Ordonnancement de produits périssables

Julien Fondrevelle

ASPECTS PRATIQUES CONCERNANT L'ORDONNANCEMENT DE PRODUITS PERISSABLES : APPLICATION AU CAS DU FLOWSHOP DE PERMUTATION

Julien FONDREVELLE : DEA GSI, sous la direction de Marie-Claude PORTMANN -
Equipe MACSI, Ecole des Mines de Nancy, Parc de Saurupt, 54042 Nancy Cedex -
julien.fondrevelle@mines.inpl-nancy.fr

Résumé : Les travaux de recherche en ordonnancement, bien que très nombreux, traitent rarement les problèmes à écarts de temps maximaux. Pourtant ce type de problèmes correspond à de nombreuses situations industrielles, notamment la fabrication de produits périssables. Il est possible d'adapter des techniques de résolution classiques en ordonnancement à ce contexte spécifique : une Procédure par Séparation et Evaluation peut en particulier être mise au point pour les problèmes de flowshop de permutation. Les différents paramètres de cette procédure (borne inférieure, borne supérieure, cheminement) doivent alors intégrer les contraintes particulières du problème étudié.

Mots clés : Ordonnancement, heuristique, procédure par séparation et évaluation, bornes inférieures, bornes supérieures.

Abstract : Although research on scheduling is very large, few papers deal with maximal time-lags problems. Yet, this kind of problems represents many industrial situations, especially when it comes to manufacturing perishable products. Classical scheduling techniques may be adapted to this specific context : a branch and bound procedure may be developed for permutation flowshop problems. Its different parameters (lower bound, upper bound, exploration) must integrate the particular constraints of the problem.

Keywords : Scheduling, heuristic, branch and bound procedure, lower bounds, upper bounds.

Introduction

Cet article présente les principaux résultats d'un projet de recherche de DEA mené au sein de l'équipe MACSI (Modélisation Analyse et Conduite des Systèmes Industriels) du LORIA. L'objectif général de ce projet était l'étude et la résolution de problèmes d'ordonnancement particuliers que nous décrivons dans la section 2. De façon plus précise, nous avons cherché dans le cadre de ce travail à mettre en évidence des résultats de dominance, afin de réduire l'exploration de l'espace des solutions à certaines régions, à mettre au point des méthodes de résolution efficaces et à élaborer des instances intéressantes correspondant à des situations réelles. Cependant, une attention toute particulière a été portée sur le côté pratique et applicable des méthodes de résolution.

Dans la section 1, nous revenons brièvement sur la problématique générale de l'ordonnancement, puis nous présentons précisément le sujet d'étude dans la section 2. La section 3 décrit le modèle utilisé, la section 4 donne un aperçu rapide de la littérature sur le sujet alors que les sections 5, 6 et 7 fournissent les principaux éléments de résolution mis en œuvre. La section 8 soulève la question de la génération des données et la section 9 présente

les résultats obtenus. On trouve en annexes un exemple numérique d'application des méthodes exposées, des figures illustrant les principes de résolution ainsi que certains résultats expérimentaux.

I- Problématique générale de l'ordonnancement

Les contraintes actuelles qui pèsent sur les entreprises en termes de respect des délais annoncés aux clients, ainsi que la nécessité de constamment réduire ces délais afin de faire face à la concurrence ont contribué au développement de la fonction ordonnancement dans le secteur de la gestion de production. On assiste actuellement à son intégration au sein d'un ensemble d'activités beaucoup plus vaste, la *supply chain* (ou chaîne logistique), qui concerne la gestion et l'optimisation des flux, depuis les fournisseurs en matières premières jusqu'aux clients finaux.

L'objectif de l'ordonnancement est de déterminer les dates d'exécution d'un ensemble de tâches à réaliser, en leur attribuant les ressources nécessaires, qu'elles soient humaines (la main d'œuvre), matérielles (machines, outils, matières premières, composants), financières (argent) … Il s'agit en fait d'un domaine issu des mathématiques discrètes dont l'objectif est de résoudre un problème en trouvant parmi un ensemble de solutions admissibles, c'est-à-dire respectant une série de contraintes, une solution dite optimale qui minimise (ou maximise) une fonction d'évaluation, dite fonction objectif. La résolution de tels problèmes est en général relativement difficile et est effectuée par l'intermédiaire d'algorithmes programmés au niveau informatique ; si l'on exige de trouver une solution véritablement optimale, il est la plupart du temps nécessaire d'explorer (astucieusement) l'ensemble des solutions, ce qui conduit néanmoins à des méthodes très coûteuses en termes de temps et parfois d'espace de stockage informatique.

II- Description du problème étudié

On considère un ensemble de travaux ou jobs, chacun d'eux pouvant correspondre à un produit dans le cas d'une production manufacturière. Chaque job est composé d'une série d'opérations élémentaires ou tâches devant être effectuées sur certaines machines. L'objectif est de déterminer les machines et les dates pour l'exécution des opérations de manière à optimiser un ou plusieurs critères fonctions de ces paramètres.

Le problème que nous étudions impose les conditions supplémentaires suivantes : à chaque tâche correspond une seule machine, de telle sorte que le problème considéré ne concerne pas l'affectation des tâches aux ressources. Le critère à minimiser est la durée totale de l'ordonnancement, c'est-à-dire la plus grande date de fin sur l'ensemble des opérations : il s'agit du Cmax (ou makespan). Chaque machine ne peut exécuter qu'une opération à la fois et chaque opération ne peut être réalisée que sur une machine à la fois. D'autre part, on suppose les tâches non préemptives, ce qui signifie qu'une tâche ne peut être divisée en sous-tâches pouvant être exécutées à des instants différents. En ce qui concerne le mode de production, on se place dans le cadre du flowshop de permutation : la succession des opérations au sein d'un travail est imposée (elle correspond à une gamme linéaire) et identique pour tous les travaux ; à chaque machine correspond une et une seule opération pour un job donné ; ainsi, il est possible de numéroter les machines de telle sorte que les jobs passent sur les machines dans l'ordre 1, 2 … m, avec m le nombre de machines. En outre, on impose que pour toutes les machines, la séquence de passage des jobs soit la même. Ces deux hypothèses correspondent à des situations assez courantes dans le monde industriel : lignes de production avec un

convoyeur entre les machines qui ne permet pas le dépassement des jobs entre les machines. On ajoute enfin une contrainte supplémentaire concernant les temps d'attente entre tâches d'un même job : ceux-ci ne doivent pas dépasser certaines valeurs appelées écarts de temps maximaux ou attentes maximales. Arbitrairement, nous considérons le temps d'attente entre deux tâches comme l'écart de temps séparant la date de fin de la première tâche et la date de début de la seconde tâche. Au cours de notre travail, nous nous sommes principalement concentrés sur des écarts de temps définis uniquement entre les opérations successives des différents jobs, mais nous verrons que d'autres hypothèses peuvent être traitées en suivant un principe analogue à celui utilisé. Dans la pratique, de tels problèmes concernent des domaines très variés :

- l'industrie agro-alimentaire, qui doit tenir compte des propriétés et des limites de conservation des produits périssables, notamment en cas de congélation / surgélation
- l'industrie chimique et pharmaceutique, dont la production peut porter sur des composés instables
- la sidérurgie et la métallurgie (en particulier lors des traitements thermiques), pour lesquelles interviennent des matériaux dont les propriétés changent selon la température
- l'informatique et l'électronique embarquée, dans le cas de systèmes experts nécessitant des temps de réponse impératifs dans des intervalles de temps fixés

III- Modélisation

Nous décrivons dans cette partie le modèle utilisé pour représenter la réalité ; celui-ci est basé sur un formalisme mathématique relativement simple et est communément employé dans la littérature.

L'ensemble $J = \{1, 2, \dots n\}$ désigne l'ensemble des n travaux à accomplir.

L'ensemble $M = \{1, 2, \dots m\}$ désigne l'ensemble des m machines utilisées pour l'exécution des tâches.

Chaque job j est composé de m opérations élémentaires (j,1), (j,2) … (j,m) devant être réalisées respectivement sur les machines 1, 2, … m et dans cet ordre : ceci correspond à l'hypothèse d'une production de type flowshop.

L'hypothèse supplémentaire selon laquelle on ne travaille qu'avec des ordonnancements de type flowshop de permutation nous permet d'utiliser une unique permutation $\pi = (\pi(1), \pi(2), \dots \pi(n))$ pour représenter la séquence des jobs sur les différentes machines.

D'autre part, pour chaque tâche (j,k), on désigne par $p_{(j,k)}$ sa durée (qui est une donnée du problème), $t_{(j,k)}$ sa date de début et $c_{(j,k)}$ sa date de fin (qui sont des variables de décision à déterminer lors de la résolution). On a donc : $\forall j \in J, \forall k \in M, c_{(j,k)} = t_{(j,k)} + p_{(j,k)}$.

Les contraintes supplémentaires d'attente maximale à ne pas dépasser sont traduites par l'existence d'écarts de temps maximaux $a_{(j,k)}$ ($j \in J, k \in M\{m\}$) entre deux tâches successives d'un même job.

Les différentes contraintes du problème peuvent donc être traduites par l'ensemble d'inégalités suivantes :

$\forall j \in J, \forall k \in M\{m\}, t_{(j,k+1)} \geq c_{(j,k)}$ (contraintes de précédence au sein des jobs)

$\forall j \in J\{n\}, \forall k \in M, t_{(\pi(j+1),k)} \geq c_{(\pi(j),k)}$ (contraintes de précédence sur les machines)

$\forall j \in J, \forall k \in M\{m\}, t_{(j,k+1)} \leq c_{(j,k)} + a_{(j,k)}$ (contraintes d'attente maximale)

Le critère que l'on souhaite optimiser ici est la durée totale définie par :

$Cmax = \max \{c_{(j,m)} / j \in J\}$ qu'il faut minimiser.

IV- Bibliographie

Alors que la recherche en ordonnancement est très active d'une manière générale (plus de 40000 articles fournis par la base de données INSPEC après une recherche sur le mot clé « scheduling »), peu de travaux sont consacrés à ces problèmes particuliers (le nombre de références passe à 40 environ si l'on rajoute le terme «time lags »), d'autant plus que pour la quasi-totalité de ceux-ci, les contraintes d'écarts de temps correspondent à des attentes minimales à respecter (comme dans Szwarc (1983) et Baker (1990)), ce qui conduit à des situations complètement différentes. Nous présentons dans cette partie les différents ouvrages que nous avons consultés et leur contribution à ce travail de recherche.

On peut trouver dans Carlier (1988) les concepts de base liés à l'ordonnancement : modèles utilisés, problèmes fondamentaux, méthodes de résolution classiques … D'autre part, les auteurs insistent particulièrement sur la question de la complexité des problèmes et des algorithmes et son importance, à la fois théorique et pratique.

Lopez (2001) est également un ouvrage général sur les problèmes d'ordonnancement. Il dresse un panorama relativement complet des catégories de problèmes et des méthodes de résolution employées qui font l'objet de recherches à l'heure actuelle.

En ce qui concerne le flowshop, Tanaev (1994) établit une liste de problèmes résolus polynomialement : on y retrouve les algorithmes de Johnson (Johnson (1954)) concernant le problème à 2 machines, et de Gilmore et Gomory pour le problème à 2 machines sans attente (Gilmore (1964)). Des heuristiques particulièrement intéressantes pour le problème à m machines sont présentées respectivement dans Nawaz (1983) pour la méthode NEH et dans Campbell (1970) pour la méthode CDS, dérivée de la règle de Johnson. Ces deux méthodes sont également décrites dans Lopez (2001).

Le principe des Procédures par Séparation et Evaluation est détaillé dans Carlier (1988) (sous le nom de méthodes arborescentes) ainsi que dans Lopez (2001). Une application au cas du flowshop est présentée dans Ignall (1965).

Une PSE efficace dédiée au problème d'ordonnancement à une machine, avec time lags positifs et négatifs, est décrite dans Brucker (1999) : les auteurs montrent comment se ramener à ce type de problème à partir de différentes situations (job shops, open shops, machines parallèles, problèmes multi-ressources …). La résolution fait appel au modèle disjonctif et à des règles d'arbitrage immédiat. Néanmoins, d'après les auteurs, cette méthode générale se révèle moins efficace pour les problèmes d'atelier classiques que les techniques spécifiques qui leur sont consacrées.

V- Principes de résolution

Dans cette partie, nous allons détailler précisément les principes sur lesquels se fondent les méthodes de résolution développées dans le cadre de ce projet. Le schéma général fait intervenir une Procédure par Séparation et Evaluation dont on a cherché à améliorer les différents paramètres. Avant tout, nous énonçons un résultat remarquable.

1) Résultat fondamental

Connaissant la séquence complète des jobs π, il est possible de construire l'ordonnancement au plus tôt correspondant en un temps polynomial. Ainsi, on peut calculer « facilement » les dates de début et de fin au plus tôt de l'ensemble des tâches à partir de la séquence des jobs.

Le principe de calcul est le suivant :

- on cale le premier job au plus tôt : $t_{(\pi(1),\,1)} = 0$, $c_{(\pi(1),1)} = t_{(\pi(1),\,1)} + p_{(\pi(1),\,1)}$

  $\forall\, k \in M\backslash\{m\}$, $t_{(\pi(1),\,k+1)} = c_{(\pi(1),\,k)}$; $c_{(\pi(1),\,k+1)} = t_{(\pi(1),\,k+1)} + p_{(\pi(1),\,k+1)}$

  Pour ce premier job, les contraintes d'attente sont respectées puisque les écarts de temps entre opérations successives sont nuls.
- on ordonnance successivement les autres jobs de la manière suivante :

  On cale chaque tâche au plus tôt, en respectant les contraintes de précédence dans le job et les contraintes de précédence sur les machines ; à l'étape j+1, $j \in J\backslash\{n\}$, on a donc :

  $t_{(\pi(j+1),\,1)} = c_{(\pi(j),\,1)}$, $c_{(\pi(j+1),\,1)} = t_{(\pi(j+1),\,1)} + p_{(\pi(j+1),\,1)}$

  $\forall\, k \in M\backslash\{m\}$, $t_{(\pi(j+1),\,k+1)} = \max\,(c_{(\pi(j+1),\,k)}\,,\, c_{(\pi(j),\,k+1)})$;

  $c_{(\pi(j+1),\,k+1)} = t_{(\pi(j+1),\,k+1)} + p_{(\pi(j+1),\,k+1)}$

  Puis on vérifie si les contraintes d'attente maximale sont satisfaites, dans l'ordre inverse du passage sur les machines (de la dernière machine à la première). Si ce n'est pas le cas, on applique la technique de « l'élastique » en retardant l'exécution de la tâche située juste avant l'attente trop grande :

  $\forall\, k \in M\backslash\{m\}$, si $t_{(\pi(j),\,k+1)} - c_{(\pi(j),\,k)} > a_{(\pi(j),\,k)}$ alors $c_{(\pi(j),\,k)} = t_{(\pi(j),\,k+1)} - a_{(\pi(j),\,k)}$;

  $t_{(\pi(j),\,k)} = c_{(\pi(j),\,k)} - p_{(\pi(j),\,k)}$

On dispose alors d'un ordonnancement au plus tôt respectant les contraintes et la séquence de jobs fixée π. Parmi les ordonnancements ayant la même séquence de jobs, celui-ci est optimal.

Nous allons maintenant justifier brièvement que l'algorithme décrit est bien polynomial : l'algorithme consiste à effectuer une boucle globale sur l'ensemble des jobs. A chaque étape, on effectue 2 boucles sur les machines : une première pour calculer les dates au plus tôt sans tenir compte des contraintes d'attente, une seconde pour tester si ces contraintes sont satisfaites et éventuellement corriger les dates au plus tôt ; pour le premier job, la seconde boucle n'est pas réalisée car elle est inutile. A l'intérieur de chaque boucle, le nombre d'opérations élémentaires (affectations, sommes, comparaisons) est borné par des constantes indépendantes des paramètres du problème. La complexité de l'algorithme est donc en O(n.m), où n et m désignent respectivement le nombre de jobs et le nombre de machines ; en fait, n.m désigne le nombre de tâches du problème.

2) Présentation des Procédures par Séparation et Evaluation (PSE)

Dans ce paragraphe, nous décrivons l'idée principale qui est à la base de la méthode de résolution que nous avons mise au point. Il s'agit d'une Procédure par Séparation et Evaluation (PSE) dont nous avons cherché à améliorer certains paramètres. On peut trouver une présentation rapide de ces méthodes dites arborescentes, appliquées à l'ordonnancement, dans les ouvrages de Carlier (1988) et Lopez (2001). Nous reprenons uniquement dans cette partie les éléments employés pour la résolution du problème considéré ici.

La PSE élaborée construit progressivement des solutions, en ajoutant à chaque étape un job à une séquence partielle. Chaque nœud de l'arborescence correspond ainsi à une séquence partielle de jobs. On passe d'un nœud à ses fils en ajoutant en fin de séquence un job parmi ceux qui ne figurent pas dans cette séquence. Les feuilles de l'arbre correspondent aux séquences complètes de jobs, donc aux permutations. Il est ainsi possible de construire toutes les permutations envisageables. Cependant, pour éviter d'explorer et de tester toutes ces

combinaisons, on associe à chaque nœud une valeur représentant une estimation de la meilleure solution pouvant être obtenue à partir de ce nœud. D'autre part, si l'on dispose lors de l'exploration de l'arbre d'une solution de bonne qualité, il est possible d'éliminer certains sous arbres sans avoir à les visiter : en effet, si la valeur d'un nœud est supérieure à celle de la meilleure solution trouvée jusque là, on est assuré de ne pas aboutir à une meilleure solution en développant le nœud. Ainsi, l'un des principaux objectifs pour mettre au point une PSE efficace est de définir des évaluations précises des nœuds (les plus grandes possibles) qui jouent le rôle de bornes inférieures, et d'obtenir très tôt des solutions de bonne qualité (les plus petites possibles) qui interviennent en tant que bornes supérieures.

La borne inférieure la plus simple que l'on peut associer à un nœud consiste à construire l'ordonnancement partiel correspondant à la sous-séquence σ du nœud puis à ajouter pour toutes les machines à la date de fin du dernier job de σ la somme des durées des tâches des jobs non encore placés et à conserver le maximum :

si longueur(σ) = i, $LB = \max \{ c_{(\sigma(i), k)} + \Sigma_{j \in J \setminus \sigma} p_{(j,k)} / k \in M \}$

Nous verrons dans la suite comment calculer des évaluations plus efficaces.

En ce qui concerne la borne supérieure, on se contente de la valeur de la meilleure solution trouvée. On peut alors procéder de deux manières :

- tant qu'on n'a pas atteint de solution, cette borne possède une très grande valeur, de manière à n'élaguer aucune branche
- on exécute avant la PSE une heuristique nous fournissant une solution de bonne qualité. La description des différentes heuristiques testées et leur comparaison font l'objet d'un paragraphe spécifique.

Le cheminement choisi pour notre PSE se base sur le principe d'exploration, en profondeur d'abord : non seulement cette technique permet d'atteindre plus rapidement les feuilles, mais elle présente aussi un avantage en termes de cheminement et en termes de stockage de données : en effet, il est facile de passer d'un père à un fils ou inversement par ajout ou suppression d'un travail à la fin d'une sous-séquence ; dans le cas de l'exploration en largeur d'abord, on est régulièrement amené à passer d'un nœud à un autre situé sur le même niveau mais dont les pères sont différents : dans ce cas, il faut modifier les deux derniers jobs de la séquence partielle ; enfin, si l'on sépare à chaque étape le nœud de plus petite borne, il est possible que les séquences partielles soient complètement différentes, non seulement au niveau du nombre de jobs, mais aussi au niveau de l'ordre entre les jobs.

VI- Améliorations de la PSE

1) Calcul des bornes inférieures

Nous présentons ici différentes bornes inférieures susceptibles d'être utilisées dans l'algorithme de la PSE. Avant toute chose, il convient de rappeler que deux critères peuvent se révéler importants pour le calcul d'une borne inférieure :

- la qualité de l'évaluation : une borne est d'autant meilleure que la valeur qu'elle fournit est proche de la valeur réelle de la meilleure solution, dans le sous-arbre
- la complexité, c'est-à-dire le nombre d'opérations effectuées pour évaluer le nœud : une borne est d'autant meilleure qu'elle fournit un résultat rapidement : ceci est d'autant plus vrai qu'on est amené à visiter un grand nombre de nœuds

La principale difficulté vient du fait que ces deux critères sont en général opposés : les bornes inférieures de qualité sont souvent sophistiquées en termes de nombre d'opérations, alors que les plus rapides fournissent des évaluations assez peu précises.

La borne citée dans le paragraphe précédent est la plus immédiate : nous rappelons qu'elle consiste à construire l'ordonnancement partiel correspondant à la sous-séquence σ du nœud puis à ajouter pour toutes les machines à la date de fin du dernier job de σ la somme des durées des tâches des jobs non encore placés et à conserver le maximum :
si longueur(σ) = i, $LB = \max \{ c_{(\sigma(i),\, k)} + \Sigma_{j \in J \setminus \sigma} p_{(j,k)} / k \in M \}$
Cette borne est extrêmement facile à calculer mais fournit des évaluations de mauvaise qualité : en effet, elle ne tient compte que des contraintes de précédence sur les machines qui conduisent à ce que des tâches devant être exécutées sur la même machine ne se déroulent pas simultanément ; les contraintes de précédence entre opérations d'un même job et les contraintes d'attente maximale ne sont pas prises en compte.

La seconde borne programmée cherche à remédier à ce problème : elle correspond à une borne tournée machine et ne se distingue de la première que par le fait qu'on considère le job placé en dernière position : pour tous les jobs j non encore placés, on procède comme suit : on complète l'ordonnancement partiel en ajoutant pour toutes les machines à la date de fin la somme des durées des tâches non encore placées sur ces machines, sauf celle qui appartient au job j. On obtient alors de nouvelles dates de fin. On cale alors au plus tôt le job j par rapport à ces dates de fin, en appliquant une boucle de l'algorithme de calcul des dates. On dispose ainsi d'une borne inférieure pour tous les ordonnancements dont la séquence commence par σ et se termine par j. Si l'on prend le minimum de ces bornes inférieures pour tous les jobs non encore placés, on obtient une borne inférieure pour tous les ordonnancements dont la séquence commence par σ.

La troisième borne programmée cherche à affiner l'évaluation de la précédente sur les deux premières machines : au lieu de compléter l'ordonnancement partiel par la somme des durées des tâches non encore placées sur les machines, elle effectue une relaxation de la contrainte d'attente sur les jobs non encore placés entre la machine 1 et la machine 2, et applique la règle de Johnson (Johnson (1954)). Nous rappelons ici brièvement son principe :

On considère un problème de flowshop à 2 machines, sans contrainte supplémentaire (seules les contraintes de précédence sur les machines et au sein des jobs sont prises en compte). Alors dans un ordonnancement optimal, le job i précède le job j si

$\min (p_{(i,1)}, p_{(j,2)}) \leq \min (p_{(i,2)}, p_{(j,1)})$

Cette règle conduit directement à un algorithme polynomial qui détermine un ordonnancement optimal pour les problèmes de flowshop à 2 machines de base : on répartit les jobs à ordonnancer dans deux groupes : $U = \{ j \in J / p_{(i,1)} < p_{(i,2)} \}$ et $V = J \setminus U$. On classe alors les jobs de U dans l'ordre croissant de leur durée sur la première machine, puis ceux de V dans l'ordre décroissant de leur durée sur la seconde machine. On concatène les deux listes de jobs trouvés et on obtient une séquence optimale.

Notons tout de même qu'il est possible de combiner sur les deux premières machines l'ordonnancement partiel déjà calculé et l'ordonnancement selon la règle de Johnson pour les travaux non encore placés : supposons que le dernier job de la séquence σ soit j. Les dates de fin de l'ordonnancement partiel sur les 2 premières machines sont $c_{(j,\, 1)}$ et $c_{(j,\, 2)}$, avec $c_{(j,\, 2)} \geq c_{(j,\, 1)}$. On se place alors à la date $c_{(j,\, 1)}$, que l'on considère comme nouvelle origine des temps. L'ordonnancement partiel est assimilé à un job unique dont la durée de traitement sur la première machine vaut 0 et celle sur la deuxième machine est $c_{(j,\, 2)} - c_{(j,\, 1)}$. D'après la règle de Johnson, ce job artificiel peut être placé en première position dans la séquence optimale, puisque sa durée sur la première machine est nulle.

On procède alors de la même manière qu'avec la borne précédente : pour chaque job non encore placé j, on complète l'ordonnancement partiel sur les deux premières machines par celui trouvé grâce à la règle de Johnson appliquée aux autres jobs non placés ; sur les

autres machines, on complète l'ordonnancement partiel par la somme des durées des opérations de ces jobs. Ceci nous conduit à de nouvelles dates de fin. On cale alors le job j au plus tôt par rapport à ces dates de fin. Là encore, nous disposons d'une borne inférieure pour les ordonnancements commençant par σ et se terminant par j. Le minimum de ces bornes calculées sur l'ensemble des jobs non placés ($\in J \setminus \sigma$) nous fournit la borne inférieure souhaitée. Comme on peut le constater, cette borne ne se distingue de la précédente que par une meilleure estimation de la durée sur la seconde machine : en effet, l'ordonnancement au plus tôt respectant la règle de Johnson est sans attente sur la première machine, donc sa durée sur cette première machine est égale à la somme des durées des tâches, estimation que l'on utilise pour la borne précédente.

Il est possible d'étendre ce principe à l'ensemble des couples de machines successives : ainsi, on applique successivement le calcul précédent aux machines 1 et 2 puis 2 et 3 et ainsi de suite jusqu'aux machines m-1 et m. On dispose alors de m-1 bornes inférieures ; la meilleure estimation est donc obtenue en conservant la plus grande de ces valeurs.

Pour vérifier l'efficacité de ces bornes, nous avons comparé les résultats qu'on obtenait avec les PSE dans lesquelles elles étaient intégrées : deux critères sont à prendre en compte : le nombre de nœuds visités par l'algorithme, qui représente la qualité de la borne en termes d'évaluation, et le temps de calcul, qui rend compte de la complexité de la borne.

2) Cheminement dans l'arborescence

De la même manière que les types de bornes inférieures et supérieures employés, le cheminement dans l'arborescence, c'est-à-dire l'ordre dans lequel on visite les nœuds, peut influer de manière non négligeable sur les performances de la PSE. Avant de revenir sur les différentes versions proposées, il convient d'insister sur une hypothèse de travail qui joue ici un rôle fondamental : on suppose que la meilleure solution contenue dans un sous-arbre est d'autant meilleure que la valeur du nœud à l'origine du sous-arbre est plus faible. Cette hypothèse, bien qu'assez naturelle, n'est pas rigoureusement exacte. Rien ne garantit ce résultat, et d'ailleurs, il est possible de rencontrer des cas où elle se révèle fausse. En outre, si cette hypothèse était toujours vérifiée, il suffirait d'explorer à chaque niveau le (ou éventuellement les) nœud pour lequel la borne inférieure est la plus petite. Pour être plus rigoureux, nous pouvons reformuler notre hypothèse de la manière suivante : la probabilité qu'un sous-arbre contienne la solution optimale est d'autant plus grande que la valeur du nœud à l'origine de ce sous-arbre est plus faible.

La première version de la PSE programmée était fondée sur un principe d'exploration en profondeur d'abord très simple : à chaque nouveau nœud qu'on visite, on génère l'ensemble de ses fils, on évalue chacun d'eux à l'aide de la borne inférieure choisie puis on passe au meilleur fils, si sa valeur est inférieure à la borne supérieure. Lorsqu'on a entièrement exploré le sous-arbre correspondant (de manière exhaustive ou en ayant coupé certaines branches), on remonte au nœud d'origine, et on passe au premier nœud dont la valeur est inférieure à la borne supérieure, en suivant l'ordre de génération des fils : rappelons que cet ordre correspond à l'ordre croissant des numéros de jobs placés en dernière position dans la séquence partielle. Cette démarche peut nous conduire à séparer un nœud dont la valeur est bien plus grande que celle d'un de ses frères, situés plus «à droite » dans l'arbre, c'est-à-dire plus loin dans l'ordre de génération. Or, d'après notre hypothèse, un telle situation est peu intéressante. Une raison supplémentaire pour modifier le cheminement initial provient du fait qu'on peut rencontrer plusieurs nœuds dont la valeur est minimale ; dans un tel cas,

seul le premier est exploré dans un premier temps, les autres sont considérés comme n'importe quel autre nœud. Pour remédier à cet obstacle, plusieurs alternatives se présentent :

- explorer successivement les fils dans l'ordre croissant de leur évaluation. Cette technique serait la plus satisfaisante compte tenu de l'hypothèse que nous avons faite. Cependant, elle engendrerait un coût non négligeable au niveau du temps de calcul : les procédures les plus efficaces pour les tris complets ont une complexité en O(n.log(n)) où n désigne le nombre d'éléments à trier, alors que la simple recherche du plus petit élément est en O(n). Il est également possible de fixer au départ un nombre d'éléments $r \leq n$ à trier : ainsi, on limite quelque peu le nombre d'opérations (si la valeur d'un nœud est supérieure au r premières valeurs, on se contente de la placer en queue).
- Répartir les nœuds issus d'un père en 3 catégories : les nœuds dont la valeur est minimale, les nœuds dont la valeur est inférieure à une certaine limite v_l et les nœuds dont la valeur est supérieure à cette limite ; v_l peut par exemple être définie à partir d'une fraction λ représentant une certaine position entre la valeur minimale v_{min} et la valeur maximale v_{max} : $v_l = v_{min} + \lambda(v_{max} - v_{min})$, avec $0 \leq \lambda \leq 1$.

  Si $\lambda = 0$, $v_l = v_{min}$ et la seconde catégorie est vide ; si $\lambda = 1$, $v_l = v_{max}$ et la troisième catégorie est vide.

  On visite alors les nœuds de la première catégorie puis ceux de la deuxième puis ceux de la troisième. La procédure de constitution des groupes a une complexité en O(n) : il suffit de lire une première fois l'ensemble des valeurs pour repérer v_{max} et v_{min}, de calculer v_l puis de parcourir une seconde fois l'ensemble des nœuds pour les répartir dans les 3 groupes.

Pour réduire davantage le temps de calcul de la PSE, deux améliorations supplémentaires ont été testées. La première consiste à fusionner les deux derniers niveaux de l'arbre : lorsqu'on se trouve au niveau n-2, chaque nœud correspond à une séquence partielle de n-2 jobs, et possède donc 2 fils, chacun possédant un fils unique (une feuille de l'arbre) fournissant une solution. Plutôt que générer ces 2 fils, les évaluer en calculant les bornes inférieures correspondantes et les explorer en respectant le cheminement défini, on choisit de construire directement les deux solutions qui en sont issues et de les comparer avec la meilleure solution trouvée jusque là. Bien que cette méthode nous oblige à bâtir et à tester les deux solutions, elle permet d'éviter le calcul des bornes qui à ce niveau conduit dans la plupart des cas à construire les deux solutions malgré tout.

La deuxième amélioration consiste à fusionner les 2 ou 3 premiers niveaux : il est en effet possible de générer directement l'ensemble des $n \times (n-1)$ séquences partielles à 2 éléments ou des $n \times (n-1) \times (n-2)$ séquences partielles à 3 éléments. On peut alors introduire des tests de dominance : si il existe deux nœuds N_1 et N_2 correspondant à des séquences possédant les mêmes éléments (dans un ordre différent), et si pour toutes les machines, la date de fin de l'ordonnancement partiel de N_1 est inférieure (au sens large) à la date de fin de l'ordonnancement partiel de N_2, alors la meilleure solution contenue dans le sous-arbre issu de N_1 est nécessairement meilleure (au sens large) que celle contenue dans le sous-arbre issu de N_2.

3) Heuristiques utilisées pour le calcul des bornes supérieures

Comme nous l'avons déjà expliqué, l'intérêt de ces heuristiques est de fournir à la PSE une première borne supérieure de la valeur optimale. L'objectif étant d'être le plus proche possible de cet optimum, on peut accepter de ces heuristiques qu'elles soient un peu coûteuses

en temps de calcul, d'autant plus qu'on ne les exécute qu'une fois, juste en aval de la PSE. Nous avons élaboré 5 heuristiques que nous présentons maintenant.

Pour la première méthode, on ajoute à chaque étape en queue de la séquence partielle le job non encore placé qui minimise le Cmax de l'ordonnancement partiel. Si σ_i désigne la séquence des i premiers jobs ($i < n$), on définit σ_{i+1} comme : $\sigma_{i+1} = \sigma_i + j$, où j est tel que $Cmax(\sigma_i + j) = \min (Cmax(\sigma_i + l) / l \in J \setminus \sigma_i)$. S'il existe plusieurs jobs vérifiant cette égalité, on choisit parmi ceux-ci le job qui minimise la durée d'inactivité sur l'ensemble des machines : pour cela, on calcule uniquement l'inactivité supplémentaire provoquée par le placement du job j à l'étape i + 1, en faisant la somme sur l'ensemble des machines de l'écart entre la fin du job placé en $i^{ième}$ position ($\sigma_i(i)$) et le début du job j : $\Sigma_{t \in M} (t_{(j,t)} - c_{(\sigma i(i),t)})$.

La seconde heuristique consiste à ajouter en queue de la séquence partielle le job non encore placé qui minimise les temps d'inactivité sur les machines : ceux-ci sont définis comme la somme des durées qui séparent la date de fin d'une opération de la date de début de l'opération suivante sur la même machine : si σ désigne une séquence partielle, que L = longueur(σ), alors le temps d'inactivité total est défini par :

$TI = \Sigma_{k \in M} \Sigma_{j<L} (t_{(\sigma(j+1),k)} - c_{(\sigma(j),k)})$

De même que précédemment, on peut se contenter de calculer à chaque étape l'inactivité supplémentaire engendrée par le job placé en dernier. Si σ_i désigne la séquence des i premiers jobs ($i < n$), on définit σ_{i+1} comme : $\sigma_{i+1} = \sigma_i + j$, où j est tel que :
$\Sigma_{t \in M} (t_{(j,t)} - c_{(\sigma i(i),t)}) = \min(\Sigma_{t \in M} (t_{(l,t)} - c_{(\sigma i(i),t)}) / l \in J \setminus \sigma_i)$. S'il existe plusieurs jobs vérifiant cette égalité, on choisit parmi ceux-ci le job qui minimise le Cmax.

La troisième heuristique est une adaptation de la méthode NEH en présence de contraintes d'écarts de temps maximaux (Nawaz (1983)). On classe les jobs dans l'ordre décroissant de leur durée de traitement totale. Puis on construit progressivement la séquence correspondant à la solution en insérant à chaque étape le job suivant dans la liste à la meilleure place dans la séquence, de manière à minimiser le Cmax.

Enfin, la quatrième heuristique est une adaptation de la méthode CDS aux conditions du problème que nous étudions (Campbell (1970)). Cette méthode fait elle aussi appel à la règle de Johnson déjà présentée. On applique celle-ci à un ensemble de problèmes à 2 machines générés de la manière suivante : le nombre de jobs est le même que celui du problème initial (n = card(J)) et pour chacun d'eux, la durée de traitement sur la première machine est la somme des durées des opérations du job initial de la machine 1 à la machine k_1 ; la durée de traitement sur la seconde machine est la somme des durées des opérations du job initial de la machine k_2 à la machine m. En faisant varier k_1 entre 1 et m-1 et k_2 entre 2 et m, on obtient ainsi $(m-1)^2$ problèmes que l'on résout avec l'algorithme basé sur la règle de Johnson et présenté dans la partie précédente. Finalement, on dispose de $(m-1)^2$ séquences (non nécessairement distinctes) qui correspondent à autant d'ordonnancements pour le problème initial. En conservant le meilleur d'entre eux, on dispose d'une solution dont la valeur est une borne supérieure pour la valeur optimale.

Pour comparer l'intérêt de ces différentes méthodes, il est de nouveau possible de prendre en compte 2 paramètres : la qualité du résultat, c'est-à-dire la différence par rapport à l'optimum, et le temps de calcul nécessaire pour atteindre ce résultat. Néanmoins, comme nous l'avons expliqué plus haut, ce temps de calcul est secondaire étant donné qu'on est amené à exécuter l'heuristique une seule fois, en préambule de la PSE. Au contraire, plus le

résultat fourni sera proche de la valeur optimale, plus le nombre de nœuds visités sera faible et plus la PSE donnera la meilleure solution rapidement.

Nous avons donc lancé ces différentes méthodes approchées sur des instances et comparé les valeurs qu'elles renvoyaient ainsi que le temps de calcul : la procédure NEH / mod conduit pratiquement toujours à des meilleurs résultats que les autres, et son temps d'exécution est tout à fait acceptable. Ceci nous amène tout naturellement à l'utiliser comme heuristique préalable à la PSE.

VII- Méthodes approchées dérivées

Malgré les améliorations apportées à la PSE, celle-ci reste très coûteuse en temps de calcul dans certains cas. Pour pouvoir résoudre efficacement des problèmes de grande taille, susceptibles d'être rencontrés dans le monde industriel, nous avons élaboré des méthodes de résolution approchée dérivées de la PSE. Nous les décrivons dans cette partie.

1) L'approche ε

L'approche ε est utilisée pour surmonter un obstacle récurrent lors de l'exécution des PSE : si l'on met en place un compteur de nœuds et qu'on étudie le numéro du nœud correspondant à la solution optimale par rapport au nombre total de nœuds visités, on remarque que dans la majorité des cas, ce numéro est relativement petit. Ceci signifie que la PSE atteint l'optimum relativement rapidement ; le grand nombre d'opérations provient du fait qu'on prouve que cette solution est la meilleure, c'est-à-dire que les autres nœuds ne conduisent pas à de meilleures solutions. Or, dans les niveaux inférieurs (proches de la racine), l'évaluation des nœuds n'est pas suffisamment précise pour garantir que ce nœud ne contient pas l'optimum.

Le principe de cette méthode est le suivant : on diminue la borne supérieure de manière à élaguer davantage de nœuds. Chaque fois que l'on trouve une nouvelle meilleure solution de valeur V, on utilise comme borne supérieure $V/(1 + \varepsilon)$.

Avec cette heuristique, nous disposons d'une majoration de la différence entre la solution fournie et l'optimum V* : en effet, si la solution S trouvée a une valeur V, tous les nœuds non séparés ont une évaluation supérieure à $V/(1 + \varepsilon)$. Par conséquent, les solutions issues de ces nœuds ont des valeurs supérieures à $V/(1 + \varepsilon)$.

D'où : $V/(1 + \varepsilon) \leq V^* \leq V$.

Donc : $(V - V^*) / V^* \leq \varepsilon$.

L'écart relatif entre la solution trouvée et la solution optimale est inférieur à ε.

Nous ne décrivons pas ici l'algorithme correspondant, puisque celui-ci diffère de la fonction PSE uniquement au niveau de la borne supérieure : il suffit de remplacer meilleure_valeur par meilleure_valeur / (1 + ε) lors de l'exploration de l'arborescence.

2) Le beam search

Cette heuristique consiste à ne conserver à chaque niveau de l'arborescence, qu'un nombre limité de nœuds. Pour cela, on effectue une exploration en largeur d'abord, en gardant uniquement les meilleurs nœuds en termes d'évaluation.

A un niveau donné, on génère l'ensemble des fils issus des nœuds conservés à l'étape précédente et on sélectionne de nouveau les meilleurs d'entre eux pour poursuivre la recherche.

La fonction d'évaluation utilisée ici est la même que celle qui sert à calculer les bornes inférieures. Par ailleurs, le nombre de nœuds conservés peut être fixe ou varier selon le niveau de l'arbre.

Bien qu'il s'agisse d'une « PSE tronquée », le type d'exploration (en largeur d'abord) conduit à un algorithme relativement différent de celui présenté pour la PSE, pour laquelle nous avons privilégié une exploration en profondeur d'abord. En particulier, l'espace mémoire utilisé ici est beaucoup plus important puisqu'il faut conserver à chaque niveau un nombre donné de nœuds, chacun correspondant à une séquence partielle de jobs.

VIII- Génération de jeux d'essais / benchmarks

Afin d'estimer la qualité des algorithmes, de les comparer entre eux, il est indispensable de les tester sur des instances particulières. En plus de cet objectif, ces jeux d'essais ou benchmarks peuvent nous conduire à distinguer des classes de données, qui conviennent bien à certains algorithmes. Enfin, pour respecter l'objectif que nous nous sommes fixés en termes d'applications pratiques, il convient de réfléchir aux différents types de problèmes rencontrés dans le milieu industriel et à leurs traductions au niveau des paramètres.
On peut ainsi construire des jeux d'essais traduisant différentes situations :

- si les durées sont homogènes, avec une faible dispersion et des machines toutes goulots, on est en présence d'une chaîne de production équilibrée récente
- si 2 grandes familles de produits cohabitent, avec un groupe de produits vérifiant les conditions précédentes et un groupe de produits pour lesquels les durées sont beaucoup plus dispersées, on est face à une chaîne de production équilibrée qui commence à vieillir : la première famille de produits est formée des produits lancés à l'origine, la seconde est constituée de produits nouveaux, pour lesquels les machines ne sont pas forcément parfaitement adaptées
- si certains produits ont des durées extrêmement dispersées, cela peut également traduire le cas d'une chaîne de production possédant certaines contraintes techniques empêchant l'équilibrage
- la présence d'écarts de temps à respecter uniquement entre certaines machines permet de rendre compte d'opérations critiques au cours du processus de production

IX- Résultats expérimentaux

Pour évaluer l'intérêt des méthodes élaborées au cours de ce projet de recherche, un grand nombre de jeux d'essai ont été générés, selon les critères décrits dans la section précédente. Parmi ceux-ci, 30 ont été conservés pour tester les programmes. Il convient de préciser que, pour des raisons de simplicité, les algorithmes ont été implémentés en Visual Basic. Plutôt que de considérer les temps de calcul dans l'absolu, il est intéressant de comparer ces valeurs entre elles. Nous avons également étudié le gain apporté au niveau du nombre de nœuds explorés (nœuds auxquels on applique la borne inférieure).

Dans le cas de durées homogènes, il est possible de résoudre en un temps raisonnable (moins de 2 heures) des problèmes comptant jusqu'à 15 travaux. Mais lorsque l'on se trouve en présence de machines dont les charges sont très hétérogènes, certains problèmes à 100 travaux peuvent être résolus en quelques secondes.

La première série de comparaisons concerne les heuristiques : sur les 30 instances, la méthode dérivée de NEH fournit dans 28 cas une meilleure solution que les autres heuristiques. La valeur obtenue diffère de l'optimum de 1,2 % en moyenne, et ne dépasse

dans aucun cas 3,6 %. Quoi qu'il en soit, il est toujours possible d'appliquer toutes les heuristiques, celles-ci ne demandant que quelques secondes d'exécution, même sur des jeux d'essai de grande taille.

Au niveau des bornes inférieures, la borne « tournée machine » conduit aux temps de calcul les plus faibles : la borne dérivée de la règle de Johnson demande en moyenne 3 fois plus de temps, alors que celle qui utilise l'extension de cette règle en demande 19 fois plus. En réalité, ces deux méthodes réduisent le nombre de nœuds explorés (de 3,7 % pour la seconde borne et de 12,9 % pour la troisième, en moyenne) ; mais ce gain n'est pas suffisant pour compenser les temps dus aux opérations supplémentaires.

L'utilisation des heuristiques pour obtenir une borne supérieure peut se révéler extrêmement efficace : on peut économiser jusqu'à plus de 80 % de temps de calcul, et le gain moyen est de 23 % , que ce soit pour la durée ou pour le nombre de nœuds explorés.

La fusion des deux derniers niveaux, le classement partiel des nœuds et les tests de dominance n'apportent pas d'amélioration significative ; dans certains cas, les performances sont même moins bonnes qu'avec une version simple de la PSE.

Le beam search a été testé en conservant 10, 50 et 90 nœuds par niveau : combiné à l'heuristique NEH, il améliore peu le résultat obtenu par cette dernière. Par contre, l'approche ε présente un certain intérêt : elle permet de s'approcher de la solution optimale autant que l'on souhaite. Néanmoins, si l'on veut atteindre le même niveau de performance que l'heuristique NEH, le temps de calcul est beaucoup plus important.

Ces différents résultats tendent à infirmer l'hypothèse formulée dans les paragraphes précédents : dans de nombreux cas, la meilleure solution contenue dans un sous-arbre n'est pas issue du nœud dont la valeur est la plus faible.

X- Conclusion

Nous avons vu dans cet article comment adapter des méthodes classiques en ordonnancement afin de résoudre des problèmes particuliers, en prenant en compte leurs contraintes spécifiques : le schéma de PSE présenté ici convient bien au contexte de production : flow shop de permutation avec des durées d'attente maximales à ne pas dépasser. Les résultats expérimentaux ont démontré l'intérêt des heuristiques à exécuter avant la PSE pour réduire les temps de calcul. Par contre, l'élaboration de bornes inférieures doit prendre en compte deux aspects opposés : la précision et la simplicité. Nous avons pu constater qu'une trop grande complexité des bornes peut pénaliser l'exécution du programme dans des proportions importantes.

Le principe de résolution adopté dans ce travail de recherche repose sur une méthode exacte. Cependant, il faut être conscient que dans le monde industriel, les situations rencontrées sont souvent beaucoup trop complexes pour être représentées intégralement par des modèles de ce genre. On a donc plutôt recours à des méthodes approchées, qui fournissent des solutions de bonne qualité, que l'on peut adapter à l'environnement en fonction des aléas et des opportunités. A ce titre, l'heuristique NEH et l'approche ε présentent un grand intérêt, la première pour sa précision et sa rapidité d'exécution sur des problèmes de grande taille, la seconde par la garantie de performances qu'elle comporte.

Bibliographie

BAKER K.R. (1990), Scheduling groups of jobs in the two-machine flow shop, Mathematical and Computer Modelling, **13**, n°3, 29-36.

BRUCKER P., HILBIG T., HURINK J. (1999), A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags, Discrete Applied Mathematics, **94**, 77-99.

CAMPBELL H.G., DUDEK R.A. et SMITH M.L. (1970), A heuristic algorithm for the n job, m machine sequencing problem, Management Science, **16**, n°10, 630-637.

CARLIER J. et CHRETIENNE P. (1988), Problèmes d'ordonnancement : modélisation / complexité / algorithmes, Masson, Paris.

GILMORE et GOMORY (1964), Sequencing a one state variable machine : a solvable case of the traveling salesman problem , Operations Research, **12**, n°5, 655-679.

IGNALL E.J. et SCHRAGE L.E. (1965), Application of the branch and bound technique to some flow-shop scheduling problems, Operations Research, **13**, n°13.

JOHNSON S.M. (1954), Optimal two and three-stage production schedules with setup times included , Naval Research Logistics Quarterly, **1**, n°1, 61-68.

LOPEZ P. et ROUBELLAT F. (sous la direction de) (2001), Ordonnancement de la production, Hermès Science Publications, Paris.

NAWAZ M., ENSCORE JR. E.E. et HAM I. (1983), A heuristic algorithm for the m-machine, n-job flow shop sequencing problem , Omega, **11**.

SZWARC W. (1983), Flow shop problems with time lags, Management Science, **29**, n°4, 477-481.

TANAEV V.S., SOTSKOV Y.N., STRUSEVICH V.A. (1994), Scheduling Theory Multi-Stage Systems, Kluwer Academic Publishers.

ANNEXES I

Exemple d'application

On considère une production concernant 4 produits, devant être traités successivement sur 3 machines. Les durées des tâches, ainsi que les attentes maximales sont données dans les tableaux 1 et 2. Supposons que la séquence de passage des produits sur les machines, déterminée préalablement, soit représentée par la séquence π = (2, 3, 1, 4). La figure 1 décrit les étapes successives de construction de l'ordonnancement correspondant, en utilisant la représentation sous forme de diagramme de Gantt.

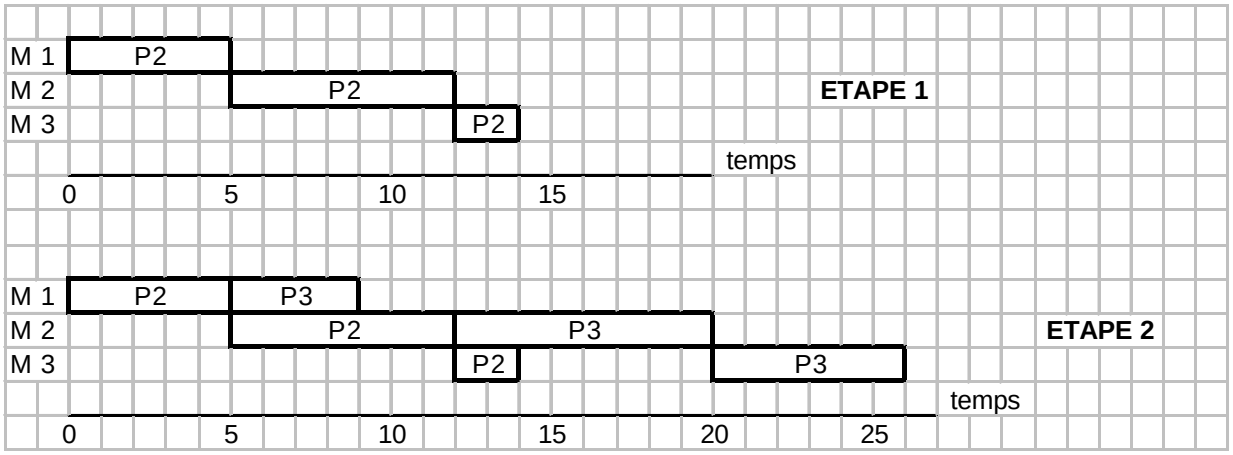
Produit / Machine	M1	M2	M3
P1	3	4	3
P2	5	7	2
P3	4	8	6
P4	6	4	5

Tableau 1
Durées des tâches

Produit / Ecart	M1-M2	M2-M3
P1	2	3
P2	2	2
P3	4	1
P4	2	0

Tableau 2
Attentes maximales autorisées

Lors de l'étape 3, la première opération du job 1 est placée dans un premier temps immédiatement après la fin de la première opération du job 3. Il en est de même sur la seconde machine. Mais cette situation conduit à un encart dépassant la limite autorisée entre les opérations 1 et 2 du job 1 (8 unités de temps au lieu de 2). En décalant à droite la première opération de manière à atteindre la valeur de l'attente maximale autorisée (« technique de l'élastique »), on obtient un ordonnancement admissible, c'est-à-dire respectant les contraintes du problème. Il en est de même à l'étape 4 où l'on est obligé de retarder de 1 unité le job 4 sur la machine 2 (étape 4 bis).



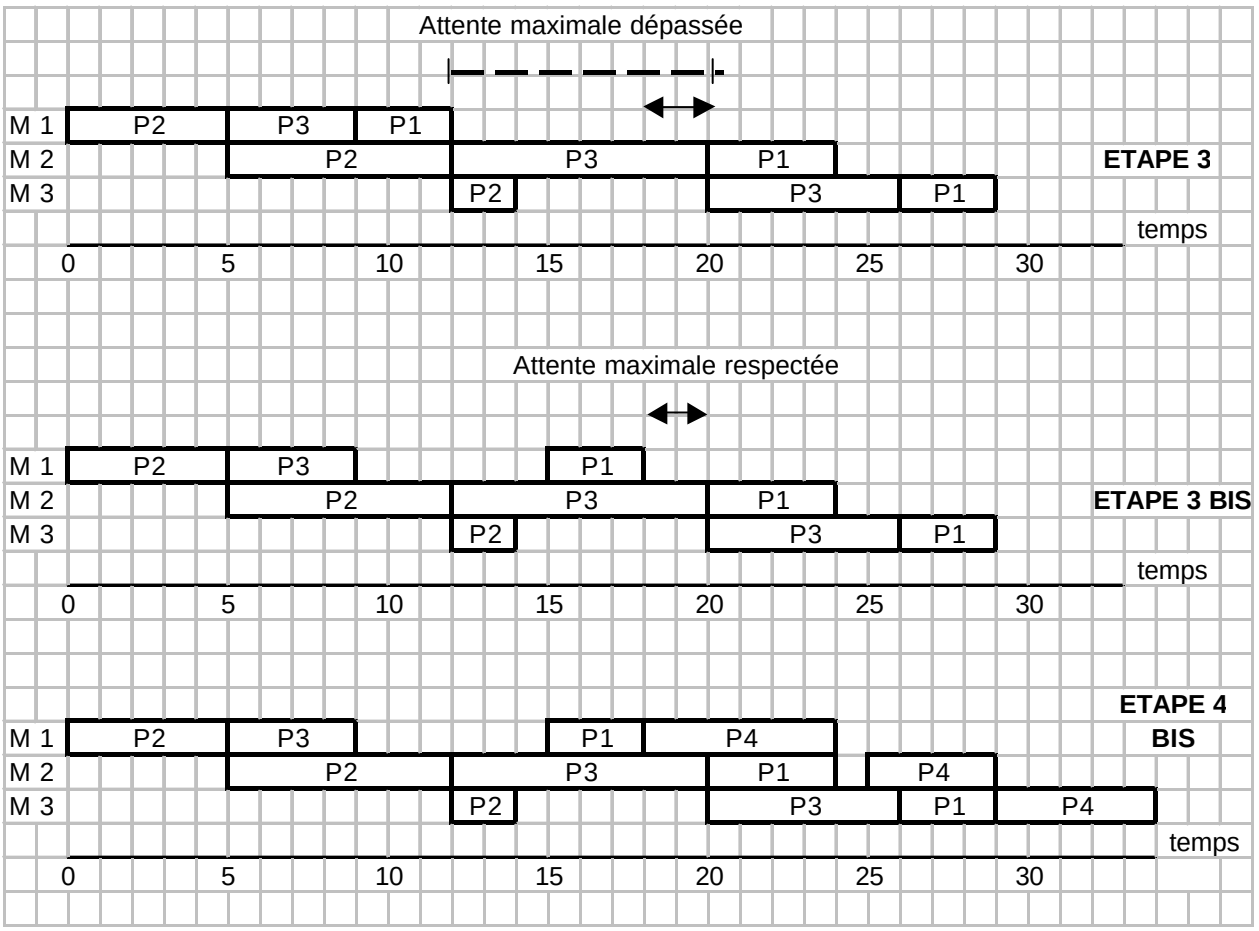


Figure 1
Etapes successives de la construction de l'ordonnancement

Bornes inférieures

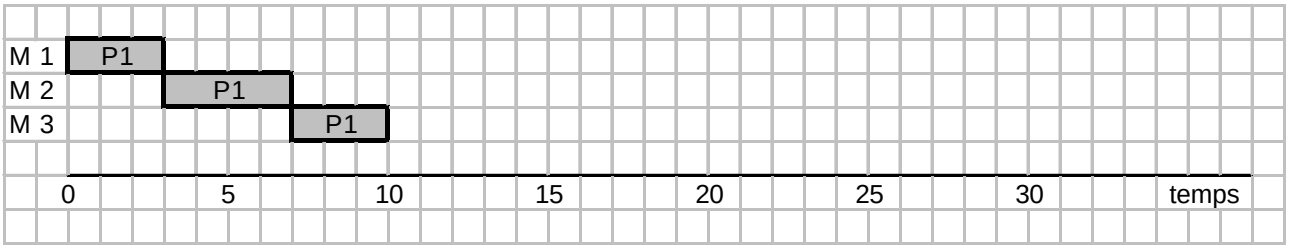


Figure 3
Ordonnancement partiel

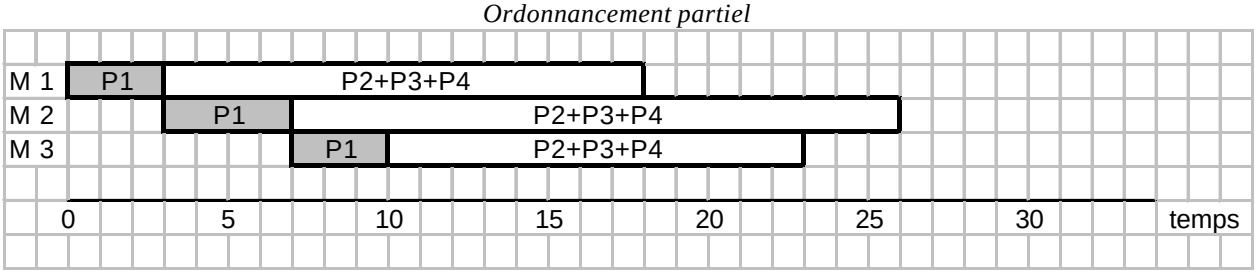


Figure 4
Première borne inférieure programmée

Les figure 3 et 4 illustrent le principe de la première borne, appliquée à la séquence partielle (1) ; les valeurs numériques sont celles des tableaux 1 et 2.

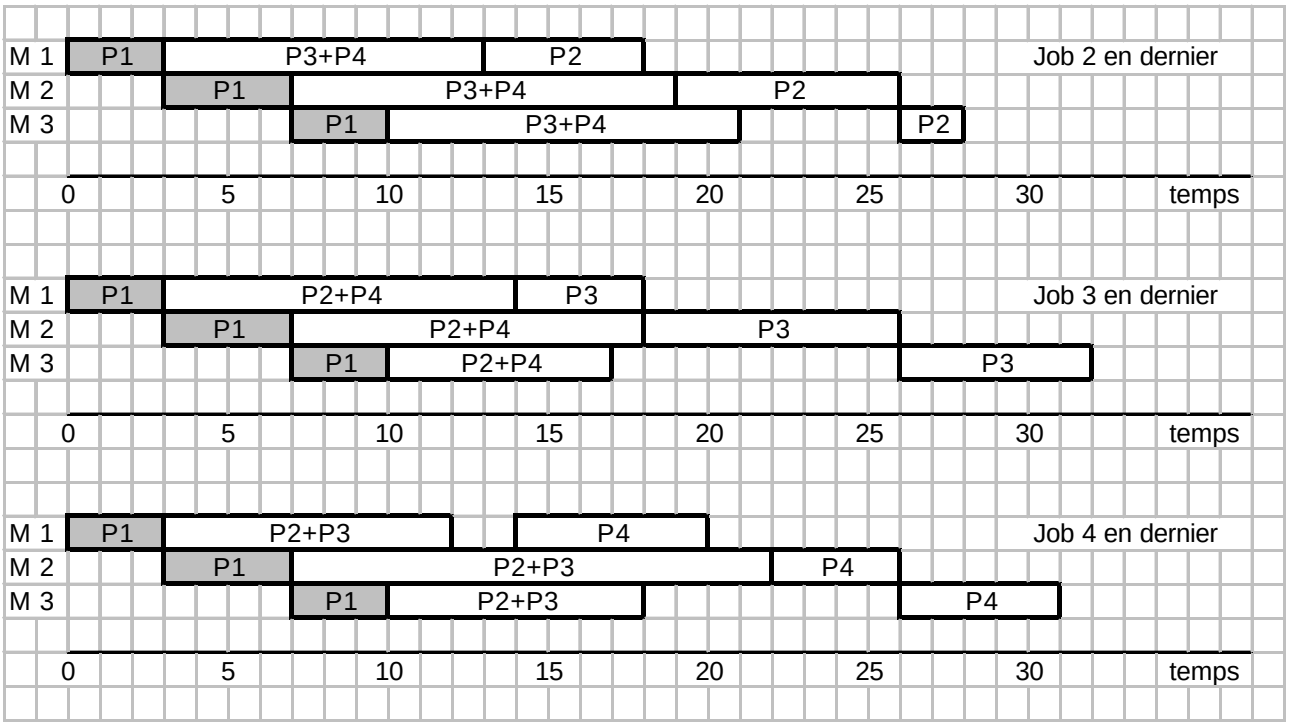
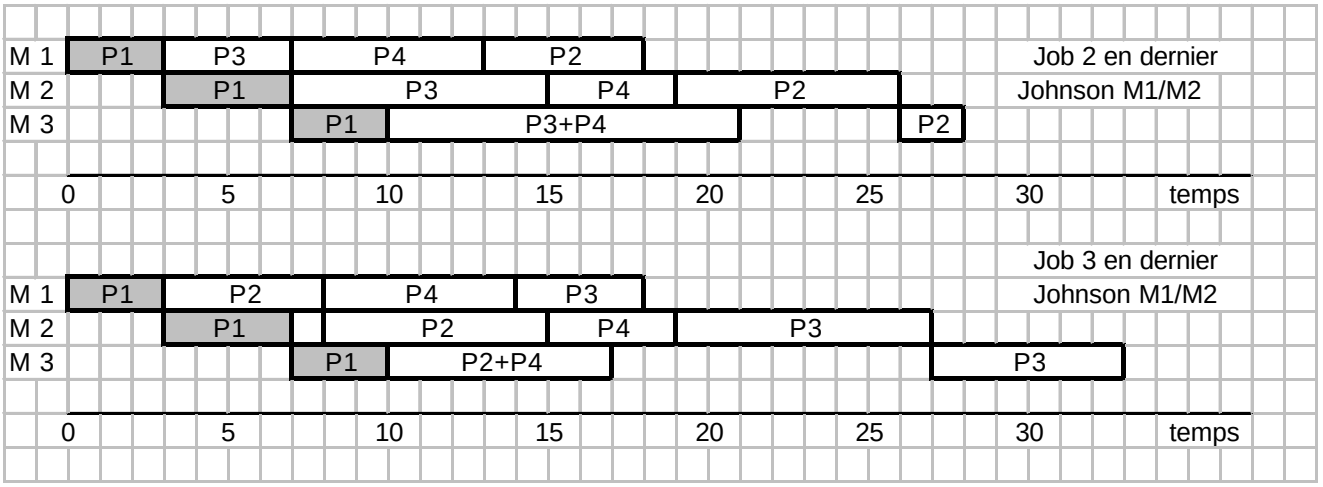


Figure 5
Borne inférieure tournée machine

La figure 5 détaille le fonctionnement de la seconde borne, sur le même exemple que précédemment ; l'ordonnancement partiel est toujours celui donné figure 3. La valeur est cette fois de 28, elle est obtenue en plaçant le job 2 en dernier. On constate sur cet exemple qu'elle est effectivement supérieure à celle fournie par la première borne inférieure.



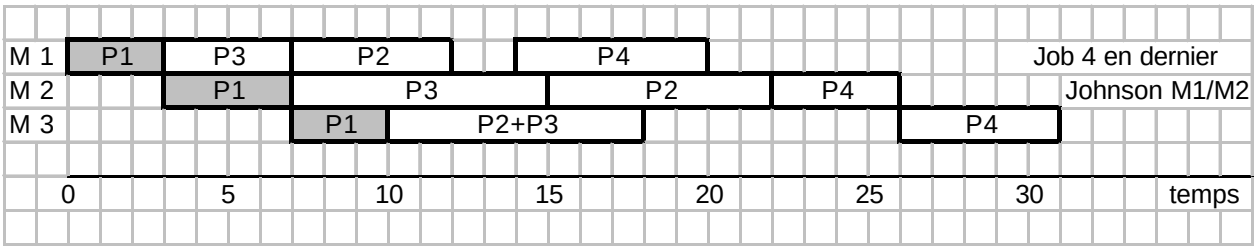


Figure 6
Borne inférieure utilisant la règle de Johnson

On trouve une application de la troisième borne sur la figure 6, qui reprend le même exemple que précédemment : lorsque l'on place le job 2 en dernier, la règle de Johnson appliquée aux jobs 3 et 4 nous fournit l'ordre 3-4 ; de même, si le job 3 (respectivement 4) est placé en dernière position, l'ordre de passage est 2-4 (respectivement 3-2). Comme on peut le constater, la valeur obtenue avec le job 3 en dernier est supérieure à celle calculée avec la seconde borne (33 au lieu de 32). Cependant, étant donné que l'on ne conserve que la valeur minimale, l'estimation du nœud reste la même : 28, valeur obtenue en plaçant le job 2 à la fin.

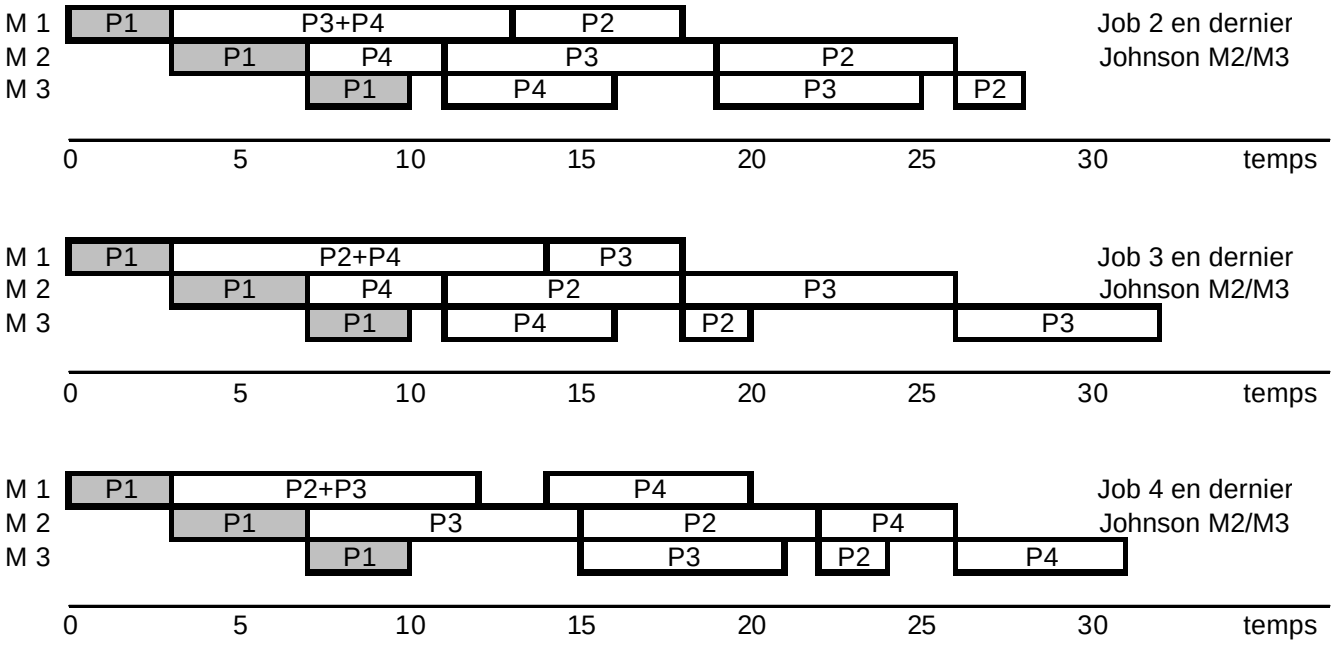
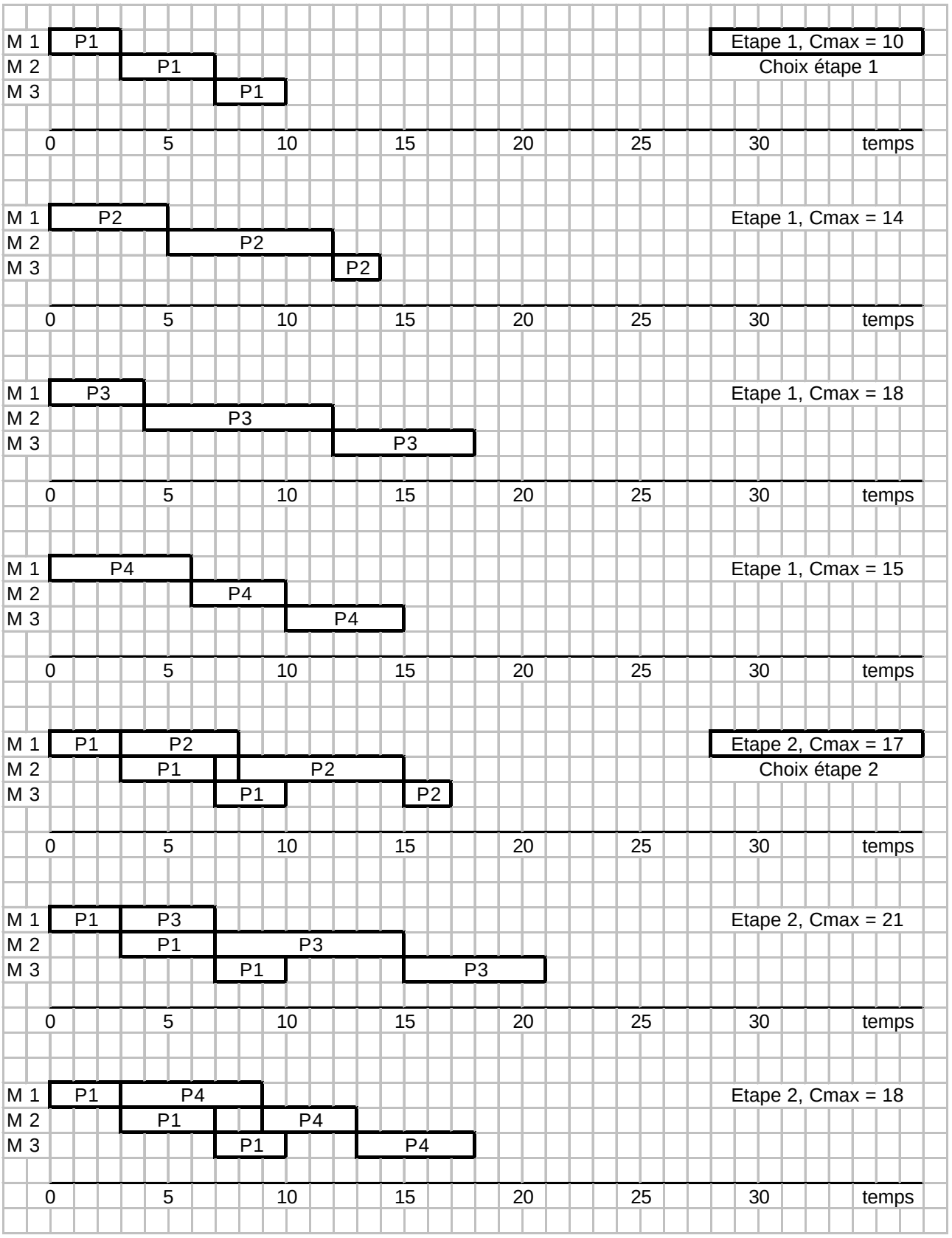


Figure 7
Borne inférieure utilisant la règle de Johson généralisée

La figure 7 nous montre les résultats obtenus avec la dernière borne inférieure, sur le même exemple. En réalité, puisque la figure 6 présente déjà les profils correspondant à l'application de la règle de Johnson sur les machines 1 et 2, on se contente sur la figure 7 de représenter ceux auxquels on aboutit avec la règle de Johnson sur les machines 2 et 3. Ainsi, pour chaque job non placé, on conserve la valeur maximale (28 pour le job 2, 32 pour le job 3 et 31 pour le job 4) : parmi toutes ces valeurs, la plus petite constitue une borne inférieure.
Sur cet exemple, la valeur retenue est issue des machines 1 et 2 et l'application de la règle de Johnson étendue à tous les couples de machines successives n'apporte rien de plus. Néanmoins, on peut vérifier que sur des problèmes de plus grande taille (notamment au niveau du nombre de machines), le gain résultant de l'utilisation de cette borne peut être particulièrement intéressant.

Bornes supérieures



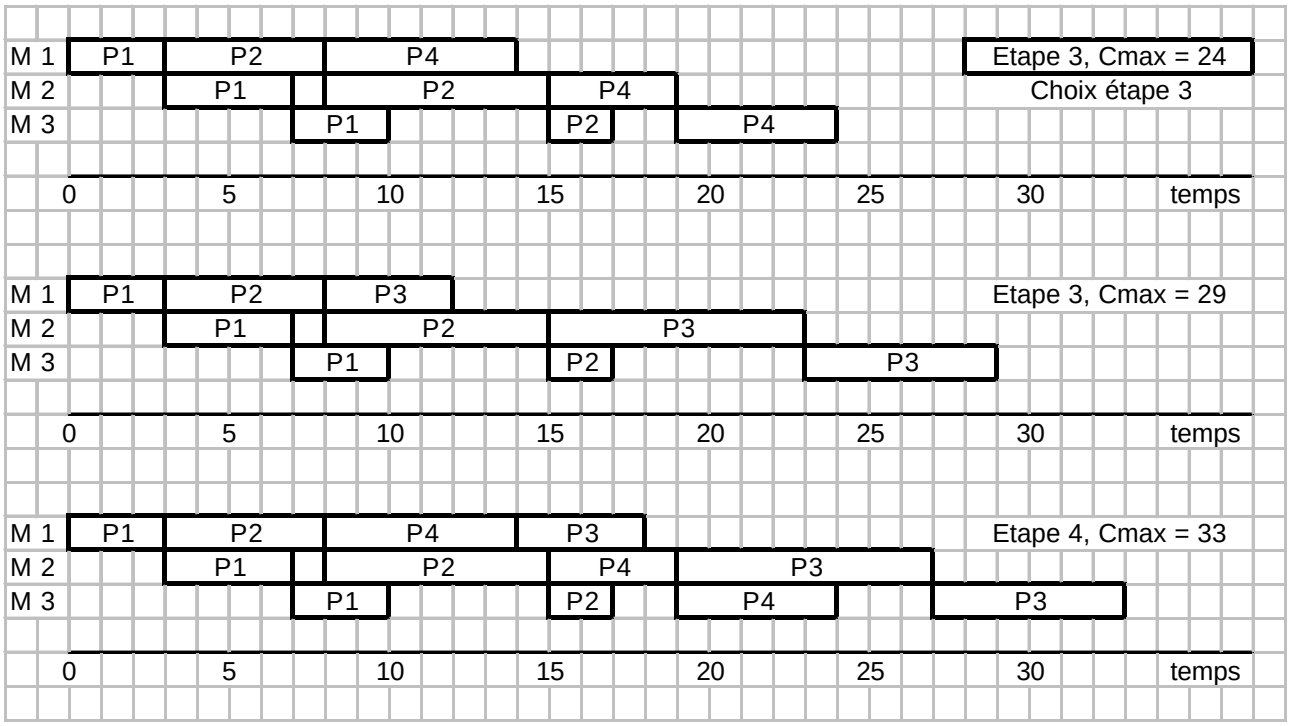
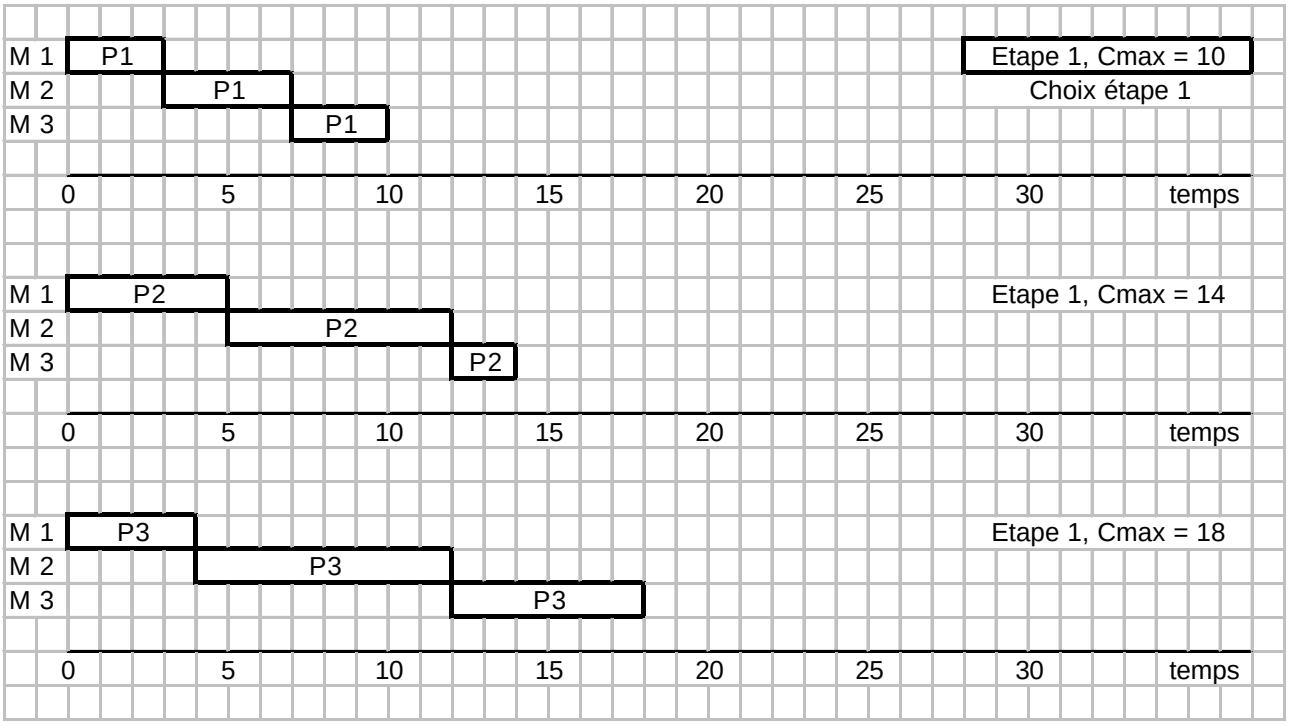


Figure 8
Première heuristique

La figure 8 illustre le déroulement de la première heuristique sur l'exemple numérique (ajouter le travail qui minimise le Cmax). La solution obtenue a une valeur de 33, alors que le meilleur ordonnancement, fourni par la PSE, a une durée de 30.



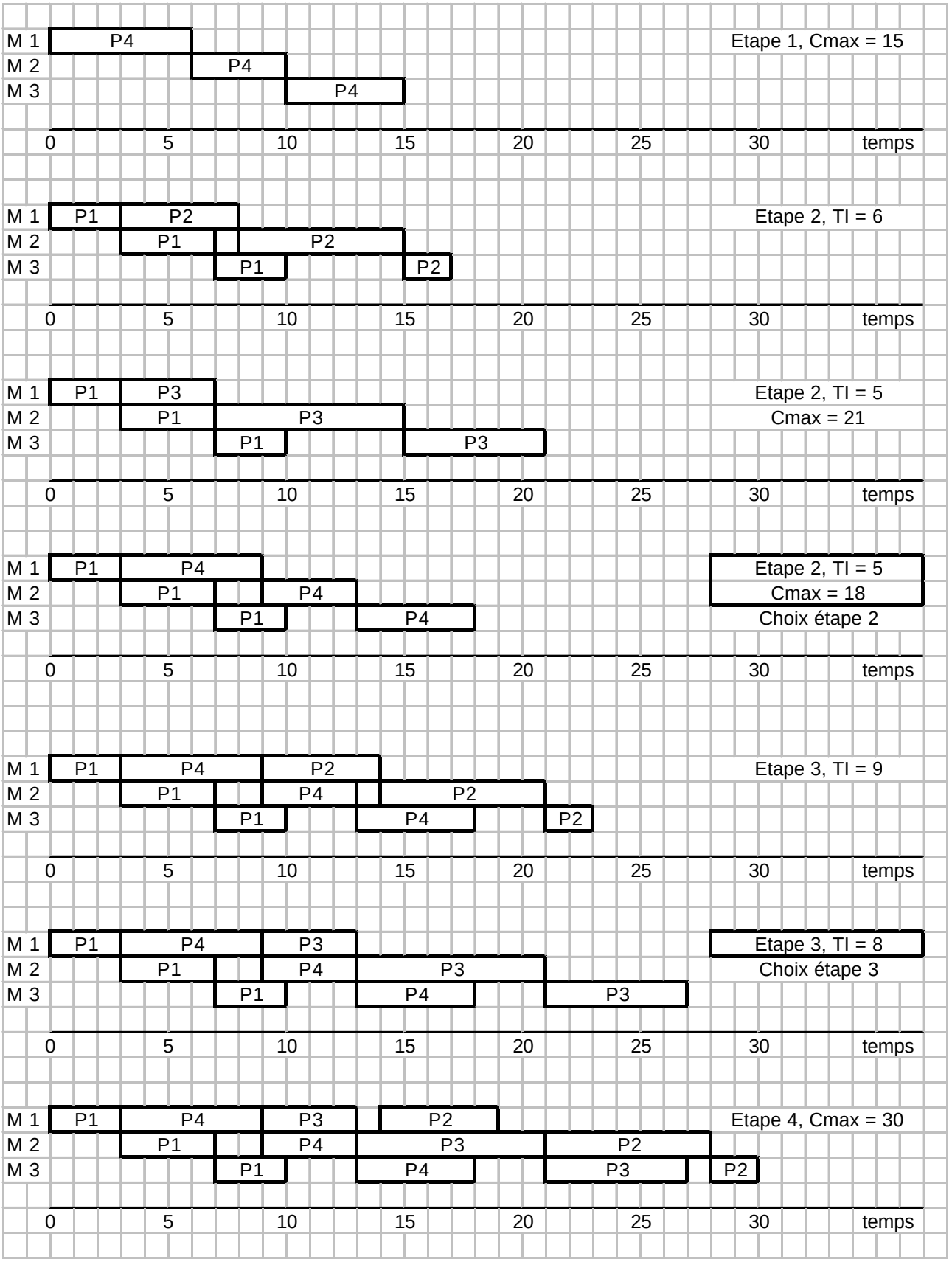
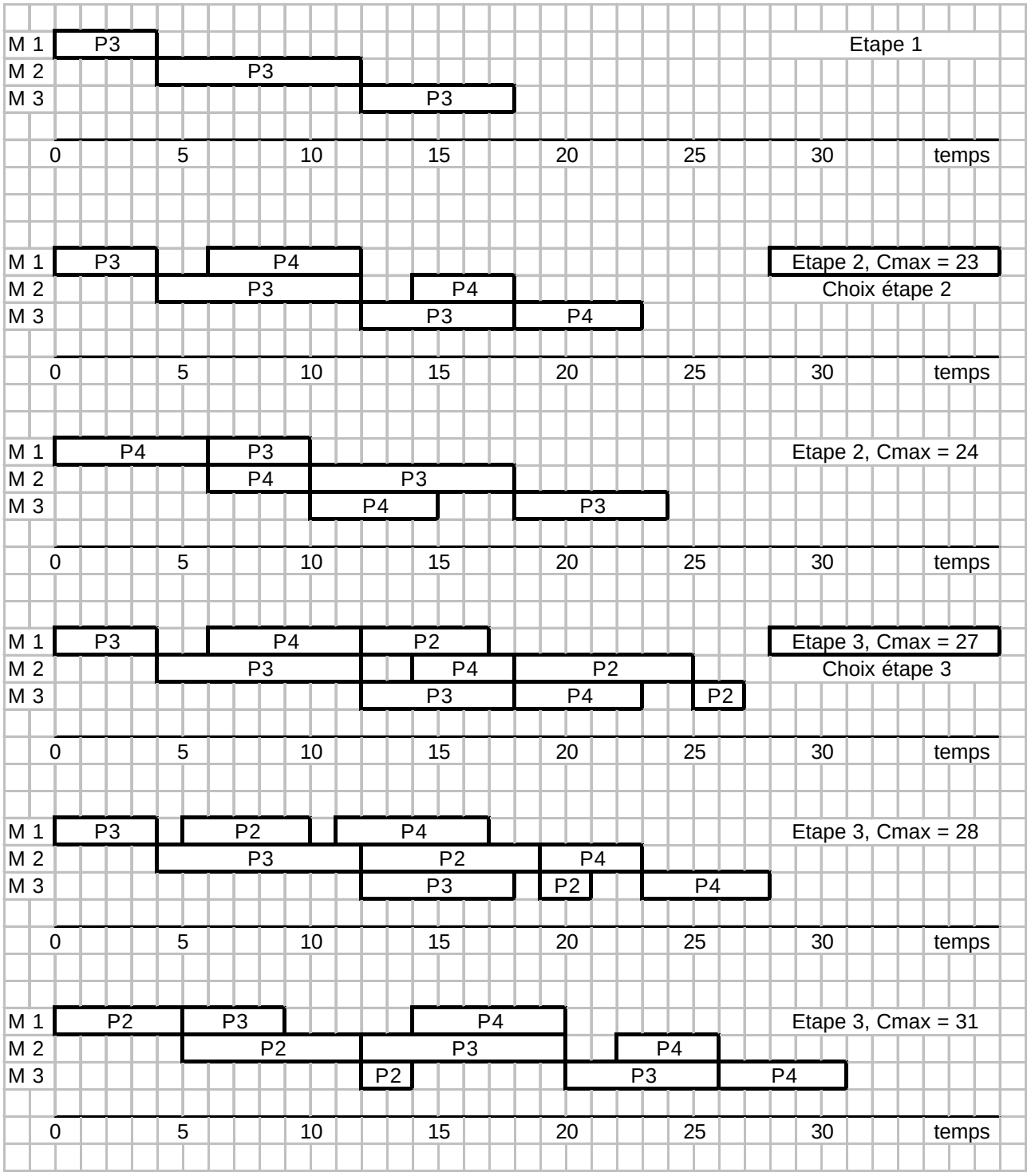


Figure 9
Deuxième heuristique

La figure 9 donne un exemple de déroulement de la seconde heuristique (ajouter le job qui minimise les temps morts et à égalité le Cmax), correspondant à l'exemple numérique. La solution obtenue est ici optimale, puisque sa valeur est égale à celle fournie par la PSE (30).



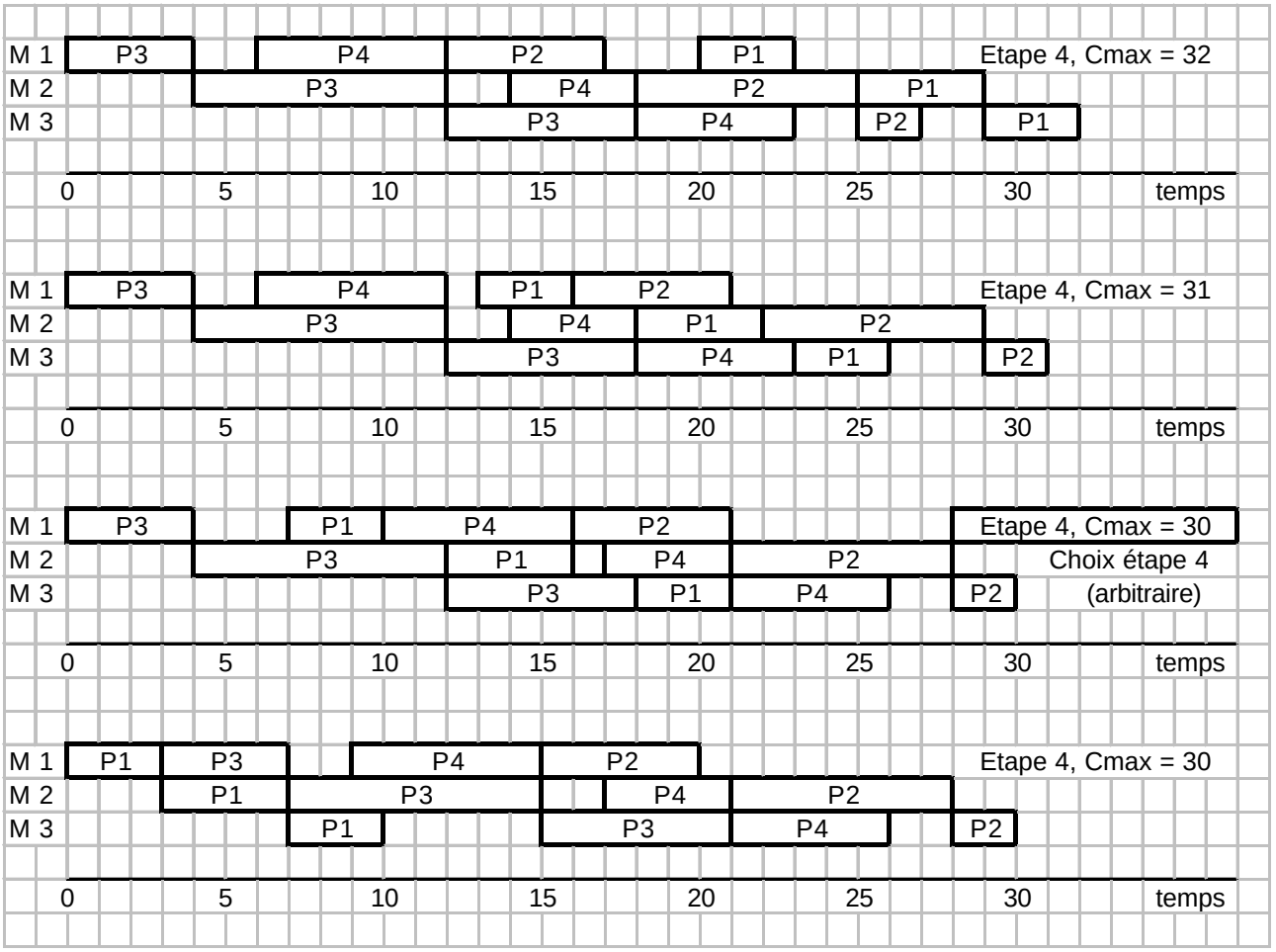


Figure 10
Heuristique NEH modifiée

Le déroulement de la méthode NEH, appliquée à l'exemple numérique, est représenté sur la figure 10 : le classement par ordre décroissant de la durée totale de traitement est 3-4-2-1. La solution obtenue est également optimale sur cet exemple (Cmax = 30).

Sur l'exemple numérique, les résultats obtenus avec la méthode CDS sont les suivants : (les nombres indiqués sont les durées des jobs sur les machines ou les groupes de machines associés à chaque colonne, sur lesquels on applique la règle de Johnson pour obtenir un ordre et donc un Cmax). Nous ne proposons pas de schémas mais seulement une présentation rapide.

	M1 / M3	M1+M2 / M3	M1 / M2+M3	M1+M2 / M2+M3
Job 1	3/3	7/3	3/7	7/7
Job 2	5/2	12/2	5/9	12/9
Job 3	4/6	12/6	4/14	12/14
Job 4	6/5	10/5	6/9	10/9
Séquence	3-4-1-2	3-4-1-2	1-3-2-4	3-2-4-1 ou 3-4-2-1
Cmax	31	31	31	31 ou 32

Tableau 3
Heuristique CDS modifiée

La solution trouvée a donc un Cmax de 31.

Annexes II

Principes de résolution

On présente maintenant un exemple pour illustrer le déroulement de la PSE de base : celui-ci reprend les valeurs numériques des tableaux 1 et 2.

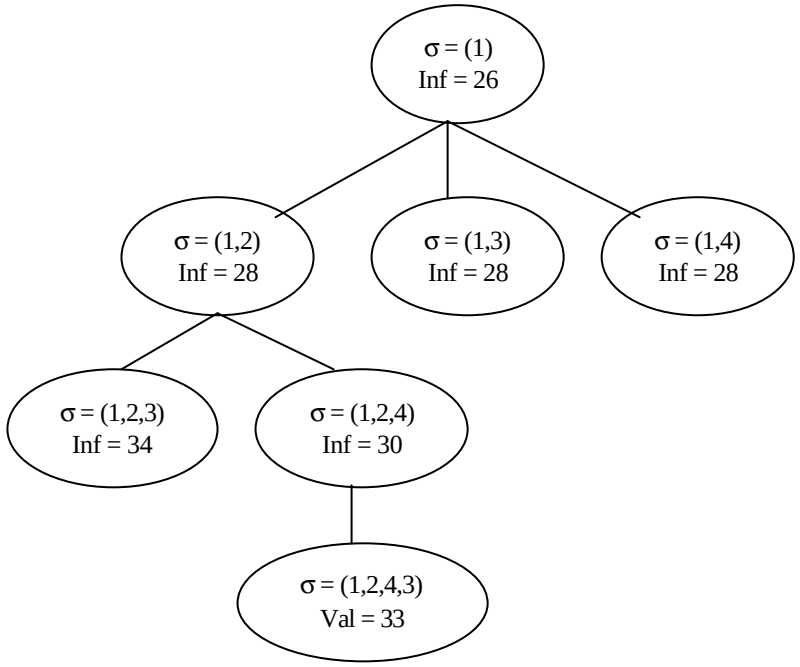


Figure 2
Déroulement de la PSE

Sur cette figure est représenté une partie du sous-arbre d'origine N1, le nœud associé à la séquence partielle (1). On suppose que ce nœud est le prochain nœud à séparer et qu'à ce moment de l'exploration, aucune solution n'a encore été trouvée ; la valeur de la borne supérieure est donc UB = +∞. Les bornes inférieures des nœuds, calculées d'après la formule du paragraphe précédent, sont données sous le nom *Inf*.

En séparant ce nœud, on génère 3 fils en ajoutant à la séquence du père un job parmi ceux qui n'ont pas encore été placés. Ces 3 fils ayant la même borne inférieure, on sépare arbitrairement le premier, correspondant à la séquence (1,2). Celui-ci conduit à 2 nouveaux nœuds. En admettant que le mode d'exploration de la PSE soit en profondeur d'abord, on sépare le meilleur des 2, c'est-à-dire celui qui représente la séquence (1,2,4). Il ne reste qu'un job à placer, donc le nœud n'a qu'un fils : celui-ci est terminal, et est associé à la permutation (1,2,4,3). En calculant le Cmax de l'ordonnancement correspondant (à l'aide de la fonction Ordo), on obtient une valeur de 33, qui devient la meilleure solution trouvée jusque là : on affecte donc cette valeur à la borne supérieure : UB = 33. Le dernier nœud visité est une feuille de l'arbre donc on remonte au niveau précédent : le nœud suivant à traiter correspond à la séquence (1,2,3). Sa borne inférieure (34) est supérieure à la borne supérieure (33) donc on est sûr qu'il n'amène pas à une solution meilleure que celle trouvée jusqu'à maintenant. Ce nœud est par conséquent éliminé. A ce niveau, l'exploration est terminée donc on remonte au niveau précédent. Le nœud suivant à traiter correspond à la séquence partielle (1,3). On répète le processus jusqu'à ce que tous les nœuds créés soient traités (c'est-à-dire séparés ou éliminés).

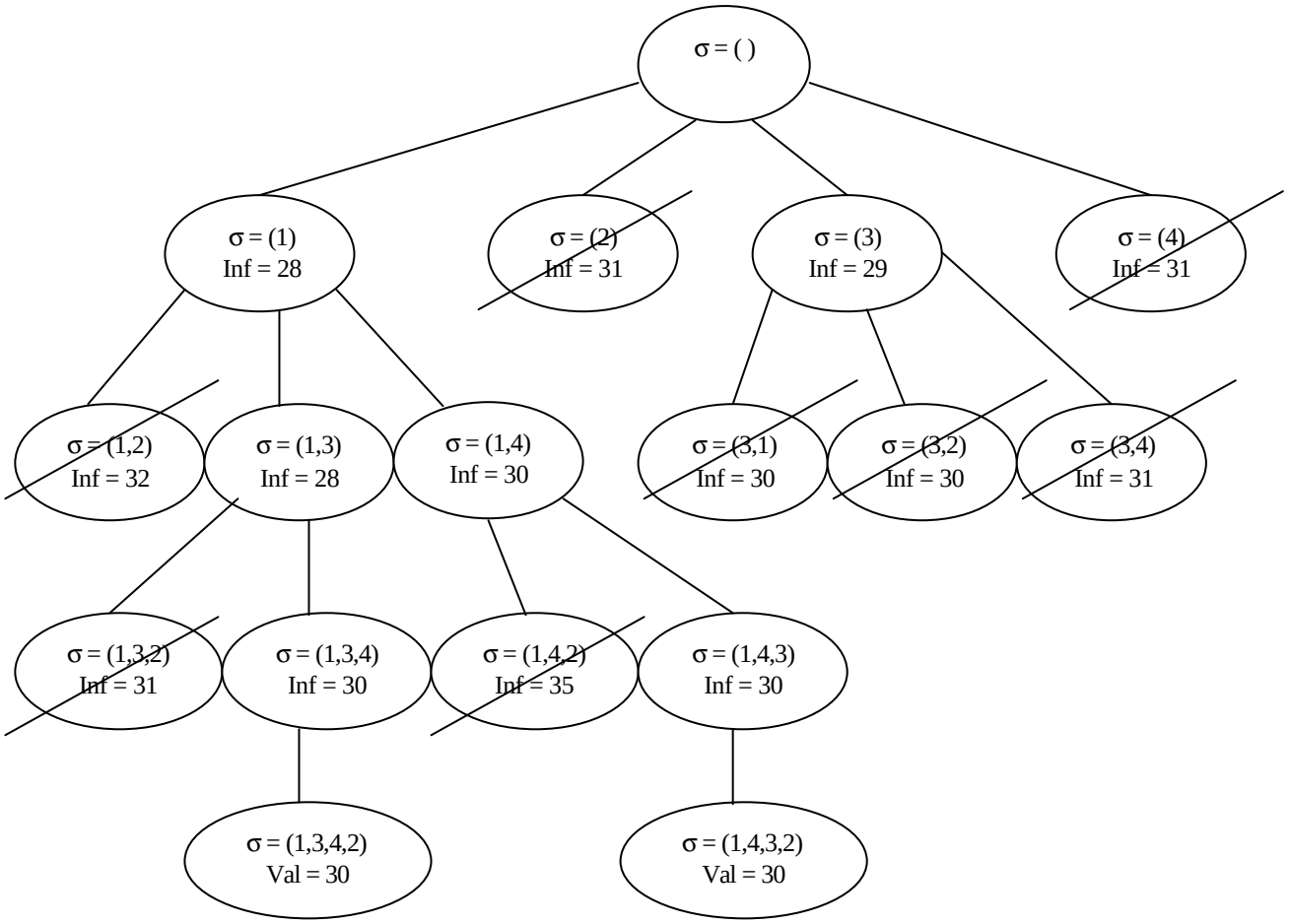


Figure 11
Beam search

La figure 11 représente le déroulement du beam search sur l'exemple numérique, avec un nombre de nœuds à conserver égal à 2; lorsque plus de 2 nœuds ont la même valeur, on sélectionne arbitrairement ceux que l'on garde (en suivant, par exemple, l'ordre de génération) : c'est le cas sur cet exemple, au niveau 2: les nœuds (1,4), (3,1) et (3,2) ont tous les trois la même valeur, mais on ne peut en conserver qu'un seul. La solution fournie par le beam search est ici optimale, puisque sa valeur est égale à la valeur de la meilleure solution obtenue par la PSE (30). L'évaluation des nœuds est ici effectuée en utilisant la borne tournée machine.

Annexes III

Résultats expérimentaux

Le tableau 4 donne l'écart relatif entre la solution optimale et la solution fournie par les différentes heuristiques, en moyenne et dans le pire cas (écart maximal) :
$(C_H - C_{opt} / C_{opt}) \times 100$

	H1	H2	NEH	CDS
Maximum	19,0878378	19,9716714	3,54291417	9,33133733
Moyenne	8,80363703	5,22914379	1,23768388	3,87790107

Tableau 4 : Résultats obtenus avec les heuristiques

Le tableau 5 concerne les performances des bornes utilisant la règle de Johnson : pour chacune d'elles, il indique l'écart relatif avec les performances de la borne « tournée machine », en termes de temps, et en termes de nœuds (une valeur positive indique une amélioration, une valeur négative un handicap) : $(Temps_{BM} - Temps_{BJ} / Temps_{BM}) \times 100$ et $(Nb\_noeuds_{BM} - Nb\_noeuds_{BJ} / Nb\_noeuds_{BM}) \times 100$

Borne Johnson		Borne Johnson étendue	
Temps	Nœuds	Temps	Nœuds
-312,005006	3,69043392	-1905,02152	12,8979788

Tableau 5 : Résultats obtenus avec les bornes inférieures / écarts relatifs avec la borne « tournée machine »

Le tableau 6 est similaire au précédent : il concerne la comparaison entre la PSE de base et les PSE plus sophistiquées, obtenues en effectuant les modifications décrites dans l'article.

				Classement des nœuds			
Fusion		Tests de dominance		$\lambda = 0\%$		$\lambda = 25\%$	
Temps	Nœuds	Temps	Noeuds	Temps	Nœuds	Temps	Nœuds
1,07323764	0,95620708	-15,4691563	0,19139624	-0,64480141	1,91728752	0,84249635	2,62620112

Tableau 6 : Résultats obtenus en modifiant la PSE / écarts relatifs avec la PSE de base