École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2013 - 2014

Rapport Projet d'option Logistique & Optimisation

Optimal workforce assignment to operations of a paced transfer line

Encadrant
Ameur SOUKHAL
ameur.soukhal@univ-tours.fr

Université François-Rabelais, Tours

Etudiants
Anaïs LINOT
anais.linot@etu.univ-tours.fr
Rémy PRADIGNAC
remy.pradignac@etu.univ-tours.fr

DI5 2013-2014

Version du 19 mai 2014

Table des matières

1 Introduction **6**

2 Présentation du problème **7**
2.1 Heuristiques pour répondre au problème 7
2.1.1 TopLong 7
2.1.2 TopShort 7
2.1.3 Conclusion sur les heuristiques 8

3 Evolutions des résultats **9**
3.1 Modifications 10
3.1.1 Premières modifications 10
3.1.2 Secondes modifications 10
3.1.3 Calcul du chemin critique 11
3.1.4 Modifications améliorant l'heuristique 11

4 Benchmark **16**
4.1 Protocole 16
4.2 Résultats 17

5 Guide d'utilisation **19**
5.1 Menu 19
5.1.1 Propre instance 19
5.1.2 Benchmark 19

6 Conclusion **21**

Table des figures

3.1 Exemple d'instance 9
3.2 Résultat obtenu avec un D fixé à 40 9
3.3 Résultat obtenu avec un D fixé à 25 10
3.4 Résultat obtenu avec un D fixé à 40 10
3.5 Résultat obtenu avec un D fixé à 40 11
3.6 Résultat obtenu avec un D fixé à 25 11
3.7 Exemple d'ordonnacement optimisable 12
3.8 Nouvelle formule de la durée 15

4.1 Début du fichier d'instance 16
4.2 Matrice de prédécesseurs 16
4.3 Première partie des résultats 17
4.4 Seconde partie des résultats 18

5.1 Menu 19
5.2 Choix de l'instance 19
5.3 Choix de l'heuristique 19
5.4 Résultat console *Propre solution* 20
5.5 Gantt de la solution *Propre solution* 20

Liste des tableaux

Introduction

Dans le cadre de notre dernière année d'études à Polytech Tours, nous devions effectuer un projet pour notre option de logistique et optimisation. L'intitulé de notre problématique est :

— Optimal workforce assignment to operations of a paced transfer line

Ce projet avait un enjeu important puisqu'il était destiné pour un industriel. Malgré la confidentialité des données, nous empêchant de travailler sur des jeux "véritables", nous avons pu avancer grâce à notre encadrant qui nous a fourni une explication de construction de jeux de tests.

Présentation du problème

Comme dit précédemment, notre projet avait une problèmatique industrielle. En effet, le but de ce programme est d'optimiser le nombre de travailleurs totals en fonction d'un temps maximum déterminé et d'un nombre de tâches.

2.1 Heuristiques pour répondre au problème

Pour répondre à cette problèmatique, deux heuristiques avaient été créé, à savoir :

— Top Long
— Top Short

Bien que très proches, c'est deux heuristiques nous ont fourni des résultats très différents.

2.1.1 TopLong

La première heuristique que nous avons vu fut "TopLong". Le principe de cette dernière est :

— Mettre à jour la liste des tâches qu'il est possible de faire ;
— Choisir dans cette liste, celle avec la durée la plus **longue** et l'affecter dès que le nombre de travailleurs disponible corresponds aux nombres de travailleurs minimum de la tâches.
— Si à la fin de ce placement, la durée à respecter est dépassée, alors :
  — On calcule le chemin critique de ce résultat
  — On ajoute une tâche à chacune de ces tâches

On réitère cette opération tant que la durée totale dépasse celle imposée. Si l'on dépasse le nombre maximum de travailleurs disponible dans l'instance, alors elle ne possède pas de solutions.

2.1.2 TopShort

La seconde heuristique que nous avons vu fut "TopShort". Le principe de cette dernière est :

— Mettre à jour la liste des tâches qu'il est possible de faire ;
— Choisir dans cette liste, celle avec la durée la plus **courte** et l'affecter dès que le nombre de travailleurs disponible corresponds aux nombres de travailleurs minimum de la tâches.
— Si à la fin de ce placement, la durée à respecter est dépassée, alors :
  — On calcule le chemin critique de ce résultat
  — On ajoute une tâche à chacune de ces tâches

On réitère cette opération tant que la durée totale dépasse celle imposée. Si l'on dépasse le nombre maximum de travailleurs disponible dans l'instance, alors elle ne possède pas de solutions.

2.1.3 Conclusion sur les heuristiques

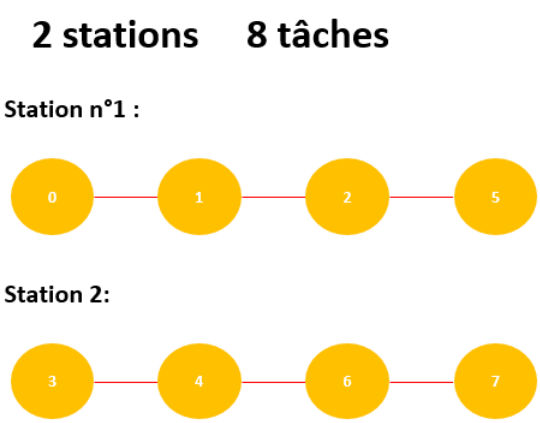
Comme nous pouvons le constater, les différences dans l'implémentation des deux heuristiques sont minimimes, cependant, les résultats possèdent d'importantes différences, comme nous le verrons dans la troisième partie de ce rapport.

Lorsque nous avons récupéré l'existant de ce projet, nous nous sommes aperçus qu'aucune des deux heuristiques n'étaient fonctionnelles.

Nous allons voir les modifications que nous avons du apporter dans la seconde partie de ce rapport.

Evolutions des résultats

Avec TopLong
Nous avons principalement travaillé avec l'instance suivante :



Numéro Tâche	Durée	Min Workers	Max Workers
0	12	4	8
1	12	2	4
2	9	3	6
3	12	2	4
4	28	4	8
5	12	4	8
6	2	1	2
7	6	3	6

FIGURE 3.1 – Exemple d'instance

Voici l'évolution des résultats avec cette instance :
Pour l'instance, on a une date de fin fixée.
Voici les résultats obtenus avec le programme récupéré :

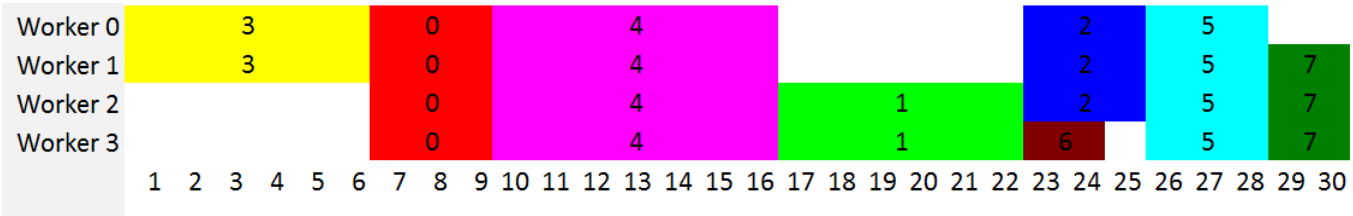


FIGURE 3.2 – Résultat obtenu avec un D fixé à 40

Problèmes avec ce résultat :
— Les tâches ne sont pas parallèlisées. Comme on peut le voir sur l'exemple, la tâche 6 pourrait se faire en même temps que la tâche 1.
— Les tâches sont mal affectées aux workers, et il est impossible d'établir un chemin critique avec ces tâches
— N+ non mis à jour

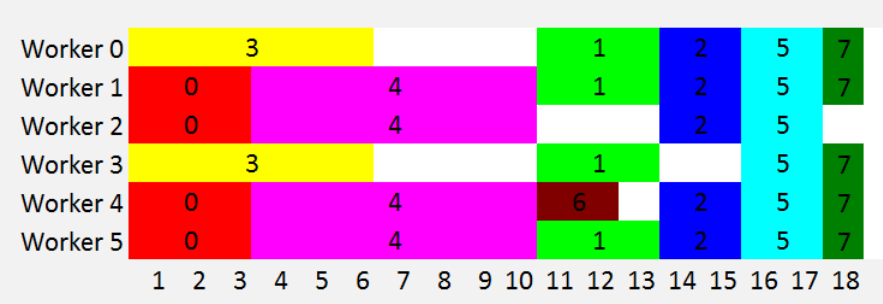


FIGURE 3.3 – Résultat obtenu avec un D fixé à 25

Problèmes avec ces résultats :
— Même problème que ceux présentés sur le résultat ci-dessus
— Non respect des contraintes de précédences (et donc de la date de début de la tâche
— Mauvais calcul du chemin critique

Après la génération de ces résultats, on se rend alors compte que de nombreuses modifications sont à apporter au programme.

3.1 Modifications

3.1.1 Premières modifications

— N+ mis à jour dès qu'une tâche est placée
— Respect des contraintes de précédence
— Tâches affectées aux workers prioritaires
  — On prend en priorité la tâche qui termine au plus ôt
  — On affecte la tâche aux workers dont l'id est le plus petit

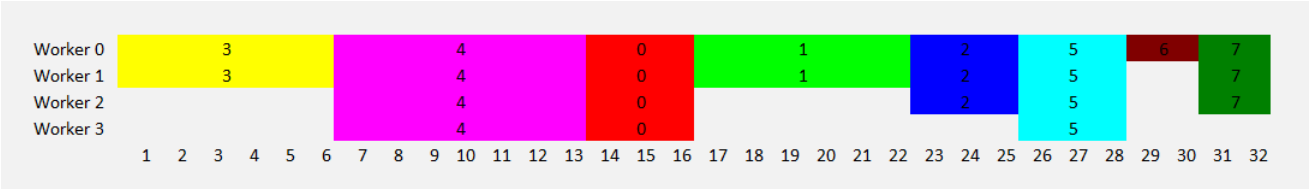


FIGURE 3.4 – Résultat obtenu avec un D fixé à 40

Cependant, cette solution n'est pas encore satisfaisante, car comme on peut le voir, les tâches ne sont pas parallélisées. Ceci est dû au fait que l'algorithme prend en priorité les tâches qui durent le plus longtemps, et ces tâches ne sont pas parallèlisables.

3.1.2 Secondes modifications

On peut donc encore améliorer cette algorithme en mettant N+ à jour lorsque on ne peut plus paralléliser les tâches comprises dans N+. C'est à dire qu'on regarde toutes les tâches comprises dans N+. On place la première tâche (celle retournée par l'heuristique). On regarde si d'autres tâches peuvent être parallélisées. Si oui, on les place, sinon, on regénère N+.

On a maintenant une solution réalisable et optimale avec TopLong. Cependant, la génération de ces résultats ne fonctionne que lorsque le Cmax est inférieur au D fixé. En effet, un des problèmes présentés au

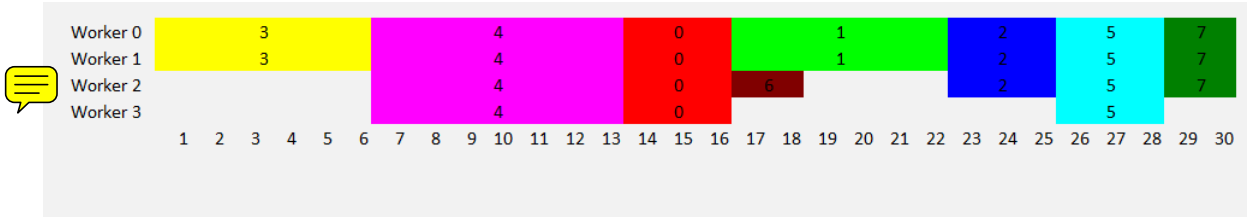


FIGURE 3.5 – Résultat obtenu avec un D fixé à 40

début était le fait que le calcul du chemin critique n'était pas faisable et ne fonctionnait pas. Nous avons maintenant des instances qui permettent plus facilement de pouvoir calculer le chemin critique.

3.1.3 Calcul du chemin critique

Nous avons donc refait le chemin critique
En remettant D à 25, voici les résultats :

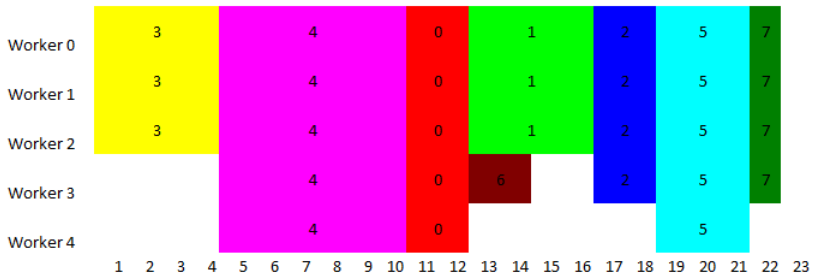


FIGURE 3.6 – Résultat obtenu avec un D fixé à 25

3.1.4 Modifications améliorant l'heuristique

L'heuristique TOP PWGS

Après avoir retravaillé sur ces heuristiques, nous nous sommes rendus compte qu'il y avait moyen d'améliorer cette heuristique. En effet, cette amélioration s'effectue au niveau du calcul du nombre de personnes à rajouter sur l'ordonnancement lors du calcul du chemin critique.

Comme on peut le voir sur les heuristiques présentées ci-dessus, lorsqu'il n'est pas possible d'ordonnancer les tâches avec les données utilisées sans dépasser la date D fixée, alors on augmente l'ensemble des tâches des chemins critiques de 1 worker, et le nombre global de worker de 1 worker également.

Cette organisation n'est pas optimale dans la mesure où on se rend compte que sur certains créneaux, aucun travailleur ne travaille, et que l'on pourrait donc rajouter des travailleurs sur ces tâches, pour ainsi réduire leur durée.

Nous allons un peu plus expliquer cette optimisation en image.

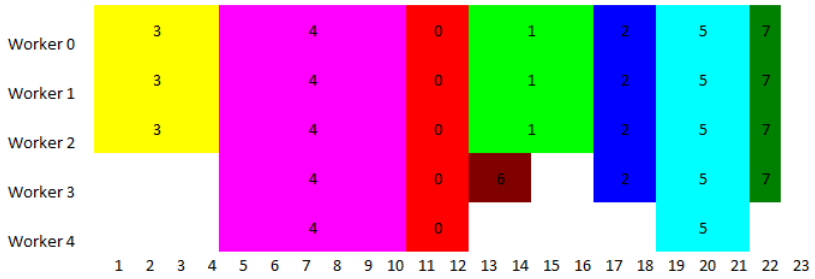


FIGURE 3.7 – Exemple d'ordonnacement optimisable

Comme on peut le voir sur cet ordonnancement, les travailleurs 3 et 4 en travaillent pas sur les temps de 1 à 4. Si on rajoute deux travailleurs sur la tâche 3, en fonction de son nombre de travailleurs maximum, alors on réduirait la durée de la tâche 3, ce qui fait que la tâche 4 pourrait commencer plus tôt et etc. L'ordonnancement pourrait finir plus tôt. Il en est de même sur les temps de 12 à 18 et 22.

La nouvelle heuristique s'appelle TOP PWGS (TOP Parallel Workers Generation Scheme).

Nous allons un peu plus expliquer son principe avec son algorithme.

L'algorithme que nous allons présenter est l'algorithme principal pour exécuter une heuristique. La seule modification qui interviendra avec l'heuristique PWGS se fera au niveau du calcul du chemin critique.

Algorithme 1 Algorithme TopPWGS

```
1: Initialisation nbWorkers et nbTaches
2: Répéter
3:     Initialisation Nplus, NPlusSize, workers
4:     //Nplus contient les tâches que l'on peut placer directement
5:     Répéter
6:         FirstTask = 1
7:         Tant Que (NplusSize! = 0) Faire
8:             //Ici on récupert l'ID de la tâche prioritaire en fonction de l'algo choisi
9:             chosenTaskIndex = selectionFunction(Nplus, NplusSize, workers) ;
10:            //On cherche la date au plus tôt de la tâche, en fonction de ses précédences
11:            DateDispo = Nplus(chosenTaskIndex)− > availabilityTime
12:            nbWorkersToAssign = Nplus[chosenTaskIndex]− > nbWorkersToAssign
13:            Si (FirstTask == 1) Alors
14:                beginningTime = CalculDatePlacementTacheAuPlusTot()
15:            Sinon
16:                //On regarde si la tâche peut-être placée en même temps que la première placée
17:            Fin Si
18:            Si (La tache peut être placée) Alors
19:                //Assignation de la tâche dans l'ordonnancement
20:                //Calcul des dates de disponibilité des autres tâches
21:                NPlusSize − −
22:            Fin Si
23:            FirstTask + +
24:        Fin Tant Que
25:        //Si toutes les tâches n'ont pas été placées
26:        Si (NPlusSize!=0) Alors
27:            //On rajoute des tâches dans N+ en fonction de celles qui ont été placées et on
               enleve de N+ les tâches placées
28:        Sinon
29:            //On rajoute des tâches dans N+ et on enleve de N+ les tâches placées
30:        Fin Si
31:    Jusqu'à GSize > 0
32:    //Calcul du CMax
33:    CMax = Max(workers− > availabilityTime)
34:    //Recherche des tâches du chemin critique
35:    //C'est à ce moment qu'intervient la différence par rapport aux autres heuristiques
36:    behinDuePWGS()
37: Jusqu'à (CMax > D & nbWorkers <= WORKERS_MAX)
```

avec les valeurs suivantes :

— D : valeur connue représentant la date de fin demandée.
— WORKERS_MAX : valeur connue représentant le nombre maximum de travailleurs.
— GSize : valeur connue représentant la taille du Graphe.
— workers : Ensemble des workers que nous avons

Algorithme 2 Algorithme RechercheTachesCheminCritique

```
1: //On sait d'avance les tâches à prendre en compte grace a la variable boolCriticalPath
2: nbTasksCriticalPathWay = 0
3: Pour (i = 0; i<nbWorkers; i++) Faire
4:     Si (workers[i]->boolCriticalPath == 0) Alors
5:         Pour (int k = 0; k < workers[i]->nbTasks; k++) Faire
6:             j = 0
7:             trouve = 0
8:             idTask = workers[i]- > doneTasks[k]- > id
9:             //On regarde si la tache est déjà dans le chemin critique ou non
10:            Tant Que (j < nbTasksCriticalPathWay & trouve == 0) Faire
11:                Si (TasksCriticalPathWay[j] == idTask) Alors
12:                    trouve = 1
13:                    j + +
14:                Fin Si
15:                //Si on n'a pas trouvé la tache dans le chemin critique, alors on l'ajoute au
                   tableau car elle doit prendre un worker
16:                Si (trouve == 0) Alors
17:                    nbTasksCriticalPathWay + +
18:                    TasksCriticalPathWay[nbTasksCriticalPathWay] = idTask
19:                Fin Si
20:            Fin Tant Que
21:        Fin Pour
22:    Fin Si
23: Fin Pour
24: //On enregistre le chemin critique actuel au cas ou on arrive pas à améliorer la solution sans
    rajouter de worker général
25: TasksCriticalPathWayRecop = TasksCriticalPathWay
```

Algorithme 3 Algorithme behinDue_PWGS

```
1: Pour (i = 0; i < nbCriticalPathWay; i++) Faire
2:     Si (le nombre de worker à assigner < nombre de workerMax & le nombre de worker à assigner
       < nbWorkers ) Alors
3:         Tache_i- > nbWorkersToAssign = minInt(Tache_i- > nbWorkersToAssign +
           1, minInt(_nbWorkers + 1, Tache_i- > workersMax))
4:     Fin Si
5: Fin Pour
6: //S'il n'y a pas de solutions malgré les changements, on rend l'ordo initial, on ajoute un worker
   aux taches du chemin critique et on rajoute un worker
```

Finalement, la seule chose qui change dans cet algorithme vis à vis des autres heuristiques, c'est le nombre de personnes affectées à chaque tâche du chemin critique, et le moment où on rajoute un travailleur global.

Recalcul de la durée d'une tâche

La durée sur laquelle était calculé le nombre de travailleurs en fonction du nombre de travailleurs était incorrecte. En effet, avant la durée écrite dans le fichier correspondait à la durée totale de la tâche. Ainsi,

pour savoir combien de temps dure la tâche, on affecte à chaque travailleur la durée de la tâche divisée par le nombre de travailleurs affectés.

Voici la nouvelle prise en compte de la durée d'une tâche.

$$Duree_i = Max\left\{1; Duree_i * \left(1 - \frac{NbWorker_i - WorkerMin_i}{WorkerMax_i}\right)\right\}$$

FIGURE 3.8 – Nouvelle formule de la durée

Avec cette prise en compte, la durée inscrite dans le fichier représente la durée de la tâche pour le nombre de travailleurs au minimum. C'est à dire que chaque travailleur doit effectuer la durée pour que la tâche soit accomplie.

Benchmark

A ce stade du projet, nous possédions donc quatre heuristiques résolvant notre problème. Notre problème était donc de savoir laquelle de ces méthodes étaient la plus performante.

4.1 Protocole

Afin de reconnaitre l'heuristique la plus performante, nous les avons toutes testées sur un nombre important et aléatoire de jeux de données. En effet, afin d'avoir un benchmark le plus objectif possible, nous avons créé une méthode de création d'instance, avant d'évaluer chacunes de nos heuristiques sur toutes les instances générées.

Voici comment nous avons généré nos instances :
- On créé un nouveau fichier ayant pour nom : **instance"nbTasks"-"nbWorkers"-"nbStations"-"TauxDePrédécesseurs"-"numInstance".txt**
- On écrit dans l'entête de fichier le nombre de stations ainsi que le nombre de tâches.
- On définit la durée de la tâche, son affection à la station, son nombre de travailleurs nécessaires minimum et maximum.
- On définit la durée maximum pour l'exécution de l'ensemble des tâches
- On définit les prédécesseurs de chaque tâches.

Avec cette méthode de création d'instance, on obtient pour **"instance60-10-10-3-0.txt** un fichier de ce type :

```
10  60              // 10 stations, 60 tâches
88  0   2   3       // 88 : durée de la tâche, 0 : numéro de la station, 2 : minimum, 3 : maximum de workers
64  8   3   5       // 64 : durée de la tâche, 8 : numéro de la station, 3 : minimum, 5 : maximum de workers
```

FIGURE 4.1 – Début du fichier d'instance

```
983                 // d = 983 ( ne dépassera jamais)
0                   // 0 prédécesseur
0
0
0
0
0
0
1   4               // 1 prédécesseur : la tâche 4
0
2   7   4           // 2 prédecesseurs : les tâches 7 et 4
0                   // 0 prédecesseur etc etc
```

FIGURE 4.2 – Matrice de prédécesseurs

4.2 Résultats

Voici les résultats que nous avons obtenu. Nous avons calculé la moyenne sur les 10 instances d'un même type, lesquelles étaient impossible et si l'heuristique trouvait le meilleur résultat.

Voici un tableau récapitulatif :

n	rmax	s	prob	TopLong			TopShort			PWGS(Long)		
				Moyenne	Infaisable	Best	Moyenne	Infaisable	Best	Moyenne	Infaisable	Best
60	10	10	0,3	0.000000	10	0	0.000000	10	0	0.000000	10	0
60	10	10	0,6	0.000000	10	0	0.000000	10	0	0.000000	10	0
60	10	10	0,9	10.000000	5	5	10.000000	4	6	10.000000	5	5
60	10	20	0,3	0.000000	10	0	0.000000	10	0	0.000000	10	0
60	10	20	0,6	0.000000	10	0	0.000000	10	0	0.000000	10	0
60	10	20	0,9	0.000000	10	0	0.000000	10	0	0.000000	10	0
60	30	10	0,3	10.300000	0	7	11.300000	0	3	10.300000	0	7
60	30	10	0,6	10.900000	0	8	11.300000	0	7	10.900000	0	8
60	30	10	0,9	11.300000	0	10	12.300000	0	3	11.300000	0	10
60	30	20	0,3	21.799999	0	4	21.799999	0	4	21.400000	0	6
60	30	20	0,6	23.400000	0	5	27.375000	2	1	22.799999	0	9
60	30	20	0,9	24.500000	0	6	27.125000	2	0	23.900000	0	8
60	40	10	0,3	10.500000	0	7	10.900000	0	3	10.500000	0	7
60	40	10	0,6	10.000000	0	10	11.200000	0	1	10.000000	0	10
60	40	10	0,9	11.100000	0	8	12.800000	0	4	11.100000	0	8
60	40	20	0,3	20.799999	0	9	22.900000	0	3	20.799999	0	9
60	40	20	0,6	23.799999	0	4	25.100000	0	4	23.299999	0	5
60	40	20	0,9	26.500000	2	5	29.250000	2	2	26.375000	2	6
80	10	10	0,3	10.000000	6	4	0.000000	10	0	10.000000	6	4
80	10	10	0,6	10.000000	6	4	10.000000	8	2	10.000000	6	4
80	10	10	0,9	10.000000	9	1	0.000000	10	0	10.000000	9	1
80	10	20	0,3	0.000000	10	0	0.000000	10	0	0.000000	10	0
80	10	20	0,6	0.000000	10	0	0.000000	10	0	0.000000	10	0
80	10	20	0,9	0.000000	10	0	0.000000	10	0	0.000000	10	0
80	30	10	0,3	10.300000	0	10	11.000000	0	5	10.300000	0	10
80	30	10	0,6	11.000000	0	9	11.900000	0	4	11.000000	0	9
80	30	10	0,9	11.100000	0	8	11.800000	0	5	11.100000	0	8
80	30	20	0,3	20.500000	0	8	23.100000	0	1	20.400000	0	9
80	30	20	0,6	22.222221	1	6	22.700001	0	4	22.222221	1	6
80	30	20	0,9	23.000000	0	10	24.666666	4	2	23.000000	0	10
80	40	10	0,3	10.500000	0	10	11.000000	0	5	10.500000	0	10
80	40	10	0,6	10.400000	0	10	11.300000	0	3	10.400000	0	10
80	40	10	0,9	11.900000	0	7	11.700000	0	6	11.900000	0	7
80	40	20	0,3	20.600000	0	8	21.500000	0	3	20.600000	0	8
80	40	20	0,6	22.100000	0	8	24.100000	0	4	21.900000	0	10
80	40	20	0,9	27.333334	1	3	28.333334	1	3	27.222221	1	3

FIGURE 4.3 – Première partie des résultats

Comme on peut le voir, nos heuristiques "PWGS" sont les meilleures dans la majorité des cas. La minimisation du nombre de travailleurs est bien plus efficace avec cette nouvelle heuristique. Si l'on regarde plus attentivement, on s'aperçoit même que Top Long, couplé avec PWGS est l'heuristique proposant le plus souvent le meilleur résultat, avec notamment sept fois un score de 10 sur 10 (dix fois le meilleur résultat, sur les dix instances).

PWGS(Short)		
Moyenne	Infaisable	Best
0.000000	10	0
0.000000	10	0
10.000000	4	6
0.000000	10	0
0.000000	10	0
0.000000	10	0
11.300000	0	3
11.300000	0	7
12.300000	0	3
21.400000	0	6
27.200001	0	3
24.125000	2	4
10.900000	0	3
11.200000	0	1
12.800000	0	4
22.000000	0	4
24.000000	0	6
28.875000	2	2
0.000000	10	0
10.000000	8	2
0.000000	10	0
0.000000	10	0
0.000000	10	0
0.000000	10	0
11.000000	0	5
11.900000	0	4
11.800000	0	5
22.600000	0	2
22.500000	0	6
25.750000	2	2
11.000000	0	5
11.300000	0	3
11.700000	0	6
21.100000	0	5
24.000000	0	4
26.555555	1	6

FIGURE 4.4 – Seconde partie des résultats

Guide d'utilisation

Voici un rapide guide d'utilisation de notre version finale du projet.

5.1 Menu

Tout d'abord, nous avons mis en place un menu en console, afin de pouvoir effectuer un calcul unique d'instance, ou un lancement du benchmark.

FIGURE 5.1 – Menu

5.1.1 Propre instance

Lorsque l'on choisit d'étudier sa propre instance, on renseigne tout d'abord le chemin du fichier, puis l'heuristique que l'on désire.

FIGURE 5.2 – Choix de l'instance

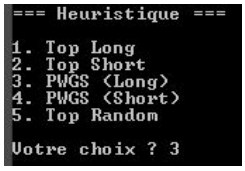


FIGURE 5.3 – Choix de l'heuristique

Une fois le calcul terminé, on obtient le résultat en console, ainsi qu'un gantt dessinant la solution.

5.1.2 Benchmark

Dans le cas du benchmark, nous avons laissé la possibilité à l'utilisateur de générer ou non, un jeu complet de nouvelles instances. Une fois ce choix fait, un fichier csv est créé en sortie afin que l'utilisateur puisse étudier les différentes heuristiques*Voir partie résultat dans la section Benchmark*.

```
Found solution: 5 workers
Processing Time: 3ms
```

FIGURE 5.4 – Résultat console *Propre solution*

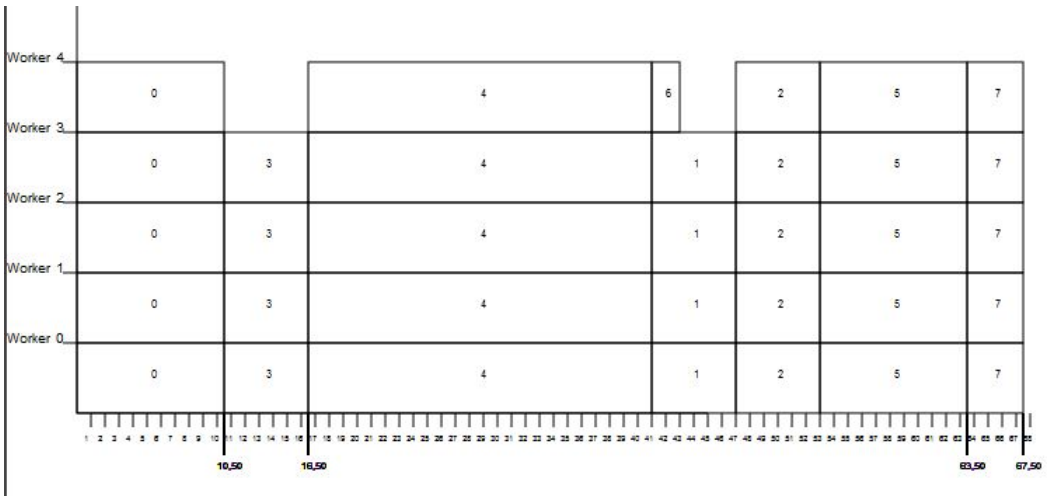


FIGURE 5.5 – Gantt de la solution *Propre solution*

De plus, comme l'intégralité des instances est sauvegardé, l'utilisateur peut toujours étudier plus précisément une instance en affichant le gantt, en effectuantla manipulation cité ci-dessus.

Conclusion

Outre l'intérêt que nous portions de base sur le projet, travailler pour un industriel, malgré les clauses de confidentialité fut pour nous un élément moteur et motivant.

En effet, avoir un problème concret de la vie professionnelle nous a aidé à entrevoir notre avenir, ainsi que des problèmatiques concrètes qu'il existe dans l'industrie.

Nous avons pu découvrir et développer de nouvelles heuristiques qui grâce à notre benchmark, un réel comparatif entre nos différentes heuristiques a pu être mis en place.

Optimal workforce assignment to operations of a paced transfer line

Département Informatique
5e année
2013 - 2014

Rapport Projet d'option Logistique & Optimisation

Résumé : Description en français

Mots clefs : Mots clés en français

Abstract: Description en anglais

Keywords: Mots clés en anglais

Encadrant
Ameur SOUKHAL
ameur.soukhal@univ-tours.fr

Université François-Rabelais, Tours

Etudiants
Anaïs LINOT
anais.linot@etu.univ-tours.fr
Rémy PRADIGNAC
remy.pradignac@etu.univ-tours.fr

DI5 2013-2014