Optimiser l'agencement d'une fabrique grâce aux diagrammes de décision

Vianney Coppé*, Xavier Gillard, Pierre Schaus

UCLouvain

{vianney.coppe, xavier.gillard, pierre.schaus}@uclouvain.be

Résumé

Cet article présente un modèle exact pour résoudre un problème d'agencement de départements dans une fabrique avec des contraintes de position, d'ordre et d'adjacence. Ce problème, communément appelé le Constrained Single-Row Facility Layout Problem, a été précédemment résolu avec la programmation linéaire mixte en nombres entiers. L'approche proposée est basée sur les diagrammes de décision. Les expériences montrent qu'elle est plus rapide et permet de résoudre des instances de plus grande taille.

Mots-clés

Optimisation discrète, Diagrammes de décision, Constrained Single-Row Facility Layout Problem

Abstract

This paper presents an exact model for the Constrained Single-Row Facility Layout Problem, an extension of the Single-Row Facility Layout Problem with positioning, ordering and adjacency constraints. A mixed-integer programming model was already proposed for this problem. The proposed approach is based on decision diagrams. It is shown to be faster and allows to solve larger instances.

Keywords

Discrete Optimization, Decision Diagrams, Constrained Single-Row Facility Layout Problem

1 Introduction

Le *Constrained Single-Row Facility Layout Problem* (cSRFLP), est un problème d'agencement de départements au sein d'une fabrique organisée sur une seule rangée, avec des contraintes de position, d'ordre et d'adjacence. Ce problème a été introduit récemment par [9] en ajoutant ces contraintes au *Single-Row Facility Layout Problem* (SRFLP). La seule approche exacte présentée pour résoudre le cSRFLP est le modèle de programmation linéaire mixte en nombres entiers (MIP) de [10]. Dans cet article, le modèle de programmation dynamique (DP) pour le SRFLP [12] est étendu pour incorporer les contraintes du cSRFLP. Il est ensuite expliqué comment ce modèle peut être intégré dans un algorithme de séparation et évaluation, basé sur les diagrammes de décision (DD) [4]. Après quoi, une comparaison entre l'approche existante et celle introduite est faite, sur base d'expériences utilisant des instances de référence pour le SRFLP, auxquelles des contraintes sont ajoutées. Le modèle présenté est nettement plus rapide et permet de résoudre des instances de plus grande taille. L'article est conclu par un résumé des contributions ainsi que des pistes d'amélioration.

*L'auteur principal est doctorant.

2 Définition du problème

Soit un ensemble de départements $N = \{1, \dots, n\}$ dans une fabrique, dont chaque département $i \in N$ possède une longueur l_i. À chaque paire de départements $i, j \in N$ est associée une quantité de trafic c_{ij} les reliant. L'objectif du cSRFLP est de placer les départements côte à côte sur une seule rangée de manière à minimiser la distance totale parcourue dans la fabrique. L'agencement des départements est soumis à trois types de contraintes définies ci-après. Les contraintes de *position* assignent à un département une position fixe dans l'arrangement. Nous utiliserons deux fonctions pour représenter ces contraintes : $position : N \to N \cup \{0\}$ et $department : N \to N \cup \{0\}$. La première donne la position à laquelle chaque département est fixé, ou 0 si la position du département n'est pas imposée. La deuxième est la relation inverse, reliant les positions aux départements. Les contraintes d'*ordre* imposent une précédence entre deux départements dans l'arrangement. La fonction $predecessors : N \to 2^N$ lie chaque département à l'ensemble de départements qui doivent être placés à gauche de celui-ci. Enfin, les contraintes d'*adjacence* sont similaires aux contraintes d'ordre, à la différence que les départements concernés doivent cette fois-ci être adjacents. La fonction $previous : N \to N \cup \{0\}$ envoie donc chaque département vers son voisin gauche direct, s'il existe, et 0 sinon. Soit la bijection $\pi : N \to N$ reliant les départements à leur position dans l'arrangement, la fonction objectif du problème est définie comme suit :

$$cSRFLP(\pi) = \sum_{k=1}^{n} l_k \sum_{\substack{i=1 \\ \pi(i)<\pi(k)}}^{n} \sum_{\substack{j=1 \\ \pi(k)<\pi(j)}}^{n} c_{ij} + K \quad (1)$$

$$K = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i<j}}^{n} c_{ij} \frac{l_i + l_j}{2} \quad (2)$$

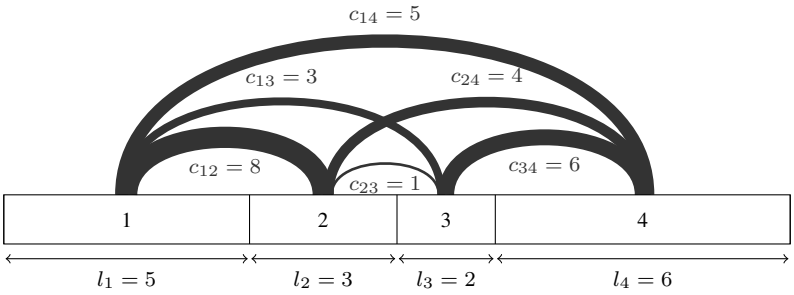

FIGURE 1 – Une instance du cSRFLP dont les départements sont arrangés de manière optimale.

à minimiser en respectant les contraintes suivantes :

$$\pi(i) = j \quad \forall i, j \in N, position(i) = j \neq 0 \quad (3)$$
$$\pi(i) > \pi(j) \quad \forall i \in N, \forall j \in predecessors(i) \quad (4)$$
$$\pi(i) = \pi(j) + 1 \quad \forall i, j \in N, previous(i) = j \neq 0. \quad (5)$$

et où K est une constante englobant les contributions des demi-longueurs de département à la fonction objectif.

3 Modèle DP

Avant de présenter comment intégrer les contraintes du cSRFLP au modèle DP pour le SRFLP [12], nous rappelons d'abord celui-ci. L'idée est de placer les départements séquentiellement de gauche à droite et de maintenir comme état l'ensemble des départements qui restent à placer. De cette manière, il est possible d'ajouter la contribution individuelle d'un département à la fonction objectif lorsqu'il est placé. Formellement, le modèle DP est composé des *variables de décision* $x_j \in D_j$ avec $j \in \{0, \ldots, n-1\}$. La variable x_j correspond au département placé à la position $j+1$. Toutes les variables ont le même domaine $D_j = N$ puisque tous les départements sont candidats à être placés à chaque position. L'ensemble des états S contient tous les sous-ensembles de N, dont l'état *racine* $\hat{r} = N$ et *terminal* $\hat{t} = \emptyset$. On définit également un état *non-admissible* $\hat{0}$. L'ensemble S est partitionné en $n+1$ ensembles $S_0, \ldots, S_n$, où l'ensemble S_j contient tous les états avec j variables assignées. Les *fonctions de transition* $t_j : S_j \times D_j \to S_{j+1}$ pour chaque $j = 0, \ldots, n-1$ spécifient l'état résultant d'une décision supplémentaire sur un état :

$$t_j\left(s^j, x_j\right) = \begin{cases} s^j \setminus \{x_j\}, & \text{si } x_j \in s^j \\ \hat{0}, & \text{sinon.} \end{cases} \quad (6)$$

Celles-ci vont de paire avec les *fonctions de coût de transition* $h_j : S_j \times D_j \to \mathbb{R}$ pour $j = 0, \ldots, n-1$ qui associent une valeur à chaque transition :

$$h_j\left(s^j, x_j\right) = \begin{cases} l_{x_j} \sum_{i \in \overline{s}^j} \sum_{k \in s^j \setminus \{x_j\}} c_{ik}, & \text{si } x_j \in s^j \\ 0, & \text{sinon.} \end{cases} \quad (7)$$

Cette formule découle de l'équation (1) puisque $\overline{s}^j$ – le complément de s^j – et $s^j \setminus \{x_j\}$ contiennent respectivement les départements placés à gauche et à droite du département considéré. Le dernier élément est la valeur donnée à la racine : $v_r = K$, voir équation (2).

La solution de ce modèle DP peut ensuite être trouvée en résolvant la formule de récurrence suivante :

$$\begin{aligned} &\min \hat{f}(x) = v_r + \sum_{j=0}^{n-1} h_j\left(s^j, x_j\right) \\ &\text{sujet à } s^{j+1} = t_j\left(s^j, x_j\right), \forall x_j \in D_j, j = 0, \ldots, n-1 \\ &\qquad s^j \in S_j, j = 0, \ldots, n. \end{aligned} \quad (8)$$

Afin d'intégrer au modèle les contraintes mentionnées dans la section 2, nous introduisons les prédicats $valid_j : S_j \times D_j \to \{true, false\}$ pour $j = 0, \ldots, n-1$. Ils servent à filtrer les solutions non-admissibles dans les fonctions de transition :

$$t_j\left(s^j, x_j\right) = \begin{cases} s^j \setminus \{x_j\}, & \text{si } x_j \in s^j \wedge valid_j(s^j, x_j) \\ \hat{0}, & \text{sinon.} \end{cases} \quad (9)$$

Pour être satisfaits, ces prédicats nécessitent que toutes les contraintes soient respectées :

$$valid_j(s^j, x_j) = p_j(s^j, x_j) \wedge o_j(s^j, x_j) \wedge r_j(s^j, x_j) \quad (10)$$

où p_j, o_j et r_j concernent respectivement les contraintes de position, ordre et adjacence :

$$\begin{aligned} &p_j(s^j, x_j) = (position(x_j) = j+1) \\ &\vee (position(x_j) = 0 \wedge department(j+1) = 0) \end{aligned} \quad (11)$$
$$o_j(s^j, x_j) = predecessors(x_j) \subseteq \overline{s}^j \quad (12)$$
$$\begin{aligned} &r_j(s^j, x_j) = (previous(x_j) \in \overline{s}^j) \vee \\ &(previous(x_j) = 0 \wedge \nexists k \in s^j : previous(k) \in \overline{s}^j). \end{aligned} \quad (13)$$

L'équation (11) vérifie que le département x_j doit être placé à la position $j+1$, ou bien que le département et la position en question ne soient pas impliqués dans une contrainte de position. Pour les contraintes d'ordre, l'équation (12) s'assure que tous les prédécesseurs du département x_j sont déjà placés. Dans l'équation (13), on s'assure que le voisin direct du département x_j soit déjà placé, ou qu'aucun des départements qu'il reste à placer n'a de contrainte d'adjacence.

Calcul des coûts de transition Pour calculer les coûts de transition plus rapidement, nous maintenons également dans chaque état les *valeurs de coupe* de chaque département libre, c'est-à-dire la somme des quantités de trafic

reliant les départements placés et le département libre en question. Pour l'état s^j et chaque département $i \in N$, on définit :

$$s^j_{cut}[i] = \begin{cases} \sum_{j \in \overline{s}^j} c_{ij}, & \text{si } i \in s^j \\ 0, & \text{sinon} \end{cases} \quad (14)$$

qui est mis à jour en $\mathcal{O}(n)$ lors des transitions $t_j\left(s^j, x_j\right)$:

$$s^{j+1}_{cut}[i] = \begin{cases} s^j_{cut}[i] + c_{ix_j}, & \text{si } i \in s^j \setminus \{x_j\} \\ 0, & \text{sinon.} \end{cases} \quad (15)$$

La formule du coût de transition devient donc :

$$h_j\left(s^j, x_j\right) = \begin{cases} l_{x_j} \sum_{i \in s^j \setminus \{x_j\}} s^j_{cut}[i], & \text{si } x_j \in s^j \\ 0, & \text{sinon} \end{cases} \quad (16)$$

qui peut être calculée en $\mathcal{O}(n)$ au lieu de $\mathcal{O}\left(n^2\right)$. Les valeurs de coupe seront également utilisées dans la section 5.2 pour obtenir une borne inférieure.

4 Représentation DD

En partant d'un modèle DP, il est aisé de construire le DD exact correspondant. En effet, un DD est un *graphe orienté acyclique* $B = (U, A, d, v, \sigma)$ dont l'ensemble de *nœuds* U est partitionné en plusieurs *couches* $L_0, \ldots, L_n$. Les *arcs* $a \in A$ reliant les nœuds de couches consécutives L_j et L_{j+1} encodent l'assignation d'une valeur à la variable de décision x_j. De plus, la fonction σ fait correspondre chaque nœud à un état du modèle DP. Ainsi, la première couche contient un seul nœud, le nœud racine, logiquement associé à l'état racine du modèle DP. Ensuite, pour chaque nœud d'une couche donnée L_j, on génère la couche L_{j+1} en ajoutant un nœud pour chaque état *distinct* obtenu grâce à la fonction de transition. Lors d'une transition, la valeur assignée à la variable de décision et le coût correspondant sont donnés respectivement par le *label* $d(a)$ et la *longueur* $v(a)$ de l'arc associé. Dès lors, un chemin $p = \left(a^{(0)}, \ldots, a^{(n-1)}\right)$ reliant le nœud racine r au nœud terminal t représente une solution au problème, avec $x_j = d\left(a^{(j)}\right)$, et dont la longueur totale $v(p) = v_r + \sum_{j=0}^{n-1} v\left(a^{(j)}\right)$ équivaut au coût. Dans un DD *exact*, qui contient toutes les solutions admissibles du problème associé, trouver la solution optimale revient donc à résoudre un problème de plus court chemin.

5 Séparation et évaluation

L'attrait des DDs pour l'optimisation discrète vient surtout de l'algorithme de séparation et évaluation basé sur ceux-ci, présenté dans [4]. Au lieu de générer un DD exact pour l'entièreté du problème considéré, l'idée est d'explorer celui-ci intelligemment en générant des DDs approximés, permettant à la fois de trouver des solutions admissibles de bonne qualité et de couper certaines branches.

5.1 Borne supérieure

Pour calculer une borne supérieure, les DDs *restreints* sont utilisés. Leur fonctionnement est simple : on impose au DD développé à partir du nœud donné une *largeur* maximale, qui correspond au nombre maximal de nœuds contenus dans une de ses couches. Pendant la création du DD en question, le surplus de nœuds générés dans chaque couche est alors supprimé. De cette manière, le DD résultant contient un sous-ensemble des solutions admissibles du problème, et fournit donc une borne supérieure. La qualité de cette borne est directement impactée par le choix des nœuds retirés de chaque couche, en pratique on retire donc ceux qui semblent les moins prometteurs.

5.2 Borne inférieure

À la façon des DDs restreints, les DDs *relaxés* permettent de calculer des bornes inférieures. Une largeur maximale est également imposée, mais cette fois-ci le surplus de nœuds est fusionné en un unique nœud. Celui-ci est réintroduit dans le DD, de manière à n'exclure aucune solution admissible. Ce faisant, il est probable que des solutions non-admissibles soient introduites dans le DD. L'opérateur de fusion mentionné doit être spécifié pour chaque problème étudié. Pour le cSRFLP, nous en avons essayé plusieurs mais avons obtenu les meilleurs résultats avec une borne inférieure heuristique.
Étant donné un nœud u et la valeur $v^*(u)$ du plus court chemin reliant le nœud racine à celui-ci, le théorème 1 montre que le coût restant à ajouter à $v^*(u)$ pour obtenir le coût complet de l'arrangement $\pi^*|_u$ peut être divisé en deux termes, que nous allons ensuite approximer séparément.

Théorème 1 *Soit un nœud u et son état $\sigma(u) = s$, soit $\pi^*|_u$ l'arrangement de coût minimal qui traverse u. Par souci de concision, on pose $\pi = \pi^*|_u$. On a alors l'égalité :*

$$SRFLP(\pi) - v^*(u) = \underbrace{\sum_{k \in s} l_k \sum_{i \in s} \sum_{\substack{j \in s \\ \pi(i) < \pi(k)\ \pi(k) < \pi(j)}} c_{ij}}_{\text{coût d'arrangement des départements libres}} + \underbrace{\sum_{k \in s} l_k \sum_{\substack{j \in s \\ \pi(k) < \pi(j)}} s_{cut}[j]}_{\text{coût lié aux départements fixés}}. \quad (17)$$

Coût d'arrangement des départements libres En observant le premier terme de l'équation (17), on remarque qu'il vaut exactement le coût de l'arrangement considérant uniquement les départements libres. Celui-ci peut être sous-évalué comme spécifié par l'algorithme 1. On commence par trier les quantités de trafic par ordre décroissant. De même, on trie les longueurs des départements par ordre croissant. On remarque ensuite qu'en plaçant n départements côte à côte, $n - k$ paires de départements seront séparés par $k - 1$ départements (voir figure 1). Une borne inférieure est obtenue en supposant que les plus importantes intensités de trafic reliant des départements libres sont multipliées par 0, n'ayant aucun département les séparant. Les $n - k$ plus grandes intensités de trafic suivantes sont multipliées par la longueur des $k - 1$ départements libres les

Algorithme 1 Calcul de $LB_{edge}(u)$.

Require: $edge = sorted_{\geq}(\{\langle c : c_{ij}, dep1 : i, dep2 : j\rangle \mid 1 \leq i < j \leq n\})$
Require: $length = sorted_{\leq}(\{\langle l : l_i, dep : i\rangle \mid 1 \leq i \leq n\})$
$s \leftarrow \sigma(u), lb \leftarrow 0, cumul\_l \leftarrow 0, i \leftarrow 1, j \leftarrow 1$
for $k \leftarrow 1$ **to** $|s| - 1$ **do**
  for $l \leftarrow 1$ **to** k **do**
    while $edge[i].dep1 \notin s \vee edge[i].dep2 \notin s$ **do**
      $i \leftarrow i + 1$
    $lb \leftarrow lb + cumul\_l \cdot edge[i].c$
    $i \leftarrow i + 1$
  while $length[j].dep \notin s$ **do**
    $j \leftarrow j + 1$
  $cumul\_l \leftarrow cumul\_l + length[j].l$
  $j \leftarrow j + 1$
return lb

Algorithme 2 Calcul de $LB_{cut}(u)$.

$s \leftarrow \sigma(u), lb \leftarrow 0, cumul\_l \leftarrow 0, order \leftarrow [\,]$
for all $i \in s$ **do**
  $order.append(\langle ratio : s_{cut}[i]/l_i, dep : i\rangle)$
$sort_{\geq}(order)$
for $k \leftarrow 1$ **to** $|s|$ **do**
  $lb \leftarrow lb + cumul\_l \cdot s_{cut}[order[k].dep]$
  $cumul\_l \leftarrow cumul\_l + l_{order[k].dep}$
return lb

moins longs. Cette borne inférieure peut être vue comme une généralisation de la *Edges method* [11] pour le *Minimum Linear Arrangement Problem*, un cas particulier du SRFLP.

Coût lié aux départements fixés Le second terme de l'équation (17) tient compte des valeurs de coupe, dont la valeur sera encore multipliée par la distance séparant chaque département du dernier département placé. Une borne inférieure est donnée par la *borne de première génération* introduite dans [13]. Comme formalisé par l'algorithme 2, elle consiste à trier les départements libres par ordre décroissant de ratio entre valeur de coupe et longueur. L'arrangement optimal des départements libres par rapport au coût lié aux valeurs de coupe est donné par le résultat du tri, sa valeur est donc une borne inférieure sur le deuxième terme de l'équation (17).
Finalement, la borne inférieure d'un nœud u est donnée par $LB(u) = v^*(u) + LB_{edge}(u) + LB_{cut}(u)$. Celle-ci peut être utilisée pour filtrer des nœuds à la fois dans l'algorithme de séparation et évaluation, et dans la génération des DDs [6].

6 Résultats expérimentaux

L'approche présentée a été implémentée en C++, en basant largement le code sur la librairie DDO [5] pour le langage Rust. Nous la comparons avec le modèle MIP introduit par Liu et al. [10] implémenté avec Gurobi [7]. Nos expériences utilisent des instances classiques du SRFLP [1, 2, 3, 8, 13] avec jusque 25 départements, auxquelles nous ajoutons des ensembles valides de contraintes, à savoir :

— des ensembles de contrainte avec $2, 4, 6, 8$ et 10 contraintes de position, d'ordre ou d'adjacence.
— des ensembles de contrainte avec $0, 2, 4, 6, 8$ et 10 contraintes de position, d'ordre et d'adjacence.

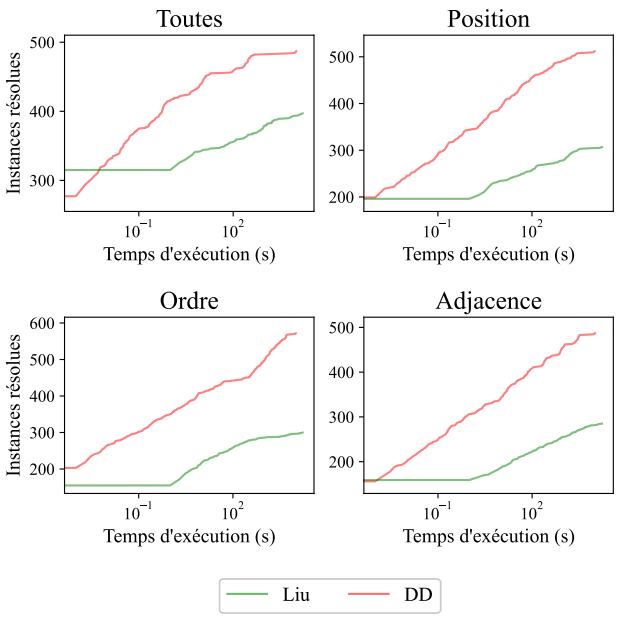

FIGURE 2 – Nombre d'instances résolues par chaque algorithme en fonction du type de contraintes imposé.

Pour chacune de ces configurations, cinq ensembles de contraintes ont été générés de façon aléatoire. Les deux algorithmes ont ensuite été confrontés à chaque combinaison valide d'instance et ensemble de contraintes, avec un temps maximal fixé à cinq heures. La figure 2 montre le résultat de ces expériences : quel que soit le type de contraintes imposé, l'approche DD est capable de résoudre toutes les instances tandis que le modèle MIP est incapable d'en résoudre presque la moitié. De plus, cette supériorité est aussi observée en termes de vitesse de résolution pour les instances résolues par les deux algorithmes.

Détails d'implémentation Pour l'approche DD, nous utilisons des DDs de largeur maximale égale à 3. Nous avons en effet remarqué que de très bonnes solutions étaient trouvées assez tôt dans la recherche malgré cette faible largeur, et avec les adaptations décrites précédemment, la largeur n'a pas d'influence sur la qualité des bornes inférieures. La sélection des nœuds à supprimer pour créer les DDs restreints se fait sur base de la borne inférieure. Enfin, nous avons adapté l'algorithme de séparation et évaluation pour parcourir les nœuds en largeur. Cela permet d'éviter d'explorer plusieurs nœuds correspondant au même état.

7 Conclusion

Dans cet article, le modèle DP du SRFLP a été étendu pour couvrir les contraintes introduites par le cSRFLP. Une approche DD basée sur ce modèle a ensuite été décrite, y compris une formule de borne inférieure. Les expériences effectuées sur un large panel d'instances ont permis de montrer la rapidité de l'approche présentée. Celle-ci pourrait certainement être améliorée dans le futur, notamment en tenant compte des contraintes dans les calculs de borne inférieure.

Références

[1] André R. S. Amaral. On the exact solution of a facility layout problem. *European Journal of Operational Research*, 173(2) :508–518, 2006.

[2] André R. S. Amaral. An exact approach to the one-dimensional facility layout problem. *Operations Research*, 56(4) :1026–1033, 2008.

[3] Miguel F. Anjos and Anthony Vannelli. Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. *INFORMS Journal on Computing*, 20(4) :611–617, 2008.

[4] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John N. Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1) :47–66, 2016.

[5] Xavier Gillard, Pierre Schaus, and Vianney Coppé. Ddo, a generic and efficient framework for mdd-based optimization. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, pages 5243–5245, 2020.

[6] Xavier Gillard, Pierre Schaus, Vianney Coppé, and André A. Cire. Improving the filtering of branch-and-bound mdd solver. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2021.

[7] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2022.

[8] Sunderesh S. Heragu and Andrew Kusiak. Efficient models for the facility layout problem. *European Journal of Operational Research*, 53(1) :1–13, 1991.

[9] Zahnupriya Kalita and Dilip Datta. A constrained single-row facility layout problem. *The international journal of advanced manufacturing technology*, 98(5) :2173–2184, 2018.

[10] Silu Liu, Zeqiang Zhang, Chao Guan, Lixia Zhu, Min Zhang, and Peng Guo. An improved fireworks algorithm for the constrained single-row facility layout problem. *International Journal of Production Research*, 59(8) :2309–2327, 2021.

[11] Jordi Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics*, 8, 2003.

[12] Jean-Claude Picard and Maurice Queyranne. On the one-dimensional space allocation problem. *Operations Research*, 29(2) :371–391, 1981.

[13] Donald M. Simmons. One-dimensional space allocation : an ordering algorithm. *Operations Research*, 17(5) :812–826, 1969.