de précédence et de synchronisation

Ces contraintes s'expriment de la même façon qu'en GNU Prolog, mis à part une petite différence de syntaxe: le '?' remplace le '#' dans IF/Prolog (voir 5.2.2.).

5.3.2. Contraintes disjonctives

IF/Prolog fournit un méta-prédicat optimisé (voir [IF/Prolog,1999]) pour exprimer les disjonctions de tâches, disjonctive/2. Ce prédicat prend deux arguments:

- une liste de tâches
- la liste de leurs durées respectives.

Pour notre problème, lorsque nous avons les tâches de densité et de pompage sur un échantillon, la réalisation de celles-ci est disjonctive c'est-à-dire qu'elles ne peuvent être réalisées en même temps.

```
Dans notre exemple, l'échantillon E627 a une tâche de densité
(E627T08) et une tâche de pompage (E627T061), dont les durées
respectives sont (E627D08) et (E627D061):

    disjonctive([E627T08, E627T061],[E627D08, E627D061])
```

5.3.3. Contraintes cumulatives

IF/Prolog fournit un méta-prédicat optimisé, cumulative/4, permettant d'exprimer les contraintes cumulatives sur l'utilisation des ressources. Il prend quatre arguments:

- une liste de tâches
- la liste de leurs durées respectives
- la liste des quantités de ressources utilisées respectivement par chaque tâches
- le nombre total de ressources disponibles

Pour notre problème, on aura une contrainte cumulative pour toutes les tâches utilisant une même ressource type. Il est à noter que chaque tâche de notre problème n'utilise qu'une seule ressource.

Pour notre exemple, deux échantillons (E124 et E698) ont chacun une tâche de refroidissement (E124T021,E698T021) utilisant la même ressource type ("cooler pour flacon de 100ml"), dont le nombre de ressources disponibles (les positions dans la station de refroidissement réservées aux flacons de 100ml) est de 12.

On écrira:

```
cumulative([E124T021,E698T021],[E124D021,E698D021],[1,1],12)
```

5.3.4. Contraintes sur les temps d'occupation des ressources

Les contraintes cumulatives et disjonctives, telles qu'exprimées en 5.3.2. et 5.3.3., ne tiennent pas compte des "temps morts" des échantillons sur les ressources (voir 3.6.2). Elles doivent donc être modifiées pour utiliser les durées d'occupation réelles des ressources par les échantillons plutôt que celles de réalisation des tâches. On introduit de nouvelles variables exprimant la durée d'occupation d'une ressource.

Pour notre exemple, la contrainte cumulative exprimée en 5.3.3. doit être modifiée:

```
E124O021 ?= E124T064 - E124T021,
E698O021 ?= E698T064 - E698T021,
cumulative([E124T021,E698T021],[E124O021,E698O021],[1,1],12).
```

5.3.5. Enumération et Minimisation

L'énumération et la minimisation s'exprime comme en GNU Prolog (5.2.5. et 5.2.6.), mis à part que IF/Prolog fournit un méta-prédicat qui permet de rechercher le minimum d'une liste de variables.

```
Soit L la liste de toutes les variables de tâches et Jobs la liste de
toutes les variables de fin de traitement de chaque échantillon:
L=[E146T01,E146T021,E146T03,E146T041,E146T07,E146T08,E146T09, ...],
Jobs = [E026,E123,E124,E698,E146,E308,E309,E312,E627],

minimize_maximum(label(L),Jobs).
```

5.3.6. Résultats des tests avec IF/Prolog

Bien que toutes les contraintes aient été exprimés dans le programme logique, celui-ci ne donne toujours pas de réponse après un jour d'exécution (même en supprimant la minimisation) sur les 9 échantillons du tableau 5.1. ! La raison de cet échec vient des contraintes vérifiant les "temps morts" d'utilisation des ressources (5.3.4).

Nous avons donc recherché une alternative a cet inconvénient en partant du fait que dévisser un flacon est contraint par deux règles (voir chapitre 2):

- la station de dévissage utilisée ne peut garder qu'au plus 6 bouchons,
- pour des raison de salubrité, le temps pendant lequel le flacon est débouché doit être le plus court possible.

Nous avons modifié notre programme logique pour supprimer la vérification des "temps morts" sur les ressources types "decapx" , "pompx" et "dens", et introduit une tâche fictive pour chaque échantillon utilisant ces ressources types:

- la tâche fictive débute en même temps que la tâche de dévissage et se termine au plus tard en même temps que la tâche de revissage;

- la durée de la tâche fictive est exactement la durée cumulée des tâches comprises entre la tâche de dévissage et la tâche de revissage dans le travail de l'échantillon;

- deux nouvelles contraintes cumulatives, impliquant les tâches fictives, sont créées pour deux ressources types fictives (une pour chaque station de dévissage / revissage) où le nombre total de ressources disponibles est égal à 6;

- les contraintes cumulatives concernant les ressources types "decapx" , "pompx" et "dens" sont modifiées pour qu'elles utilisent les temps réels des tâches.

Par ces modifications, nous avons imposé qu'il n'y ait plus de "temps morts" sur les stations entre le moment où un échantillon est débouché et le moment où il est rebouché. Evidement, cette modification va enlever de la souplesse dans les possibilités de choix d'ordonnancement qu'aura IF/Prolog.

L'exécution de ce programme, s'avère encore trop long[4] (plus de 9 heures !) . Nous décidons de ne pas rechercher la minimisation. Cette fois-ci, IF/Prolog fournit une réponse en moins de 1 seconde sur un Pentium II, 450 Mhz (voir paragraphe suivant).

5.4. Résultats comparés

Dans ce paragraphe, nous présentons les résultats obtenus par l'heuristique actuelle (2.7.), l'heuristique "gloutonne" (voir 4.2.1.) et CLP(FD) (voir 5.3.6) sur l'exemple décrit par le tableau 5.1.

5.4.1. Heuristique actuelle

Dans notre test, l'heuristique actuelle a pris les échantillons dans l'ordre: [E026,E123,E124,E698,E146,E308,E309,E312,E627]

L'heuristique actuelle est pénalisée par son comportement :
- elle choisit au hasard les échantillons qu'elle prend dans la station d'entrée.
- elle impose un ordre arbitraire entre la tâche de densité et la tâche de pompage.
- elle ne réalise la tâche de dévissage que si elle a réservé la ressource de la tâche suivante.

Le tableau 5.3 présente les temps de début et de fin de chaque tâche obtenus par cette méthode.

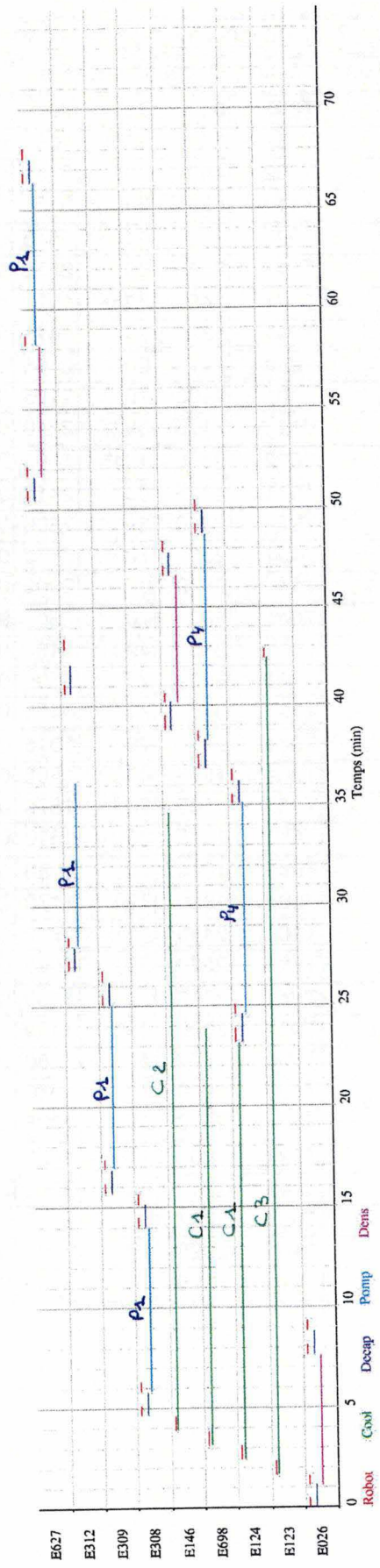
La figure 5.1 présente le diagramme de Gantt de l'ordonnancement réalisé.

[4] Cette solution pourrait déjà être acceptable sur un ordinateur plus puissant

Tableau 5.3 Ordonnancement par l'heuristique actuelle

Echantillons	Tâches	Début	Durée	Fin	temps mort
E026	T042	0	70	70	0
E026	T08	70	390	460	0
E026	T102	460	70	530	0
E026	T11	530	35	565	0
E123	T023	100	2445	2545	0
E123	T11	2545	30	2575	0
E124	T021	145	1245	1390	0
E124	T041	1390	85	1475	0
E124	T064	1475	630	2105	0
E124	T101	2105	70	2175	0
E124	T11	2175	35	2210	0
E146	T022	235	1845	2080	245
E146	T041	2325	85	2410	0
E146	T08	2410	390	2800	0
E146	T101	2800	70	2870	0
E146	T11	2870	35	2905	0
E308	T041	280	70	350	0
E308	T061	350	490	840	0
E308	T101	840	70	910	0
E308	T11	910	35	945	0
E309	T041	945	70	1015	0
E309	T061	1015	490	1505	0
E309	T101	1505	70	1575	0
E309	T11	1575	35	1610	0
E312	T041	1610	70	1680	0
E312	T061	1680	490	2170	270
E312	T101	2440	70	2510	0
E312	T11	2510	35	2545	0
E627	T041	3030	70	3100	0
E627	T08	3100	390	3490	0
E627	T061	3490	490	3980	0
E627	T101	3980	70	4050	0
E627	T11	4050	35	4085	0
E698	T021	190	1245	1435	775
E698	T041	2210	85	2295	0
E698	T064	2295	630	2925	0
E698	T101	2925	70	2995	0
E698	T11	2995	35	3030	0

Figure 5.1 Diagramme de Gantt de l'ordonnancement réalisé par l'heuristique actuelle



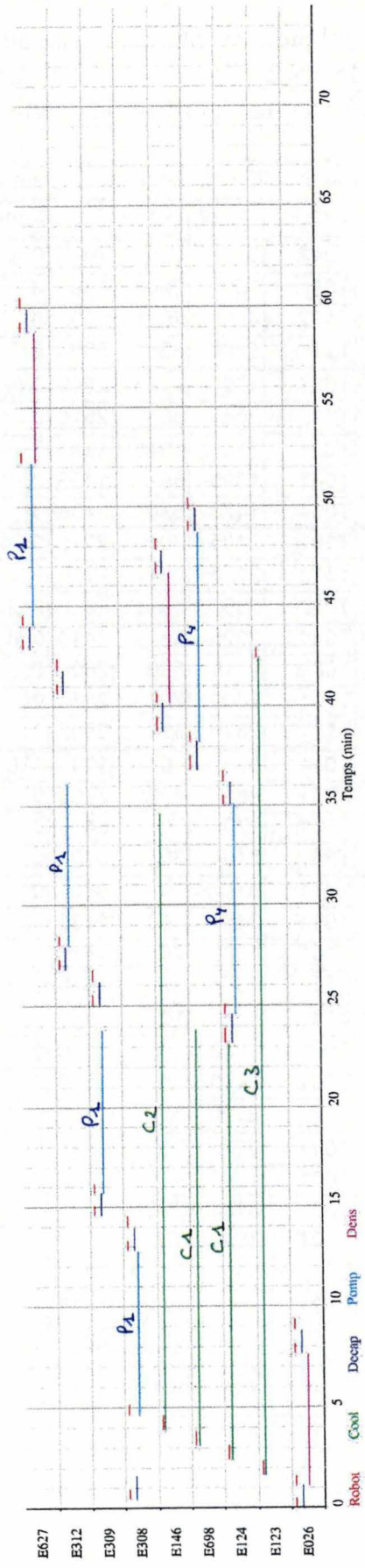
5.4.2. Heuristique "gloutonne"

Nous avons simulé l'algorithme avec la même liste que celle des échantillons pris de manière aléatoire par le système actuel (5.4.1.) . Le tableau 5.4 et la figure 5.2 (diagramme de Gantt) présente la planification obtenue:

Tableau 5.4 Planification par heuristique gloutonne

Echantillons	Tâches	Début	Durée	Fin	temps mort
E026	T042	0	70	70	0
E026	T08	70	390	460	0
E026	T102	460	70	530	0
E026	T11	530	35	565	0
E123	T023	100	2445	2545	0
E123	T11	2545	30	2575	0
E124	T021	145	1245	1390	0
E124	T041	1390	85	1475	0
E124	T064	1475	630	2105	0
E124	T101	2105	70	2175	0
E124	T11	2175	35	2210	0
E146	T022	235	1845	2080	245
E146	T041	2325	85	2410	0
E146	T08	2410	390	2800	0
E146	T101	2800	70	2870	0
E146	T11	2870	35	2905	0
E308	T041	30	70	100	180
E308	T061	280	490	770	0
E308	T101	770	70	840	0
E308	T11	840	35	875	0
E309	T041	875	70	945	0
E309	T061	945	490	1435	70
E309	T101	1505	70	1575	0
E309	T11	1575	35	1610	0
E312	T041	1610	70	1680	0
E312	T061	1680	490	2170	270
E312	T101	2440	70	2510	0
E312	T11	2510	35	2545	0
E627	T041	2575	70	2645	0
E627	T061	2645	490	3135	0
E627	T08	3135	390	3525	0
E627	T101	3525	70	3595	0
E627	T11	3595	35	3630	0
E698	T021	190	1245	1435	775
E698	T041	2210	85	2295	0
E698	T064	2295	630	2925	0
E698	T101	2925	70	2995	0
E698	T11	2995	35	3030	0

Figure 5.2 Diagramme de Gantt de la planification obtenu par l'heuristique "gloutonne"



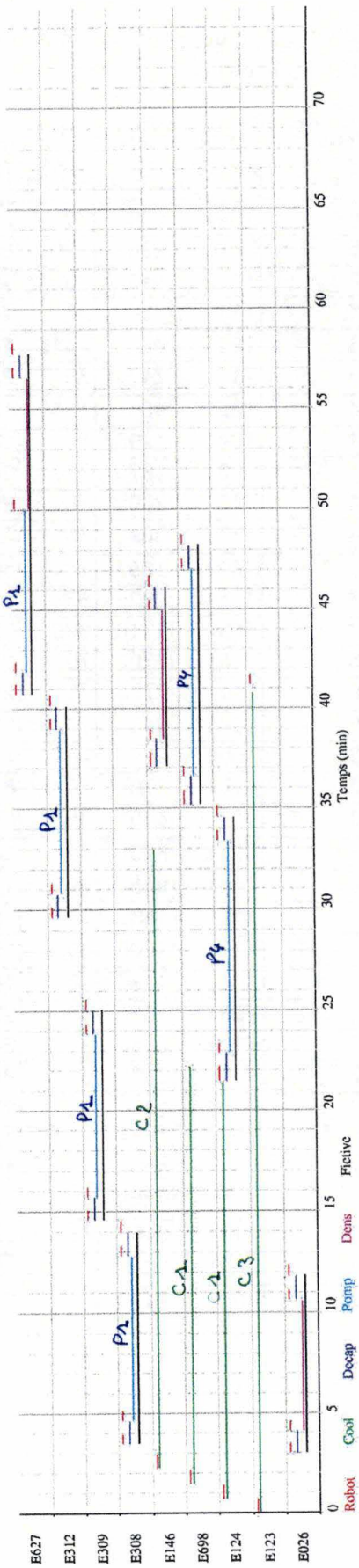
5.4.3. CLP(FD)

Le tableau 5.5 et la figure 5.3 (diagramme de Gantt) présente la planification obtenue par notre programme logique.

Tableau 5.5 Planification par CLP(FD)

Echantillons	Tâches	Début	Durée	Fin	temps mort
E026	T042	180	70	250	0
E026	T08	250	390	640	0
E026	T102	640	70	710	0
E026	T11	710	35	745	0
E123	T023	0	2445	2445	30
E123	T11	2475	30	2505	0
E124	T021	45	1245	1290	0
E124	T041	1290	85	1375	0
E124	T064	1375	630	2005	0
E124	T101	2005	70	2075	0
E124	T11	2075	35	2110	0
E146	T022	135	1845	1980	245
E146	T041	2225	85	2310	0
E146	T08	2310	390	2700	0
E146	T101	2700	70	2770	0
E146	T11	2770	35	2805	0
E308	T041	210	70	280	0
E308	T061	280	490	770	0
E308	T101	770	70	840	0
E308	T11	840	35	875	0
E309	T041	875	70	945	0
E309	T061	945	490	1435	0
E309	T101	1435	70	1505	0
E309	T11	1505	35	1540	0
E312	T041	1780	70	1850	0
E312	T061	1850	490	2340	0
E312	T101	2340	70	2410	0
E312	T11	2410	35	2445	0
E627	T041	2445	70	2515	0
E627	T061	2515	490	3005	0
E627	T08	3005	390	3395	0
E627	T101	3395	70	3465	0
E627	T11	3465	35	3500	0
E698	T021	90	1245	1335	775
E698	T041	2110	85	2195	0
E698	T064	2195	630	2825	0
E698	T101	2825	70	2895	0
E698	T11	2895	35	2930	0

Figure 5.3 Diagramme de Gantt de la planification obtenu par CLP(FD)



5.4.1. Récapitulatif

Le tableau 5.6 donne le récapitulatif des résultats obtenus par les différentes méthodes.

Tableau 5.6 Comparaison des méthodes		Echantillons (sec)									TOTAL
		E026	E123	E124	E698	E146	E308	E309	E312	E627	
Heuristique actuelle	début du travail	0	100	145	190	235	280	945	1610	3030	0
	fin du travail	565	2575	2210	3030	2905	945	1610	2545	4085	4085
	durée totale du travail	565	2475	2065	2840	2670	665	665	935	1055	13935
	durée totale des temps morts	0	0	0	775	245	0	0	270	0	1290
Heuristique gloutonne	début du travail	0	100	145	190	235	30	875	1610	2575	0
	fin du travail	565	2575	2210	3030	2905	875	1610	2545	3630	3630
	durée totale du travail	565	2475	2065	2840	2670	845	735	935	1055	14185
	durée totale des temps morts	0	0	0	775	245	180	70	270	0	1540
CLP(FD)	début du travail	180	0	45	90	135	210	875	1780	2445	0
	fin du travail	745	2505	2110	2930	2805	875	1540	2445	3500	3500
	durée totale du travail	565	2505	2065	2840	2670	665	665	665	1055	13695
	durée totale des temps morts	0	30	0	775	245	0	0	0	0	1050

5.5. Application finale

Par manque de temps, nous n'avons pas développé l'application finale qui permettrait au système robotique de générer la planification et d'exécuter les mouvements du bras robotiques en suivant celle-ci. Mais de manière générale, nous avons imaginé la dynamique d'exécution (voir figure 5.4) suivante pour cette application.

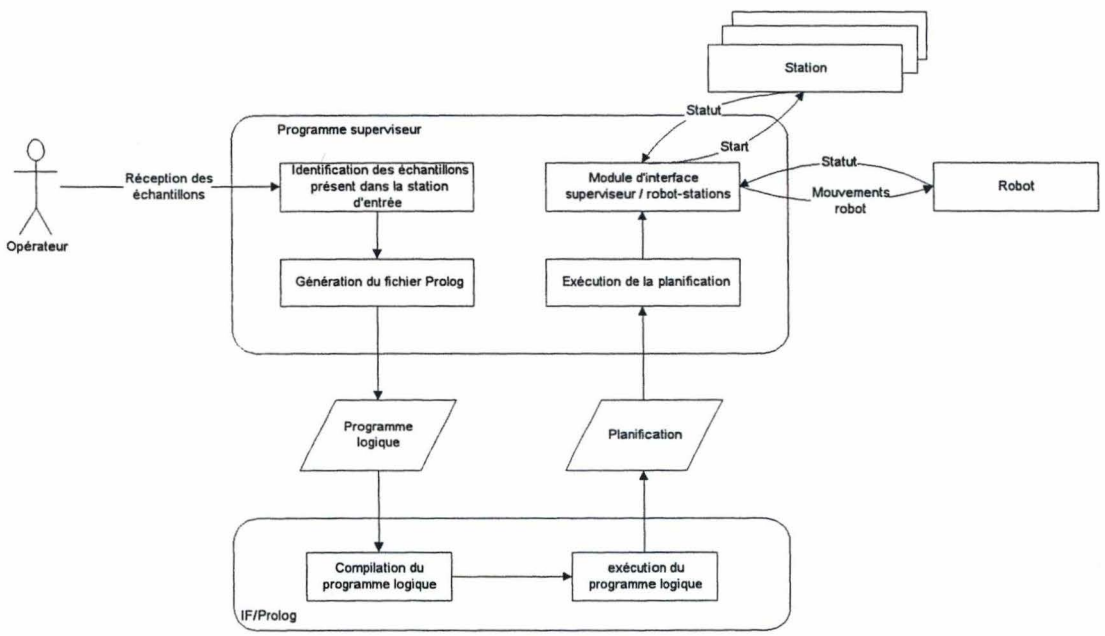


Figure 5.4 Architecture générale de l'application finale

5.6. Conclusion

Dans ce chapitre, nous avons vu comment utiliser la CLP(FD) pour l'appliquer à notre problème. Nous avons vu également que, des deux outils utilisés, seul IF/Prolog dispose de prédicats adéquats et optimisés pour réaliser un ordonnancement comme celui de notre problème.

Malheureusement, les prédicats de IF/Prolog nécessitent de connaître la durée d'utilisation des ressources. Or, pour notre problème, une ressource n'est libérée qu'au moment où la tâche suivante commence et non quand la tâche courante se termine ! Il nous faut donc vérifier la durée d'occupation réel des ressources. Cette vérification augmente considérablement la difficulté du problème.

Après des modifications apportés au programme logique initial, nous avons obtenu une solution réalisable (5.3.6). Ce programme logique ne permet pas d'obtenir une solution optimale. Néanmoins, sans tirer des conclusions définitives, les tests réalisés sur un petit nombre d'échantillons montrent que cette méthode est déjà meilleure que l'heuristique actuelle et qu'une heuristique gloutonne créée pour ce problème.

Conclusion

Dans ce travail, nous avons étudié le problème de l'optimisation d'un atelier robotisé de laboratoire à partir d'une étude de cas réel. Nous avons mis en évidence la difficulté de ce problème d'ordonnancement et des contraintes qui lui sont liées.

L'exploration de différentes méthodes de résolution nous a conduit à envisager plus particulièrement la programmation logique sous contraintes sur des domaines finis. En l'appliquant à notre problème, nous avons montré les possibilités que ce langage offre pour l'expression de contraintes du monde réel.

Cependant, un problème d'ordonnancement reste toujours un problème combinatoire. Dans un problème comme le nôtre, l'accumulation des contraintes rend la recherche d'une solution optimale impossible dans un temps compatible avec le fonctionnement d'un atelier robotisé.

Nous avons réalisé un prototype de programme logique en admettant certaines restrictions à notre problème d'origine, et surtout en diminuant nos prétentions quant à la recherche d'une solution optimale. En l'utilisant sur un nombre d'échantillons restreint, nous avons obtenu une planification réalisable dans un temps compatible avec le fonctionnement d'un atelier robotisé. Cette solution n'est malheureusement pas une solution optimale (du moins en théorie[5]), mais elle est cependant meilleure que l'heuristique actuelle et qu'une heuristique gloutonne réalisée pour ce problème. Par ailleurs, cette solution n'est pas incompatible avec la vue réaliste des entreprises qui considèrent une solution comme acceptable si elle est meilleure que celle actuelle.

Dans cet ordre d'idées, plusieurs perspectives s'offrent à nous. D'abord, en continuant sur la voie de la programmation logique sous contraintes:
- confirmer sur un nombre plus important d'échantillons les résultats obtenus,
- explorer les possibilités offertes par les outils (GNU Prolog, If/Prolog) pour enrichir le compilateur de nouveaux prédicats afin d'améliorer le programme.

Ensuite, l'implémentation efficace des méthodes "approchées" envisagées au chapitre 4 devra également être envisagée afin de vérifier leur comportement face à un nombre plus important d'échantillons.

Nous pouvons conclure que l'étude de la programmation logique sous contraintes et les outils envisagés (IF/Prolog, GNU Prolog) nous ont ouvert des perspectives d'applications industrielles très importantes. En effet, les problèmes industriels comportent généralement de nombreuses contraintes souvent mal définies ou mal exprimées par les utilisateurs. L'expressivité du langage nous a permis de modéliser des contraintes complexes et de les modifier facilement pour les adapter à de nouvelles conditions. De plus, la description du problème qui a été réalisée au chapitre 3 est préalable à toute implémentation qu'elle soit en programmation logique ou autre. Elle sera utile pour les prochaines implémentations de quelques natures qu'elles soient.

[5] En effet, le problème peut être suffisamment contraint pour donner "fortuitement" une solution optimale

Références bibliographiques

A. Aho, J. Ullman, *Concepts fondamentaux de l'informatique*, Dunod, p95-162, 1993.

Y. Caseau, F. Laburthe, *Disjonctive scheduling with Task Intervals*, Liens Technical Report n°95-25, Laboratoire d'Informatique de l'Ecole Normale Supérieure, 1995.

Y. Caseau, F. Laburthe, *Cumulative scheduling with Task Intervals*, Proceedings of the Joint International Conference and Symposium on Logic Programming, MIT Press, 1996.

T. Cormen, C. Leiserson, R. Rivest, *Introduction à l'algorithmique*, Dunod, p930, 1994.

D. Diaz, *Thèse de doctorat*, Université d'Orléans, France, annexe H, 1995.

D. Diaz, *GNU Prolog: Reference Manual*, GNU, 1999.

IF/Prolog, *Constraints Package*, Siemens AG Austria,1999.

C. Le Pape, Ph. Baptiste, *Claire Schedule 1.1, Reference Manual*, Bouygues scientific department technical report, St Quentin, 1997.

S. Pieret, *Organisation d'un laboratoire de contrôle COMPOUNDS intégrant un système robotique*, Université de Liège, 1992.

Ch. Prins, *Algorithmes de graphes*, Eyrolles, Paris, France, p50-52, 1994.

Ch. Schulte, G. Smolka, *Finite Domain Constraint Programming in Oz*, p75-96, 2000.

P. Van Hentenryck, *Constraint Satifaction in Logic Programming*, MIT Press, p162-174, 1989.

Annexes

A. Table des traitements de l'atelier SICP
B. Spécification du bras robotique de SICP
C. Prototype d'un programme logique en GNU Prolog
D. Prototype d'un programme logique en IF/Prolog

Fabrication	Nom du point	Libellé	Nom	Code	Ugence	Flacon	Refroidir	Refroidissement (sec)	Densité	Tps densité (sec)	Matrice	Pompe	Tps Pomp (sec)	Couloir Sortie	
PCM	A030	HYPO PF. MOYEN DES MAG. N° 6 ET 7	A030/E2/1/1111	26		0 PLT500	FAUX		VRAI	360		0			5
UEM	C003	EAU DEMINERALISEE ENTREE UEM	C115/E1/1/0111	739		0 PLT500	VRAI		2400 FAUX			0			3
UEM	C110	SAUMURE ENTREE RESINES	C115/E1/1/1000	123		0 PLT500	VRAI		2400 FAUX			0			3
UEM	C110	SAUMURE ENTREE RESINES	C115/E3_A/1/1111	124		0 PLT100	VRAI		1200 FAUX		SAUMURE	4	600		5
UEM	C115	SAUMURE ENTREE CELLULES	C115/E3_B/1/1111	698		0 PLT100	VRAI		1200 FAUX		SAUMURE	4	600		5
UEM	C115	SAUMURE ENTREE CELLULES	C425/E1/1/1111	146		0 PLT250	VRAI		1800 VRAI	360		0			3
UEM	C115	SAUMURE ENTREE CELLULES	G330/E1/1/1111	308		0 PLT100	FAUX		FAUX		ACIDE	1	460		8
UEM	C115	SAUMURE ENTREE CELLULES	G410/E1/1/1111	309		0 PLT100	FAUX		FAUX		ACIDE	1	460		8
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	G510/E1/1/1111	15023		0 PLT100	FAUX		FAUX		ACIDE	1	460		8
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	G520/E1/1/1111	15186		0 PLT100	FAUX		FAUX		ACIDE	1	460		8
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	G530/E1/1/1111	312		0 PLT100	FAUX		FAUX		ACIDE	1	460		8
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	G751/E1/1/1111	627		0 PLT250	FAUX		VRAI	360	ACIDE	1	460		8
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	G752/E1/1/1111	632		0 PLT250	FAUX		VRAI	360	ACIDE	1	460		8
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	G753/E1/1/1111	495		0 PLT250	FAUX		VRAI	360	ACIDE	1	460		8
UEM	C212	SAUMURE EPUISEE RECOLTEUR PRINCIPAL	G754/E1/1/1111	704		0 PLT250	FAUX		VRAI	360	ACIDE	1	460		8
UEM	C410	LESSIVE SORTIE CELLULE N	C212/E1/1/1111	681		1 PLT250	VRAI		1800 FAUX			0			5
UEM	C410	LESSIVE SORTIE CELLULE N	C415/E1/1/1111	144		1 PLT250	VRAI		1800 FAUX			0			3
UEM	C410	LESSIVE SORTIE CELLULE N	E530/E1/1/1111	570		1 PLT500	VRAI		2400 VRAI	360		0			4
UEM	C410	LESSIVE SORTIE CELLULE N	G804/E1/1/1111	258		1 PLT100	VRAI		1200 FAUX		EAU PROPRE	2	600		7
UEM	C410	LESSIVE SORTIE CELLULE N	G806/E1/1/1111	260		1 PLT100	VRAI		1200 FAUX		EAU PROPRE	2	600		7
UEM	C410	LESSIVE SORTIE CELLULE N	G824/E1/1/1111	734		1 PLT250	VRAI		1800 FAUX		EAU PROPRE	2	600		7
UEM	C415	LESSIVE SORTIE COLLECTEUR PRINCIPAL	G826/E1/1/1111	735		1 PLT250	VRAI		1800 FAUX		EAU PROPRE	2	600		7
UEM	C420	LESSIVE SORTIE EVAPORATEUR BERTRAMS	G844/E1/1/1111	262		1 PLT100	VRAI		1200 FAUX		EAU PROPRE	2	600		7
UEM	C420	LESSIVE SORTIE EVAPORATEUR BERTRAMS	G846/E1/1/1111	264		1 PLT100	VRAI		1200 FAUX		EAU PROPRE	2	600		7
UEM	C425	LESSIVE SORTIE MAGASIN 7	G864/E1/1/1111	266		1 PLT100	VRAI		1200 FAUX		EAU PROPRE	2	600		7
EAUX	D207	EAU TRG RESEAU 7	G866/E1/1/1111	269		1 PLT100	VRAI		1200 FAUX		EAU PROPRE	2	600		7
UEHG	E030	SAUMURE ENTREE SALLE	G884/E1/1/1111	252		1 PLT100	VRAI		1200 FAUX		EAU PROPRE	2	600		7
UEHG	E425	SORTIE FILTRE E0820	G886/E1/1/1111	254		1 PLT100	VRAI		1200 FAUX		EAU PROPRE	2	600		7
UEHG	E426	SORTIE FILTRE E0821	G888/E1/1/1111	256		1 PLT100	VRAI		1200 FAUX		EAU PROPRE	2	600		7
UEHG	E427	SORTIE FILTRE E0822	V870/E1/1/1111	368		1 PLT500	VRAI		2400 FAUX			0			11
UEHG	E430	EFFLUENTS SORTIE DEMERCURISATION	V870/E2/1/1111	508		1 PEN100	VRAI		1200 FAUX			0			11
UEHG	E530	LESSIVE RS 8	C003/E1/1/1111	120		2 PLT100	FAUX		FAUX		EAU PROPRE	2	360		3
UEHG	E615	SAUMURE NAVETTE SCRUBBER	C110/E1/1/1111	202		2 PLT100	VRAI		1200 FAUX			0			3
UPHCL	G330	HCL SORTIE C.A.	C110/E3/1/1111	203		2 PLT100	VRAI		1200 FAUX		SAUMURE	4	360		3
UPHCL	G410	HCL SORTIE RESINES	C210/E12/1/1111	127		2 PLT250	VRAI		1800 FAUX			0			5
UPHCL	G510	HCL RS 28% N°1	C210/E22/1/1111	129		2 PLT250	VRAI		1800 FAUX			0			5
UPHCL	G520	HCL RS 28% N°2	C210/E32/1/1111	131		2 PLT250	VRAI		1800 FAUX			0			5
UPHCL	G530	HCL RS 28% N°4	C210/E42/1/1111	133		2 PLT250	VRAI		1800 FAUX			0			5
UPHCL	G751	HCl STOCK 34% RS 1	C210/E52/1/1111	135		2 PLT250	VRAI		1800 FAUX			0			5
UPHCL	G752	HCl STOCK 34% RS 2	C210/E62/1/1111	137		2 PLT250	VRAI		1800 FAUX			0			5
UPHCL	G753	HCl STOCK 34% RS 3	C410/E11/1/1111	563		2 PLT250	VRAI		1800 FAUX			0			5
UPHCL	G754	HCl STOCK 34% RS 4	C410/E21/1/1111	564		2 PLT250	VRAI		1800 FAUX			0			5
GN	G804	GN 6 : ALIMENTATION APRES REACTIFS	C410/E31/1/1111	565		2 PLT250	VRAI		1800 FAUX			0			5
GN	G806	GN 6 : EAU DU BALLON	C410/E41/1/1111	566		2 PLT250	VRAI		1800 FAUX			0			5
GN	G824	GN 9 : ALIMENTATION APRES REACTIFS	C410/E51/1/1111	567		2 PLT250	VRAI		1800 FAUX			0			5
GN	G826	GN 9 : EAU DU BALLON	C410/E61/1/1111	568		2 PLT250	VRAI		1800 FAUX			0			5
GN	G844	GN 69 : ALIMENTATION APRES REACTIFS	C420/E1/1/0100	145		2 PLT250	VRAI		1800 VRAI	360		0			3
GN	G846	GN 69 : EAU DU BALLON	C420/E1/1/1011	740		2 PLT250	VRAI		1800 FAUX			0			3
GN	G864	GN 71 : ALIMENTATION APRES REACTIFS	D207/E2/1/1111	513		2 PLT100	FAUX		FAUX		EAU CONTAMINEE	5	460		6
GN	G866	GN 71 : EAU DU BALLON	E030/E1/1/1111	48		2 PLT500	VRAI		2400 FAUX			0			4
GN	G884	GN 72 : ALIMENTATION APRES REACTIFS	E425/E1/1/1111	55		2 PLT100	FAUX		FAUX		MERCURE	6	360		4
GN	G886	GN 72 : EAU DU BALLON	E426/E1/1/1111	56		2 PLT100	FAUX		FAUX		MERCURE	6	360		4
GN	G888	GN 72 : ECLATEUR	E427/E1/1/1111	57		2 PLT100	FAUX		FAUX		MERCURE	6	360		4
ENVIRONNEMENT	N305	EXUTOIRE DE L'EGOUT 3	E430/E1/1/1111	58		2 PLT100	FAUX		FAUX		MERCURE	6	360		4
VC/DCE	V870	PIED COLONNE X381/X281	E615/E1/1/1111	510		2 PLT500	VRAI		2400 FAUX			0			4
VC/DCE	V870	PIED COLONNE X381/X281	N305/E1/1/1111	15092		2 PLT250	FAUX		FAUX		MERCURE	6	600		12

Fabrication	Nom du point	Libellé	Nom	Code	Ugence	Flacon	Refroidir	Refroidissement (sec)	Densité	Tps densité (sec)	Matrice	Pompe	Tps Pomp (sec)	Couloir Sortie	
PCM	A030	HYPO PF. MOYEN DES MAG. N° 6 ET 7	A030/E2/2/1111	28		0 PLT500	FAUX		VRAI		360		0		5
UEM	C425	LESSIVE SORTIE MAGASIN 7	C425/E1/2/1111	173		0 PLT250	VRAI		1800 VRAI		360		0		3
UPHCL	G330	HCL SORTIE C.A.	G330/E1/2/1111	318		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G410	HCL SORTIE RESINES	G410/E1/2/1111	518		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G510	HCL RS 28% N°1	G510/E1/2/1111	15183		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G530	HCL RS 28% N°4	G530/E1/2/1111	517		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G751	HCI STOCK 34% RS 1	G751/E1/2/1111	514		0 PLT250	FAUX		VRAI		360 ACIDE		1	460	8
UPHCL	G752	HCI STOCK 34% RS 2	G752/E1/2/1111	628		0 PLT250	FAUX		VRAI		360 ACIDE		1	460	8
UPHCL	G753	HCI STOCK 34% RS 3	G753/E1/2/1111	705		0 PLT250	FAUX		VRAI		360 ACIDE		1	460	8
UPHCL	G754	HCI STOCK 34% RS 4	G754/E1/2/1111	706		0 PLT250	FAUX		VRAI		360 ACIDE		1	460	8
UEM	C115	SAUMURE ENTREE CELLULES	C115/E1/2/1111	149		1 PLT250	VRAI		1800 FAUX				0		3
UEM	C212	SAUMURE EPUISEE RECOLTEUR PRINCIPAL	C212/E1/2/1111	164		1 PLT250	VRAI		1800 FAUX				0		5
UEM	C415	LESSIVE SORTIE COLLECTEUR PRINCIPAL	C415/E1/2/1111	172		1 PLT250	VRAI		1800 FAUX				0		3
UEHG	E530	LESSIVE RS 8	E530/E1/2/1111	574		1 PLT250	VRAI		1800 VRAI		360		0		4
INTEROX	H620	E7: H2O2 SORTIE EPURATION	H620/E1/2/0100	13		1 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
VC.DCE	V170	EAU DE TREMPE X82/1	V170/E1/2/1111	376		1 PLT250	FAUX		FAUX		EAU CONTAMINEE		5	360	11
VC.DCE	V171	EAU DE TREMPE X82/2	V171/E1/2/1111	377		1 PLT250	FAUX		FAUX		EAU CONTAMINEE		5	360	11
VC.DCE	V172	EAU DE TREMPE X82/3	V172/E1/2/1111	378		1 PLT250	FAUX		FAUX		EAU CONTAMINEE		5	360	11
PCM	A225	CACL2 NAVETTE N° 1	A225/E1/2/1111	29		2 PLT500	FAUX		VRAI		360 EAU CONTAMINEE		5	360	2
PCM	A230	CACL2 NAVETTE N 2	A230/E1/2/1111	30		2 PLT500	FAUX		VRAI		360 EAU CONTAMINEE		5	360	2
PCM	A232	CACL2 NAVETTE N 3	A232/E1/2/1111	31		2 PLT500	FAUX		VRAI		360 EAU CONTAMINEE		5	360	2
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E12/2/1111	153		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E22/2/1111	155		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E32/2/1111	157		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E42/2/1111	159		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E52/2/1111	161		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E62/2/1111	163		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E11/2/1111	557		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E21/2/1111	558		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E31/2/1111	559		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E41/2/1111	560		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E51/2/1111	561		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E61/2/1111	562		2 PLT250	VRAI		1800 FAUX				0		5
UEHG	E425	SORTIE FILTRE E0820	E425/E1/2/1111	68		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E426	SORTIE FILTRE E0821	E426/E1/2/1111	69		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E427	SORTIE FILTRE E0822	E427/E1/2/1111	70		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E430	EFFLUENTS SORTIE DEMERCURISATION	E430/E1/2/1111	71		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E525	LESSIVE RS 6	E525/E1/2/1111	585		2 PLT250	VRAI		1800 FAUX				0		4
UEHG	E525	LESSIVE RS 6	E525/E2/2/1000	586		2 PLT100	VRAI		1200 FAUX				0		4
INTEROX	H720	H17: H2O2	H720/E35/2/1111	15307		2 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H720	H17: H2O2	H720/E50/2/1111	751		2 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H720	H17: H2O2	H720/E60/2/1111	14		2 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H720	H17: H2O2	H720/E70/2/1111	753		2 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H770	H27: H2O2	H770/E35/2/1111	15308		2 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H770	H27: H2O2	H770/E50/2/1111	752		2 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H770	H27: H2O2	H770/E60/2/1111	15		2 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H770	H27: H2O2	H770/E70/2/1111	754		2 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
ENVIRONNEMENT	N305	EXUTOIRE DE L'EGOUT 3	N305/E1/2/1111	15093		2 PLT250	FAUX		FAUX		MERCURE		6	600	12
SALINE	S110	SAUMURE VIERGE ARRIVEE HAM	S110/E1/2/0010	749		2 PLT500	FAUX		VRAI		360 SAUMURE		4	460	2
SALINE	S110	SAUMURE VIERGE ARRIVEE HAM	S110/E1/2/1101	528		2 PLT500	FAUX		VRAI		360 SAUMURE		4	460	2
SALINE	S130	SAUMURE EPURANTE CARBONATEE RS121	S130/E1/2/1010	271		2 PLT500	FAUX		VRAI		360		0		2
SALINE	S165	SAUMURE EPUREE SORTIE RS413	S165/E1/2/1111	529		2 PLT500	FAUX		VRAI		360 SAUMURE		4	460	2
SALINE	S165	SAUMURE EPUREE SORTIE RS413	S165/E2/2/1111	530		2 PLT500	FAUX		FAUX		SAUMURE		4	460	2
SALINE	S265	CONDENSATS SORTIE PREMIER EFFET E005	S265/E1/2/1111	41		2 PLT250	VRAI		1800 FAUX				0		2
SALINE	S295	EAUX CONDENSEES SORTIE 007	S295/E1/2/1111	42		2 PLT250	FAUX		FAUX		EAU CONTAMINEE		5	360	2
SALINE	S315	EAUX MERES SALINE SORTIE EPAISSISSEURS	S315/E1/2/1111	43		2 PLT500	FAUX		VRAI		360 SAUMURE		4	460	2

Fabrication	Nom du point	Libelle	Nom	Code	Ugence	Flacon	Refroidir	Refroidissement (sec)	Densité	Tps densité (sec)	Matrice	Pompe	Tps Pomp (sec)	Couloir Sortie	
PCM	A030	HYPO PF, MOYEN DES MAG. N° 6 ET 7	A030/E2/3/1111	32		0 PLT500	FAUX		VRAI		360		0		5
UEM	C115	SAUMURE ENTREE CELLULES	C115/E1/3/1111	177		0 PLT250	VRAI		1800	FAUX			0		3
UEM	C115	SAUMURE ENTREE CELLULES	C115/E3_A/3/1111	178		0 PLT100	VRAI		1200	FAUX		SAUMURE	4	600	5
UEM	C115	SAUMURE ENTREE CELLULES	C115/E3_B/3/1111	700		0 PLT100	VRAI		1200	FAUX		SAUMURE	4	600	5
UEM	C425	LESSIVE SORTIE MAGASIN 7	C425/E1/3/1111	199		0 PLT250	VRAI		1800	VRAI	360		0		3
UPHCL	G330	HCL SORTIE C.A.	G330/E1/3/1111	327		0 PLT100	FAUX			FAUX		ACIDE	1	460	8
UPHCL	G410	HCL SORTIE RESINES	G410/E1/3/1111	449		0 PLT100	FAUX			FAUX		ACIDE	1	460	8
UPHCL	G510	HCL RS 28% N°1	G510/E1/3/1111	15184		0 PLT100	FAUX			FAUX		ACIDE	1	460	8
UPHCL	G520	HCL RS 28% N°2	G520/E1/3/1111	15188		0 PLT100	FAUX			FAUX		ACIDE	1	460	8
UPHCL	G530	HCL RS 28% N°4	G530/E1/3/1111	331		0 PLT100	FAUX			FAUX		ACIDE	1	460	8
UPHCL	G751	HCI STOCK 34% RS 1	G751/E1/3/0100	629		0 PLT250	FAUX			VRAI	360	ACIDE	1	460	8
UPHCL	G751	HCI STOCK 34% RS 1	G751/E1/3/1011	746		0 PLT250	FAUX			VRAI	360	ACIDE	1	460	8
UPHCL	G752	HCI STOCK 34% RS 2	G752/E1/3/0100	448		0 PLT250	FAUX			VRAI	360	ACIDE	1	460	8
UPHCL	G752	HCI STOCK 34% RS 2	G752/E1/3/1011	744		0 PLT250	FAUX			VRAI	360	ACIDE	1	460	8
UPHCL	G753	HCI STOCK 34% RS 3	G753/E1/3/0100	707		0 PLT250	FAUX			VRAI	360	ACIDE	1	460	8
UPHCL	G753	HCI STOCK 34% RS 3	G753/E1/3/1011	747		0 PLT250	FAUX			VRAI	360	ACIDE	1	460	8
UPHCL	G754	HCI STOCK 34% RS 4	G754/E1/3/0100	708		0 PLT250	FAUX			VRAI	360	ACIDE	1	460	8
UPHCL	G754	HCI STOCK 34% RS 4	G754/E1/3/1011	748		0 PLT250	FAUX			VRAI	360	ACIDE	1	460	8
VC/DCE	V870	PIED COLONNE X381/X281	ASUPPV870/E1/3/1111	395		1 PLT500	VRAI		2400	FAUX			0		11
UEM	C212	SAUMURE EPUISEE RECOLTEUR PRINCIPAL	C212/E1/3/1111	682		1 PLT250	VRAI		1800	FAUX			0		5
UEM	C415	LESSIVE SORTIE COLLECTEUR PRINCIPAL	C415/E1/3/1111	197		1 PLT250	VRAI		1800	FAUX			0		3
UEHG	E530	LESSIVE RS 8	E530/E1/3/1111	534		1 PLT250	VRAI		1800	VRAI	360		0		4
VC/DCE	V870	PIED COLONNE X381/X281	V870/E1/3/1111	445		1 PLT500	VRAI		2400	FAUX		EAU CONTAMINEE	5	360	11
VC/DCE	V870	PIED COLONNE X381/X281	V870/E2/3/1111	444		1 PEN100	VRAI		1200	FAUX			0		11
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E12/3/1111	180		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E22/3/1111	182		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E32/3/1111	184		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E42/3/1111	186		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E52/3/1111	188		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E62/3/1111	190		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E11/3/1111	551		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E21/3/1111	552		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E31/3/1111	553		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E41/3/1111	554		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E51/3/1111	555		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E61/3/1111	556		2 PLT250	VRAI		1800	FAUX			0		5
UEM	C420	LESSIVE SORTIE EVAPORATEUR BERTRAMS	C420/E1/3/1111	198		2 PLT250	VRAI		1800	FAUX			0		5
EAUX	D005	ORNEAU ENTREE DECANTEUR	D005/E1/3/1111	396		2 PLT1000	FAUX			FAUX		MERCURE	6	400	6
EAUX	D005	ORNEAU ENTREE DECANTEUR	D005/E3/3/1111	759		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	400	6
EAUX	D010	ORNEAU SORTIE DECANTEUR No3	D010/E2/3/1111	398		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	360	6
EAUX	D015	ORNEAU SORTIE DECANTEUR No4	D015/E2/3/1111	400		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	360	6
EAUX	D025	SORTIE FILTRE ORNEAU	D025/E2/3/1111	660		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	600	6
EAUX	D028	SORTIE FILTRE PUITS	D028/E2/3/1111	661		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	600	6
EAUX	D120	EAU DEMINE DE DISTRIBUTION	D120/E1/3/1111	286		2 PLT500	FAUX			FAUX		EAU PROPRE	2	600	6
EAUX	D201	EAU TRG RESEAU 1	D201/E2/3/1111	337		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	460	6
EAUX	D203	EAU TRG RESEAU 3	D203/E2/3/1111	676		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	460	6
EAUX	D204	EAU TRG RESEAU 4	D204/E2/3/1111	665		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	460	6
EAUX	D205	EAU TRG RESEAU 5	D205/E2/3/1111	678		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	460	6
EAUX	D206	EAU TRG RESEAU 6	D206/E2/3/1111	535		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	460	6
EAUX	D207	EAU TRG RESEAU 7	D207/E2/3/1111	603		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	460	6
EAUX	D208	EAU TRG RESEAU 8	D208/E2/3/1111	273		2 PLT100	FAUX			FAUX		EAU CONTAMINEE	5	460	6
UEHG	E425	SORTIE FILTRE E0820	E425/E1/3/1111	81		2 PLT100	FAUX			FAUX		MERCURE	6	360	4
UEHG	E426	SORTIE FILTRE E0821	E426/E1/3/1111	82		2 PLT100	FAUX			FAUX		MERCURE	6	360	4
UEHG	E427	SORTIE FILTRE E0822	E427/E1/3/1111	83		2 PLT100	FAUX			FAUX		MERCURE	6	360	4
UEHG	E430	EFFLUENTS SORTIE DEMERCURISATION	E430/E1/3/1111	84		2 PLT100	FAUX			FAUX		MERCURE	6	360	4
UEHG	E510	LESSIVE SORTIE SALLE	E510/E1/3/1111	85		2 PLT250	VRAI		1800	FAUX			0		4
ENVIRONNEMENT	N305	EXUTOIRE DE L'EGOUT 3	N305/E1/3/1111	15094		2 PLT250	FAUX			FAUX		MERCURE	6	600	12
ENVIRONNEMENT	N542	ORNEAU AMONT	N542/E1/3/1111	15105		2 PLT1000	FAUX			FAUX		MERCURE	6	600	12
VC/DCE	V165	EAU CONDENSEE X31/1	V165/E1/3/1111	411		2 PLT100	VRAI		1200	FAUX		EAU PROPRE	2	360	11
VC/DCE	V165	EAU CONDENSEE X31/1	V165/E2/3/1111	598		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V165	EAU CONDENSEE X31/1	V165/E3/3/1111	666		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V166	EAU CONDENSEE X31/2	V166/E1/3/1111	413		2 PLT100	VRAI		1200	FAUX		EAU PROPRE	2	360	11
VC/DCE	V166	EAU CONDENSEE X31/2	V166/E2/3/1111	600		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V166	EAU CONDENSEE X31/2	V166/E3/3/1111	667		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V167	EAU CONDENSEE X31/3	V167/E1/3/1111	415		2 PLT100	VRAI		1200	FAUX		EAU PROPRE	2	360	11
VC/DCE	V167	EAU CONDENSEE X31/3	V167/E2/3/1111	602		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V167	EAU CONDENSEE X31/3	V167/E3/3/1111	668		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V320	VAPEUR 10 ATA	V320/E1/3/1111	402		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V435	EAU CONDENSEE P134/1	V435/E1/3/1111	407		2 PLT100	VRAI		1200	FAUX		EAU PROPRE	2	360	11
VC/DCE	V435	EAU CONDENSEE P134/1	V435/E2/3/1111	408		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V440	EAU CONDENSEE P134/2	V440/E1/3/1111	409		2 PLT100	VRAI		1200	FAUX		EAU PROPRE	2	360	11
VC/DCE	V440	EAU CONDENSEE P134/2	V440/E2/3/1111	410		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V445	EAU CONDENSEE P62	V445/E1/3/1111	405		2 PLT100	VRAI		1200	FAUX		EAU PROPRE	2	360	11
VC/DCE	V533	EAU G18	V533/E1/3/1111	403		2 PLT100	VRAI		1200	FAUX		EAU PROPRE	2	360	11
VC/DCE	V533	EAU G18	V533/E2/3/1111	404		2 PLT100	VRAI		1200	FAUX			0		11
VC/DCE	V915	EAU CONDENSEE N31	V915/E1/3/1111	427		2 PLT100	VRAI		1200	FAUX		EAU PROPRE	2	360	11
VC/DCE	V915	EAU CONDENSEE N31	V915/E2/3/1111	428		2 PLT250	VRAI		1800	FAUX			0		11
VC/DCE	V915	EAU CONDENSEE N31	V915/E3/3/1111	674		2 PLT100	VRAI		1200	FAUX			0		11

Fabrication	Nom du point	Libellé	Nom	Code	Ugence	Flacon	Refroidir	Refroidissement (sec)	Densité	Tps densité (sec)	Matrice	Pompe	Tps Pomp (sec)	Couloir Sortie	
PCM	A030	HYPO PF. MOYEN DES MAG. N° 6 ET 7	A030:E2.4.1111	33		0 PLT500	FAUX		VRAI	360			0		2
UEM	C425	LESSIVE SORTIE MAGASIN 7	C425:E1:4:0111	741		0 PLT500	VRAI	2400	VRAI	360			0		3
UEM	C425	LESSIVE SORTIE MAGASIN 7	C425:E1:4:1000	225		0 PLT500	VRAI	2400	VRAI	360			0		3
UPHCL	G330	HCL SORTIE C.A.	G330:E1.4.1111	338		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G410	HCL SORTIE RESINES	G410:E1.4.1111	339		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G510	HCL RS 28% N°1	G510:E.1:4:1111	15185		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G520	HCL RS 28% N°2	G520:E.1.4.1111	15025		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G530	HCL RS 28% N°4	G530:E.1.4.1111	342		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G751	HCl STOCK 34% RS 1	G751:E.1:4.1111	461		0 PLT250	FAUX		VRAI	360	ACIDE		1	460	8
UPHCL	G752	HCl STOCK 34% RS 2	G752:E.1:4.1111	630		0 PLT250	FAUX		VRAI	360	ACIDE		1	460	8
UPHCL	G753	HCl STOCK 34% RS 3	G753:E.1:4.1111	709		0 PLT250	FAUX		VRAI	360	ACIDE		1	460	8
UPHCL	G754	HCl STOCK 34% RS 4	G754:E.1:4.1111	710		0 PLT250	FAUX		VRAI	360	ACIDE		1	460	8
UEM	C115	SAUMURE ENTREE CELLULES	C115:E1:4.1111	205		1 PLT250	VRAI	1800	FAUX				0		3
UEM	C212	SAUMURE EPUISEE RECOLTEUR PRINCIPAL	C212:E1:4.1111	683		1 PLT250	VRAI	1800	FAUX				0		5
UEM	C415	LESSIVE SORTIE COLLECTEUR PRINCIPAL	C415:E1.4.1111	470		1 PLT250	VRAI	1800	FAUX				0		3
UEHG	E530	LESSIVE RS 8	E530:E1:4:0100	577		1 PLT500	VRAI	2400	VRAI	360			0		4
UEHG	E530	LESSIVE RS 8	E530:E1:4:1011	745		1 PLT500	VRAI	2400	VRAI	360			0		4
INTEROX	H953	H2O2 35% RS 693	H953:E.1.4:0001	15037		1 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H954	H2O2 50% RS 694	H954:E.1:4.0001	15039		1 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H992	H2O2 BURGO 50% RS 792	H992:E.1:4:0001	15304		1 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H993	H2O2 60% RS 793	H993:E.1:4.0001	15073		1 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H994	H2O2 60% RS 794	H994:E.1:4.0001	15078		1 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H995	H2O2 70% RS 795	H995:E.1:4.0001	15081		1 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
INTEROX	H996	H2O2 70% RS 796	H996:E.1:4.0001	15083		1 PLT250	FAUX		FAUX		EAU OXYGENEE		3	360	6
VC/DCE	V170	EAU DE TREMPE X82/1	V170:E1.4.1111	417		1 PLT250	FAUX		FAUX		EAU CONTAMINEE		5	360	11
VC/DCE	V171	EAU DE TREMPE X82/2	V171:E1:4:1111	418		1 PLT250	FAUX		FAUX		EAU CONTAMINEE		5	360	11
VC/DCE	V172	EAU DE TREMPE X82/3	V172:E1:4:1111	419		1 PLT250	FAUX		FAUX		EAU CONTAMINEE		5	360	11
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210:E12:4:1111	208		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210:E22:4:1111	210		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210:E32:4:1111	212		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210:E42:4:1111	214		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210:E52:4:1111	216		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210:E62:4:1111	218		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410:E11:4:1111	545		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410:E21:4:1111	546		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410:E31:4:1111	547		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410:E41:4:1111	548		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410:E51:4:1111	549		2 PLT250	VRAI	1800	FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410:E61:4:1111	550		2 PLT250	VRAI	1800	FAUX				0		5
UEHG	E030	SAUMURE ENTREE SALLE	E030:E1:4:0111	737		2 PLT500	VRAI	2400	FAUX		SAUMURE		4	600	4
UEHG	E030	SAUMURE ENTREE SALLE	E030:E1:4:1000	93		2 PLT500	VRAI	2400	VRAI	360	SAUMURE		4	600	4
UEHG	E030	SAUMURE ENTREE SALLE	E030:E3:4:1000	94		2 PLT250	VRAI	1800	FAUX				0		4
UEHG	E035	SAUMURE SORTIE SALLE	E035:E1:4:0111	738		2 PLT500	VRAI	2400	FAUX		SAUMURE		4	600	4
UEHG	E035	SAUMURE SORTIE SALLE	E035:E1:4:1000	95		2 PLT500	VRAI	2400	VRAI	360	SAUMURE		4	600	4
UEHG	E035	SAUMURE SORTIE SALLE	E035:E2:4:1000	96		2 PLT250	VRAI	1800	FAUX				0		4
UEHG	E130	SORTIE FILTRE SCHEIBLER	E130:E1.4.1111	98		2 PLT250	FAUX		FAUX		SAUMURE		4	360	4
UEHG	E425	SORTIE FILTRE E0820	E425:E1:4:1111	100		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E426	SORTIE FILTRE E0821	E426:E1:4:1111	101		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E427	SORTIE FILTRE E0822	E427:E1:4:1111	102		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E430	EFFLUENTS SORTIE DEMERCURISATION	E430:E1:4:1111	103		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E525	LESSIVE RS 6	E525:E1:4:1111	576		2 PLT250	VRAI	1800	FAUX				0		4
UPHCL	G110	H2O BALLON UPHCI	G110:E1:4:1111	347		2 PLT100	FAUX		FAUX		EAU PROPRE		2	360	8
ENVIRONNEMENT	N305	EXUTOIRE DE L'EGOUT 3	N305:E1:4:1111	15095		2 PLT250	FAUX		FAUX		MERCURE		6	600	12

Fabrication	Nom du point	Libellé	Nom	Code	Ugence	Flacon	Refroidir	Refroidissement (sec)	Densité	Tps densité (sec)	Matrice	Pompe	Tps Pomp (sec)	Couloir Sortie	
PCM	A030	HYPO PF. MOYEN DES MAG. N° 6 ET 7	A030/E2/5/1111	34		0 PLT500	FAUX		VRAI		360		0		5
UEM	C115	SAUMURE ENTREE CELLULES	C115/E1/5/1111	229		0 PLT250	VRAI		1800 FAUX				0		3
UEM	C115	SAUMURE ENTREE CELLULES	C115/E3_A/5/1111	230		0 PLT100	VRAI		1200 FAUX		SAUMURE		4	600	5
UEM	C115	SAUMURE ENTREE CELLULES	C115/E3_B/5/1111	702		0 PLT100	VRAI		1200 FAUX		SAUMURE		4	600	5
UEM	C425	LESSIVE SORTIE MAGASIN 7	C425/E1/5/1111	251		0 PLT250	VRAI		1800 VRAI		360		0		3
UPHCL	G330	HCL SORTIE C.A.	G330/E1/5/1111	349		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G410	HCL SORTIE RESINES	G410/E1/5/1111	476		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G510	HCL RS 28% N°1	G510/E1/5/1111	15024		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G520	HCL RS 28% N°2	G520/E1/5/1111	15189		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G530	HCL RS 28% N°4	G530/E1/5/1111	353		0 PLT100	FAUX		FAUX		ACIDE		1	460	8
UPHCL	G751	HCl STOCK 34% RS 1	G751/E1/5/1111	475		0 PLT250	FAUX		VRAI		360 ACIDE		1	460	8
UPHCL	G752	HCl STOCK 34% RS 2	G752/E1/5/1111	631		0 PLT250	FAUX		VRAI		360 ACIDE		1	460	8
UPHCL	G753	HCl STOCK 34% RS 3	G753/E1/5/1111	711		0 PLT250	FAUX		VRAI		360 ACIDE		1	460	8
UPHCL	G754	HCl STOCK 34% RS 4	G754/E1/5/1111	712		0 PLT250	FAUX		VRAI		360 ACIDE		1	460	8
UEM	C212	SAUMURE EPUISEE RECOLTEUR PRINCIPAL	C212/E1/5/1111	684		1 PLT250	VRAI		1800 FAUX				0		5
UEM	C415	LESSIVE SORTIE COLLECTEUR PRINCIPAL	C415/E1/5/1111	249		1 PLT250	VRAI		1800 FAUX				0		3
UEHG	E530	LESSIVE RS 8	E530/E1/5/1111	119		1 PLT250	VRAI		1800 VRAI		360		0		4
VC/DCE	V870	PIED COLONNE X381/X281	V870/E1/5/1111	438		1 PLT500	VRAI		2400 FAUX				0		11
VC/DCE	V870	PIED COLONNE X381/X281	V870/E2/5/1111	483		1 PEN100	VRAI		1200 FAUX				0		11
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E12/5/1111	232		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E22/5/1111	234		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E32/5/1111	236		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E42/5/1111	238		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E52/5/1111	240		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C210	SAUMURE EPUISEE SORTIE CELLULE N	C210/E62/5/1111	242		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E11/5/1111	539		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E21/5/1111	540		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E31/5/1111	541		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E41/5/1111	542		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E51/5/1111	543		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C410	LESSIVE SORTIE CELLULE N	C410/E61/5/1111	544		2 PLT250	VRAI		1800 FAUX				0		5
UEM	C420	LESSIVE SORTIE EVAPORATEUR BERTRAMS	C420/E1/5/1111	250		2 PLT250	VRAI		1800 FAUX				0		5
EAUX	D201	EAU TRG RESEAU 1	D201/E2/5/1111	359		2 PLT100	FAUX		FAUX		EAU CONTAMINEE		5	460	6
EAUX	D206	EAU TRG RESEAU 6	D206/E2/5/1111	581		2 PLT100	FAUX		FAUX		EAU CONTAMINEE		5	460	6
EAUX	D207	EAU TRG RESEAU 7	D207/E2/5/1111	610		2 PLT100	FAUX		FAUX		EAU CONTAMINEE		5	460	6
UEHG	E425	SORTIE FILTRE E0820	E425/E1/5/1111	111		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E426	SORTIE FILTRE E0821	E426/E1/5/1111	112		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E427	SORTIE FILTRE E0822	E427/E1/5/1111	113		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
UEHG	E430	EFFLUENTS SORTIE DEMERCURISATION	E430/E1/5/1111	114		2 PLT100	FAUX		FAUX		MERCURE		6	360	4
ENVIRONNEMENT	N305	EXUTOIRE DE L'EGOUT 3	N305/E1/5/1111	15096		2 PLT250	FAUX		FAUX		MERCURE		6	600	12
SALINE	S165	SAUMURE EPUREE SORTIE RS413	S165/E2/5/1111	46		2 PLT500	FAUX		FAUX		SAUMURE		4	460	2

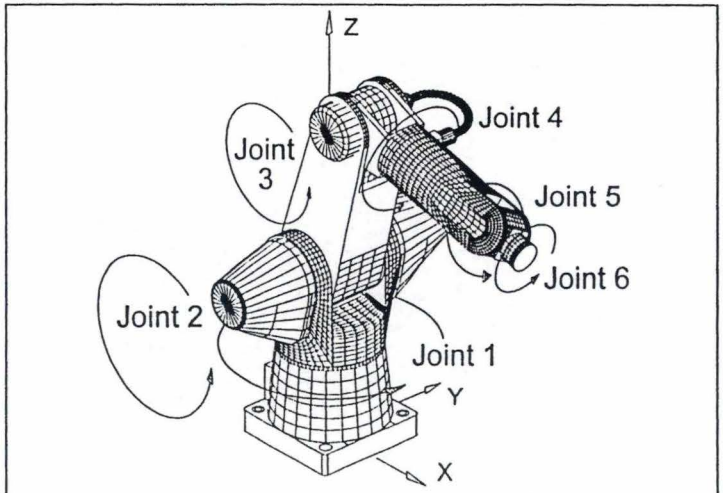
CHAPTER 1

Introduction

An A465 robot consists of a robot arm and a robot controller running RAPL-II operating software. It is designed to automate tasks such as machine loading, parts handling, product testing, spray painting, polishing and deburring.

The A465 can be operated with a serial teach pendant, a computer running Robcomm communication software, or a video terminal (VDT) with a serial I/O.

The arm is articulated with six joints or axes, providing it six degrees of freedom. This allows the arm to move a gripper or other tool to cartesian spatial coordinates and orientation defined by X, Y, Z, Yaw, Pitch, and Roll.



The A465 arm has six axes of motion (joints): 1 (waist), 2 (shoulder), 3 (elbow), 4 (wrist rotate), 5 (wrist pitch), and 6 (tool roll).

Component Parts

Before installing the robot, locate and check that you have received all the components.

- If you did not order options, the robot system is packaged in two containers.
- If you ordered options, the robot system is packaged in three or more containers.
- Options may include: gripper, teach pendant, arm user's guide, controller user's guide, software user's guide(s) and diskette(s), cable extensions, spares, etc.

NO OPTIONS	
Container	**Contents**
labeled "DESCRIPTION: ARM"	• Arm • Hex key
labeled "DESCRIPTION: CTRL"	• Controller • Feedback cable • Motor power cable • Fuse kit with AC power cable • Override plug

OPTIONS	
Container	**Contents**
labeled "DESCRIPTION: ARM"	• Arm • Hex key
labeled "DESCRIPTION: CTRL"	• Controller
labeled "OPTIONS"	• Feedback cable • Motor power cable • Fuse kit with AC power cable • Override plug • Options

Optional Equipment

The following options are available from CRS or an authorized CRS distributor.

End Effectors

- Servo Gripper and Adapter
- Pneumatic Gripper and Adapter

Hardware and Equipment

- Teach Pendant
- End Effector Custom Cabling
- Linear Track Axis and Mounting
- Remote Operator Panel
- Force Sensor
- Pressurized Suit

Software

- Robcomm Host Computer Interface Software
- Visualizer Work Cell Simulation Software
- Scheduler Laboratory Automation Software

CHAPTER 2

Specifications

This chapter describes the A465 arm's:

- Range of Motion, Dimensions, and Weight
- Reach
- Torque Ratings
- Joint Speed and Acceleration Rates
- Payload
- Resolution
- Brakes
- Proximity Sensors
- Gripper

Range of Motion, Dimensions, and Weight

The arm's range of motion depends on the dimensions of each arm section (base, links, tool flange) and the extent of travel of each joint. These measurements determine the shape of the arm's workspace. See the following figures and tables.

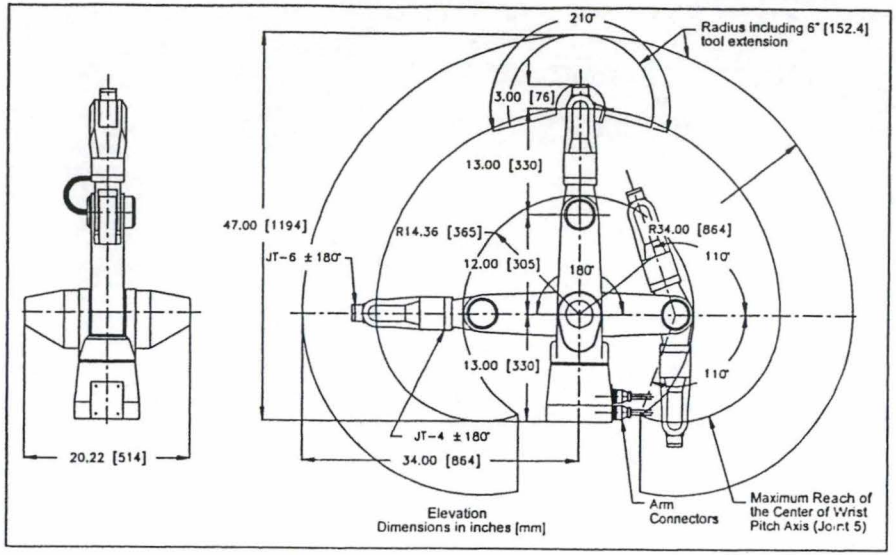
Note: A dimension involving a joint is measured to the joint axis.

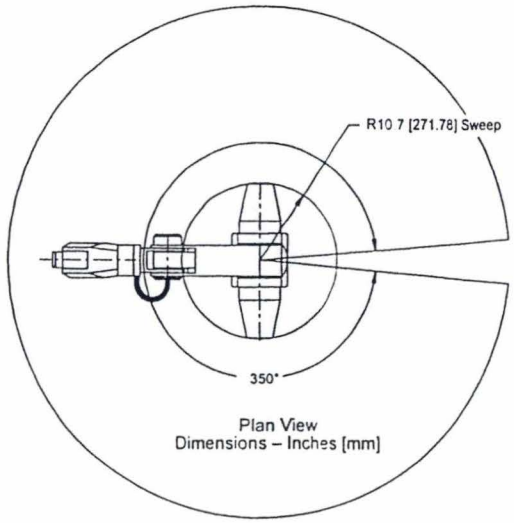
The arm weighs approximately 67 lb. [32 kg].

Weight (approx.)	lb.	kg
Arm	67	32

Joint	Axis	Range of Motion
Waist	1	+175° - 175°
Shoulder	2	+90° - 90°
Elbow	3	+110° - 110°
Wrist rotate	4	+180° - 180°
Wrist pitch	5	+105° - 105°
Tool roll	6	+180° - 180°

Section	Dimension	
	inch	mm
Base mounting surface to shoulder	13	30
Shoulder to elbow	12	305
Elbow to wrist pivot (joint 5)	13	330
Wrist pivot to tool flange surface	3	76





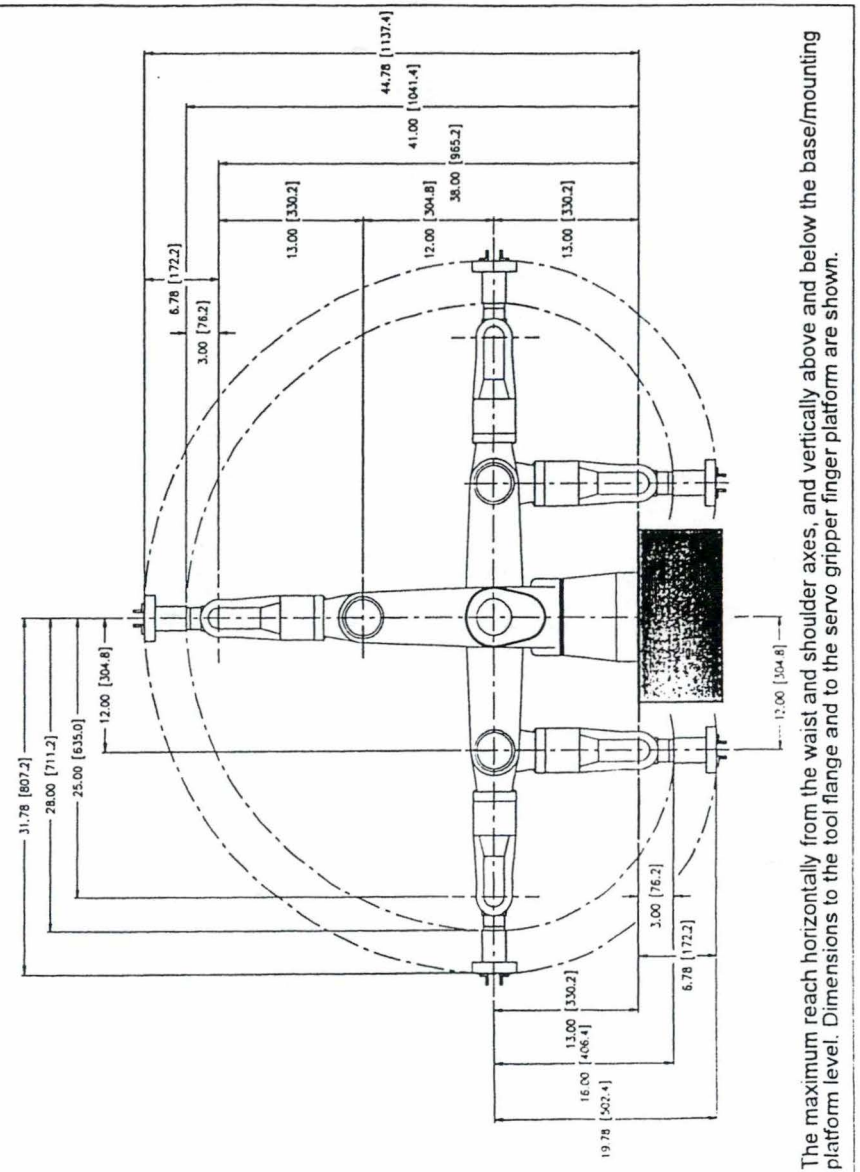
The range of motion of the joints and the dimensions of the sections of the robot arm determine the workspace. Units are inches [mm].

Reach

The maximum reach of the arm is calculated horizontally outward from the shoulder joint (axis 2) and vertically upward from the bottom of the base.

The arm can reach points below the level of the bottom of the base.

A465 Maximum Arm Reach			
		inch	mm
Horizontal outward from the shoulder axis in the X-Y plane	to tool flange	28.00	711.2
	to finger platform of servo gripper	31.78	807.2
Vertical upwards along the Z axis	to tool flange	41.00	1041.0
	to finger platform of servo gripper	44.78	1137.0
Vertical downwards below the base level	to tool flange	3.00	76.2
	to finger platform of servo gripper	6.78	172.2



The maximum reach horizontally from the waist and shoulder axes, and vertically above and below the base/mounting platform level. Dimensions to the tool flange and to the servo gripper finger platform are shown.

Torque Ratings

This table shows the torque rating for each arm joint.

Continuous Torque Rating			
Joint	Axis	Torque	
		in-lb.	N-m
Waist	1	350.0	39.50
Shoulder	2	350.0	39.50
Elbow	3	350.0	39.50
Wrist rotate	4	61.0	6.89
Wrist pitch	5	60.4	6.82
Tool roll	6	22.1	2.50

Joint Speeds and Acceleration Rates

The standard pick and place cycle is 1.2 seconds.

These following tables show the speed and acceleration rates for each arm joint.

Maximum Speeds	
Motion	m/s
Compounded joint interpolated motions	4.35
Linear and path motions	1.02

Joint Speeds At 100% Program Speed (RAPL 2.60)					
Joint	Axis	pulse/ms	Gear Reduction	Maximum Speed	
				rad/s	deg/s
Waist	1	50.0	100:1	3.14	180
Shoulder	2	50.0	100:1	3.14	180
Elbow	3	50.0	100:1	3.14	180
Wrist rotate	4	24.0	101:1	2.99	171
Wrist pitch	5	24.0	100:1	3.02	173
Tool roll	6	24.0	101:1	2.99	171

Default Acceleration Rates				
Joint	Axis	pulse/s²	rad/s²	deg/s²
Waist	1	2000	12.6	720
Shoulder	2	2000	12.6	720
Elbow	3	2000	12.6	720
Wrist rotate	4	2000	24.9	1430
Wrist pitch	5	2000	25.1	1430
Tool roll	6	2000	24.9	1430

Payload

Payload is the amount of mass (weight) carried by the arm and/or the amount of force the arm can exert on an object. This includes the end effect and any load that it carries.

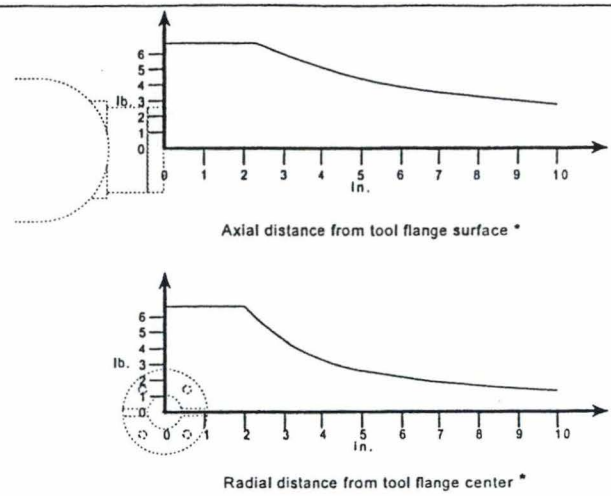
The maximum and nominal payloads are determined for speed and acceleration that maintain rated precision. For rated precision, the arm travels at a reduced speed to carry the maximum payload, and the arm must carry a reduced payload to travel at maximum speed.

The maximum payload depends on the distance between the center of the tool flange surface and the center of gravity of the payload as shown in the payload derating tables and curves.

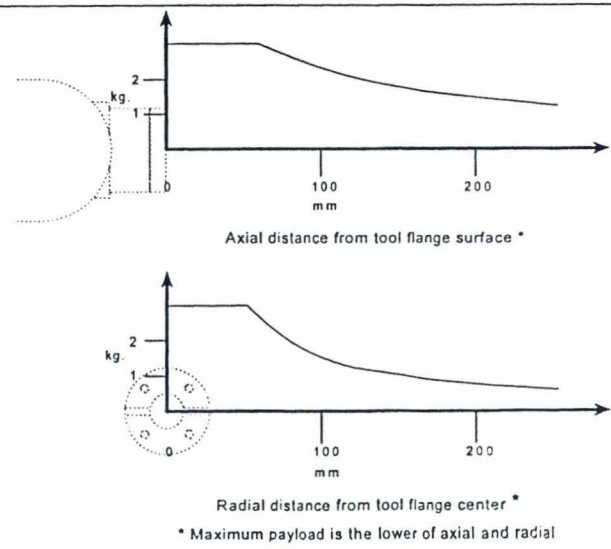
Payload	Speed	Mass
Maximum	80% speed or acceleration	3.0 kilograms
Nominal	100% speed or acceleration	2.0 kilograms

Axial Distance From The Tool Flange Surface			
Distance		Mass	
inch	mm	lb.	kg
2.0	51	6.61	3.00
2.3	58	6.61	3.00
4.0	102	5.00	2.27
6.0	152	3.89	1.76
8.0	203	3.18	1.44
10.0	254	2.69	1.22

Radial Distance From The Tool Flange Center			
Distance		Mass	
inch	mm	lb.	kg
2.0	51	6.61	3.00
4.0	102	3.25	1.47
6.0	152	2.17	0.98
8.0	203	1.63	0.74
10.0	254	1.30	0.59



Imperial: Payload decreases with the distance to its center of gravity.



* Maximum payload is the lower of axial and radial

Metric: Payload decreases with the distance to its center of gravity.

Resolution

Resolution is the smallest increment of motion or distance that can be detected or controlled. Resolution depends on the distance between the center of the tool flange surface and the center of gravity of the payload as shown in the resolution derating tables and curves.

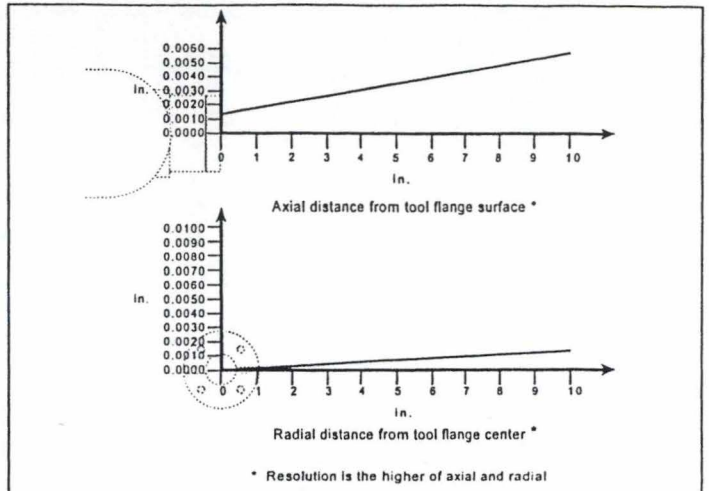
Axial Distance From The Tool Flange Surface			
Distance		Resolution	
inch	mm	inch	mm
0.0	0.0	0.0013	0.0330
2.0	51.0	0.0022	0.0559
4.0	102.0	0.0031	0.0787
6.0	152.0	0.0039	0.0991
8.0	203.0	0.0048	0.1219
10.0	254.0	0.0057	0.1448

Radial Distance From The Tool Flange Center			
Distance		Resolution	
inch	mm	inch	mm
0.0	0.0	0.0000	0.0000
2.0	51.0	0.0003	0.0076
4.0	102.0	0.0005	0.0127
6.0	152.0	0.0008	0.0203
8.0	203.0	0.0010	0.0254
10.0	254.0	0.0013	0.0330

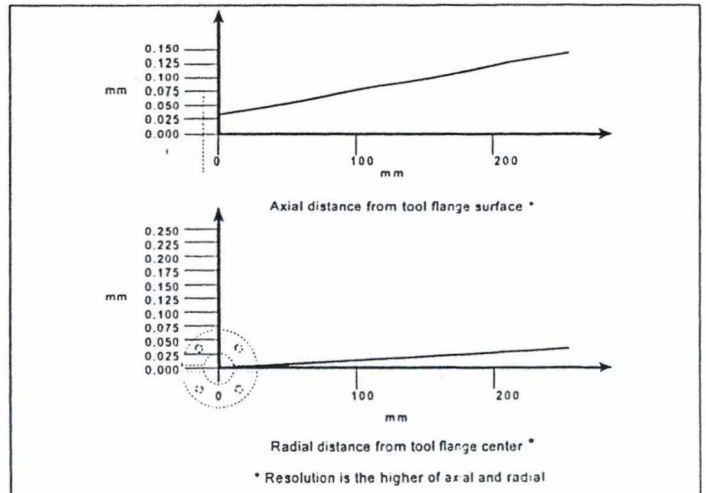
Repeatability

Repeatability is the ability to repeat the same motion or achieve the same points of location when presented with the same control signals. It can also be defined as the cycle-to-cycle error when trying to perform a specific task.

Repeatability	± 0.002 inch	± 0.05 mm



Imperial: Resolution decreases with the distance to the payload's center of gravity.



Metric: Resolution decreases with the distance to the payload's center of gravity.

Brakes

Fail-safe brakes prevent the robot from moving under the influence of gravity or inertia when power is removed. Each brake consists of a spring-loaded clamp on a rotating disk. A 35 Volt DC signal energizes a magnetic solenoid which unloads the clamp.

Brakes are installed on joints 2 and 3.

Do not move the joints by hand when the brakes are engaged.
This may damage some components.

Proximity Sensors

Each joint in the A465 arm contains a proximity sensor (in earlier models, a mechanical home switch) which is used when homing the arm. Each joint also contains a cam or end-of-travel pin location which the proximity sensor uses to find its transition point.

Proximity sensors are not used to limit the positive or negative travel of any joint under normal operation. This is done by software limits.

Homing

When the robot is homed with the internal home command, the controller moves individual joints in sequence until, for each joint, the transition point is found. The resolution of a proximity sensor by itself is not accurate enough for homing a joint. The controller uses the next zero pulse from the joint's encoder to accurately determine the arm position.

Customized Homing

Because of the limitations presented by obstacles in a workcell (peripherals, equipment, walls, etc.), the regular homing sequence may have to be revised and a customized homing program used instead. Proximity sensors do not have to be used as part of a customized homing sequence.

Transition Points

The location of the transition points are listed in the following table:

Axis	Joint	Transition Location
1	Waist	+175° (positive end of travel limit)
2	Shoulder	+10° (vertical is 0 degrees)
3	Elbow	– 60°
4	Wrist rotate	+180° (positive end of travel limit)
5	Wrist pitch	+25°
6	Tool roll	+180° (positive end of travel limit)

An arm in the @CALRDY position (links posed vertical upwards from the base) has all joints at 0 degrees.

Gripper

An A465 robot requires the attachment of a gripper (or other end effector) to perform its intended tasks. Custom-designed grippers and other end effectors are available from CRS.

Standard Grippers

Servo Gripper (SGRIP)
An electric, servo-controlled, parallel motion, two-finger gripper, capable of measuring objects between its fingers. Finger travel is 0–2 in. [50.8 mm] with programmable position and force.

Servo Gripper with Microplate Fingers (SEC-B0-645)
An electric servo-controlled, parallel motion, two-finger gripper, with fingers specially designed for handling laboratory microplate containers. Finger travel is 0–2 in. [50.8 mm] with programmable position and force.

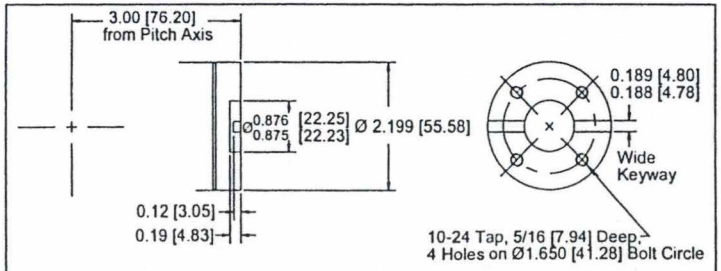
Pneumatic Gripper (PGR112/3)
A two-jaw, double acting, air gripper with 3 in. [76.2 mm] long, angular motion, re-toolable fingers. Fingers can be machined to meet specific needs. Travel is 0–10 degrees per finger.

Grippers (other than CRS)

Grippers must be designed and constructed, so that:

- Power failure does not cause the release of the load, or result in a hazardous condition.
- Static and dynamic forces exerted by the load and the gripper together are within the load capacity and dynamic response of the robot.

To attach a standard gripper to the A465 tool flange, you need a tool flange adapter, refer to page 69 for detailed instructions.



Mounting dimensions of the tool flange for the gripper in inches [mm].

```
/*-------------------------------------------------------------------------*/
/* Prototype de programme logique GNU Prolog                               */
/* pour le problème de l'ordonnancement des tâches d'un atelier robotisé   */
/*                                                                         */
/* File Name       : robot.pl                                              */
/* Title           : optimisation du travail d'un atelier robotique        */
/* Author          : M. Marmoro                                            */
/* Date            : Juillet 2000                                          */
/* Version         : GNU PROLOG 1.1.2                                      */
/*                   ressources avec capacité de 1                         */
/* Remarque        : ce programme s'inspire de P. Van Hentenryck et D. Diaz */
/*                   (voir références bibliographiques)                    */
/*-------------------------------------------------------------------------*/

/* constraint definitions */

smallereqc(X,Y,C):-
        X #=< Y+C.

greatereqc(X,Y,C):-
        X #>= Y+C.

q:-     statistics(runtime,_),
        robot(Ld,End,Lv), statistics(runtime,[_,Y]),
        write(Ld), nl,
        write(End), nl,
        write(Lv), nl,
        write('time : '), write(Y), nl.

/*************************************************************************/
/* prédicat principal                                                    */
/*************************************************************************/

robot(K,Ende,Lv):-
        data_structure(K,Ende,Disj,Lv),
        minimize_maximum(K,Disj,Lv).

/*************************************************************************/
/* création de la structure de données                                   */
/*************************************************************************/

data_structure(K,Ende,Disj,Lv):-
        jobs(L),
        make_vars(L,K),
        member([stop,_,Ende],K),
        contraintes(K),
        ressources(R),
        make_disj(R,K,[],Disj1,Lv),
        reverse(Disj1,Disj).

/*************************************************************************/
/* création des variables de domaine = le moment de début de chaque tâche */
/*************************************************************************/

make_vars([],[]).
make_vars([H|T],[[H,D,A]|R]):-
        duration(H,D),
        fd_domain(A,0,28800),
        make_vars(T,R).
```

```
/*****************************************************************************/
/* contraintes                                                               */
/*****************************************************************************/

contraintes(K):-
        precedence(M),
        make_precedence(M,K),
        synchro(M1),
        make_synchro(M1,K).

/*****************************************************************************/
/* contraintes de précédence                                                 */
/*****************************************************************************/

make_precedence([],_).
make_precedence([[A,B]|R],L):-
        member([A,Ad,Aa],L),
        member([B,_Bd,Ba],L),
        greatereqc(Ba,Aa,Ad),                  % Ba #>= Aa+Ad,
        make_precedence(R,L).

/*****************************************************************************/
/* contraintes de synchronisation                                            */
/*****************************************************************************/

make_synchro([],_).
make_synchro([[A,B]|R],L):-
        member([A,_Ad,Aa],L),
        member([B,_Bd,Ba],L),
        Ba #= Aa,
        make_synchro(R,L).

/*****************************************************************************/
/* création d'une liste des contraintes de disjonctions                      */
/*****************************************************************************/

make_disj([],_R,D,D,[]).
make_disj([[_H,R]|T],K,Din,Dout,[Atleast|Lv]):-
        el_list(R,K,R1),
        check_start(R1,Atleast),
        make_disj1(R1,Din,D1),
        make_disj(T,K,D1,Dout,Lv).

/*****************************************************************************/
/* contraintes redondantes                                                   */
/*****************************************************************************/

check_start([],0).
check_start([[B,D]|Ls],Atleast):-
        check_start1(Ls,B,D,Atleast).

check_start1([],Bmin,SDi,Atleast):-
        Atleast #= Bmin + SDi.
check_start1([[B,D]|Ls],Bi,SDi,Ci):-
        SDnew is D + SDi,
        Bmin #= min(B,Bi),
        check_start1(Ls,Bmin,SDnew,Ci).
```

```
/*****************************************************************************/
/* recherche des durées de chaque tâches                                     */
/*****************************************************************************/

el_list([],_,[]).
el_list([H|T],L,[[A,D]|S]):-
        member([H,D,A],L),
        el_list(T,L,S).

/*****************************************************************************/
/* création de la sous-liste de contraintes disjonctives                     */
/*****************************************************************************/

make_disj1([],D,D).
make_disj1([H|T],Din,Dout):-
        make_disj2(H,T,Din,D1),
        make_disj1(T,D1,Dout).

make_disj2(_H,[],D,D).
make_disj2([A,B],[[C,D]|S],Din,Dout):-
        make_disj2([A,B],S,[[A,B,C,D]|Din],Dout).

/*****************************************************************************/
/* minimise la valeur maximale d'une liste                                   */
/*****************************************************************************/

minimize_maximum(K,Disj,[H|T]):-
        maximum(T,H,Ende),
        fd_minimize(choice(Disj,K),Ende).

/*****************************************************************************/
/* recherche la valeur maximale d'une liste                                  */
/*****************************************************************************/

maximum([],B,B).
maximum([H|T],Bi,Ci):-
        Bnew #= max(H,Bi),
        maximum(T,Bnew,Ci).

/*****************************************************************************/
/* choix des variables et énumuration                                        */
/*****************************************************************************/

choice(Disj,K):-
        disjunct(Disj),
        label(K).

/*****************************************************************************/
/* énumuration des valeurs pour les variables                                */
/*****************************************************************************/

label([]).
label([[_A,_Ad,Aa]|R]):-
        fd_labeling(Aa),
        label(R).
```

```
/*************************************************************************/
/* disjonctions de tâches                                                */
/*************************************************************************/

disjunct([]).
disjunct([[A,B,C,D]|R]):-
        disj(A,B,C,D),
        disjunct(R).

disj(Aa,Ad,Ba,_Bd):-
        greatereqc(Ba,Aa,Ad).                % Ba #>= Aa+Ad.

disj(Aa,_Ad,Ba,Bd):-
        greatereqc(Aa,Ba,Bd).                % Aa #>= Ba+Bd.


/*************************************************************************/
/* DATA                                                                  */
/*************************************************************************/
/*************************************************************************/

/*************************************************************************/
/* liste des tâches                                                      */
/*************************************************************************/

jobs([start,
      e026t03,e026t042,e026t07,e026t08,e026t09,e026t102,e026t11,
      e123t01,e123t023,e123t11,
      e124t01,e124t021,e123t03,e123t041,e123t05,e123t064,e123t09,e123t101,e123t11,
      e698t01,e698t021,e698t03,e698t041,e698t05,e698t064,e698t09,e698t101,e698t11,
      e146t01,e146t022,e146t03,e146t041,e146t07,e146t08,e146t09,e146t101,e146t11,
      e308t03,e308t041,e308t05,e308t061,e308t09,e308t101,e308t11,
      e309t03,e309t041,e309t05,e309t061,e309t09,e309t101,e309t11,
      e312t03,e312t041,e312t05,e312t061,e312t09,e312t101,e312t11,
      e627t03,e627t041,e627t05,e627t061,e627t07,e627t08,e627t09,e627t101,e627t11,
      stop]).

/*************************************************************************/
/* durée par tâches                                                      */
/*************************************************************************/

duration(start,0).
duration(e026t03,30).
duration(e026t042,70).
duration(e026t07,30).
duration(e026t08,390).
duration(e026t09,30).
duration(e026t102,70).
duration(e026t11,35).
duration(e123t01,45).
duration(e123t023,2445).
duration(e123t11,30).
duration(e124t01,45).
duration(e124t021,1245).
duration(e123t03,45).
duration(e123t041,85).
duration(e123t05,30).
duration(e123t064,630).
duration(e123t09,30).
duration(e123t101,70).
```

```
duration(e123t11,35).
duration(e698t01,45).
duration(e698t021,1245).
duration(e698t03,45).
duration(e698t041,85).
duration(e698t05,30).
duration(e698t064,630).
duration(e698t09,30).
duration(e698t101,70).
duration(e698t11,35).
duration(e146t01,45).
duration(e146t022,1845).
duration(e146t03,45).
duration(e146t041,85).
duration(e146t07,30).
duration(e146t08,390).
duration(e146t09,30).
duration(e146t101,70).
duration(e146t11,35).
duration(e308t03,30).
duration(e308t041,70).
duration(e308t05,30).
duration(e308t061,490).
duration(e308t09,30).
duration(e308t101,70).
duration(e308t11,35).
duration(e309t03,30).
duration(e309t041,70).
duration(e309t05,30).
duration(e309t061,490).
duration(e309t09,30).
duration(e309t101,70).
duration(e309t11,35).
duration(e312t03,30).
duration(e312t041,70).
duration(e312t05,30).
duration(e312t061,490).
duration(e312t09,30).
duration(e312t101,70).
duration(e312t11,35).
duration(e627t03,30).
duration(e627t041,70).
duration(e627t05,30).
duration(e627t061,490).
duration(e627t07,30).
duration(e627t08,390).
duration(e627t09,30).
duration(e627t101,70).
duration(e627t11,35).
duration(stop,0).
```

```
/*************************************************************************/
/* précédence des tâches                                                 */
/*************************************************************************/

precedence([[start,e026t03],[start,e026t042],[e026t042,e026t07],
            [e026t042,e026t08],[e026t08,e026t09],[e026t08,e026t102],
            [e026t102,e026t11],[e026t11,stop],[start,e123t01],[start,e123t023],
            [e123t023,e123t11],[e123t11,stop],[start,e124t01],[start,e124t021],
            [e124t021,e123t03],[e124t021,e123t041],[e123t041,e123t05],
            [e123t041,e123t064],[e123t064,e123t09],[e123t064,e123t101],
            [e123t101,e123t11],[e123t11,stop],[start,e698t01],[start,e698t021],
            [e698t021,e698t03],[e698t021,e698t041],[e698t041,e698t05],
            [e698t041,e698t064],[e698t064,e698t09],[e698t064,e698t101],
            [e698t101,e698t11],[e698t11,stop],[start,e146t01],[start,e146t022],
            [e146t022,e146t03],[e146t022,e146t041],[e146t041,e146t07],
            [e146t041,e146t08],[e146t08,e146t09],[e146t08,e146t101],
            [e146t101,e146t11],[e146t11,stop],[start,e308t03],[start,e308t041],
            [e308t041,e308t05],[e308t041,e308t061],[e308t061,e308t09],
            [e308t061,e308t101],[e308t101,e308t11],[e308t11,stop],
            [start,e309t03],[start,e309t041],[e309t041,e309t05],[e309t041,e309t061],
            [e309t061,e309t09],[e309t061,e309t101],[e309t101,e309t11],[e309t11,stop],
            [start,e312t03],[start,e312t041],[e312t041,e312t05],[e312t041,e312t061],
            [e312t061,e312t09],[e312t061,e312t101],[e312t101,e312t11],[e312t11,stop],
            [start,e627t03],[start,e627t041],[e627t041,e627t05],[e627t041,e627t061],
            [e627t041,e627t07],[e627t041,e627t08],[e627t061,e627t09],
            [e627t061,e627t101],[e627t08,e627t09],[e627t08,e627t101],
            [e627t101,e627t11],[e627t11,stop]]).

/*************************************************************************/
/* tâches synchronisées                                                  */
/*************************************************************************/

synchro([e026t03,e026t042],[e026t07,e026t08],[e026t09,e026t102],
        [e123t01,e123t023],
        [e124t01,e124t021],[e123t03,e123t041],[e123t05,e123t064],[e123t09,e123t101],
        [e698t01,e698t021],[e698t03,e698t041],[e698t05,e698t064],[e698t09,e698t101],
        [e146t01,e146t022],[e146t03,e146t041],[e146t07,e146t08],[e146t09,e146t101],
        [e308t03,e308t041],[e308t05,e308t061],[e308t09,e308t101],
        [e309t03,e309t041],[e309t05,e309t061],[e309t09,e309t101],
        [e312t03,e312t041],[e312t05,e312t061],[e312t09,e312t101],
        [e627t03,e627t041],[e627t05,e627t061],[e627t07,e627t08],[e627t09,e627t101]).


/*************************************************************************/
/* partage des ressources                                                */
/*************************************************************************/

ressources([[robot,[e026t03,e026t07,e026t09,e026t11,e123t01,e123t11,e124t01,e123t03,
                    e123t05,e123t09,e123t11,e698t01,e698t03,e698t05,e698t09,e698t101,
                    e698t11,e146t01,e146t03,e146t07,e146t09,e146t11,e308t03,e308t05,
                    e308t09,e308t11,e309t03,e309t05,e309t09,e309t11,e312t03,e312t05,
                    e312t09,e312t11,e627t03,e627t05,e627t07,e627t09,e627t11]],
            [cooler021,[e124t021,e698t021]],
            [cooler022,[e146t022]],
            [cooler023,[e123t023]],
            [decap041,[e123t041,e698t041,e146t041,e308t041,e309t041,e312t041,
                       e627t041,e123t101,e698t101,e146t101,e308t101,e309t101,
                       e312t101,e627t101]],
            [decap042,[e026t042,e026t102]],
            [pomp061,[e308t061,e309t061,e312t061,e627t061]],
            [pomp064,[e123t064,e698t064]],
```

```
            [dens,[e026t08,e146t08,e627t08]],
            [fictif,[e627t061,e627t08]]]).

:- initialization(q).
```

```
/*-------------------------------------------------------------------------*/
/* File Name      : planification.pro                                      */
/* Title          : Prototype de programme logique avec IF/Prolog          */
/*                  pour le problème de l'ordonnancement des tâches d'un   */
/*                  atelier robotisé                                       */
/*                                                                         */
/* Author         : M. Marmoro                                             */
/* Date           : Août 2000                                              */
/* Version        : IF/Prolog 5.1                                          */
/*                                                                         */
/* Remarques      :                                                        */
/*                                                                         */
/* 1) Les contraintes cumulatives portent sur les ressources types:        */
/*         M01 = Robot                                                     */
/*         M021 = Cooler1                                                  */
/*         M022 = Cooler2                                                  */
/*         M023 = Cooler3                                                  */
/*         M041 = Decap1                                                   */
/*         M042 = Decap2                                                   */
/*         M061 = Pomp1                                                    */
/*         M062 = Pomp2                                                    */
/*         M063 = Pomp3                                                    */
/*         M064 = Pomp4                                                    */
/*         M065 = Pomp5                                                    */
/*         M08 = Dens                                                      */
/*                                                                         */
/* 2) nous avons ajouté 2 ressources types fictives pour modéliser la      */
/*    capacité de maximum 6 bouchons dans une station de dévissage /       */
/*    revissage                                                            */
/*         MF1 = simule une ressource type sur le decap 1                  */
/*         MF2 = simule une ressource type sur le décap 2                  */
/*-------------------------------------------------------------------------*/

/***************************************************************************/
/* importe le résolveur de contraintes                                     */
/***************************************************************************/

:- import(const_domain).

/***************************************************************************/
/* Programme logique                                                       */
/***************************************************************************/

program :-
/***************************************************************************/
/* Chaque échantillon a un travail à réaliser: les variables de JOB sont   */
/* les temps au plus tard de réalisation des travaux qu'il faut minimiser  */
/***************************************************************************/

        Jobs = [E026,E123,E124,E698,E146,E308,E309,E627,E312],

/***************************************************************************/
/* Data                                                                    */
/***************************************************************************/
/***************************************************************************/
```

```
/*************************************************************************/
/* ressources types fictives et les variables associées                  */
/*************************************************************************/

          MF1_Tsk = [E124TF01,E698TF01,E146TF01,E308TF01,E309TF01,E312TF01,
                     E627TF01],
          MF1_Dur = [E124DF01,E698DF01,E146DF01,E308DF01,E309DF01,E312DF01,
                     E627DF01],
          MF1_Dur = [     785,     785,     545,     630,     630,     630,
                         1020],
          MF1_Use = [       1,       1,       1,       1,       1,       1,
                            1],

          MF2_Tsk = [E026TF01],
          MF2_Dur = [E026DF01],
          MF2_Dur = [     530],
          MF2_Use = [       1],

/*************************************************************************/
/* ressource type robot et les temps de déplacement associés             */
/*************************************************************************/

          M01_Tsk = [E123T01,E124T01,E698T01,E146T01,E026T03,E124T03,E698T03,
                     E146T03,E308T03,E309T03,E312T03,E627T03,E124T05,E698T05,
                     E308T05,E309T05,E312T05,E627T05,E026T07,E146T07,E627T07,
                     E026T09,E124T09,E698T09,E146T09,E308T09,E309T09,E312T09,
                     E627T09,E026T11,E123T11,E124T11,E698T11,E146T11,E308T11,
                     E309T11,E312T11,E627T11],
          M01_Dur = [E123D01,E124D01,E698D01,E146D01,E026D03,E124D03,E698D03,
                     E146D03,E308D03,E309D03,E312D03,E627D03,E124D05,E698D05,
                     E308D05,E309D05,E312D05,E627D05,E026D07,E146D07,E627D07,
                     E026D09,E124D09,E698D09,E146D09,E308D09,E309D09,E312D09,
                     E627D09,E026D11,E123D11,E124D11,E698D11,E146D11,E308D11,
                     E309D11,E312D11,E627D11],
          M01_Dur = [     45,     45,     45,     45,     30,     45,     45,
                          45,     30,     30,     30,     30,     30,     30,
                          30,     30,     30,     30,     30,     30,     30,
                          30,     30,     30,     30,     30,     30,     30,
                          30,     35,     15,     35,     35,     35,     35,
                          35,     35,     35],
          M01_Use = [      1,      1,      1,      1,      1,      1,      1,
                           1,      1,      1,      1,      1,      1,      1,
                           1,      1,      1,      1,      1,      1,      1,
                           1,      1,      1,      1,      1,      1,      1,
                           1,      1,      1,      1,      1,      1,      1,
                           1,      1,      1],

/*************************************************************************/
/* ressources types "refroidisseur"                                      */
/*************************************************************************/

          E124D021 ?= 1245,
          E698D021 ?= 1245,
          E146D022 ?= 1845,
          E123D023 ?= 2445,

          M021_Tsk = [E124T021,E698T021],
          M021_Dur = [E124O021,E698O021],
          M021_Use = [       1,       1],
```

```
          M022_Tsk = [E146T022],
          M022_Dur = [E146O022],
          M022_Use = [       1],

          M023_Tsk = [E123T023],
          M023_Dur = [E123O023],
          M023_Use = [       1],

/*************************************************************************/
/* ressources types dévissage / revissage                                */
/*************************************************************************/

          M04a_Dur = [E124D041,E698D041,E146D041,E308D041,E309D041,E312D041,
                      E627D041,E124D101,E698D101,E146D101,E308D101,E309D101,
                      E312D101,E627D101],
          M04a_Dur = [      85,      85,      85,      70,      70,      70,
                            70,      70,      70,      70,      70,      70,
                            70,      70],
          M04b_Dur = [E026D042,E026D102],
          M04b_Dur = [      70,      70],

          M041_Tsk = [E124T041,E698T041,E146T041,E308T041,E309T041,E312T041,
                      E627T041,E124T101,E698T101,E146T101,E308T101,E309T101,
                      E312T101,E627T101],
          M041_Dur = [E124D041,E698D041,E146D041,E308D041,E309D041,E312D041,
                      E627D041,E124O101,E698O101,E146O101,E308O101,E309O101,
                      E312O101,E627O101],
          M041_Use = [       1,       1,       1,       1,       1,       1,
                             1,       1,       1,       1,       1,       1,
                             1,       1],

          M042_Tsk = [E026T042,E026T102],
          M042_Dur = [E026D042,E026O102],
          M042_Use = [       1,       1],

/*************************************************************************/
/* ressources types pompages                                             */
/*************************************************************************/

          M061_Tsk = [E308T061,E309T061,E312T061,E627T061],
          M061_Dur = [E308D061,E309D061,E312D061,E627D061],
          M061_Dur = [     490,     490,     490,     490],
          M061_Use = [       1,       1,       1,       1],


          M064_Tsk = [E124T064,E698T064],
          M064_Dur = [E124D064,E698D064],
          M064_Dur = [     630,     630],
          M064_Use = [       1,       1],

/*************************************************************************/
/* ressource type "densimètre"                                           */
/*************************************************************************/

          M08_Tsk = [E026T08,E146T08,E627T08],
          M08_Dur = [E026D08,E146D08,E627D08],
          M08_Dur = [    390,    390,    390],
          M08_Use = [      1,      1,      1],
```

```
/*************************************************************************/
/* liste des variables des débuts de tâche (une par tâche)               */
/*************************************************************************/

          Tasks = [E123T01,E124T01,E698T01,E146T01,E026T03,E124T03,E698T03,
                   E146T03,E308T03,E309T03,E312T03,E627T03,E124T05,E698T05,
                   E308T05,E309T05,E312T05,E627T05,E026T07,E146T07,E627T07,
                   E026T09,E124T09,E698T09,E146T09,E308T09,E309T09,E312T09,
                   E627T09,E026T11,E123T11,E124T11,E698T11,E146T11,E308T11,
                   E309T11,E312T11,E627T11,E124T021,E698T021,E146T022,E123T023,
                   E124T041,E698T041,E146T041,E308T041,E309T041,E312T041,E627T041,
                   E124T101,E698T101,E146T101,E308T101,E309T101,E312T101,E627T101,
                   E026T042,E026T102,E308T061,E309T061,E312T061,E627T061,E124T064,
                   E698T064,E026T08,E146T08,E627T08,E026TF01,E124TF01,E698TF01,
                   E146TF01,E308TF01,E309TF01,E312TF01,E627TF01],

/*************************************************************************/
/* liste des variables des durées de tâche (une par tâche)               */
/*                uniquement pour l'output des résultats                 */
/*************************************************************************/

       Durations = [E123D01,E124D01,E698D01,E146D01,E026D03,E124D03,E698D03,
                    E146D03,E308D03,E309D03,E312D03,E627D03,E124D05,E698D05,
                    E308D05,E309D05,E312D05,E627D05,E026D07,E146D07,E627D07,
                    E026D09,E124D09,E698D09,E146D09,E308D09,E309D09,E312D09,
                    E627D09,E026D11,E123D11,E124D11,E698D11,E146D11,E308D11,
                    E309D11,E312D11,E627D11,E124D021,E698D021,E146D022,E123D023,
                    E124D041,E698D041,E146D041,E308D041,E309D041,E312D041,E627D041,
                    E124D101,E698D101,E146D101,E308D101,E309D101,E312D101,E627D101,
                    E026D042,E026D102,E308D061,E309D061,E312D061,E627D061,E124D064,
                    E698D064,E026D08,E146D08,E627D08,E026DF01,E124DF01,E698DF01,
                    E146DF01,E308DF01,E309DF01,E312DF01,E627DF01],

/*************************************************************************/
/* Déclaration des domaines des variables                                */
/*************************************************************************/

          Tasks in 0..18800,   Jobs in 0..18800,

/*************************************************************************/
/* contraintes de précédence et de synchronisation                       */
/*************************************************************************/

          E026T042 ?=  E026T03,
          E026T08  ?>= E026T03  + E026D03,
          E026T08  ?>= E026T042 + E026D042,
          E026T08  ?=  E026T07,
          E026T102 ?>= E026T07  + E026D07,
          E026T102 ?>= E026T08  + E026D08,
          E026T102 ?=  E026T09,
          E026T11  ?>= E026T09  + E026D09,
          E026T11  ?>= E026T102 + E026D102,
          E026     ?=  E026T11  + E026D11,

          E123T023 ?=  E123T01,
          E123T11  ?>= E123T01  + E123D01,
          E123T11  ?>= E123T023 + E123D023,
          E123     ?=  E123T11  + E123D11,

          E124T021 ?=  E124T01,
```

```
E124T041 ?>= E124T01  + E124D01,
E124T041 ?>= E124T021 + E124D021,
E124T041 ?=  E124T03,
E124T064 ?>= E124T03  + E124D03,
E124T064 ?>= E124T041 + E124D041,
E124T064 ?=  E124T05,
E124T101 ?>= E124T05  + E124D05,
E124T101 ?>= E124T064 + E124D064,
E124T101 ?=  E124T09,
E124T11  ?>= E124T09  + E124D09,
E124T11  ?>= E124T101 + E124D101,
E124     ?=  E124T11  + E124D11,

E698T021 ?=  E698T01,
E698T041 ?>= E698T01  + E698D01,
E698T041 ?>= E698T021 + E698D021,
E698T041 ?=  E698T03,
E698T064 ?>= E698T03  + E698D03,
E698T064 ?>= E698T041 + E698D041,
E698T064 ?=  E698T05,
E698T101 ?>= E698T05  + E698D05,
E698T101 ?>= E698T064 + E698D064,
E698T101 ?=  E698T09,
E698T11  ?>= E698T09  + E698D09,
E698T11  ?>= E698T101 + E698D101,
E698     ?=  E698T11  + E698D11,

E146T022 ?=  E146T01,
E146T041 ?>= E146T01  + E146D01,
E146T041 ?>= E146T022 + E146D022,
E146T041 ?=  E146T03,
E146T08  ?>= E146T03  + E146D03,
E146T08  ?>= E146T041 + E146D041,
E146T08  ?=  E146T07,
E146T101 ?>= E146T07  + E146D07,
E146T101 ?>= E146T08  + E146D08,
E146T101 ?=  E146T09,
E146T11  ?>= E146T09  + E146D09,
E146T11  ?>= E146T101 + E146D101,
E146     ?=  E146T11  + E146D11,

E308T041 ?=  E308T03,
E308T061 ?>= E308T03  + E308D03,
E308T061 ?>= E308T041 + E308D041,
E308T061 ?=  E308T05,
E308T101 ?>= E308T05  + E308D05,
E308T101 ?>= E308T061 + E308D061,
E308T101 ?=  E308T09,
E308T11  ?>= E308T09  + E308D09,
E308T11  ?>= E308T101 + E308D101,
E308     ?=  E308T11  + E308D11,

E309T041 ?=  E309T03,
E309T061 ?>= E309T03  + E309D03,
E309T061 ?>= E309T041 + E309D041,
E309T061 ?=  E309T05,
E309T101 ?>= E309T05  + E309D05,
E309T101 ?>= E309T061 + E309D061,
E309T101 ?=  E309T09,
E309T11  ?>= E309T09  + E309D09,
E309T11  ?>= E309T101 + E309D101,
```

```
          E309     ?=  E309T11  + E309D11,

          E312T041 ?=  E312T03,
          E312T061 ?>= E312T03  + E312D03,
          E312T061 ?>= E312T041 + E312D041,
          E312T061 ?=  E312T05,
          E312T101 ?>= E312T05  + E312D05,
          E312T101 ?>= E312T061 + E312D061,
          E312T101 ?=  E312T09,
          E312T11  ?>= E312T09  + E312D09,
          E312T11  ?>= E312T101 + E312D101,
          E312     ?=  E312T11  + E312D11,

          E627T041 ?=  E627T03,
          E627T061 ?>= E627T03  + E627D03,
          E627T061 ?>= E627T041 + E627D041,
          E627T08  ?>= E627T03  + E627D03,
          E627T08  ?>= E627T041 + E627D041,

          E627T061 ?=  E627T05,
          E627T101  ?>= E627T05  + E627D05,
          E627T101  ?>= E627T061 + E627D061,

          E627T08  ?=  E627T07,
          E627T101 ?>= E627T07  + E627D07,
          E627T101 ?>= E627T08  + E627D08,

          E627T101 ?=  E627T09,
          E627T11  ?>= E627T09  + E627D09,
          E627T11  ?>= E627T101 + E627D101,
          E627     ?=  E627T11  + E627D11,

/*************************************************************************/
/* calcul de la durée réel d'occupation des échantillons dans les        */
/*        ressources types "refroidisseur" et "decap"                    */
/*************************************************************************/

          E123O023 ?= (E123T11 - E123T023),
          E124O021 ?= (E124T041 - E124T021),
          E698O021 ?= (E698T041 - E698T021),
          E146O022 ?= (E146T041 - E146T022),

          E026O102 ?= (E026T11 - E026T102),
          E124O101 ?= (E124T11 - E124T101),
          E698O101 ?= (E698T11 - E698T101),
          E146O101 ?= (E146T11 - E146T101),
          E308O101 ?= (E308T11 - E308T101),
          E309O101 ?= (E309T11 - E309T101),
          E312O101 ?= (E312T11 - E312T101),
          E627O101 ?= (E627T11 - E627T101),

/*************************************************************************/
/* Synchronisation des tâches fictives couvrant la durée du dévissage au */
/*       revissage des échantillons                                      */
/*************************************************************************/

          E026TF01 ?= E026T042,
          E124TF01 ?= E124T041,
          E146TF01 ?= E146T041,
          E698TF01 ?= E698T041,
          E308TF01 ?= E308T041,
```

```
            E309TF01 ?= E309T041,
            E312TF01 ?= E312T041,
            E627TF01 ?= E627T041,

            E026T102 ?= E026TF01 + E026DF01 - E026D102,
            E124T101 ?= E124TF01 + E124DF01 - E124D101,
            E698T101 ?= E698TF01 + E698DF01 - E698D101,
            E146T101 ?= E146TF01 + E146DF01 - E146D101,
            E308T101 ?= E308TF01 + E308DF01 - E308D101,
            E309T101 ?= E309TF01 + E309DF01 - E309D101,
            E312T101 ?= E312TF01 + E312DF01 - E312D101,
            E627T101 ?= E627TF01 + E627DF01 - E627D101,

/*************************************************************************/
/* contraintes cumulative sur les ressources types                       */
/*************************************************************************/

            cumulative(M01_Tsk,M01_Dur,M01_Use,1),
            cumulative(M021_Tsk,M021_Dur,M021_Use,12),
            cumulative(M022_Tsk,M022_Dur,M022_Use,8),
            cumulative(M023_Tsk,M023_Dur,M023_Use,4),
            cumulative(M041_Tsk,M041_Dur,M041_Use,1),
            cumulative(M042_Tsk,M042_Dur,M042_Use,1),
            cumulative(M061_Tsk,M061_Dur,M061_Use,1),
            cumulative(M064_Tsk,M064_Dur,M064_Use,1),
            cumulative(M08_Tsk,M08_Dur,M08_Use,1),

/*************************************************************************/
/* contrainte disjonctive entre tâche de pompage et tâche de densité  */
/*************************************************************************/

            disjunctive([E627T061,E627T08],[E627D061,E627D08]),

            cumulative(MF1_Tsk,MF1_Dur,MF1_Use,6),
            cumulative(MF2_Tsk,MF2_Dur,MF2_Use,6),

/*************************************************************************/
/* énumération des valeurs                                               */
/*************************************************************************/

            label(Tasks),
/*            minimize_maximum(label(Tasks),Jobs), */

/*************************************************************************/
/* sortie des résultats                                                  */
/*************************************************************************/

            tell('resultats.txt'),
            print((Tasks,Durations,Jobs)),told.
```