Optimisation d'une ligne d'assemblage

Table des matières

1. Contexte

Dans toutes les usines de fabrication à la chaîne est utilisée une ligne d'assemblage qui permet d'assembler toutes les parties d'un produit. Pour illustrer le propos dans le cadre de ce projet, nous travaillerons sur la ligne d'assemblage de la future voiture électrique ECElecar. Mais le logiciel que l'on développera sera capable en réalité d'optimiser n'importe quelle ligne d'assemblage.

Pour développer l'ECElecar, la ligne doit effectuer certaines actions, comme monter les roues sur les essieux, monter les essieux sur le châssis, placer la carrosserie sur le châssis, placer le moteur, le volant, les sièges dans la carrosserie, peindre la carrosserie…

Une ligne d'assemblage est une ligne sur laquelle les voitures en construction se déplacent et autour de laquelle sont placés un certain nombre de robots et/ou d'ouvriers qui effectuent des opérations répétitives (fig. 1).

Figure 1 : Ligne d'assemblage de l'entreprise Car Bench qui équipe un grand nombre de constructeurs automobiles

Les différents robots ou ouvriers présents le long de la ligne sont appelés **station**. Chaque station est capable de réaliser une ou plusieurs opérations. Il suffit pour cela de programmer le robot ou de former l'ouvrier. Nous nous plaçons dans le cas ou n'importe quelle station est capable de réaliser n'importe quelle opération.

Il existe plusieurs contraintes pour réaliser les opérations sur les stations :

1. Contraintes d'exclusion : certaines opérations ne peuvent pas être effectuées par la même station. Par exemple, une même machine ne peut pas à la fois poser le circuit électrique et peindre la carrosserie.
2. Contraintes de précédence : certaines opérations doivent être effectuées avant d'autres. Par exemple, on ne peut monter le moteur que si la carrosserie a déjà été posée avant.

3. Temps de cycle : les voitures ne restent qu'un temps donné auprès de chaque station. Ce temps est fixe et le même pour toutes les stations, ce qui fait que toutes les voitures passent à la station suivante en même temps.

Intéressons-nous en détail à chacune de ces contraintes. Mais avant cela, voici les notations utilisées dans ce document :

- V est l'ensemble des opérations à effectuer pour assembler le produit,
- t_j est le temps d'exécution d'une opération j, $j \in V$
- m désigne le nombre de stations de la ligne d'assemblage
- $M = \{1, 2, ..., m\}$ est l'ensemble des stations
- $G = \{V, A\}$ est un graphe orienté acyclique qui représente les contraintes de précédence
- E est un ensemble de paires d'opérations qui ne peuvent pas être affectées ensemble à la même station.
- T_0 est le temps de cycle

Soient une opération $j \in V$ et une station $k \in M$. On définit la variable de décision suivante :

$$y_{jk} = \begin{cases} 1, & \text{si l'opération } j \in V \text{ est affectée à la station de travail } k \\ 0, & \text{sinon} \end{cases}$$

1.1. Contraintes d'exclusion

Les contraintes d'exclusion expriment le fait que deux opérations ne peuvent pas être effectuées par la même station :

$$\forall \{i,j\} \in E, \forall k \in M,\ y_{ik} + y_{jk} \leq 1$$

Par exemple, étant donnés un ensemble d'opérations $V = \{1, 2, 3, \dots, 34, 35\}$, on donne l'ensemble de paires d'opérations ne pouvant pas être affectées à la même station :
E = {{1, 4}, {1, 17}, {1, 20}, {2, 11}, {3, 24}, {4,15}, {5, 22}, {6, 24}, {8, 21}, {9, 22}, {10, 15}, {11, 31}, {12, 13}, {12, 20}, {15, 17}, {16, 17}, {22, 26}, {30, 33}, {31, 32}, {33, 35}}
Les opérations 1 et 4 ne pourront donc pas être affectées à la même station, de même que les opérations 1 et 17 etc. À partir de ces contraintes d'exclusion, une répartition possible des opérations est montrée figure 2. Chaque WS désigne une station (workstation en anglais).

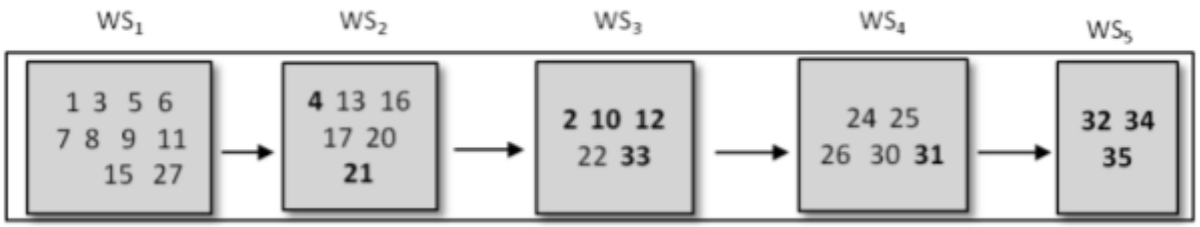

Figure 2 : exemple de ligne après prise en compte des contraintes d'exclusion

1.2. Contraintes de précédence

Les contraintes de précédence imposent que certaines opérations soient effectuées avant d'autres. $G = \{V, A\}$ étant le graphe de précédence, les contraintes de précédence s'expriment de la façon suivante :

$$\forall (i,j) \in A, \sum_{k \in M} k\, y_{ik} \leq \sum_{k \in M} k\, y_{jk}$$

La figure 3 montre un exemple de graphe de précédence. On voit sur ce graphe que par exemple, l'opération doit être effectuée avant l'opération 6, qui elle-même doit être effectuée avant les opérations 27, 4, 9, etc.

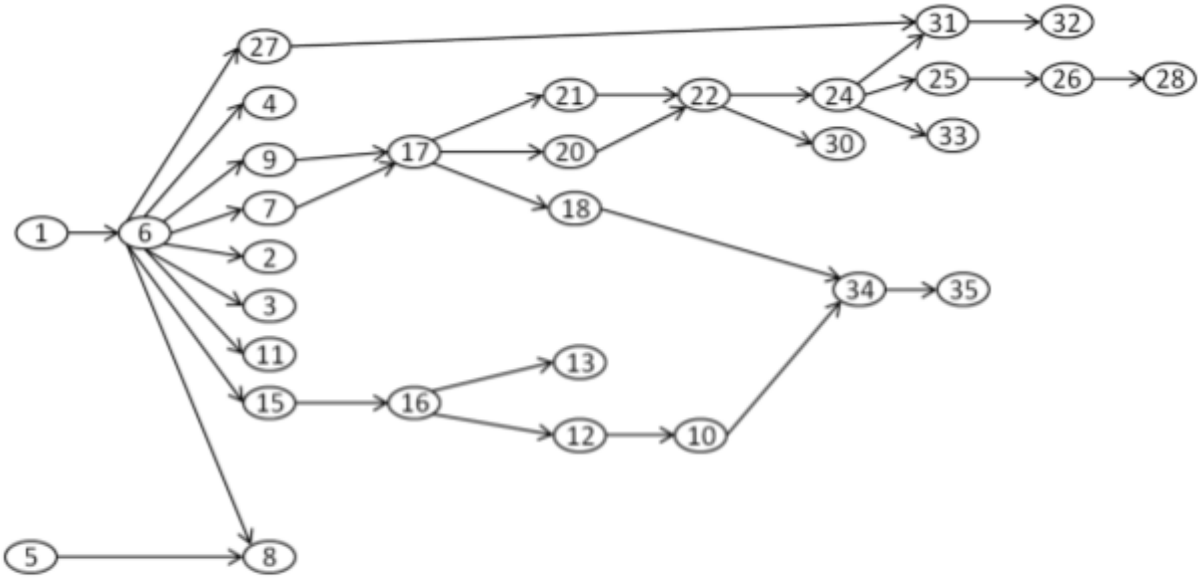

Figure 3 : exemple de graphe de précédence

1.3. Contrainte de temps de cycle

Le temps de cycle T_0 est une constante. La durée totale des opérations affectées à la station de travail k en doit pas dépasser le temps de cycle :

$$\forall k \in M, \sum_{j \in V} t_j y_{jk} \leq T_0$$

Par exemple, le temps de cycle peut valoir $T_0 = 10\ s$ et les durées de chaque opération sont notées dans le tableau de la figure 4.

Opération	Temps (s)	Opération	Temps (s)	Opération	Temps (s)
1	0.93	11	1	24	0.09
2	1.06	12	0.78	25	0.17
3	0.68	13	0.64	26	0.09
4	0.16	15	0.09	27	0.12
5	0.68	16	0.17	30	0.91
6	0.16	17	0.09	31	0.72
7	0.68	18	0.12	32	0.15
8	0.16	20	1	33	0.19
9	0.68	21	0.78	34	0.33
10	0.16	22	0.6	35	0.97

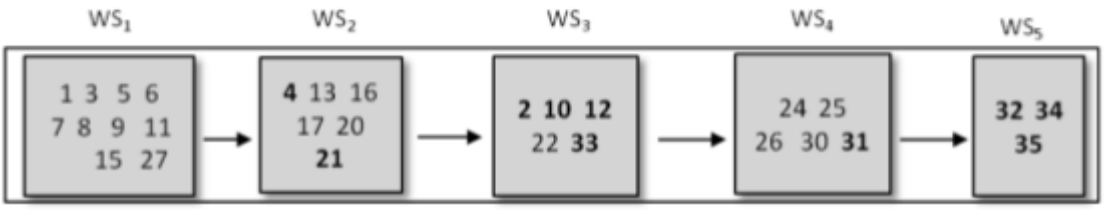

Figure 4 : durée de chaque opération

Sur cet exemple, on voit que chaque station respecte bien le temps de cycle.

1.4. Optimisation

L'objectif principal de ce projet est d'optimiser le nombre de stations, ici donc de minimiser le nombre de stations. Chaque station ayant un coût non négligeable, cela permet par ricochet de minimiser le coût de production.

2. Cahier des charges

Les différentes contraintes devront être stockées dans un ou plusieurs fichiers texte et seront les données d'entrée de votre programme. L'architecture des fichiers est laissée à votre convenance.

Dans le contexte de l'industrie, le plus important est d'obtenir un résultat, ici une répartition des opérations par station pour savoir comment construire notre ligne d'assemblage. L'esthétique importe peu. Vous vous concentrerez donc sur la réponse à la problématique, à l'aide d'algorithmes de théorie des graphes. Le programme pourra donc s'exécuter entièrement en console. Si toutefois vous choisissez de développer une interface graphique, sachez qu'elle ne fera l'objet d'aucun point dans la notation. Faites en sorte que l'affichage soit clair et compréhensible en console.

Vous pourrez utiliser certains algorithmes de théorie des graphes vus en cours, mais vous êtes libres de chercher d'autres algorithmes dans la littérature scientifique, auquel cas vous ne manquerez pas de citer vos sources. Vous pouvez aussi inventer vos propres algorithmes. Dans tous les cas, lors de la soutenance, vous présenterez les algorithmes que vous avez utilisés, sous forme de pseudo-code en français (comme en cours). Rappel : ne **jamais** présenter de code source en soutenance, c'est beaucoup trop indigeste pour le jury…

2.1. Première approche simpliste

Dans un premier temps, développez des algorithmes pour répondre aux contraintes indépendamment les unes des autres. C'est-à-dire :

- Cherchez à répondre à la contrainte d'exclusion, sans prendre en compte les autres contraintes et proposez une répartition des opérations par station en fonction de cette contrainte seule.
- Cherchez à prendre en compte les contraintes de précédence et de temps de cycle. En effet, prendre en compte uniquement les contraintes de précédence est trop simpliste : si le temps de cycle est infini, il suffit d'affecter toutes les opérations sur une seule station, et le tour est joué !

Dans chacun des cas, rappelez-vous l'objectif du projet : minimiser le nombre de stations.

Votre interface affichera les répartitions par station et le nombre de stations total pour chaque contrainte, de manière à pouvoir les comparer facilement pour que les décideurs puissent facilement analyser les différentes propositions et prendre les bonnes décisions. Les résultats de chaque contrainte pourront être affichés côte-à-côte ou les uns au-dessus des autres en fonction de ce que vous trouvez le plus esthétique ou le plus clair.

2.2. Dans un second temps : multi-contraintes

Une fois cette phase effectuée, vous pourrez proposer de prendre en compte les contraintes différemment, par exemple :

- contrainte d'exclusion et de précédence
- contrainte d'exclusion et temps de cycle
- et pourquoi pas les trois à la fois !

2.3. Extension : heuristique

Cette partie fera l'objet de points bonus. Ces points ne seront comptabilisés que si les objectifs principaux du point 2.1 (prise en compte des contraintes indépendamment les unes des autres) sont déjà opérationnels. Ceci dit, la mise en place d'une heuristique peut aider à prendre en compte plusieurs contraintes pour optimiser le nombre de stations et minimiser le temps de calcul.

Optimiser un problème sur lequel interviennent plusieurs contraintes peut vite devenir très complexe à résoudre. La recherche d'une solution optimale exacte est souvent chronophage et incompatible avec une prise de décision rapide. On ne peut pas se permettre de chercher la solution optimale s'il faut 1 an de calcul pour la trouver. Souvent, on se contentera d'une "bonne" solution si on peut obtenir la réponse rapidement.

Il existe un grand nombre d'heuristiques pour trouver de "bonnes" solutions dans un temps acceptable, et qui donnera de plus ou moins bonnes solutions. L'une de ces heuristiques s'appelle COMSOAL (Computer Method of Sequencing Operations for Assembly Lines). Une explication claire et concise de cette heuristique se trouve dans le document suivant en page 45 :

https://depot-e.uqtr.ca/id/eprint/1363/1/030033031.pdf

Trois autres heuristiques sont d'ailleurs expliquées dès la page 41.

2.4. Fichiers de contraintes

Toutes les données utiles pour lancer l'optimisation de la ligne d'assemblage seront stockées dans des fichiers textes. Il y a aura 4 fichiers textes qui devront respecter une certaine nomenclature. Les données pourront bien entendu être modifiées pour les tests. En soutenance, le jury pourra vous donner des fichiers sur lesquels lancer votre programme. Il est très important de respecter strictement le format de ces fichiers.

2.4.1. exclusions.txt

Le fichier **exclusions.txt** (au pluriel) contiendra la liste des paires d'opérations qui ne peuvent pas être exécutées sur la même station. Chaque paire sera écrite sur une ligne suivant le format suivant où op1, op2, … sont des numéros d'opération :

```
op1 op2
op3 op4
```

```
op5 op6
…
```

Voici par exemple à quoi ressemblerait ce fichier pour les contraintes d'exclusion listées au point 1.1 :

```
1 4
1 17
1 20
2 11
3 24
4 15
5 22
6 24
8 21
9 22
10 15
11 31
12 13
12 20
15 17
16 17
22 26
30 33
31 32
33 35
```

Vous remarquerez que nulle part n'est indiqué le nombre de contraintes listées dans le fichier. Si vous avez besoin de ce nombre, c'est à votre algorithme de le trouver. Le fichier doit être le plus facile et le plus rapide possible à modifier. Dans l'industrie, on n'aime pas perdre son temps à écrire des informations qui peuvent se retrouver par simple calcul ou comptage…

2.4.2. precedences.txt

Le fichier **precedences.txt** (au pluriel et sans accent ; les accents, c'est le mal…) contiendra le graphe de précédences en listant les arcs du graphe. Il respectera le format suivant où op1, op2… sont des opérations, (op1,op2) étant un arc, (op3, op4) un autre :

```
op1 op2
op3 op4
op5 op6
```

Ce fichier ressemblerait à ceci pour le graphe présent au point 1.2 :

```
1 6
6 27
6 4
6 9
6 7
6 2
6 3
6 11
6 15
6 8
5 8
27 31
31 32
9 17
7 17
17 20
17 18
17 21
21 22
20 22
22 24
22 30
24 31
24 25
24 33
25 26
26 28
18 34
34 35
15 16
16 13
16 12
12 10
10 34
```

A nouveau, nulle part n'est indiqué le nombre d'arcs du fichier. Votre algorithme devra le déduire automatiquement à partir de votre fichier.

2.4.3. operations.txt

Le fichier **operations.txt** (au pluriel) contiendra la liste des opérations ainsi que leur temps d'exécution suivant le format suivant, où le couple op1 désigne le numéro de l'opération et t1 son temps d'exécution :

```
op1 t1
op2 t2
op3 t3
```

Voici par exemple à quoi ressemblerait ce fichier pour le tableau donné au point 1.3 :

```
1 0.93
2 1.06
3 0.68
4 0.16
5 0.68
6 0.16
7 0.68
8 0.16
9 0.68
10 0.16
11 1
12 0.78
13 0.64
15 0.09
16 0.17
17 0.09
18 0.12
20 1
21 0.78
22 0.6
24 0.09
25 0.17
26 0.09
27 0.12
28 0.10
30 0.91
31 0.72
32 0.15
33 0.19
34 0.33
35 0.97
```

Encore une fois, le nombre d'opérations n'est pas écrit dans le fichier. C'est à votre programme de le trouver s'il en a besoin. Au passage, vous remarquerez qu'il n'y a pas 35 opérations dans cet exemple (eh non !). Il n'y a par exemple pas d'opération 14, 19 ou 23… C'est comme ça, il faudra respecter les numéros des opérations qui sont donnés dans ce fichier… Notre programme doit être le plus souple possible.

2.4.4. temps_cycle.txt

Le fichier **temps_cycle.txt** contiendra une seule information : la durée du temps de cycle, en secondes. Si comme dans le point 1.3, le temps de cycle est de 10s, ce fichier ressemblera donc à ceci :

```
10
```

3. Organisation

Ce projet a pour but de vous faire travailler la **Théorie des graphes** implémentée en **langage C**.

Ce projet sera à effectuer par **équipes de 3 ou 4 étudiants au sein du même groupe de TP**. Vous devez inscrire votre équipe sur la page Boostcamp de votre campus (Paris ou Lyon) avant la date inscrite sur cette même page.

La date de démarrage de votre projet sera affichée sur la page Boostcamp de votre campus. Idem pour la date de rendu de vos livrables, ainsi que les dates de soutenances.

3.1. Git

Vous devrez utiliser git pour effectuer ce projet. Les enseignants de votre campus vous donneront accès sur la page Boostcamp à un dépôt git pour votre équipe via githubclassroom.

Apprenez à utiliser les branches git pour vous répartir le travail (par exemple une branche `main` principale qui contiendra un code fonctionnel et débogué et une branche au nom de chaque étudiant pour développer). Faites des commit/push quotidiens pour vous assurer une sauvegarde de votre travail et assurer un historique en cas de plantage, décision de reprendre un ancien code… Apprenez à fusionner les branches entre elles. Un fichier contenant les commandes git de base vous est fourni sur la page Boostcamp. Apprenez à utiliser les commandes git en *mode console*, ce qui vous permet d'avoir des messages en cas de problème et de pouvoir faire des recherches sur Internet.

Faites un fichier **.gitignore** pour exclure du git tout fichier non en lien direct avec votre projet (fichiers de compilation, exécutable). Ne laissez que les fichiers .c, .h, .txt (dont le CMakeLists et les fichiers de contraintes).

Le dépôt githubclassroom **est** le lieu de dépôt final de votre projet. Une date butoir sera définie. Passée cette date, le dépôt **interdira** tous les push vers le dépôt. Prévoyez donc de la marge pour effectuer votre dernier push.

3.2. Livrables

Vous aurez deux livrables à rendre :

1. Un diaporama de soutenance à rendre sur la page Boostcamp de votre campus contenant :
    - Page de garde avec titre, groupe de TP, numéro d'équipe et noms (1 slide)
    - **Versionning git :** screenshot screenshot montrant la bonne utilisation et répartition des tâches du code entre coéquipiers (nombre de commits, nombre de lignes). (1 slide)

- **Méthodologie de votre travail** : (8 slides maximum)
  - Répartition des tâches par personne
  - Diagramme PERT de gestion de votre projet avec dates au plus tôt, au plus tard et chemin critique
  - Algorithmes utilisés écrits en pseudo-code (en français) ou sous forme d'algorigramme, mais **jamais** de code source
  - Tous schémas, diagrammes, formules mathématiques ou screenshots permettant d'éclaircir votre propos et de rendre votre présentation attractive
- **Bilan individuel et collectif** sans blabla de l'état du travail effectué. (1 à 2 slides)
- **Bibliographie** précise de toutes vos sources (web, livres etc.). (1 slide). Toute source non citée est considérée de facto comme un plagiat

2. Dépôt de votre projet sur githubclassroom comme expliqué précédemment. Ce dépôt ne doit contenir **que** vos codes sources et fichiers textes.
   Citez vos sources précisément si vous reprenez du code : auteur, liens vers site etc…
   L'algorithme anti-plagiat MOSS sera lancé sur les codes sources de toutes les équipes des deux campus et les résultats seront rendus publics. Un taux de ressemblance trop élevé entre deux équipes entraînera la note de 0/20 pour les 2 équipes ainsi qu'une convocation en conseil de discipline.

La note de projet est la moyenne pondérée de la **présentation du diaporama (coefficient 1)** et de la **démonstration du code (coefficient 2)** avec au *maximum 4 points de bonus pour les extensions mentionnées dans le sujet **mais** pas prioritaires, la note étant plafonnée à 20.*

Rappelez-vous qu'en soutenance vous êtes là pour "vendre" un produit, comme si vous étiez devant des clients, alors veillez un minimum à votre tenue. Bannissez les tenues trop décontractées comme les casquettes, shorts ou tongues… (ne riez pas, j'en ai déjà vu certains sont comme ça en soutenance). N'en faites pas trop non plus, pas la peine de venir en smoking ou robe de soirée, juste en tenue professionnelle !