Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Fast local search for single row facility layout

Gintaras Palubeckis*

Faculty of Informatics, Kaunas University of Technology, Studentu 50-408, 51368 Kaunas, Lithuania

ARTICLE INFO

Article history:
Received 11 April 2014
Accepted 20 May 2015
Available online 30 May 2015

Keywords:
Combinatorial optimization
Single row facility layout
Local search
Variable neighborhood search

ABSTRACT

Given n facilities of prescribed lengths and a flow matrix, the single row facility layout problem (SRFLP) is to arrange the facilities along a straight line so as to minimize the total arrangement cost, which is the sum of the products of the flows and center-to-center distances between facilities. We propose interchange and insertion neighborhood exploration (NE) procedures with time complexity $O(n^2)$, which is an improvement over $O(n^3)$-time NE procedures from the literature. Numerical results show that, for large SRFLP instances, our insertion-based local search (LS) algorithm is two orders of magnitude faster than the best existing LS techniques. As a case study, we embed this LS algorithm into the variable neighborhood search (VNS) framework. We report computational results for SRFLP instances of size up to 300 facilities. They indicate that our VNS implementation offers markedly better performance than the variant of VNS that uses a recently proposed $O(n^3)$-time insertion-based NE procedure.

1. Introduction

An active line of research in the area of combinatorial optimization is concerned with developing various algorithms for a wide set of problems whose solutions are permutations. An important member of this set is the *single row facility layout problem* (SRFLP for short). Given a number of facilities and the flows between them, the SRFLP is to arrange the facilities along a straight line so as to minimize the total arrangement cost, which is the sum of the products of the flows and center-to-center distances between facilities. Suppose that there are n facilities having lengths $L_1, \ldots, L_n$, respectively. Let $W = (w_{ij})$ be a symmetric $n \times n$ matrix whose entry w_{ij} represents the flow of material between facilities i and j. Our intention in this paper is to deal with a version of the SRFLP where clearances between facilities, denoted as γ_{ij}, $i, j \in \{1, \ldots, n\}$, $i \neq j$, are not all equal to a (nonnegative) constant value. We emphasize that our approach to the SRFLP is applicable even when the matrix of clearances, $\Gamma = (\gamma_{ij})$, is not assumed to be necessarily symmetric. Because of this, and also to avoid losing the generality of the SRFLP formulation, we do not require symmetry in the matrix Γ. Certainly, the main diagonal of Γ is zero, and all other entries are nonnegative. With these notations, the SRFLP can be expressed as

$$\min_{p \in \Pi} F(p) = \sum_{k=1}^{n-1} \sum_{l=k+1}^{n} w_{p(k)p(l)} d_{p(k)p(l)}, \qquad (1)$$

where Π is the set of all permutations of $\{1, \ldots, n\}$, $p(k)$ is the facility in the k-th position of permutation p, and $d_{p(k)p(l)}$ is the distance between the centroids of facilities $p(k)$ and $p(l)$. Let us assume that $k < l$. Then the distance is calculated according to the following equation:

$$d_{p(k)p(l)} = L_{p(k)}/2 + \sum_{\substack{m=k+1 \\ \text{if } l>k+1}}^{l-1} L_{p(m)} + L_{p(l)}/2 + \sum_{m=k}^{l-1} \gamma_{p(m)p(m+1)}. \qquad (2)$$

We note that the formulation (1)–(2) was used by Datta, Amaral, and Figueira (2011) in their paper on a genetic algorithm approach to single row facility layout.

The SRFLP is a challenging research problem which has several real-life applications. In the area of flexible manufacturing systems, it models the linear layout of machines within manufacturing cells. In this type of layout, the machines are placed along a straight path travelled by an automated guided vehicle (Heragu & Kusiak, 1988). Other applications of the SRFLP include arranging a number of rooms on one side of a corridor in supermarkets, hospitals and office buildings (Simmons, 1969), arranging books on a shelf in a library (Picard & Queyranne, 1981), and design of warehouse layouts (Picard & Queyranne, 1981).

Because of the practical importance of the SRFLP, considerable attention has been given to the development of algorithms for its solution. Existing exact methods for the SRFLP include branch-and-bound (Simmons, 1969), dynamic programming (Picard & Queyranne, 1981), mixed-integer linear programming (Amaral, 2006, 2008; Heragu & Kusiak, 1991), cutting plane (Amaral, 2009), branch-and-cut (Amaral & Letchford, 2013), and semidefinite programming approaches (Anjos & Vannelli, 2008; Hungerländer & Rendl, 2013). Branch-and-bound

* Tel.: +370 37 300353.
E-mail address: gintaras@soften.ktu.lt, gintaras.palubeckis@ktu.lt

http://dx.doi.org/10.1016/j.ejor.2015.05.055
0377-2217/© 2015 Elsevier B.V. and Association of European Operational Research Societies (EURO) within the International Federation of Operational Research Societies (IFORS). All rights reserved.

(Palubeckis, 2012) and semidefinite programming (Hungerländer, 2014) algorithms were also applied for solving a special case of the problem in which all facilities have the same length. A computational comparison of the state-of-the-art exact methods for the SRFLP is given in the paper by Hungerländer and Rendl (2013). They report that the largest SRFLP instance solved to prove optimality involves 42 facilities. For the purpose of finding good but not necessarily optimal solutions for larger instances of the problem, a number of heuristic algorithms have been developed.

The fastest methods to generate feasible solutions for large-scale SRFLP instances are construction heuristics. Among them, a greedy-like algorithm of Heragu and Kusiak (1988) and an iterative construction procedure of Djellab and Gourgand (2001) can be mentioned. However, it is widely acknowledged that construction heuristics are not able to produce solutions of high quality. They can be applied in situations where computation time is a critical factor.

Another group of heuristic algorithms construct a permutation of facilities from the results obtained by solving either a mixed-integer or a semidefinite program (SDP). In particular, an algorithm relying on a mixed-integer programming model was presented by Heragu and Kusiak (1991). Recently, Amaral and Letchford (2013) have proposed an approach which allows obtaining a suboptimal solution as a byproduct of the branch-and-cut method. The crux of their approach is the use of a multi-dimensional scaling technique. Anjos, Kennings, and Vannelli (2005) were the first who proposed an SDP-based heuristic to produce a single row facility layout. The same strategy to obtain solutions to the SRFLP was followed by Anjos and Yen (2009) and Hungerländer and Rendl (2013).

The other way to approach the problem is to use metaheuristic search methods. The application of metaheuristics for the SRFLP dates back at least to Romero and Sánchez-Flores (1990) and Heragu and Alfa (1992), who developed simulated annealing algorithms for the problem. de Alvarenga, Negreiros-Gomes, and Mestria (2000) proposed another simulated annealing implementation for the SRFLP. The same authors also presented a tabu search algorithm and tested both metaheuristics on a small set of instances of size $n \leq 30$. More recent variants of tabu search strategy were proposed by Samarghandi and Eshghi (2010) and Kothari and Ghosh (2013a). Solimanpur, Vrat, and Shankar (2005) developed an ant algorithm for the SRFLP. Teo and Ponnambalam (2008) investigated a hybrid approach, combining ant colony optimization and particle swarm optimization (PSO) techniques. A pure PSO algorithm for the problem was proposed by Samarghandi, Taabayan, and Jahantigh (2010). Recently, Kothari and Ghosh (2013b) presented an insertion-based Lin–Kernighan heuristic for producing good quality layouts. The heuristic was shown to be competitive with other high-performance algorithms. There are also several layout methods available which follow the genetic paradigm. These include genetic algorithms of Ficko, Brezocnik, and Balic (2004) and Datta et al. (2011) as well as hybrid genetic algorithms of Ozcelik (2012) and Kothari and Ghosh (2014a). A similar evolutionary technique called the imperialist competitive algorithm was presented by Lian, Zhang, Gao, and Shao (2011). Single row layout algorithms based on the scatter search metaheuristic were proposed by Kumar, Asokan, Kumanan, and Varma (2008) and Kothari and Ghosh (2014b). The second of them was reported to yield very good solutions for popular benchmark SRFLP instances. For recent surveys on the single row facility layout problem, the reader is referred to Anjos and Liers (2012), Hungerländer and Rendl (2013), Keller and Buscher (2015), and Kothari and Ghosh (2012).

From the literature, it can be seen that many algorithms for the SRFLP incorporate a local search procedure (Amaral & Letchford, 2013; Heragu & Alfa, 1992; Heragu & Kusiak, 1991; Kothari & Ghosh, 2013a, 2014a, 2014b; Kumar et al., 2008; Ozcelik, 2012; Samarghandi & Eshghi, 2010; Samarghandi et al., 2010; Solimanpur et al., 2005; Teo & Ponnambalam, 2008). Two main types of local searches have been used for this problem (Kothari & Ghosh, 2013b). The first of them is based on pairwise interchanges of facilities (Amaral & Letchford, 2013; Heragu & Alfa, 1992; Heragu & Kusiak, 1991; Kothari & Ghosh, 2013a; Samarghandi & Eshghi, 2010; Samarghandi et al., 2010; Solimanpur et al., 2005; Teo & Ponnambalam, 2008), whereas the second one proceeds by executing insertion moves, where a facility is moved from one position to another in the permutation (Kothari & Ghosh, 2013a, 2014a, 2014b; Kumar et al., 2008; Ozcelik, 2012). The performance of local search (LS) algorithms greatly depends on the neighborhood exploration (NE) procedures (Kothari & Ghosh, 2013a). A straightforward implementation of such a procedure for each type of LS has time complexity $O(n^4)$ (Kothari & Ghosh, 2013a). Indeed, for example, in the case of interchange-based LS, there are $n(n-1)/2$ pairs of facilities, and computation of the objective function value for a permutation obtained by interchanging two facilities takes $O(n^2)$ operations. Recently, Kothari and Ghosh (2013a) developed NE procedures (for both interchange and insertion neighborhood structures) whose time complexity is $O(n^3)$. The algorithms of Kothari and Ghosh (2013a, 2013b, 2014a, 2014b) use these procedures and show good performance compared to other methods in the literature.

Many studies on the SRFLP (Amaral and Letchford, 2013; Anjos et al., 2005; Kothari and Ghosh, 2013a, among others) assumed that the clearance between each pair of facilities is equal to a constant value. In such a situation, by adequately adjusting the length of the facilities, zero clearances can be achieved. In this paper, however, our intention is to consider a more general model in which clearances between adjacent facilities are not necessarily all equal. There are two main reasons for such a choice. First, as emphasized by Solimanpur et al. (2005), allowing different clearances is important in real life manufacturing. Solimanpur et al. (2005) listed several factors that affect the clearance spaces required between facilities. They mentioned that an analytic approach, e.g. queuing models or simulation study, can be used to obtain the required data. Second, the assumption of different clearances between facilities adds no principal difficulties to our approach to constructing fast local search algorithms for the SRFLP.

The primary motivation of this paper is to develop more efficient neighborhood exploration algorithms than those presented by Kothari and Ghosh (2013a). We propose three NE procedures, one for searching pairwise interchange neighborhoods and the other two for searching insertion neighborhoods. The time complexity of each procedure is $O(n^2)$. We present empirical results comparing the performance of our NE procedures against those of Kothari and Ghosh (2013a). We embed these procedures in the variable neighborhood search algorithm for solving the SRFLP. We report on computational experiments on SRFLP instances of size up to 300 facilities.

The outline of the rest of the paper is as follows. In Section 2, we rephrase the objective function of the problem and introduce some preliminary notations. In Sections 3 and 4, we propose interchange-based and, respectively, insertion-based local search algorithms for the SRFLP. Their experimental evaluation is presented in Section 5. In Section 6, we provide a case study focused on the application of the variable neighborhood search metaheuristic for the considered problem. Concluding remarks are given in Section 7. Proofs of some results appear in Appendix A.

2. Preliminaries

The SRFLP (1)–(2) can be restated using an alternative form of the objective function. To present this form, we fix a permutation $p \in \Pi$ and consider a family of cuts induced by subsets of facilities $V_m = \{p(k) \mid k = 1, \ldots, m\}$, $m \in \{1, 2, \ldots, n-1\}$. Let $m \in \{1, 2, \ldots, n-1\}$. We call the sum $c_m = \sum_{k=1}^{m} \sum_{l=m+1}^{n} w_{p(k)p(l)}$ the *cut value* and the vector $C = (c_1, \ldots, c_{n-1})$ indexed by cuts the *cut vector*. We define $\lambda_r = L_r/2$ to be the half-length of the facility r, $r \in \{1, \ldots, n\}$. With these definitions, the objective function in (1) can be rewritten as follows.

Proposition 1. *For $p \in \Pi$,*

$$F(p) = \sum_{m=1}^{n-1} c_m(\lambda_{p(m)} + \lambda_{p(m+1)} + \gamma_{p(m)p(m+1)}). \tag{3}$$

Proof. For $m \in \{1, \ldots, n\}$, let us denote by $\lambda^{\text{left}}_{p(m)}$ (respectively $\lambda^{\text{right}}_{p(m)}$) the left half length (respectively, right half length) of the facility placed in the m-th position in the permutation p. Then $F(p)$, given by (1) and (2), can be rewritten as (for notational simplicity, we assume here and in what follows that a sum is zero if the lower limit of the summation index is greater than the upper limit)

$$\begin{aligned} F(p) &= \sum_{k=1}^{n-1}\sum_{l=k+1}^{n} w_{p(k)p(l)} \\ &\quad\times\left(\lambda^{\text{right}}_{p(k)} + \sum_{j=k+1}^{l-1}\left(\lambda^{\text{left}}_{p(j)} + \lambda^{\text{right}}_{p(j)}\right) + \lambda^{\text{left}}_{p(l)} + \sum_{j=k}^{l-1}\gamma_{p(j)p(j+1)}\right) \\ &= \sum_{k=1}^{n-1}\sum_{l=k+1}^{n} w_{p(k)p(l)} \\ &\quad\times\left(\sum_{j=k+1}^{l}\lambda^{\text{left}}_{p(j)} + \sum_{j=k}^{l-1}\lambda^{\text{right}}_{p(j)} + \sum_{j=k}^{l-1}\gamma_{p(j)p(j+1)}\right). \end{aligned} \tag{4}$$

We combine all the terms in (4) containing $\lambda^{\text{left}}_{p(m)}$, $m \in \{2, \ldots, n\}$. It can be checked that the resulting expression is

$$\lambda^{\text{left}}_{p(m)}\sum_{k=1}^{m-1}\sum_{l=m}^{n} w_{p(k)p(l)}, \quad m \in \{2, \ldots, n\}. \tag{5}$$

Similarly, all the terms with $\lambda^{\text{right}}_{p(m)}$ can be combined into a single term:

$$\lambda^{\text{right}}_{p(m)}\sum_{k=1}^{m}\sum_{l=m+1}^{n} w_{p(k)p(l)}, \quad m \in \{1, \ldots, n-1\}. \tag{6}$$

Finally, the terms with $\gamma_{p(m)p(m+1)}$ contribute

$$\gamma_{p(m)p(m+1)}\sum_{k=1}^{m}\sum_{l=m+1}^{n} w_{p(k)p(l)}, \quad m \in \{1, \ldots, n-1\}. \tag{7}$$

The double summation in (6) (as well as (7)) is equal to the cut value c_m and that in (5) is equal to the cut value c_{m-1}. Thus we can write

$$F(p) = \sum_{m=2}^{n}\lambda^{\text{left}}_{p(m)}c_{m-1} + \sum_{m=1}^{n-1}\left(\lambda^{\text{right}}_{p(m)}c_m + \gamma_{p(m)p(m+1)}c_m\right). \tag{8}$$

The first summand in (8) is equal to $\sum_{m=1}^{n-1}\lambda^{\text{left}}_{p(m+1)}c_m$. Hence

$$F(p) = \sum_{m=1}^{n-1} c_m\left(\lambda^{\text{right}}_{p(m)} + \lambda^{\text{left}}_{p(m+1)} + \gamma_{p(m)p(m+1)}\right). \tag{9}$$

After getting rid of the "right" and "left" superscripts in (9), we arrive at (3). □

For a permutation $p \in \Pi$, the components of the cut vector C can be computed using the following recurrence:

$$c_m = c_{m-1} - \sum_{k=1}^{m-1} w_{p(m)p(k)} + \sum_{l=m+1}^{n} w_{p(m)p(l)}, \quad m = 1, \ldots, n-1, \tag{10}$$

with the initial condition $c_0 = 0$. To show (10), we assume that $m > 1$. From the definition of c_m, we have

$$c_m = \sum_{k=1}^{m-1}\sum_{l=m+1}^{n} w_{p(k)p(l)} + \sum_{l=m+1}^{n} w_{p(m)p(l)}. \tag{11}$$

Adding to and subtracting from the right-hand side of (11) the sum $\sum_{k=1}^{m-1} w_{p(k)p(m)}$ gives

$$c_m = \sum_{k=1}^{m-1}\sum_{l=m}^{n} w_{p(k)p(l)} - \sum_{k=1}^{m-1} w_{p(k)p(m)} + \sum_{l=m+1}^{n} w_{p(m)p(l)}. \tag{12}$$

By replacing the double summation in (12) with c_{m-1} we obtain (10).

Given a cut vector $C = (c_1, \ldots, c_{n-1})$, we denote by $C^- = (c^-_1, \ldots, c^-_n)$ the n-vector with components $c^-_1 = c_1$, $c^-_m = c_m - c_{m-1}$ for $m = 2, \ldots, n-1$, and $c^-_n = -c_{n-1}$. We similarly define a vector C^+: $c^+_1 = c_1$, $c^+_m = c_m + c_{m-1}$ for $m = 2, \ldots, n-1$, and $c^+_n = c_{n-1}$. We will also use two $n \times n$ matrices computed from the lengths of facilities: (λ^-_{rs}) with entries $\lambda^-_{rs} = \lambda_r - \lambda_s = (L_r - L_s)/2$ and (λ^+_{rs}) with entries $\lambda^+_{rs} = \lambda_r + \lambda_s$. We will write $D = (d_{rs})$ to denote the distance matrix, where d_{rs} for $r = p(k)$ and $s = p(l)$ is given by (2). The components of the vectors C^-, C^+ and entries of the matrices (λ^-_{rs}), (λ^+_{rs}) are used in formulas to calculate the change in the objective function value resulting from applying a pairwise interchange or insertion move (see (15) and (24)–(27) in the next two sections). In fact, the purpose of the introduction of C^-, C^+, (λ^-_{rs}) and (λ^+_{rs}) is twofold. First, the above-mentioned formulas become slightly shorter. Second, and more importantly, the usage of the auxiliary arrays and matrices allows us to reduce the execution time of the algorithms based on our approach. For example, at each iteration of interchange-based local search, the vectors C^- and C^+ are computed only once and then used $O(n^2)$ times while examining pairs of n facilities. When vectors C^- and C^+ are not maintained, then extra addition/subtraction operations are needed. We also note that the matrices (λ^-_{rs}) and (λ^+_{rs}) can be precomputed in advance.

In the next sections, we derive some formulas related to the LS algorithms. In order to keep them compact, we make the following conventions: $p(0) = 0$, $p(n+1) = 0$, $c_0 = 0$, $c_n = 0$, and $\gamma_{0r} = \gamma_{r0} = 0$ for each $r = 1, \ldots, n$.

3. Interchange-based local search

In this section, we describe an implementation of the local search algorithm based on a fast procedure for exploring the pairwise interchange neighborhood of a SRFLP solution in the search space. Our aim is to provide an efficient way to compare the quality of two solutions differing in the positions of precisely two facilities. For this purpose, a couple of auxiliary matrices along with some other data are used.

For $p \in \Pi$, let $\Delta(p, k, l)$ denote the change in the objective function value that will result from swapping positions of the facilities $p(k)$ and $p(l)$ in the permutation p. Formally, let p' be the permutation obtained by performing this swapping operation. Then $\Delta(p, k, l) = F(p') - F(p)$. In order to present a fast method for computing $\Delta(p, k, l)$ we introduce two $n \times n$ matrices denoted by $A = (a_{uj})$ and $B = (b_{uj})$. Both matrices are associated with a permutation of facilities, say $p \in \Pi$. The rows of the matrices correspond to facilities and the columns correspond to permutation positions. To define the matrices A and B, we suppose that the facility u is assigned to the m-th position of the permutation p, that is, $u = p(m)$. The entry a_{uj} of the matrix A represents the total flow of material between facility u and facilities in the segment of p from position m exclusively to position j inclusively. Formally, for $u, j \in \{1, \ldots, n\}$

$$a_{uj} = \begin{cases} \sum_{i=j}^{m-1} w_{up(i)} & \text{if } j < m \\ \sum_{i=m+1}^{j} w_{up(i)} & \text{if } j > m \\ 0 & \text{if } j = m. \end{cases} \tag{13}$$

The entries of the matrix B are computed by summing over the same permutation positions as in the case of the matrix A. The summand for $i \neq j$ is a product of a_{ui} and the distance between the centroids of facilities $p(i)$ and either $p(i-1)$ or $p(i+1)$ depending on whether $p(i)$ is to the left or the right of the facility u. If $i = j$, then calculations are performed assuming that u substitutes for $p(i-1)$ (or $p(i+1)$). Thus, the entry b_{uj}, $u, j \in \{1, \dots, n\}$, of the matrix B is defined as follows:

$$b_{uj} = \begin{cases} a_{uj}(\lambda_u + \lambda_{p(j)} + \gamma_{up(j)}) \\ \quad + \sum_{i=j+1}^{m-1} a_{ui}(\lambda_{p(i-1)} + \lambda_{p(i)} + \gamma_{p(i-1)p(i)}) & \text{if } j < m \\ a_{uj}(\lambda_{p(j)} + \lambda_u + \gamma_{p(j)u}) \\ \quad + \sum_{i=m+1}^{j-1} a_{ui}(\lambda_{p(i)} + \lambda_{p(i+1)} + \gamma_{p(i)p(i+1)}) & \text{if } j > m \\ 0 & \text{if } j = m. \end{cases} \tag{14}$$

The matrices A and B arise when manipulating the expression for $\Delta(p, k, l)$. They also play an important role in constructing fast insertion-based local search algorithms for the problem (see Section 4).

Next we provide a formula to calculate the value of $\Delta(p, k, l)$.

Proposition 2. *For $p \in \Pi$, $k \in \{1, \dots, n-1\}$, $l \in \{k+1, \dots, n\}$, $r = p(k)$, and $s = p(l)$,*

$$\begin{aligned}\Delta(p,k,l) = {} & (c_l^- - c_k^- + 2w_{rs})(d_{rs} + \gamma^1 + \gamma^2) + 2(b_{r,l-1} + b_{s,k+1}) \\ & + \lambda_{rs}^-(c_l^+ - c_k^+) + c_{k-1}(\gamma_{p(k-1)s} - \gamma_{p(k-1)r}) \\ & + c_k\gamma^1 + c_{l-1}\gamma^2 + c_l(\gamma_{rp(l+1)} - \gamma_{sp(l+1)}),\end{aligned} \tag{15}$$

where

$$\gamma^1 = \begin{cases} \gamma_{sp(k+1)} - \gamma_{rp(k+1)} & \text{if } l > k+1 \\ \gamma_{sr} - \gamma_{rs} & \text{if } l = k+1, \end{cases}$$

$$\gamma^2 = \begin{cases} \gamma_{p(l-1)r} - \gamma_{p(l-1)s} & \text{if } l > k+1 \\ 0 & \text{if } l = k+1. \end{cases}$$

The proof of this result can be found in Appendix A. Notice that in the case of zero clearances between neighboring facilities the right-hand side of (15) becomes much simpler. Indeed, in this case, the last four terms as well as γ^1 and γ^2 in its first term vanish.

Armed with the formula for $\Delta(p, k, l)$, we are now ready to present our first local search algorithm for the SRFLP. We start with a description of its main ingredient, a neighborhood exploration procedure, called NE1 (we add the digit 1 at the end of NE to distinguish between this and two other procedures which are provided in the next section).

```
NE1($p, C, C^-, C^+, A, B, D$)

1. Set $h^* := 0$.
2. For each pair $(k, l)$, $k = 1, \dots, n-1$, $l = k+1, \dots, n$, do the following:
   2.1. Compute $h = \Delta(p, k, l)$ by Eq. (15).
   2.2. Check whether $h < h^*$. If so, then set $h^* := h$, $k' := k$ and $l' := l$.
3. If $h^* = 0$, then return with $\rho$ set to 1. Otherwise, perform pairwise interchange of facilities $p(k')$ and $p(l')$ in the permutation $p$, update matrices $A, B, D$ and vectors $C, C^-, C^+$ used in $\Delta$ calculations, and return with $\rho = 0$.
```

The described procedure takes a permutation p, vectors C, C^-, C^+ and matrices A, B, D as input parameters and searches for the best solution in the interchange neighborhood of p defined as $N_2(p) = \{p' \in \Pi \mid p'(m) \neq p(m)$ for precisely two positions $m\}$. Such a solution is specified by position indices k' and l'. The return value $\rho = 1$ says that p is locally optimal with respect to $N_2(p)$. On the other hand, $\rho = 0$ means that p has been replaced by a better permutation. We note that the matrix A is used indirectly via the matrix B. The entries of A are not present in (15), however, they are needed to update the matrix B (the formulas derived from (14) will be given later in this section). Using NE1, implementation of the local search algorithm is straightforward as given below.

```
LS1($p$)

1. Initialize matrices $A, B, D$ and vectors $C, C^-$ and $C^+$.
2. Apply NE1($p, C, C^-, C^+, A, B, D$). Let $\rho$ denote the local optimality flag returned by NE1.
3. Check whether $\rho = 1$. If so, then stop with the solution $p$. Otherwise, go to 2.
```

An important question that remains to be addressed is how efficiently the entries of the matrices A, B and D can be updated (Step 3 of NE1) after the replacement of p by a permutation p' chosen from the neighborhood $N_2(p)$. Suppose that p' is obtained from p by interchanging the facilities r and s. Let $r = p(k)$, $s = p(l)$, $k < l$, and $u = p'(m)$, $m \in \{1, \dots, n\}$. Then, we can write the following formulas that are involved in updating the matrices A, B and D:

$$a_{uj} = a_{u,j-1} + w_{up'(j)}, \quad j > m, \tag{16}$$

$$a_{uj} = a_{u,j+1} + w_{up'(j)}, \quad j < m, \tag{17}$$

$$\begin{aligned} b_{uj} = {} & b_{u,j-1} + a_{u,j-1}\big(\lambda_{p'(j)u}^- + \gamma_{p'(j-1)p'(j)} - \gamma_{p'(j-1)u}\big) \\ & + a_{uj}\big(\lambda_{p'(j)u}^+ + \gamma_{p'(j)u}\big), \quad j > m, \end{aligned} \tag{18}$$

$$\begin{aligned} b_{uj} = {} & b_{u,j+1} + a_{u,j+1}\big(\lambda_{p'(j)u}^- + \gamma_{p'(j)p'(j+1)} - \gamma_{up'(j+1)}\big) \\ & + a_{uj}(\lambda_{p'(j)u}^+ + \gamma_{up'(j)}), \quad j < m, \end{aligned} \tag{19}$$

$$d_{up'(j)} = d_{up'(j-1)} + \lambda_{p'(j-1)p'(j)}^+ + \gamma_{p'(j-1)p'(j)}, \quad j > m. \tag{20}$$

To show (16), we decompose the right-hand side of (13):

$$a_{uj} = \sum_{i=m+1}^{j} w_{up'(i)} = \sum_{i=m+1}^{j-1} w_{up'(i)} + w_{up'(j)} = a_{u,j-1} + w_{up'(j)}.$$

Eq. (17) is derived similarly (the first case of the right-hand side of (13) is used). To obtain (18), we apply the same trick with respect to (14):

$$\begin{aligned} b_{uj} = {} & a_{uj}(\lambda_{p'(j)} + \lambda_u + \gamma_{p'(j)u}) \\ & + \sum_{i=m+1}^{j-2} a_{ui}(\lambda_{p'(i)} + \lambda_{p'(i+1)} + \gamma_{p'(i)p'(i+1)}) \\ & + a_{u,j-1}(\lambda_{p'(j-1)} + \lambda_{p'(j)} + \gamma_{p'(j-1)p'(j)}). \end{aligned} \tag{21}$$

Adding to and subtracting from (21) the term $a_{u,j-1}(\lambda_u + \gamma_{p'(j-1)u})$ gives

$$\begin{aligned} b_{uj} = {} & a_{u,j-1}(\lambda_{p'(j-1)} + \lambda_u + \gamma_{p'(j-1)u}) \\ & + \sum_{i=m+1}^{j-2} a_{ui}(\lambda_{p'(i)} + \lambda_{p'(i+1)} + \gamma_{p'(i)p'(i+1)}) \\ & + a_{u,j-1}(\lambda_{p'(j)} + \gamma_{p'(j-1)p'(j)} - \lambda_u - \gamma_{p'(j-1)u}) \\ & + a_{uj}(\lambda_{p'(j)} + \lambda_u + \gamma_{p'(j)u}) \\ = {} & b_{u,j-1} + a_{u,j-1}(\lambda_{p'(j)u}^- + \gamma_{p'(j-1)p'(j)} - \gamma_{p'(j-1)u}) \\ & + a_{uj}\big(\lambda_{p'(j)u}^+ + \gamma_{p'(j)u}\big). \end{aligned}$$

To show (19) we proceed analogously and obtain

$$\begin{aligned} b_{uj} = {} & a_{uj}(\lambda_u + \lambda_{p'(j)} + \gamma_{up'(j)}) \\ & + \sum_{i=j+2}^{m-1} a_{ui}(\lambda_{p'(i-1)} + \lambda_{p'(i)} + \gamma_{p'(i-1)p'(i)}) \\ & + a_{u,j+1}(\lambda_{p'(j)} + \lambda_{p'(j+1)} + \gamma_{p'(j)p'(j+1)}). \end{aligned}$$

Continuing as in the previous case, we find that

$$\begin{aligned} b_{uj} &= a_{u,j+1}(\lambda_u + \lambda_{p'(j+1)} + \gamma_{up'(j+1)}) \\ &\quad + \sum_{i=j+2}^{m-1} a_{ui}(\lambda_{p'(i-1)} + \lambda_{p'(i)} + \gamma_{p'(i-1)p'(i)}) \\ &\quad + a_{u,j+1}(\lambda_{p'(j)} + \gamma_{p'(j)p'(j+1)} - \lambda_u - \gamma_{up'(j+1)}) \\ &\quad + a_{uj}(\lambda_u + \lambda_{p'(j)} + \gamma_{up'(j)}) \\ &= b_{u,j+1} + a_{u,j+1}(\lambda^-_{p'(j)u} + \gamma_{p'(j)p'(j+1)} - \gamma_{up'(j+1)}) \\ &\quad + a_{uj}\big(\lambda^+_{p'(j)u} + \gamma_{up'(j)}\big). \end{aligned}$$

Finally, we prove (20). Using (2), we can compute the distance between facilities u and $p'(j)$

$$d_{up'(j)} = \lambda_u + \sum_{i=m+1}^{j-1} L_{p'(i)} + \lambda_{p'(j)} + \sum_{i=m}^{j-1} \gamma_{p'(i)p'(i+1)}.$$

Equivalently, we can write

$$\begin{aligned} d_{up'(j)} &= \lambda_u + \sum_{i=m+1}^{j-2} L_{p'(i)} + 2\lambda_{p'(j-1)} + \lambda_{p'(j)} \\ &\quad + \sum_{i=m}^{j-2} \gamma_{p'(i)p'(i+1)} + \gamma_{p'(j-1)p'(j)}. \end{aligned} \tag{22}$$

Replacing $\lambda_u + \sum_{i=m+1}^{j-2} L_{p'(i)} + \lambda_{p'(j-1)} + \sum_{i=m}^{j-2} \gamma_{p'(i)p'(i+1)}$ in (22) by $d_{up'(j-1)}$, we arrive at (20).

The procedure for updating entries of A, B and D related to facility $u = p'(m)$ is summarized below:

1. If $m < k$, then compute a_{uj}, $j = k, \ldots, l-1$, by (16), b_{uj}, $j = k, \ldots, n-1$, by (18), and $d_{up'(j)} = d_{p'(j)u}$, $j = k, \ldots, n$, by (20). Go to 4.
2. If $m > l$, then compute a_{uj}, $j = l, l-1, \ldots, k+1$, by (17) and b_{uj}, $j = l, l-1, \ldots, 2$, by (19). Go to 4.
3. If $k < m < l$, then set $j_1 := k$ and $j_2 := l$. Otherwise, set $j_1 := m-1$, $j_2 := m+1$, $a_{um} := 0$ and $b_{um} := 0$. Compute a_{uj}, $j = j_1, j_1 - 1, \ldots, 1$, by (17) and b_{uj}, $j = j_1, j_1 - 1, \ldots, 2$, by (19). Also, compute a_{uj}, $j = j_2, \ldots, n$, by (16), b_{uj}, $j = j_2, \ldots, n-1$, by (18), and $d_{up'(j)} = d_{p'(j)u}$, $j = j_2, \ldots, n$, by (20).
4. Stop.

Additionally, in Step 3 of NE1, the cut vector C is updated. The computation is based on the following variation of (10):

$$c_m = c_{m-1} + a_{p'(m)n} - a_{p'(m)1}, \tag{23}$$

where $m = k, \ldots, l-1$.

Another remark is that the formulas (16)–(19) and (23) (with p' replaced by p) can also be used to initialize the matrices A, B and vector C in Step 1 of LS1. Notice that, before applying (16)–(19), the entries $a_{p(j)j}$ and $b_{p(j)j}$ for each $j = 1, \ldots, n$ must be set to zero. It is easy to see that the initialization step of the local search algorithm takes $O(n^2)$ time.

Example 1. Let us consider the 5-facility SRFLP instance taken from Simmons (1969). The flows of material between facilities are given by the following matrix:

$$W = \begin{bmatrix} 0 & 2 & 1 & 0 & 1 \\ 2 & 0 & 0 & 2 & 2 \\ 1 & 0 & 0 & 6 & 3 \\ 0 & 2 & 6 & 0 & 4 \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}.$$

The vector of lengths of facilities is (1, 3, 4, 6, 7). In order to avoid fractional numbers, we multiply its components by 2. Thus, $L_1 = 2$, $L_2 = 6$, $L_3 = 8$, $L_4 = 12$, and $L_5 = 14$. There are no clearances between facilities specified in this instance, so the formula (15) for calculating Δ values becomes simpler.

Suppose that LS1 is applied to the permutation $p = (3, 5, 1, 2, 4)$ (the corresponding layout is depicted in the left part of Fig. 1). Taking into account the fact that we doubled the length of each facility, the objective function value of the solution p is 386. Step 1 of LS1 initializes the data needed for the procedure NE1. Starting with $a_{p(j)j} = 0$ for all j and using (16) and (17), the matrix A for p is constructed:

$$A = \begin{bmatrix} 2 & 1 & 0 & 2 & 2 \\ 4 & 4 & 2 & 0 & 2 \\ 0 & 3 & 4 & 4 & 10 \\ 12 & 6 & 2 & 2 & 0 \\ 3 & 0 & 1 & 3 & 7 \end{bmatrix}.$$

Then, using (18) and (19), the entries of the matrix B can be computed:

$$B = \begin{bmatrix} 21 & 8 & 0 & 8 & 32 \\ 88 & 56 & 8 & 0 & 18 \\ 0 & 33 & 44 & 68 & 176 \\ 210 & 102 & 22 & 18 & 0 \\ 33 & 0 & 8 & 34 & 122 \end{bmatrix}.$$

The distance matrix corresponding to p is as follows:

$$D = \begin{bmatrix} 0 & 4 & 19 & 13 & 8 \\ 4 & 0 & 23 & 9 & 12 \\ 19 & 23 & 0 & 32 & 11 \\ 13 & 9 & 32 & 0 & 21 \\ 8 & 12 & 11 & 21 & 0 \end{bmatrix}.$$

The cut vector is $C = (10, 14, 14, 12)$. From it, we obtain the vectors $C^- = (10, 4, 0, -2, -12)$ and $C^+ = (10, 24, 28, 26, 12)$. Now, using (15), the value of Δ for each pair of facilities can easily be calculated. For example, consider the second and fifth facilities in the permutation p. In this case, $k = 2$, $l = 5$, $r = 5$, $s = 4$, and $\lambda^-_{5,4} = 7 - 6 = 1$. From (15), we obtain $\Delta(p, 2, 5) = (-12 - 4 + 2 \cdot 4)21 + 2(34 + 22) + 1 \cdot (12 - 24) = -68$. Other Δ values are calculated similarly: $\Delta(p, 1, 2) = -42$, $\Delta(p, 1, 3) = -16$, $\Delta(p, 1, 4) = -60$, $\Delta(p, 1, 5) = 16$, $\Delta(p, 2, 3) = 8$, $\Delta(p, 2, 4) = 16$, $\Delta(p, 3, 4) = 12$, $\Delta(p, 3, 5) = -24$, $\Delta(p, 4, 5) = -12$. The maximum decrease in objective function value is achieved by interchanging the second and fifth facilities in p (the obtained layout is shown in the right part of Fig. 1). In Step 3 of NE1, the matrices and vectors are updated to correspond to the new permutation $p = (3, 4, 1, 2, 5)$. This is done using the above given four-step procedure based on formulas (16)–(20). The matrix A takes the following form:

$$A = \begin{bmatrix} 1 & 0 & 0 & 2 & 3 \\ 4 & 4 & 2 & 0 & 2 \\ 0 & 6 & 7 & 7 & 10 \\ 6 & 0 & 0 & 2 & 6 \\ 10 & 7 & 3 & 2 & 0 \end{bmatrix}.$$

A relevant part of the matrix B is given below (the entries of B in the first and last columns, except $b_{p(1)1}$ and $b_{p(n)n} = b_{p(5)5}$, are not used in (15) and are not recalculated by the algorithm; therefore, we replace them by asterisks):

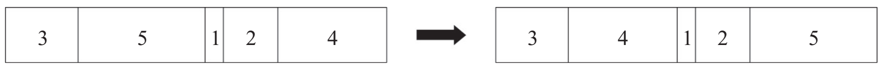

Fig. 1. Pairwise interchange of facilities 4 and 5.

$$B = \begin{bmatrix} * & 0 & 0 & 8 & * \\ * & 50 & 8 & 0 & * \\ 0 & 60 & 77 & 119 & * \\ * & 0 & 0 & 18 & * \\ * & 120 & 32 & 20 & 0 \end{bmatrix}.$$

The new cut vector is $C = (10, 10, 12, 10)$. In the next iteration, NE1 explores the interchange neighborhood of the updated permutation p. This is done in the same way as we have described above.

We end the section with the following statement concerning the effectiveness of our approach.

Proposition 3. *The computational complexity of the procedure NE1 is $O(n^2)$.*

Proof. First, notice that the vectors C, C^-, C^+ and matrices B and D remain unchanged during execution of Step 2 of NE1. These vectors and matrices are passed as parameters to the procedure. Thus, using (15), $\Delta(p, k, l)$ can be computed in Step 2 in constant time. Hence, the complexity of Step 2 of NE1 is $O(n^2)$. If $h^* < 0$, then NE1 updates the data used in Δ calculations. These data will be returned to NE1 at the next call to this procedure in Step 2 of LS1. In order to update the matrices A, B and D, Step 3 of NE1 applies the above-described four-step method based on equations (16)–(20). It is clear that, for each facility, this method runs in $O(n)$ time. Therefore, in total, it performs $O(n^2)$ operations. Since the vectors C, C^- and C^+ can be updated in linear time using (23), the overall complexity of Step 3, and hence of the entire procedure, is $O(n^2)$. □

4. Insertion-based local search

Another strategy in the development of local search techniques for the SRFLP is to use the insertion neighborhood structure. We will describe two practical procedures for exploring this neighborhood. The first one takes advantage of the matrices A and B introduced in the previous section. The second procedure is somewhat simpler because it does not use these two auxiliary matrices. However, this procedure suffers from the restriction imposed on the order in which solutions are explored.

4.1. The first approach for the insertion neighborhood exploration

For $p \in \Pi$, $k, l \in \{1, \ldots, n\}$, $k \neq l$, let p' be the permutation obtained from p by removing the facility $p(k)$ from position k and inserting it at position l. Let us denote the resulting change in the objective function value by $\delta(p, k, l) = F(p') - F(p)$. An efficient way to compute $\delta(p, k, l)$ is provided by the proposition stated below and proved in Appendix A.

Proposition 4. *Let $p \in \Pi$, $k, l \in \{1, \ldots, n\}$, $k \neq l$, and $r = p(k)$. Then, for $l > k$,*

$$\begin{aligned} \delta(p, k, l) = {} & 2b_{rl} - c_k^-(d_{p(k+1)p(l)} + \lambda^+_{p(k+1)p(l)}) + L_r(c_l - c_k) \\ & + c_{k-1}(\gamma_{p(k-1)p(k+1)} + \gamma_{p(l)r} - \gamma_{p(k-1)r}) \\ & - c_k(\gamma_{rp(k+1)} + \gamma_{p(l)r}) \\ & + c_l(\gamma_{p(l)r} + \gamma_{rp(l+1)} - \gamma_{p(l)p(l+1)}) \end{aligned} \tag{24}$$

and, for $l < k$,

$$\begin{aligned} \delta(p, k, l) = {} & 2b_{rl} + c_k^-(d_{p(l)p(k-1)} + \lambda^+_{p(l)p(k-1)}) + L_r(c_{l-1} - c_{k-1}) \\ & + c_{l-1}(\gamma_{p(l-1)r} + \gamma_{rp(l)} - \gamma_{p(l-1)p(l)}) \\ & - c_{k-1}(\gamma_{rp(l)} + \gamma_{p(k-1)r}) \\ & + c_k(\gamma_{rp(l)} + \gamma_{p(k-1)p(k+1)} - \gamma_{rp(k+1)}). \end{aligned} \tag{25}$$

We will refer to the neighborhood exploration and local search algorithms based on (24) and (25) as NE2 and LS2, respectively. The former is similar to NE1 with a few differences. First, Step 2 is executed for all pairs (k, l) such that $k, l \in \{1, \ldots, n\}$ and $k \neq l$. Second, of course, Δ is replaced by δ calculated using either Eq. (24) or (25) depending on whether $l > k$ or $l < k$. Third, in Step 3, the permutation p is updated by relocating the facility $p(k')$ from position k' to position l'. Fourth, the vector C^+ is eliminated. The LS2 algorithm is almost identical to LS1. The only differences are that the procedure NE2 instead of NE1 is invoked and the vector C^+ is not used.

The routine for updating the matrices A, B and D is slightly more complicated than in the case of pairwise interchanges. For their entries corresponding to facility $u = p'(m)$, it goes as follows.

1. Let $j_1 = \min(k, l)$, $j_2 = \max(k, l)$.
2. If $m < j_1$, then compute a_{uj}, $j = j_1, \ldots, j_2 - 1$, by (16), b_{uj}, $j = j_1, \ldots, n$, by (18), and $d_{up'(j)} = d_{p'(j)u}$, $j = j_1, \ldots, n$, by (20). Go to 6.
3. If $m > j_2$, then compute a_{uj}, $j = j_2, j_2 - 1, \ldots, j_1 + 1$, by (17) and b_{uj}, $j = j_2, j_2 - 1, \ldots, 1$, by (19). Go to 6.
4. If $m = l$ (that is, $u = p'(l) = p(k)$), then perform the following operations. Set $a_{um} := 0$ and $b_{um} := 0$. For $j = l - 1, l - 2, \ldots, 1$, compute a_{uj} by (17) and b_{uj} by (19). Also, for $j = l + 1, \ldots, n$, compute a_{uj} by (16), b_{uj} by (18) and $d_{up'(j)} = d_{p'(j)u}$ by (20). Go to 6.
5. If $k < l$, then set $j_1 := k - 1$, $j_2 := l$ and $a_{uj} := a_{u,j+1}$, $b_{uj} := b_{u,j+1}$ for $j = k, \ldots, l - 1$. Otherwise, set $j_1 := l$, $j_2 := k + 1$ and $a_{uj} := a_{u,j-1}$, $b_{uj} := b_{u,j-1}$ for $j = k, k - 1, \ldots, l + 1$. In both cases, for $j = j_1, j_1 - 1, \ldots, 1$, compute a_{uj} by (17) and b_{uj} by (19). Also, for $j = j_2, \ldots, n$, compute a_{uj} by (16), b_{uj} by (18), and $d_{up'(j)} = d_{p'(j)u}$ by (20).
6. Stop.

We note that the cut vector C can be updated in the same way as it is done in the procedure NE1. However, unlike NE1, the procedure NE2 does not maintain the vector C^+. Owing to the similarity between NE1 and NE2, the following result should be clear.

Proposition 5. *The computational complexity of the procedure NE2 is $O(n^2)$.*

Example 2. Consider the same SRFLP instance as in Example 1. Suppose that LS2 and NE2 are applied to the same permutation as before, that is, $p = (3, 5, 1, 2, 4)$. To illustrate Proposition 5, we compute the value of δ for the case when the last facility in p is moved to the first position (the layouts corresponding to p as well as to the new permutation are shown in Fig. 2). In this case, $k = 5$, $l = 1$, $r = 4$, and $\lambda^+_{p(l)p(k-1)} = \lambda^+_{3,2} = 7$. The initial matrices A, B, D and vectors C and C^- are the same as in Example 1. Therefore, using (25), we have $\delta(p, 5, 1) = 2 \cdot 210 + (-12)(23 + 7) + 12(0 - 12) = -84$. Other δ values are as follows: $\delta(p, 1, 2) = -42$, $\delta(p, 1, 3) = -40$, $\delta(p, 1, 4) = -68$, $\delta(p, 1, 5) = -68$, $\delta(p, 2, 1) = -42$, $\delta(p, 2, 3) = 8$, $\delta(p, 2, 4) = 8$, $\delta(p, 2, 5) = -32$, $\delta(p, 3, 1) = 14$, $\delta(p, 3, 2) = 8$, $\delta(p, 3, 4) = 12$, $\delta(p, 3, 5) = 36$, $\delta(p, 4, 1) = 44$, $\delta(p, 4, 2) = 56$, $\delta(p, 4, 3) = 12$, $\delta(p, 4, 5) = -12$, $\delta(p, 5, 2) = -84$, $\delta(p, 5, 3) = -28$, $\delta(p, 5, 4) = -12$. Since $\delta(p, 5, 1) \leq \delta(p, k, l)$ for all $k, l = 1, \ldots, 5$, $k \neq l$, facility 4 is moved to position 1. Then, using the above given six-step procedure based on (16)–(20), the matrices A and B are updated:

$$A = \begin{bmatrix} 2 & 2 & 1 & 0 & 2 \\ 6 & 4 & 4 & 2 & 0 \\ 6 & 0 & 3 & 4 & 4 \\ 0 & 6 & 10 & 10 & 12 \\ 7 & 3 & 0 & 1 & 3 \end{bmatrix},$$

$$B = \begin{bmatrix} 45 & 21 & 8 & 0 & 8 \\ 154 & 88 & 56 & 8 & 0 \\ 60 & 0 & 33 & 44 & 68 \\ 0 & 60 & 196 & 216 & 294 \\ 121 & 33 & 0 & 8 & 34 \end{bmatrix}.$$

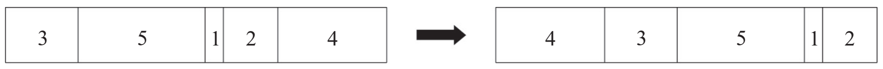

Fig. 2. Relocating facility 4 to position 1.

The new cut vector is $C = (12, 10, 6, 6)$. The resulting layout is shown in the right part of Fig. 2. The corresponding objective function value with respect to original (not doubled) facility lengths is 151. We notice that the obtained layout is optimal (see Amaral (2006)).

4.2. The second approach for the insertion neighborhood exploration

Another efficient method for exploring the insertion neighborhood rests on the idea to include the value of δ, calculated in the previous step, in an expression for $\delta(p, k, l)$. In the following, we present our realization of this idea.

Proposition 6. *Let $p \in \Pi$, $k \in \{1, \ldots, n\}$, and $r = p(k)$. Then, for $l = k+1, k+2, \ldots, n$,*

$$\begin{aligned}\delta(p,k,l) = {} & \delta(p,k,l-1) + L_r(w_{rp(l)} + c_l^-) + L_{p(l)}(e_{l-1} + e_l - c_k^-) \\ & + 2w_{p(l)r}\gamma_{p(l)r} + (2e_{l-1} - c_k^-)(\gamma_{up(l)} + \gamma_{p(l)r} - \gamma_{ur}) \\ & - c_{l-1}(\gamma_{ur} + \gamma_{rp(l)} - \gamma_{up(l)}) \\ & + c_l(\gamma_{p(l)r} + \gamma_{rp(l+1)} - \gamma_{p(l)p(l+1)}), \qquad (26)\end{aligned}$$

where $e_l = e_{l-1} + w_{rp(l)}$, $u = p(k-1)$ if $l = k+1$, $u = p(l-1)$ if $l \neq k+1$, and initially $\delta(p,k,k) = 0$ and $e_k = 0$.

For $l = k-1, k-2, \ldots, 1$,

$$\begin{aligned}\delta(p,k,l) = {} & \delta(p,k,l+1) + L_r(w_{rp(l)} - c_l^-) + L_{p(l)}(e_l + e_{l+1} + c_k^-) \\ & + 2w_{rp(l)}\gamma_{rp(l)} + (2e_{l+1} + c_k^-)(\gamma_{rp(l)} + \gamma_{p(l)u} - \gamma_{ru}) \\ & + c_{l-1}(\gamma_{p(l-1)r} + \gamma_{rp(l)} - \gamma_{p(l-1)p(l)}) \\ & - c_l(\gamma_{p(l)r} + \gamma_{ru} - \gamma_{p(l)u}), \qquad (27)\end{aligned}$$

where $e_l = e_{l+1} + w_{rp(l)}$, $u = p(k+1)$ if $l = k-1$, $u = p(l+1)$ if $l \neq k-1$, and initial conditions are as in (26).

The proof of Proposition 6 is given in Appendix A. When the clearance matrix Γ is zero, Eqs. (26) and (27) take simplified forms. In this case, the right-hand side of each of them, besides the previous value of δ, has only two terms.

Let A_q, $q \in \{1, \ldots, n\}$, denote the q-th column vector of the matrix A. Our third neighborhood exploration procedure for the SRFLP can be described as follows.

```
NE3($p, A_1, A_n, C, C^-$)
1. Set $h^* := 0$.
2. For $k = 1, \ldots, n$ do the following:
   2.1. Set $\delta(p,k,k) := 0$ and $e_k := 0$.
   2.2. For $l = k-1, k-2, \ldots, 1$, perform the following operations.
        Set $e_l := e_{l+1} + w_{p(k)p(l)}$. Compute $h = \delta(p,k,l)$ by Eq. (27). If
        $h < h^*$, then set $h^* := h$, $k' := k$ and $l' := l$.
   2.3. For $l = k+1, \ldots, n$, perform the following operations. Set
        $e_l := e_{l-1} + w_{p(k)p(l)}$. Compute $h = \delta(p,k,l)$ by Eq. (26). If $h <$
        $h^*$, then set $h^* := h$, $k' := k$ and $l' := l$.
3. If $h^* = 0$, then return with $\rho$ set to 1. Otherwise, first remove the
   facility $p(k')$ from position $k'$ and insert it at position $l'$. Then update
   the vectors $A_1$ and $A_n$ as well as the vectors $C$ and $C^-$ used in
   $\delta$ calculations, and return with $\rho = 0$.
```

Example 3. We apply NE3 to the same SRFLP instance as in the previous examples. Again, the initial layout is defined by the permutation $p = (3, 5, 1, 2, 4)$. From Example 1, we know that $C^- = (10, 4, 0, -2, -12)$. The execution of Step 2 of NE3 is illustrated in Table 1, in which e_{previous} stands for e_{l-1} if $l > k$, and for e_{l+1} if $l < k$. Furthermore, δ_{added} is used to denote the sum of the second and third terms on the right-hand side of (26) (or (27)), which is added to $\delta(p, k, l-1)$ (or $\delta(p, k, l+1)$) to obtain $\delta(p, k, l)$.

Table 1
Insertion evaluation using procedure NE3.

k	l	r	e_{previous}	e_l	$\delta(p,k,l-1)$ or $\delta(p,k,l+1)$	δ_{added}	$\delta(p,k,l)$
1	2	3	0	3	0	−42	−42
1	3	3	3	4	−42	2	−40
1	4	3	4	4	−40	−28	−68
1	5	3	4	10	−68	0	−68
2	1	5	0	3	0	−42	−42
2	3	5	0	1	0	8	8
2	4	5	1	3	8	0	8
2	5	5	3	7	8	−40	−32
3	2	1	0	1	0	8	8
3	1	1	1	2	8	6	14
3	4	1	0	2	0	12	12
3	5	1	2	2	12	24	36
4	3	2	0	2	0	12	12
4	2	2	2	4	12	44	56
4	1	2	4	4	56	−12	44
4	5	2	0	2	0	−12	−12
5	4	4	0	2	0	−12	−12
5	3	4	2	2	−12	−16	−28
5	2	4	2	6	−28	−56	−84
5	1	4	6	12	−84	0	−84

Let us consider the variants of relocating the facility $r = 4$ in more detail. The loop in Step 2.2 of NE3 starts with $k = 5$ and $l = 4$. Using (27), we get $\delta_{\text{added}} = \delta(p, 5, 4) = 12(2 - (-2)) + 6(2 + 0 + (-12)) = -12$. When $k = 5$ and $l = 3$, we have $\delta_{\text{added}} = 12(0 - 0) + 2(2 + 2 + (-12)) = -16$ and $\delta(p, 5, 3) = -12 - 16 = -28$. Similarly, for $l = 2$, $\delta_{\text{added}} = 12(4 - 4) + 14(6 + 2 + (-12)) = -56$ and $\delta(p, 5, 2) = -28 - 56 = -84$. Finally, for $l = 1$, $\delta_{\text{added}} = 12(6 - 10) + 8(12 + 6 + (-12)) = 0$ and $\delta(p, 5, 1) = -84$. The results of these calculations are summarized in the last four rows of Table 1. By moving facility 4 to either position 1 or 2 we obtain an optimal solution.

It is important to note that NE3 requires significantly less data to calculate δ in comparison with NE2. Indeed, no permutation-dependent matrix is involved in δ calculations (see (26) and (27)). Instead, only four vectors, C, C^-, A_1 and A_n, need to be maintained. Let us assume that $h^* < 0$ in Step 3 of the procedure NE3, and let the permutation obtained by shifting the facility $r = p(k')$ to position l' be denoted by p'. Having p', the procedure first updates a_{u1} and a_{un} for each facility $u = p(m)$ such that its position m in p' belongs to the interval $[\min(k', l'), \max(k', l')]$. Suppose that $k' < l'$ (the case of $k' > l'$ can be processed similarly). If $m \neq l'$, then a_{u1} is decreased by w_{ur} and a_{un} is increased by the same value. Clearly, NE3 spends constant time per facility in this case. If however $m = l'$, then $u = r$, and a_{u1} is increased (and a_{un} decreased) by the sum $\sum_{j=k'}^{l'-1} w_{up'(j)}$. Since summation is performed only once, the computational complexity of updating vectors A_1 and A_n is $O(n)$. Once A_1 and A_n are ready, the procedure can use (23) to update the cut vector C (or more precisely, its components with indices in the range $\min(k', l')$ through $\max(k', l') - 1$). Finally, the vector C^- is computed. Updating C and C^- has linear-time complexity in the worst case. Summarizing, we can state that Step 3 of NE3 takes only $O(n)$ time. The efficiency of the entire procedure is determined by that of Step 2, where a double loop is performed. Thus, we have the following result.

Proposition 7. *The computational complexity of the procedure NE3 is $O(n^2)$.*

In what follows, we let LS3 denote the local search algorithm relying on the procedure NE3. Its structure is the same as that of LS1 and LS2.

The main difference between two approaches described in this section is the order in which the algorithms examine candidate positions for facilities. In the case of NE3, for each facility, there are generally two ordered sequences of positions, and the algorithm is forced to follow them. Meanwhile, in NE2, the pairs of type (facility, its new position) are allowed to be considered in an arbitrary order. However, such flexibility in the search process is achieved at the expense of an increase in the amount of computation due to the necessity of maintaining the matrices A, B and D.

5. Numerical comparison

The goal of this section is to provide empirical evidence of the effectiveness of our local search algorithms for the SRFLP. For comparison purposes, we also implemented two local search methods of Kothari and Ghosh (2013a). One of them minimizes the objective function of the SRFLP by performing pairwise interchanges of facilities, while another does the same using the insertion neighborhood structure. By adopting the same naming style as in Kothari and Ghosh (2013a), we refer to these two algorithms as LS-2OPT and LS-INSERT, respectively. The method of experimentation is quite straightforward: we run each of the algorithms in a multi-start mode. An initial permutation of facilities for each restart is generated randomly. The main metric used for comparison is the cumulative CPU time of all restarts.

5.1. Experimental setup

All the algorithms have been coded in C++ using a uniform programming style. The source code of LS1, LS2 and LS3 is publicly available in Palubeckis (2013). The tests have been carried out on a PC with an Intel Core 2 Duo CPU running at 3.0 GHz. As a testbed for the algorithms, two sets of SRFLP instances from the literature as well as additional instances of our own were considered. The first set of benchmark instances was introduced by Anjos et al. (2005). It consists of four subsets, each of cardinality 5. The size of the instances in the first to fourth subsets is 60, 70, 75 and 80, respectively. The second dataset is composed of 20 QAP (quadratic assignment problem)-based *sko* instances (Skorin-Kapov, 1990) tailored for the SRFLP in Anjos and Yen (2009). Their size ranges from 64 to 100 facilities. Both datasets have been used in several recent studies, including Anjos and Yen (2009), Hungerländer and Rendl (2013), Kothari and Ghosh (2013a, 2013b, 2014a, 2014b) and Ozcelik (2012).

In order to test the algorithms more thoroughly, we generated a set of 20 SRFLP instances of larger size (with $n \in [110, 120, \ldots, 300]$). In these instances, the lengths of facilities and the nondiagonal entries of the flow matrix are integer numbers drawn uniformly at random from the intervals [1, 10] and [0, 10], respectively. Of course, the flow matrix is symmetric. The clearances between facilities are assumed to be zero. This choice is conditioned by our intention to compare our algorithms with those of Kothari and Ghosh (2013a, 2013b). These latter algorithms, together with the underlying formulas, are presented for the SRFLP formulation where the clearances between facilities are not taken into account. The third set of SRFLP instances can be found in Palubeckis (2013).

As already mentioned before, the experiments were conducted by running the algorithms in a multi-start mode. For each algorithm, the number of restarts was limited to 500, 1000 and 100 for problem instances in the first, second and third set, respectively. The choice of a smaller number of restarts for the first set, as compared to the second one, is justified by the fact that 500 restarts of insertion-based LS algorithms are sufficient to achieve the best known objective function values for all instances in this set (see Supplementary Appendix B). In order to obtain robust results, we run the algorithms 10 times on each instance in the datasets. Thus, for example, for a benchmark instance of Anjos et al. (2005), we performed 500 restarts of each tested algorithm 10 times, and averaged the results over these 10 runs.

5.2. Comparative results

Table 2 shows the performance comparison of the tested algorithms on the dataset of Anjos et al. (2005). Its first column contains the instance names. The first integer in the name gives the problem dimension, that is, the number of facilities. The second column presents the average time taken by multi-start LS-2OPT. The average is computed over 10 runs of this heuristic. Each run consisted of 500 restarts of LS-2OPT. The standard deviation of the running time for LS-2OPT is displayed in the third column. The remaining columns report the same type of information for other local search algorithms. The overall results, averaged over 20 instances, are presented in the bottom row. The main message of Table 2 concerns the

Table 2
Performance of local search algorithms on the benchmark instances of Anjos et al. (2005): the average running time and its standard deviation σ (both in seconds).

Instance	LS-2OPT		LS-INSERT		LS1		LS2		LS3	
	Time	σ	Time	σ	Time	σ	Time	σ	Time	σ
Anjos-60-01	25.2	0.1	33.6	0.4	1.7	0.02	2.1	0.02	1.0	0.01
Anjos-60-02	25.3	0.1	34.4	0.3	1.7	0.01	2.1	0.02	1.1	0.02
Anjos-60-03	24.7	0.2	32.4	0.3	1.7	0.01	2.0	0.01	1.0	0.01
Anjos-60-04	25.4	0.1	34.4	0.3	1.7	0.01	2.1	0.02	1.1	0.01
Anjos-60-05	25.6	0.1	33.9	0.3	1.7	0.01	2.1	0.02	1.1	0.01
Anjos-70-01	47.7	0.1	62.3	0.2	2.8	0.01	3.4	0.02	1.7	0.01
Anjos-70-02	48.7	0.2	62.9	0.3	2.8	0.03	3.4	0.02	1.7	0.02
Anjos-70-03	48.7	0.2	64.8	0.3	2.8	0.03	3.5	0.02	1.8	0.02
Anjos-70-04	48.0	0.2	65.4	0.4	2.8	0.02	3.6	0.02	1.8	0.01
Anjos-70-05	48.1	0.1	62.3	0.3	2.8	0.01	3.4	0.03	1.7	0.01
Anjos-75-01	62.5	0.3	80.0	0.3	3.4	0.02	4.1	0.02	2.1	0.01
Anjos-75-02	63.5	0.3	81.0	0.2	3.4	0.02	4.2	0.02	2.1	0.02
Anjos-75-03	62.5	0.3	80.7	0.3	3.4	0.02	4.1	0.02	2.1	0.02
Anjos-75-04	62.3	0.2	81.4	0.4	3.4	0.02	4.2	0.02	2.1	0.02
Anjos-75-05	63.4	0.2	84.3	0.4	3.4	0.02	4.3	0.03	2.2	0.02
Anjos-80-01	83.9	0.6	109.9	0.5	4.3	0.04	5.4	0.02	2.7	0.02
Anjos-80-02	83.7	0.3	115.3	0.7	4.3	0.02	5.6	0.03	2.8	0.03
Anjos-80-03	83.2	0.3	106.8	0.3	4.3	0.01	5.2	0.02	2.6	0.01
Anjos-80-04	83.2	0.2	107.1	0.4	4.3	0.04	5.2	0.03	2.6	0.02
Anjos-80-05	84.4	0.4	113.9	0.6	4.3	0.02	5.5	0.03	2.7	0.03
Average	55.0	0.2	72.3	0.4	3.0	0.02	3.8	0.02	1.9	0.02

Table 3
Performance of local search algorithms on the adapted *sko* instances (Anjos & Yen, 2009): the average running time and its standard deviation σ (both in seconds).

Instance	LS-2OPT		LS-INSERT		LS1		LS2		LS3	
	Time	σ	Time	σ	Time	σ	Time	σ	Time	σ
sko64-1	67.0	0.5	98.8	0.5	4.2	0.02	5.6	0.03	2.9	0.02
sko64-2	65.2	0.2	90.7	0.7	4.1	0.02	5.2	0.02	2.7	0.01
sko64-3	64.2	0.3	89.5	0.8	4.1	0.02	5.2	0.02	2.6	0.02
sko64-4	64.6	0.2	92.2	0.8	4.1	0.03	5.3	0.03	2.7	0.02
sko64-5	64.6	0.2	89.9	0.6	4.1	0.02	5.2	0.02	2.6	0.02
sko72-1	112.6	0.6	166.0	0.8	6.3	0.04	8.7	0.05	4.4	0.02
sko72-2	106.8	0.4	148.1	0.9	6.0	0.03	7.8	0.05	3.9	0.03
sko72-3	104.9	0.3	141.3	0.4	5.9	0.03	7.5	0.03	3.8	0.03
sko72-4	104.6	0.3	141.9	0.5	5.9	0.03	7.5	0.03	3.8	0.02
sko72-5	106.8	0.4	151.0	0.7	6.0	0.03	7.9	0.05	4.0	0.03
sko81-1	177.4	0.7	259.7	1.2	8.9	0.04	12.2	0.06	6.2	0.03
sko81-2	174.2	0.6	241.7	0.8	8.8	0.03	11.6	0.04	5.8	0.03
sko81-3	171.4	0.4	237.0	0.9	8.6	0.03	11.3	0.05	5.7	0.04
sko81-4	174.2	0.7	242.9	1.0	8.8	0.03	11.6	0.06	5.8	0.04
sko81-5	170.7	0.7	235.4	0.8	8.6	0.04	11.3	0.05	5.6	0.02
sko100-1	445.7	2.2	677.3	3.8	18.3	0.10	26.6	0.15	13.2	0.07
sko100-2	418.9	1.3	592.4	2.1	17.2	0.05	23.6	0.07	11.5	0.06
sko100-3	414.0	1.4	568.1	1.6	17.1	0.07	22.7	0.08	11.1	0.04
sko100-4	409.0	1.0	567.1	2.2	16.9	0.06	22.6	0.08	11.0	0.05
sko100-5	414.7	1.0	585.9	1.4	17.1	0.05	23.3	0.05	11.4	0.03
Average	191.6	0.7	270.8	1.1	9.0	0.04	12.1	0.05	6.0	0.03

amount of time required to arrive at a locally optimal solution. The table shows that our algorithms run much faster than both LS-2OPT and LS-INSERT. We find that LS3 achieves the best performance in terms of computational time. This algorithm took less than 40 seconds for the whole dataset. On average, this is less than the time required by each of LS-2OPT and LS-INSERT (55 seconds and 72.3 seconds, respectively) for only one problem instance in the table. We can also see from the table that the standard deviation of the running time is very small. The values of this measure for LS1, LS2 and LS3 are constantly less than 0.05.

The results of solving SRFLP instances in the second dataset are summarized in Table 3. The information in this table is organized in the same manner as in Table 2. The first column contains the instance names in which the integer following *sko* indicates the number of facilities. From Table 3, we notice that again LS3 is faster than other algorithms involved in the comparison. We also see that LS-2OPT and LS-INSERT are many times slower than the local search techniques described in this paper.

In Table 4, we report the results of experiments on larger scale SRFLP instances. The number of facilities is encoded in the instance name. As Table 4 indicates, our local search algorithms are appropriate when dealing with higher-dimensional problem instances as well. They are much more efficient than previous approaches. In particular, the best of our procedures, LS3, is two orders of magnitude faster than the existing insertion-based algorithm LS-INSERT. Supplementary Appendix B contains the comparison of the local search algorithms in terms of solution quality. The results show that such a simple technique as multi-start local search is not suitable if one aims at achieving satisfactory results for problem instances with more than 100 facilities. In such a case, more sophisticated approaches

Table 4
Performance of local search algorithms on larger problem instances (Palubeckis, 2013): the average running time and its standard deviation σ (both in seconds).

Instance	LS-2OPT		LS-INSERT		LS1		LS2		LS3	
	Time	σ	Time	σ	Time	σ	Time	σ	Time	σ
p110	60.1	0.5	82.8	0.9	2.2	0.02	3.0	0.03	1.5	0.03
p120	86.7	0.6	119.2	1.1	3.0	0.02	4.0	0.02	1.9	0.02
p130	119.6	0.8	165.9	1.3	3.8	0.03	5.3	0.04	2.5	0.03
p140	163.9	1.3	224.3	2.0	4.9	0.04	6.7	0.06	3.2	0.04
p150	214.8	1.6	290.7	2.8	6.1	0.04	8.2	0.08	3.8	0.05
p160	283.1	2.2	390.0	3.3	7.5	0.06	10.3	0.08	4.8	0.04
p170	359.9	1.8	497.3	3.8	9.1	0.05	12.5	0.09	5.8	0.06
p180	459.7	3.3	635.1	8.9	11.1	0.08	15.2	0.21	7.0	0.10
p190	575.9	6.8	804.4	6.3	13.7	0.12	18.5	0.14	8.4	0.09
p200	705.3	4.8	972.5	9.3	15.4	0.10	21.2	0.19	9.6	0.09
p210	863.7	6.0	1199.0	14.8	18.0	0.13	25.0	0.30	11.3	0.12
p220	1051.5	8.9	1468.9	6.4	21.0	0.18	29.4	0.14	13.3	0.06
p230	1273.4	8.3	1798.2	21.3	24.6	0.16	34.5	0.39	15.6	0.14
p240	1505.2	11.0	2090.5	21.1	28.0	0.19	38.8	0.36	17.4	0.15
p250	1770.2	12.5	2481.6	17.5	31.8	0.22	44.3	0.30	19.7	0.11
p260	2082.4	17.3	2933.3	25.3	36.3	0.30	50.6	0.39	22.5	0.18
p270	2446.8	15.6	3461.9	32.4	41.3	0.25	57.8	0.55	25.6	0.18
p280	2809.7	20.2	3936.7	35.7	46.4	0.31	64.2	0.56	28.1	0.22
p290	3268.9	12.7	4630.6	44.3	52.3	0.19	72.9	0.62	31.8	0.23
p300	3770.9	24.0	5360.5	58.4	58.7	0.36	81.9	0.72	35.7	0.44
Average	1193.6	8.0	1677.2	15.8	21.8	0.14	30.2	0.26	13.5	0.12

are required. An example of such an approach is given in the next section.

6. A case study: variable neighborhood search

The proposed local search procedures can be incorporated into various heuristic algorithms for solving the SRFLP. In order to illustrate the benefits of using our procedures, we have developed a *variable neighborhood search* (VNS) algorithm for this problem. Different LS techniques can be embedded into the VNS framework. Our purpose is to compare the best of the techniques presented in Sections 3 and 4 with the LS-INSERT method of Kothari and Ghosh (2013a). We prefer LS-INSERT because, as Kothari and Ghosh discovered, this method performs better than LS-2OPT.

6.1. The algorithm

The variable neighborhood search metaheuristic is a general-purpose optimization method which combines a neighborhood change rule and solution perturbation component with a local search technique. In recent years, algorithms based on the VNS metaheuristic have been successfully applied to a variety of combinatorial optimization problems. The general schemes of VNS and their applications are reviewed by Hansen and Mladenović (2001) and by Hansen, Mladenović, and Moreno Pérez (2008, 2010).

Before presenting our VNS algorithm for the SRFLP, we need to first define the neighborhood structures with respect to the solution space. Let $p \in \Pi$ be fixed. We say that the permutation p' belongs to the neighborhood $N_i(p)$ of p if p and p' differ on exactly i entries. Our algorithm makes use of the neighborhoods $N_2, N_4, N_6, \ldots$. It generates a permutation $p' \in N_{2k}(p)$ by performing k pairwise interchanges of facilities under the restriction that no facility may be relocated more than once. By changing the value of k, the algorithm can select neighborhoods of different sizes. An important feature of the algorithm is that, at each iteration, it uses a neighborhood of the best solution found so far. Our implementation of the variable neighborhood search method for the SRFLP can be described as follows.

```
VNS
1. Randomly generate an initial solution $p$ to the given instance of
   the SRFLP. Apply a local search procedure to $p$. Let $p'$ denote the
   solution returned by it. Initialize $p^*$ with $p'$ and $F^*$ with $F(p')$.
2. Set $k := k_{\min}$.
3. While $k \le k_{\max}$ do the following:
    3.1. Set $p := p^*$ and $R := \{1, \ldots, n\}$.
    3.2. Generate a permutation in $N_{2k}(p^*)$ by repeating $k$ times the
         following steps:
        3.2.1. Randomly select facilities $i, j \in R$ and interchange
               their positions in $p$.
        3.2.2. Remove both $i$ and $j$ from $R$.
    3.3. Calculate the distance matrix $D$ corresponding to the per-
         mutation $p$.
    3.4. Apply a local search procedure to $p$ and let $p'$ denote the
         resulting solution.
    3.5. Check whether $F(p') < F^*$. If so, then set $p^* := p'$, $F^* := F(p')$
         and $k := k_{\min}$. Otherwise, increase $k$ by $k_{\text{step}}$.
    3.6. Check if the termination condition is satisfied. If so, then
         stop with the solution $p^*$ of value $F^*$.
4. Go to 2.
```

In the algorithm, k stands for the perturbation depth, which is the number of pairwise interchanges performed during execution of Step 3.2. Basically, Steps 3.1 and 3.2 implement the shaking mechanism of VNS. A new starting solution for local search is always generated from the currently best permutation, denoted as p^*. The value of k is controlled by the VNS parameters $k_{\min}$, k_{step} and $k_{\max}$. The last of them depends on n: we take $k_{\max} = k'_{\max} n$, where $k'_{\max}$ is a positive number not exceeding 0.5. Step 3.3 of VNS initiates a local optimization phase. Its inclusion is justified by the fact that VNS needs to recalculate the distance matrix when permutation changes occur. Step 3.5 implements a neighborhood change function. The next neighborhood is determined either by updating the permutation p^* or/and by assigning a new value to the variable k. In Step 3.6 of VNS, a termination condition is required to be specified. We performed computational experiments using a CPU-time limit as a stopping criterion. Provided the time limit is not exceeded, the search repeats the loop 3.1–3.6 if $k \le k_{\max}$, or otherwise proceeds to Step 4 and immediately to Step 2, which drops k to the minimum possible value, and the loop 3.1–3.6 starts again.

It is clear that the quality of solutions produced by VNS greatly depends on the local search procedure employed in the optimization phase of the algorithm. Within the VNS framework, we have tested the procedures proposed in the current paper as well as the LS-INSERT procedure from Kothari and Ghosh (2013a). In what follows, we will refer to the obtained configurations of VNS as VNS-LS1, VNS-LS2, VNS-LS3 and, respectively, VNS-LS-INSERT.

6.2. Computational results

We conducted computational experiments with VNS on three datasets: the QAP-based *sko* instances (Anjos & Yen, 2009), three instances of size 110 taken from Amaral and Letchford (2011), and the set of larger scale instances introduced in Section 5.1. The results summarized in the current section were obtained by running variations of VNS 10 times on each problem instance in these datasets. Maximum CPU time limits for a run were as follows: 10 seconds for *sko* instances, 100 seconds for $n \in \{110, 120, \ldots, 200\}$, and 600 seconds for $n \in \{210, 220, \ldots, 300\}$. After some preliminary testing, the parameter $k'_{\max}$ of VNS was fixed at 0.4. Two other parameters, $k_{\min}$ and k_{step}, were set to 1.

In the experiments on the *sko* instances (Anjos & Yen, 2009) and the dataset of Amaral and Letchford (2011), we compare the results of the VNS algorithm with the best known values reported in the literature. For the third dataset, the situation is slightly different. For each test case, we need a reference point as a basis of comparison. In this regard, we decided to perform one long run of VNS-LS3 for each instance in the set {p110, p120,… , p300}. We preferred VNS-LS3 to VNS-LS1 and VNS-LS2 because this variation of VNS produced the best results in our preliminary experiments. The cutoff time for each long run was 5 hours.

In Table 5, we compare the performance of our LS algorithms when used as search intensification components within the VNS framework. For each VNS variation, we provide two statistics: the average value of the objective function and the success rate, which is defined to be the number of runs (out of 10) in which the algorithm obtained the best solution (the best values are displayed in the second column of Tables 6 and 8). Table 5 discloses that the VNS-LS3 algorithm performs slightly better than both VNS-LS1 and VNS-LS2. We content ourselves by presenting results on a small sample of instances. As a matter of fact, the same conclusion on the superiority of VNS-LS3 could have been reached by considering the full set of

Table 5
Comparison of VNS-LS1, VNS-LS2 and VNS-LS3.

Instance	VNS-LS1		VNS-LS2		VNS-LS3	
	Average value	Succ.	Average value	Succ.	Average value	Succ.
sko100-5	1033346.6	4/10	1033249.0	2/10	1033152.5	7/10
p150	10624934.5	3/10	10624775.9	4/10	10624711.5	5/10
p200	27487785.0	2/10	27486922.8	3/10	27488566.1	6/10
p250	54535414.4	1/10	54531574.9	2/10	54530870.5	1/10
p300	95970033.2	1/10	95969012.7	3/10	95948194.9	5/10

Table 6
Performance of VNS algorithms on the adapted *sko* instances (Anjos & Yen, 2009): best (G_{best}) and average (G_{aver}) solution gaps to the best known result and the standard deviation σ.

Instance	Best known value	VNS-LS3 G_{best} (G_{aver})	VNS-LS3 σ	VNS-LS-INSERT G_{best} (G_{aver})	VNS-LS-INSERT σ
sko64-1	96881[d]	0 (0)	0	0 (14.5)	14.5
sko64-2	634332.5[a]	0 (0)	0	0 (0)	0
sko64-3	414323.5[c]	0 (0)	0	0 (420.0)	650.2
sko64-4	297129[d]	0 (0)	0	0 (141.6)	129.3
sko64-5	501922.5[a]	0 (0)	0	0 (0)	0
sko72-1	139150[c]	0 (0)	0	0 (2.2)	4.7
sko72-2	711998[d]	0 (0)	0	0 (183.7)	473.7
sko72-3	1054110.5[b]	0 (0)	0	0 (361.7)	734.0
sko72-4	919586.5[d]	0 (0)	0	0 (7.2)	11.7
sko72-5	428226.5[d]	0 (0)	0	0 (52.0)	103.5
sko81-1	205106[e]	0 (0)	0	0 (360.5)	400.4
sko81-2	521391.5[c]	0 (0)	0	0 (360.1)	592.8
sko81-3	970796[d]	0 (0)	0	0 (842.1)	812.6
sko81-4	2031803[d]	0 (0)	0	0 (78.7)	102.1
sko81-5	1302711[d]	0 (0)	0	0 (1602.5)	1669.8
sko100-1	378234[f]	0 (0)	0	0 (2282.2)	2215.0
sko100-2	2076008.5[f]	0 (0)	0	2778 (9564.4)	7926.5
sko100-3	16145614.5[f]	0 (0)	0	0 (47118.0)	50635.3
sko100-4	3232522[f]	0 (0)	0	0 (10211.1)	7004.4
sko100-5	1033080.5[f]	0 (72.0)	110.0	240 (2764.5)	2104.2
Average		0 (3.6)	5.5	150.9 (3818.3)	3779.2

[a] Indicates the best known value reported in Amaral and Letchford (2013).
[b] Indicates the best known value reported in Kothari and Ghosh (2013a).
[c] Indicates the best known value reported in Kothari and Ghosh (2013b).
[d] Indicates the best known value reported in Kothari and Ghosh (2014a).
[e] Indicates the best known value reported in Kothari and Ghosh (2014b).
[f] Indicates the best known value reported in Ozcelik (2012).

results. We, thus, choose VNS-LS3 as a representative of a group of VNS algorithms that are based on our local search procedures.

We note that the main goal of experiments with VNS algorithms was to test their performance in terms of solution quality. A secondary objective was to compare the average time taken by each of the tested algorithms to first find a solution that is best in the run (these statistics are presented in Supplementary Appendix B). The main results of solving SRFLP instances in the *sko* dataset (Anjos & Yen, 2009) are reported in Table 6. The best known solution values are given in its second column. They are taken from various recent studies. The third column shows the gap, G_{best}, of the value of the best solution out of 10 runs (in parentheses, the gap, G_{aver}, of the average value of 10 solutions) found by VNS-LS3 to the best known value displayed in the second column. The fourth column provides the standard deviation of solution values for VNS-LS3. The last two columns stand for VNS-LS-INSERT. The bottom row of the table gives the averages of solution gaps and standard deviation over the whole dataset. As we can see from Table 6, the combination of VNS with our local search procedure competes very favorably with VNS-LS-INSERT. Specifically, VNS-LS3 obtained best known results in 197 runs out of 200. It did not reach the best solution three times for the last instance in the set. The performance of VNS-LS-INSERT is less impressive. All 10 runs of this algorithm were successful in two test cases only. For two instances of size 100, VNS-LS-INSERT failed to match the best value in all runs.

Table 7 shows the results of the VNS algorithms on the instances from Amaral and Letchford (2011). The size of each instance is 110. The second column displays the best known values reported in the paper by Ozcelik (2012). The obtained results indicate that VNS-LS3 again significantly outperforms VNS-LS-INSERT.

We have made an attempt to improve the best known solution for each problem instance in the *sko* dataset by Anjos and Yen (2009) as well as for each instance introduced by Amaral and Letchford (2011). To accomplish this task, the VNS-LS3 algorithm has been used. We have limited the computational time for each run to one day. However, for each instance, VNS-LS3 produced a solution with precisely the same objective function value as that shown in Table 6 or 7. One can reasonably conjecture that the best known solutions reported in these tables are optimal.

Table 7
Performance of VNS algorithms on the benchmark instances of Amaral and Letchford (2011): best (G_{best}) and average (G_{aver}) solution gaps to the best known result and the standard deviation σ.

Instance	Best known value[a]	VNS-LS3 $G_{best} = G_{aver}$	VNS-LS3 σ	VNS-LS-INSERT G_{best} (G_{aver})	VNS-LS-INSERT σ
SRFLP-110-1	144296664.5	0	0	0 (76983.9)	51335.8
SRFLP-110-2	86050037	0	0	0 (2241.5)	6685.6
SRFLP-110-3	2234743.5	0	0	0 (0)	0
Average		0	0	0 (26408.5)	19340.5

[a] Best known value reported by Ozcelik (2012).

The results achieved in Table 6 suggest that SRFLP instances in the *sko* dataset do not pose a perceptible challenge to the VNS-LS3 algorithm. The benchmark instances of Amaral and Letchford are relatively easy for this algorithm as well. Therefore, we have carried out a computational study on larger problem instances, which have already been used in the investigation of LS strategies in Section 5. The results of the tested algorithms on these instances are summarized in Table 8. We remind that the second column of this table contains solution values obtained by letting VNS-LS3 run for 5 hours. The best solutions for instances in Table 8 as well as for other SRFLP instances used in our computational experiments can be found in Supplementary Appendix B. By contrasting columns 3–4 of Table 8 with the last two columns we ascertain that VNS-LS3 is unequivocally superior to VNS-LS-INSERT. The former algorithm was successful in finding the best solutions for all instances in the dataset. We note, however, that in three cases (p250, p280, p290) the best value was obtained only once. From the table, we also see that the VNS-LS-INSERT algorithm had low performance relative to VNS-LS3. It was able to produce the best quality solutions for the first four problem instances only. As a more general conclusion from the numerical experiments, we can

Table 8
Performance of VNS algorithms on larger problem instances (Palubeckis, 2013): best (G_{best}) and average (G_{aver}) solution gaps to the best objective value and the standard deviation σ.

Instance	Best value[a]	VNS-LS3 G_{best} (G_{aver})	VNS-LS3 σ	VNS-LS-INSERT G_{best} (G_{aver})	VNS-LS-INSERT σ
p110	4435868	0 (0)	0	0 (0)	0
p120	6282721	0 (0)	0	0 (139.7)	279.9
p130	7880929.5	0 (0)	0	0 (5708.4)	6872.2
p140	9257162	0 (0)	0	0 (4858.6)	8621.1
p150	10624389.5	0 (322.0)	322.0	136 (10999.1)	7902.8
p160	14873277	0 (0)	0	200 (33593.9)	17791.5
p170	16630187	0 (3680.5)	3680.5	11192 (27491.1)	12407.6
p180	18746031.5	0 (0)	0	749 (38959.5)	36491.4
p190	24453272	0 (481.5)	735.5	5461 (44958.9)	35283.6
p200	27482649.5	0 (5916.6)	8499.6	24629 (65294.4)	32112.9
p210	29512000.5	0 (242.3)	726.9	4335 (33605.6)	27742.5
p220	37351850.5	0 (618.6)	1855.8	17367 (43421.7)	17138.8
p230	46744886	0 (0)	0	242 (92635.4)	60007.8
p240	46717781	0 (34.8)	53.2	5674 (66498.8)	31536.0
p250	54526293.5	0 (4577.0)	3851.5	13012 (66894.5)	44842.9
p260	63300360.5	0 (13012.8)	11173.6	57191 (107156.8)	24841.5
p270	68960438.5	0 (0)	0	984 (197207.5)	87079.6
p280	73845821	0 (23596.5)	13997.7	44225 (151676.6)	57850.9
p290	86255267.5	0 (4446.1)	7419.5	35414 (117686.0)	65911.9
p300	95937735	0 (10459.9)	19335.3	59767 (151732.3)	79125.5
Average		0 (3369.4)	3582.6	14028.9 (63025.9)	32692.0

[a] Solution values obtained by letting VNS-LS3 run for 5 hours.

state that the VNS metaheuristic is a viable alternative that can provide a basis for the development of effective algorithms for solving the SRFLP.

7. Conclusions

The main contribution of this work is fast and easy to implement neighborhood exploration (NE) procedures for the single row facility layout problem. Their time complexity is $O(n^2)$. Because the objective function of the problem involves the summation of $n(n-1)/2$ terms, it is not reasonable to expect any NE procedure to be (asymptotically) faster than those presented in this paper. The local search (LS) algorithms based on the developed NE procedures appear to provide markedly better performance than existing state-of-the-art LS approaches for the SRFLP. We tested our LS algorithms on a range of previously studied SRFLP benchmark instances, and have found that they are capable of obtaining good solutions in a short period of time, even when they operate in the stand-alone mode.

We have shown that the usefulness of the proposed LS algorithms significantly increases when they are embedded into a VNS metaheuristic scheme. We carried out computational experiments on three sets of SRFLP instances of size up to 300 facilities. The results indicate that the speedup provided by using the fastest of our LS algorithms is rather spectacular. The VNS based on this algorithm offers much better performance than the variant of VNS that uses an $O(n^3)$-time NE technique from the literature. This finding underscores the importance of utilizing an effective LS heuristic within a VNS framework.

The approach we have presented affords opportunities to develop high-performance algorithms for the SRFLP. Besides VNS, there are other metaheuristics and their hybrids which contain a local search component. An attractive way to implement such a component is to use neighborhood exploration procedures proposed in this paper.

Appendix A. Proofs of Propositions 2, 4 and 6

Throughout the proofs, we will, mostly tacitly, use the following equation obtained from (23)

$$c_m^- = a_{p(m)n} - a_{p(m)1}. \tag{28}$$

Proof of Proposition 2. To get an initial expression for $\Delta(p, k, l)$, we rewrite (3) as $F(p) = \sum_{m=1}^{n-1} f_m(p)$ where $f_m(p)$ stands for the m-th term in the right-hand side of (3). Let p' denote the permutation obtained from p by interchanging the facilities $r = p(k)$ and $s = p(l)$ in positions k and, respectively, $l > k$. Denoting the difference $f_m(p') - f_m(p)$ by Δ_m and taking into account the fact that $f_m(p') = f_m(p)$ for $m = 1, \ldots, k-2$ and $m = l+1, \ldots, n-1$ we can write

$$\Delta(p,k,l) = F(p') - F(p) = \sum_{m=k-1}^{l} \Delta_m \tag{29}$$

where $\Delta_0 = 0$ if $k = 1$ and $\Delta_n = 0$ if $l = n$.

Now let $c'_1, \ldots, c'_{n-1}$ be the cut values for the solution p'. To continue, there are five cases which need to be handled.

(i) $m = k-1, k > 1$. It is easy to see that $f_{k-1}(p') = c'_{k-1}(\lambda_{p(k-1)} + \lambda_s + \gamma_{p(k-1)s})$ and $f_{k-1}(p) = c_{k-1}(\lambda_{p(k-1)} + \lambda_r + \gamma_{p(k-1)r})$. Since $c'_{k-1} = c_{k-1}$ it follows that

$$\begin{aligned}\Delta_{k-1} &= f_{k-1}(p') - f_{k-1}(p)\\ &= c_{k-1}(\lambda_s - \lambda_r + \gamma_{p(k-1)s} - \gamma_{p(k-1)r}).\end{aligned} \tag{30}$$

(ii) $m = l, l < n$. Similarly as in case (i) we have

$$\Delta_l = c_l(\lambda_r - \lambda_s + \gamma_{rp(l+1)} - \gamma_{sp(l+1)}). \tag{31}$$

(iii) $m = k$. In this case

$$\begin{aligned}c'_k &= c_k + \sum_{i=1}^{k-1} w_{rp(i)} - \left(\sum_{i=k+1}^{n} w_{rp(i)} - w_{rs}\right) + \sum_{i=l+1}^{n} w_{sp(i)}\\ &\quad - \left(\sum_{i=1}^{l-1} w_{sp(i)} - \sum_{i=k+1}^{l-1} w_{sp(i)} - w_{rs}\right) + \sum_{i=k+1}^{l-1} w_{sp(i)}.\end{aligned} \tag{32}$$

By using the definition of the matrix A and Eq. (28) with respect to $r = p(k)$ and $s = p(l)$, (32) can be rewritten as

$$\begin{aligned}c'_k &= c_k + a_{r1} - a_{rn} + a_{sn} - a_{s1} + 2a_{s,k+1} + 2w_{rs}\\ &= c_k + c_l^- - c_k^- + 2a_{s,k+1} + 2w_{rs}.\end{aligned}$$

Now suppose that $l > k+1$. Then we have

$$\begin{aligned}\Delta_k &= f_k(p') - f_k(p)\\ &= (c_k + c_l^- - c_k^- + 2a_{s,k+1} + 2w_{rs})\\ &\quad \times (\lambda_s + \lambda_{p(k+1)} + \gamma_{sp(k+1)})\\ &\quad - c_k(\lambda_r + \lambda_{p(k+1)} + \gamma_{rp(k+1)})\\ &= (c_l^- - c_k^- + 2a_{s,k+1} + 2w_{rs})(\lambda_s + \lambda_{p(k+1)} + \gamma_{sp(k+1)})\\ &\quad + c_k(\lambda_s - \lambda_r + \gamma_{sp(k+1)} - \gamma_{rp(k+1)}).\end{aligned} \tag{33}$$

If $l = k+1$, then $a_{s,k+1} = a_{sl} = 0$, and the expression for Δ_k is slightly different:

$$\begin{aligned}\Delta_k &= (c_k + c_l^- - c_k^- + 2w_{rs})(\lambda_s + \lambda_r + \gamma_{sr}) - c_k(\lambda_r + \lambda_s + \gamma_{rs})\\ &= (c_l^- - c_k^- + 2w_{rs})(\lambda_s + \lambda_r + \gamma_{sr}) + c_k(\gamma_{sr} - \gamma_{rs}).\end{aligned} \tag{34}$$

(iv) $m = l-1$. By the same derivation as in case (iii), we have

$$\begin{aligned}c'_{l-1} &= c_{l-1} + \sum_{i=1}^{k-1} w_{rp(i)} - \left(\sum_{i=k+1}^{n} w_{rp(i)} - \sum_{i=k+1}^{l-1} w_{rp(i)} - w_{rs}\right)\\ &\quad + \sum_{i=k+1}^{l-1} w_{rp(i)} + \sum_{i=l+1}^{n} w_{sp(i)} - \left(\sum_{i=1}^{l-1} w_{sp(i)} - w_{rs}\right)\\ &= c_{l-1} + a_{r1} - a_{rn} + a_{sn} - a_{s1} + 2a_{r,l-1} + 2w_{rs}\\ &= c_{l-1} + c_l^- - c_k^- + 2a_{r,l-1} + 2w_{rs}.\end{aligned}$$

Notice that $l > k+1$ since otherwise we get case (iii). Therefore

$$\begin{aligned}\Delta_{l-1} &= (c_{l-1} + c_l^- - c_k^- + 2a_{r,l-1} + 2w_{rs})\\ &\quad \times (\lambda_{p(l-1)} + \lambda_r + \gamma_{p(l-1)r})\\ &\quad - c_{l-1}(\lambda_{p(l-1)} + \lambda_s + \gamma_{p(l-1)s})\\ &= (c_l^- - c_k^- + 2a_{r,l-1} + 2w_{rs})(\lambda_{p(l-1)} + \lambda_r + \gamma_{p(l-1)r})\\ &\quad + c_{l-1}(\lambda_r - \lambda_s + \gamma_{p(l-1)r} - \gamma_{p(l-1)s}).\end{aligned} \tag{35}$$

(v) $m \in [k+1, \ldots, l-2]$. Now

$$\begin{aligned}c'_m &= c_m + \sum_{i=1}^{k-1} w_{rp(i)} - \left(\sum_{i=k+1}^{n} w_{rp(i)} - \sum_{i=k+1}^{m} w_{rp(i)} - w_{rs}\right)\\ &\quad + \sum_{i=k+1}^{m} w_{rp(i)} + \sum_{i=l+1}^{n} w_{sp(i)}\\ &\quad - \left(\sum_{i=1}^{l-1} w_{sp(i)} - \sum_{i=m+1}^{l-1} w_{sp(i)} - w_{rs}\right) + \sum_{i=m+1}^{l-1} w_{sp(i)}\\ &= c_m + a_{r1} - a_{rn} + a_{sn} - a_{s1} + 2a_{rm} + 2a_{s,m+1} + 2w_{rs}\end{aligned}$$

and hence

$$\begin{aligned}\Delta_m &= (c_l^- - c_k^- + 2a_{rm} + 2a_{s,m+1} + 2w_{rs})\\ &\quad \times (\lambda_{p(m)} + \lambda_{p(m+1)} + \gamma_{p(m)p(m+1)}).\end{aligned} \tag{36}$$

The obtained expressions for Δ_m, $m = k-1, \ldots, l$, can be combined to calculate the value of $\Delta(p, k, l)$. First assume that $l > k+1$. Substituting (30), (31), (33), (35) and (36) into (29) and making some minor rearrangements, we have

$$\Delta(p,k,l) = T_1 + T_2 + T_3 + T_4 + T_5,$$

where

$$T_1 = (c_l^- - c_k^- + 2w_{rs})\left[\sum_{m=k+1}^{l-2}(\lambda_{p(m)} + \lambda_{p(m+1)} + \gamma_{p(m)p(m+1)}) + \lambda_s + \lambda_{p(k+1)} + \gamma_{sp(k+1)} + \lambda_{p(l-1)} + \lambda_r + \gamma_{p(l-1)r}\right],$$

$$T_2 = 2\left[a_{r,l-1}(\lambda_{p(l-1)} + \lambda_r + \gamma_{p(l-1)r}) + \sum_{m=k+1}^{l-2} a_{rm}(\lambda_{p(m)} + \lambda_{p(m+1)} + \gamma_{p(m)p(m+1)})\right],$$

$$T_3 = 2\left[a_{s,k+1}(\lambda_s + \lambda_{p(k+1)} + \gamma_{sp(k+1)}) + \sum_{m=k+1}^{l-2} a_{s,m+1}(\lambda_{p(m)} + \lambda_{p(m+1)} + \gamma_{p(m)p(m+1)})\right], \quad (37)$$

$$T_4 = (\lambda_r - \lambda_s)(c_l + c_{l-1} - c_k - c_{k-1}),$$
$$T_5 = c_{k-1}(\gamma_{p(k-1)s} - \gamma_{p(k-1)r}) + c_k\gamma^1 + c_{l-1}\gamma^2 + c_l(\gamma_{rp(l+1)} - \gamma_{sp(l+1)}). \quad (38)$$

To simplify the term T_1, we notice that

$$\lambda_r + \lambda_{p(k+1)} + \gamma_{rp(k+1)} + \sum_{m=k+1}^{l-2}(\lambda_{p(m)} + \lambda_{p(m+1)} + \gamma_{p(m)p(m+1)}) + \lambda_{p(l-1)} + \lambda_s + \gamma_{p(l-1)s} = d_{rs}.$$

Thus $T_1 = (c_l^- - c_k^- + 2w_{rs})(d_{rs} + \gamma^1 + \gamma^2)$. Next, from the definition of the matrix B, the case of $j > k$, it can be immediately seen that $T_2 = 2b_{r,l-1}$. A similar conclusion can be reached for the term T_3. Indeed, by replacing $m+1$ by i, the sum in (37) can be written as $\sum_{i=k+2}^{l-1} a_{si}(\lambda_{p(i-1)} + \lambda_{p(i)} + \gamma_{p(i-1)p(i)})$. Now, remembering that $s = p(l)$, it should be clear that $T_3 = 2b_{s,k+1}$. Finally, the term T_4 is equal to $\lambda_{rs}^-(c_l^+ - c_k^+)$. Putting T_1, T_2, T_3, T_4 and T_5 together yields (15).

If $l = k+1$, then $\Delta(p, k, l)$ is obtained by summing up (30), (31) and (34). Keeping the same notations as in the general case, we find that $T_1 = (c_l^- - c_k^- + 2w_{rs})(\lambda_r + \lambda_s + \gamma_{sr})$, $T_2 = T_3 = 0$, $T_4 = (\lambda_r - \lambda_s)(c_l - c_{k-1})$, and T_5 is given by (38). Since $\lambda_r + \lambda_s + \gamma_{sr} = d_{rs} + \gamma_{sr} - \gamma_{rs}$ and $c_l - c_{k-1} = c_l^+ - c_k^+$ for $l = k+1$ it follows that the sum of T_1, T_4 and T_5 is equal to the right-hand side of (15). Notice that, in the case of $l = k+1$, $b_{r,l-1} = b_{rk} = 0$ and $b_{s,k+1} = b_{sl} = 0$. □

Proof of Proposition 4. Let p' denote the permutation obtained from p by relocating the facility $r = p(k)$ from position k to position $l \in \{1, \ldots, n\} \setminus \{k\}$ and let $c'_1, \ldots, c'_{n-1}$ be the cut values for p'. First consider the case where $l > k$. Like in the proof of Proposition 2, we can write

$$\delta(p,k,l) = F(p') - F(p) = \sum_{m=k-1}^{l} \Delta_m$$

where $\Delta_0 = \Delta_n = 0$ as before.

We proceed by deriving expressions for Δ_m, $m = k-1, \ldots, l$. To this end, we distinguish between the following five cases.

(i) $m = k-1$, $k > 1$. Obviously, $c'_{k-1} = c_{k-1}$. Hence

$$\begin{aligned}\Delta_{k-1} &= c_{k-1}(\lambda_{p(k-1)} + \lambda_{p(k+1)} + \gamma_{p(k-1)p(k+1)}) \\ &\quad - c_{k-1}(\lambda_{p(k-1)} + \lambda_r + \gamma_{p(k-1)r}) \\ &= c_{k-1}(\lambda_{p(k+1)} - \lambda_r + \gamma_{p(k-1)p(k+1)} - \gamma_{p(k-1)r}). \quad (39)\end{aligned}$$

(ii) $m = l$, $l < n$. In this case, we have

$$\Delta_l = c_l(\lambda_r - \lambda_{p(l)} + \gamma_{rp(l+1)} - \gamma_{p(l)p(l+1)}). \quad (40)$$

(iii) $m = k$. The k-th cut value for p' is

$$\begin{aligned}c'_k &= c_k + \sum_{i=1}^{k-1} w_{rp(i)} - \left(\sum_{i=k+1}^{n} w_{rp(i)} - w_{rp(k+1)}\right) + \sum_{i=k+2}^{n} w_{p(k+1)p(i)} - \left(\sum_{i=1}^{k} w_{p(k+1)p(i)} - w_{rp(k+1)}\right) \\ &= c_k + a_{r1} - a_{rn} + a_{p(k+1)n} - a_{p(k+1)1} + 2w_{rp(k+1)} \\ &= c_k + c_{k+1}^- - c_k^- + 2w_{rp(k+1)}.\end{aligned}$$

We remark that if $l = k+1$, then case (iv) is considered. So we can assume that $l > k+1$. Then

$$\begin{aligned}\Delta_k &= (c_k + c_{k+1}^- - c_k^- + 2w_{rp(k+1)}) \\ &\quad \times (\lambda_{p(k+1)} + \lambda_{p(k+2)} + \gamma_{p(k+1)p(k+2)}) \\ &\quad - c_k(\lambda_r + \lambda_{p(k+1)} + \gamma_{rp(k+1)}) \\ &= (c_{k+1}^- - c_k^- + 2a_{r,k+1})(\lambda_{p(k+1)} + \lambda_{p(k+2)} + \gamma_{p(k+1)p(k+2)}) \\ &\quad + c_k(\lambda_{p(k+2)} - \lambda_r + \gamma_{p(k+1)p(k+2)} - \gamma_{rp(k+1)}). \quad (41)\end{aligned}$$

(iv) $m = l-1$. In this case

$$\begin{aligned}c'_{l-1} &= c_{l-1} + \sum_{i=1}^{k-1} w_{rp(i)} - \left(\sum_{i=k+1}^{n} w_{rp(i)} - \sum_{i=k+1}^{l-1} w_{rp(i)} - w_{rp(l)}\right) \\ &\quad + \sum_{i=k+1}^{l-1} w_{rp(i)} + \sum_{i=l+1}^{n} w_{p(l)p(i)} - \left(\sum_{i=1}^{l-1} w_{p(l)p(i)} - w_{rp(l)}\right) \\ &= c_{l-1} + a_{r1} - a_{rn} + a_{p(l)n} - a_{p(l)1} + 2(a_{r,l-1} + w_{rp(l)}) \\ &= c_{l-1} + c_l^- - c_k^- + 2a_{rl}. \quad (42)\end{aligned}$$

Using (42) we get

$$\begin{aligned}\Delta_{l-1} &= (c_{l-1} + c_l^- - c_k^- + 2a_{rl})(\lambda_{p(l)} + \lambda_r + \gamma_{p(l)r}) \\ &\quad - c_{l-1}(\lambda_{p(l-1)} + \lambda_{p(l)} + \gamma_{p(l-1)p(l)}) \\ &= (c_l^- - c_k^- + 2a_{rl})(\lambda_{p(l)} + \lambda_r + \gamma_{p(l)r}) \\ &\quad + c_{l-1}(\lambda_r - \lambda_{p(l-1)} + \gamma_{p(l)r} - \gamma_{p(l-1)p(l)}). \quad (43)\end{aligned}$$

(v) $m \in [k+1, \ldots, l-2]$. As before, we first identify the cut value:

$$\begin{aligned}c'_m &= c_m + \sum_{i=1}^{k-1} w_{rp(i)} - \left(\sum_{i=k+1}^{n} w_{rp(i)} - \sum_{i=k+1}^{m} w_{rp(i)} - w_{rp(m+1)}\right) \\ &\quad + \sum_{i=k+1}^{m} w_{rp(i)} + \sum_{i=m+2}^{n} w_{p(m+1)p(i)} \\ &\quad - \left(\sum_{i=1}^{m} w_{p(m+1)p(i)} - w_{rp(m+1)}\right) \\ &= c_m + a_{r1} - a_{rn} + a_{p(m+1)n} - a_{p(m+1)1} \\ &\quad + 2(a_{rm} + w_{rp(m+1)}) = c_m + c_{m+1}^- - c_k^- + 2a_{r,m+1}.\end{aligned}$$

Then we can write

$$\begin{aligned}\Delta_m &= (c_m + c_{m+1}^- - c_k^- + 2a_{r,m+1}) \\ &\quad \times (\lambda_{p(m+1)} + \lambda_{p(m+2)} + \gamma_{p(m+1)p(m+2)}) \\ &\quad - c_m(\lambda_{p(m)} + \lambda_{p(m+1)} + \gamma_{p(m)p(m+1)}) \\ &= (c_{m+1}^- - c_k^- + 2a_{r,m+1})\end{aligned}$$

$$\times(\lambda_{p(m+1)}+\lambda_{p(m+2)}+\gamma_{p(m+1)p(m+2)}) + c_m(\lambda_{p(m+2)}-\lambda_{p(m)}+\gamma_{p(m+1)p(m+2)}-\gamma_{p(m)p(m+1)}). \tag{44}$$

Following the same strategy as in the proof of Proposition 2, we combine (39)–(41), (43) and (44) in a way that is suitable for performing further manipulations:

$$\delta(p,k,l) = T_1 - T_2 + T_3 + T_4 + T_5 + T_6 \tag{45}$$

where in the case of $l > k+1$

$$T_1 = 2\sum_{m=k}^{l-2} a_{r,m+1}(\lambda_{p(m+1)}+\lambda_{p(m+2)}+\gamma_{p(m+1)p(m+2)}) + 2a_{rl}(\lambda_{p(l)}+\lambda_r+\gamma_{p(l)r}),$$

$$T_2 = c_k^-\left[\sum_{m=k}^{l-2}(\lambda_{p(m+1)}+\lambda_{p(m+2)}+\gamma_{p(m+1)p(m+2)})+\lambda_{p(l)}+\lambda_r\right], \tag{46}$$

$$T_3 = \sum_{m=k}^{l-2} c_{m+1}^-(\lambda_{p(m+1)}+\lambda_{p(m+2)})+c_l^-(\lambda_{p(l)}+\lambda_r), \tag{47}$$

$$T_4 = c_{k-1}(\lambda_{p(k+1)}-\lambda_r)+c_l(\lambda_r-\lambda_{p(l)})+c_k(\lambda_{p(k+2)}-\lambda_r) + c_{l-1}(\lambda_r-\lambda_{p(l-1)})+\sum_{m=k+1}^{l-2} c_m(\lambda_{p(m+2)}-\lambda_{p(m)}), \tag{48}$$

$$T_5 = \sum_{m=k}^{l-2} c_{m+1}^-\gamma_{p(m+1)p(m+2)} + \sum_{m=k+1}^{l-2} c_m(\gamma_{p(m+1)p(m+2)}-\gamma_{p(m)p(m+1)}), \tag{49}$$

$$T_6 = (c_l^- - c_k^-)\gamma_{p(l)r}+c_{k-1}(\gamma_{p(k-1)p(k+1)}-\gamma_{p(k-1)r}) + c_l(\gamma_{rp(l+1)}-\gamma_{p(l)p(l+1)})+c_k(\gamma_{p(k+1)p(k+2)}-\gamma_{rp(k+1)}) + c_{l-1}(\gamma_{p(l)r}-\gamma_{p(l-1)p(l)}). \tag{50}$$

If $l = k+1$, then (48) and (50) must be adjusted by removing the third and, respectively, the fourth term.

We now simplify the above expressions. First of all, observe that $T_1 = 2b_{rl}$. Next, it is easy to see that the sum over m in (46) is equal to $d_{p(k+1)p(l)}$. Therefore, $T_2 = c_k^-(d_{p(k+1)p(l)}+\lambda_{p(l)}+\lambda_r)$. Further, it is not hard to check that, using equations $c_{m+1}^- = c_{m+1}-c_m$, $m = k,\dots,l-1$, (47) can be rewritten as

$$T_3 = c_{k+1}^-\lambda_{p(k+1)} + c_l^-\lambda_r + \sum_{m=k+2}^{l}(c_m - c_{m-2})\lambda_{p(m)}. \tag{51}$$

Similarly, simplifying (48) we get

$$T_4 = (c_l^+ - c_k^+)\lambda_r - \sum_{m=k+1}^{l}(c_m - c_{m-2})\lambda_{p(m)}. \tag{52}$$

Notice that (52) holds also for $l = k+1$, since in this case only the first two terms in (48) can be nonzero and $c_{k+1}^+ - c_k^+ = c_{k+1}-c_{k-1}$. Summing (51) and (52) gives $T_3+T_4 = (2c_l - c_k^+)\lambda_r - c_k^-\lambda_{p(k+1)}$. Suppose that $l > k+1$. Then by expanding the sums in (49), we find that $T_5 = c_{l-1}\gamma_{p(l-1)p(l)} - c_k\gamma_{p(k+1)p(k+2)}$. Substituting $T_1,\dots,T_6$ in (45), and performing minor manipulations yields (24). In the case of $l = k+1$, $T_5 = 0$ and the terms of (24) containing γ values are deduced from T_6 alone (with the adjustment mentioned right below (50)).

To finish the proof, we address the case where $l < k$. Our strategy is to take advantage of formula (24) we have just derived. For a permutation $p \in \Pi$, let $\tilde{p}$ be its reverse defined by $\tilde{p}(m) = p(n-m+1)$, $m = 1,\dots,n$. Denote the transpose of the matrix Γ by $\Gamma^{\mathrm{T}} = (\gamma_{ij}^{\mathrm{T}})$, where $\gamma_{ij}^{\mathrm{T}} = \gamma_{ji}$. We will write $\tilde{\delta}$ for the δ values (and $\tilde{b}$ for the entries of the matrix B) defined with respect to the SRFLP instance given by the triplet $(W, L, \Gamma^{\mathrm{T}})$, $L = (L_1,\dots,L_n)$. We will use $\tilde{c}_1,\dots,\tilde{c}_{n-1}$ to refer to the cut values for the solution $\tilde{p}$. By applying (24) to $\tilde{\delta}$, we have

$$\begin{aligned}\delta(p,k,l) &= \tilde{\delta}(\tilde{p}, n-k+1, n-l+1)\\ &= 2\tilde{b}_{r,n-l+1} - \tilde{c}_{n-k+1}^-(d_{\tilde{p}(n-k+2)\tilde{p}(n-l+1)}+\lambda^+_{\tilde{p}(n-k+2)\tilde{p}(n-l+1)})\\ &\quad + L_r(\tilde{c}_{n-l+1}-\tilde{c}_{n-k+1})+\tilde{c}_{n-k}\\ &\quad \times(\gamma^{\mathrm{T}}_{\tilde{p}(n-k)\tilde{p}(n-k+2)}+\gamma^{\mathrm{T}}_{\tilde{p}(n-l+1)r}-\gamma^{\mathrm{T}}_{\tilde{p}(n-k)r})\\ &\quad - \tilde{c}_{n-k+1}(\gamma^{\mathrm{T}}_{r\tilde{p}(n-k+2)}+\gamma^{\mathrm{T}}_{\tilde{p}(n-l+1)r})\\ &\quad + \tilde{c}_{n-l+1}(\gamma^{\mathrm{T}}_{\tilde{p}(n-l+1)r}+\gamma^{\mathrm{T}}_{r\tilde{p}(n-l+2)}-\gamma^{\mathrm{T}}_{\tilde{p}(n-l+1)\tilde{p}(n-l+2)}),\end{aligned}$$

where $\tilde{c}_{n-k+1}^- = \tilde{c}_{n-k+1}-\tilde{c}_{n-k}$. Making use of the identities $\tilde{b}_{r,n-l+1} = b_{rl}$, $\tilde{c}_{n-l+1} = c_{l-1}$, $\tilde{c}_{n-k+1} = c_{k-1}$, $\tilde{c}_{n-k} = c_k$, $\tilde{c}_{n-k+1}^- = -c_k^-$ and translating back to the permutation p and matrix Γ, we arrive at (25). □

Proof of Proposition 6. First we show that (26) holds. We start with the following equality:

$$\delta(p,k,l) = \delta(p,k,l-1)+\Delta_{l-2}+\Delta_{l-1}+\Delta_l, \tag{53}$$

where $\Delta_m = f_m(p') - f_m(p'')$, $m = l-2, l-1, l$, and p' (respectively, p'') is the permutation obtained from p by relocating the facility $r = p(k)$ from position k to position l (respectively, to position $l-1$). It is assumed that $p'' = p$ if $l = k+1$. Letting c_i', $i = 1,\dots,n-1$, denote the cut values for p', we consider the following three cases.

(i) $m = l$. Since $c_l' = c_l$ it follows that

$$\Delta_l = c_l(\lambda_r - \lambda_{p(l)}+\gamma_{rp(l+1)}-\gamma_{p(l)p(l+1)}). \tag{54}$$

(ii) $m = l-1$. The cut value for p' is $c_{l-1}' = c_{l-1}+\sum_{i=1}^{l-1}w_{rp(i)} - (\sum_{i=l}^{n}w_{rp(i)} - w_{rp(l)}) + \sum_{i=l+1}^{n}w_{p(l)p(i)} - (\sum_{i=1}^{l-1}w_{p(l)p(i)} - w_{rp(l)}) = c_{l-1}+a_{r1}+a_{r,l-1}-(a_{rn}-a_{r,l-1}-w_{rp(l)})+a_{p(l)n} - (a_{p(l)1}-w_{rp(l)})$. Simplifying, we get:

$$c_{l-1}' = c_{l-1} - c_k^- + c_l^- + 2a_{rl}. \tag{55}$$

Making use of (55) gives

$$\begin{aligned}\Delta_{l-1} &= (c_{l-1}-c_k^-+c_l^-+2a_{rl})(\lambda_{p(l)}+\lambda_r+\gamma_{p(l)r})\\ &\quad - c_{l-1}(\lambda_r+\lambda_{p(l)}+\gamma_{rp(l)})\\ &= (c_l^- - c_k^- + 2a_{rl})(\lambda_{p(l)}+\lambda_r+\gamma_{p(l)r})\\ &\quad + c_{l-1}(\gamma_{p(l)r}-\gamma_{rp(l)}).\end{aligned} \tag{56}$$

(iii) $m = l-2$. First suppose that $l > k+1$. Since swapping the facilities r and $p(l)$ does not affect the cut value c_{l-2}', we can write

$$\begin{aligned}\Delta_{l-2} &= c_{l-2}'(\lambda_{p(l-1)}+\lambda_{p(l)}+\gamma_{p(l-1)p(l)})\\ &\quad - c_{l-2}'(\lambda_{p(l-1)}+\lambda_r+\gamma_{p(l-1)r})\\ &= c_{l-2}'(\lambda_{p(l)}-\lambda_r+\gamma_{p(l-1)p(l)}-\gamma_{p(l-1)r}).\end{aligned}$$

Using an analogue of (55) for $l-2$, we obtain

$$\Delta_{l-2} = (c_{l-1}-c_k^-+2a_{r,l-1})(\lambda_{p(l)}-\lambda_r+\gamma_{p(l-1)p(l)}-\gamma_{p(l-1)r}). \tag{57}$$

If $l = k+1$, then Δ_{l-2} takes a simpler form:

$$\Delta_{l-2} = c_{l-2}(\lambda_{p(l)}-\lambda_r+\gamma_{p(k-1)p(l)}-\gamma_{p(k-1)r}). \tag{58}$$

Substituting (54), (56) and (57) in (53) yields

$$\delta(p,k,l) = \delta(p,k,l-1)+2\lambda_{p(l)}(a_{r,l-1}+a_{rl}-c_k^-) + 2\lambda_r(w_{rp(l)}+c_l^-)+T, \tag{59}$$

where

$$\begin{aligned} T = {} & 2a_{rl}\gamma_{p(l)r} + 2a_{r,l-1}(\gamma_{up(l)} - \gamma_{ur}) \\ & - c_k^-(\gamma_{up(l)} + \gamma_{p(l)r} - \gamma_{ur}) \\ & - c_{l-1}(\gamma_{ur} + \gamma_{rp(l)} - \gamma_{up(l)}) \\ & + c_l(\gamma_{p(l)r} + \gamma_{rp(l+1)} - \gamma_{p(l)p(l+1)}) \end{aligned} \tag{60}$$

with u as defined in the formulation of the proposition. We emphasize that (59) and (60) also cover the case of $l = k+1$. This can be checked by combining (54), (56) and (58) and taking into account the following identities, valid for $l = k+1$: $a_{r,l-1} = 0$, $a_{rl} = w_{rp(l)}$, $c_k^- = c_{l-1} - c_{l-2}$.

Using the equation $a_{rl} = a_{r,l-1} + w_{rp(l)}$, the first three terms in (60) can be rewritten as $2w_{rp(l)}\gamma_{p(l)r} + (2a_{r,l-1} - c_k^-)(\gamma_{up(l)} + \gamma_{p(l)r} - \gamma_{ur})$. Now substituting the value of T in (59) and, for ease of exposition, renaming $a_{r,l-1}$ to e_{l-1} and, respectively, a_{rl} to e_l we get (26).

For $l = k-1, k-2, \ldots, 1$, we apply the same trick as in the proof of the case $l < k$ of Proposition 4. In terms of the notations used there, from (26) we have

$$\begin{aligned} \delta(p,k,l) = {} & \tilde{\delta}(\tilde{p}, n-k+1, n-l+1) \\ = {} & \tilde{\delta}(\tilde{p}, n-k+1, n-l) + L_r(w_{r\tilde{p}(n-l+1)} + \tilde{c}^-_{n-l+1}) \\ & + L_{\tilde{p}(n-l+1)}(\tilde{e}_{n-l} + \tilde{e}_{n-l+1} - \tilde{c}^-_{n-k+1}) \\ & + 2w_{\tilde{p}(n-l+1)r}\gamma^{\mathrm{T}}_{\tilde{p}(n-l+1)r} \\ & + (2\tilde{e}_{n-l} - \tilde{c}^-_{n-k+1})\left(\gamma^{\mathrm{T}}_{u\tilde{p}(n-l+1)} + \gamma^{\mathrm{T}}_{\tilde{p}(n-l+1)r} - \gamma^{\mathrm{T}}_{ur}\right) \\ & - \tilde{c}_{n-l}\left(\gamma^{\mathrm{T}}_{ur} + \gamma^{\mathrm{T}}_{r\tilde{p}(n-l+1)} - \gamma^{\mathrm{T}}_{u\tilde{p}(n-l+1)}\right) \\ & + \tilde{c}_{n-l+1}\left(\gamma^{\mathrm{T}}_{\tilde{p}(n-l+1)r} + \gamma^{\mathrm{T}}_{r\tilde{p}(n-l+2)} - \gamma^{\mathrm{T}}_{\tilde{p}(n-l+1)\tilde{p}(n-l+2)}\right), \end{aligned} \tag{61}$$

where $\tilde{e}_{n-l+1} = \tilde{e}_{n-l} + w_{r\tilde{p}(n-l+1)}$, $u = \tilde{p}(n-k)$ if $n-l+1 = n-k+2$, and $u = \tilde{p}(n-l)$ if $n-l+1 \neq n-k+2$. Now (27) follows from (61) by applying the identities $\tilde{p}(m) = p(n-m+1)$, $\tilde{c}_m = c_{n-m}$, $\tilde{c}^-_m = -c^-_{n-m+1}$, $\tilde{e}_m = e_{n-m+1}$, $m = 1, \ldots, n$, and $\gamma^{\mathrm{T}}_{ij} = \gamma_{ji}$, $i, j = 1, \ldots, n$. □

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.ejor.2015.05.055

References

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

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

Amaral, A. R. S. (2009). A new lower bound for the single row facility layout problem. *Discrete Applied Mathematics, 157*(1), 183–190.

Amaral, A. R. S., & Letchford, A. N. (2011). *A polyhedral approach to the single row facility layout problem. Technical Report*. UK: The Department of Management Science, Lancaster University.

Amaral, A. R. S., & Letchford, A. N. (2013). A polyhedral approach to the single row facility layout problem. *Mathematical Programming, 141*(1–2), 453–477.

Anjos, M. F., Kennings, A., & Vannelli, A. (2005). A semidefinite optimization approach for the single-row layout problem with unequal dimensions. *Discrete Optimization, 2*(2), 113–122.

Anjos, M. F., & Liers, F. (2012). Global approaches for facility layout and VLSI floorplanning. In M. F. Anjos, & J. B. Lasserre (Eds.), *Handbook on semidefinite, conic and polynomial optimization. International series in operations research & management science: 166* (pp. 849–877). New York, USA: Springer.

Anjos, M. F., & Vannelli, A. (2008). Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. *INFORMS Journal on Computing, 20*(4), 611–617.

Anjos, M. F., & Yen, G. (2009). Provably near-optimal solutions for very large single-row facility layout problems. *Optimization Methods and Software, 24*(4–5), 805–817.

Datta, D., Amaral, A. R. S., & Figueira, J. R. (2011). Single row facility layout problem using a permutation-based genetic algorithm. *European Journal of Operational Research, 213*(2), 388–394.

de Alvarenga, A. G., Negreiros-Gomes, F. J., & Mestria, M. (2000). Metaheuristic methods for a class of the facility layout problem. *Journal of Intelligent Manufacturing, 11*(4), 421–430.

Djellab, H., & Gourgand, M. (2001). A new heuristic procedure for the single-row facility layout problem. *International Journal of Computer Integrated Manufacturing, 14*(3), 270–280.

Ficko, M., Brezocnik, M., & Balic, J. (2004). Designing the layout of single- and multiple-rows flexible manufacturing system by genetic algorithms. *Journal of Materials Processing Technology, 157–158*, 150–158.

Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: principles and applications. *European Journal of Operational Research, 130*(3), 449–467.

Hansen, P., Mladenović, N., & Moreno Pérez, J. A. (2008). Variable neighbourhood search: methods and applications. *4OR, 6*(4), 319–360.

Hansen, P., Mladenović, N., & Moreno Pérez, J. A. (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research, 175*, 367–407.

Heragu, S. S., & Alfa, A. S. (1992). Experimental analysis of simulated annealing based algorithms for the layout problem. *European Journal of Operational Research, 57*(2), 190–202.

Heragu, S. S., & Kusiak, A. (1988). Machine layout problem in flexible manufacturing systems. *Operations Research, 36*(2), 258–268.

Heragu, S. S., & Kusiak, A. (1991). Efficient models for the facility layout problem. *European Journal of Operational Research, 53*(1), 1–13.

Hungerländer, P. (2014). Single-row equidistant facility layout as a special case of single-row facility layout. *International Journal of Production Research, 52*(5), 1257–1268.

Hungerländer, P., & Rendl, F. (2013). A computational study and survey of methods for the single-row facility layout problem. *Computational Optimization and Applications, 55*(1), 1–20.

Keller, B., & Buscher, U. (2015). Single row layout models. *European Journal of Operational Research, 245*(3), 629–644.

Kothari, R., & Ghosh, D. (2012). The single row facility layout problem: state of the art. *OPSEARCH, 49*(4), 442–462.

Kothari, R., & Ghosh, D. (2013a). Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods. *European Journal of Operational Research, 224*(1), 93–100.

Kothari, R., & Ghosh, D. (2013b). Insertion based Lin-Kernighan heuristic for single row facility layout. *Computers and Operations Research, 40*(1), 129–136.

Kothari, R., & Ghosh, D. (2014a). An efficient genetic algorithm for single row facility layout. *Optimization Letters, 8*(2), 679–690.

Kothari, R., & Ghosh, D. (2014b). A scatter search algorithm for the single row facility layout problem. *Journal of Heuristics, 20*(2), 125–142.

Kumar, R. M. S., Asokan, P., Kumanan, S., & Varma, B. (2008). Scatter search algorithm for single row layout problem in FMS. *Advances in Production Engineering and Management, 3*(4), 193–204.

Lian, K., Zhang, C., Gao, L., & Shao, X. (2011). Single row facility layout problem using an imperialist competitive algorithm. In *Proceedings of the 41st international conference on computers & industrial engineering* (pp. 578–586).Los Angeles, CA, USA

Ozcelik, F. (2012). A hybrid genetic algorithm for the single row layout problem. *International Journal of Production Research, 50*(20), 5872–5886.

Palubeckis, G. (2012). A branch-and-bound algorithm for the single-row equidistant facility layout problem. *OR Spectrum, 34*(1), 1–21.

Palubeckis, G. (2013). Single row facility layout. http://www.proin.ktu.lt/~gintaras/srflp.html.

Picard, J.-C., & Queyranne, M. (1981). On the one-dimensional space allocation problem. *Operations Research, 29*(2), 371–391.

Romero, D., & Sánchez-Flores, A. (1990). Methods for the one-dimensional space allocation problem. *Computers and Operations Research, 17*(5), 465–473.

Samarghandi, H., & Eshghi, K. (2010). An efficient tabu algorithm for the single row facility layout problem. *European Journal of Operational Research, 205*(1), 98–105.

Samarghandi, H., Taabayan, P., & Jahantigh, F. F. (2010). A particle swarm optimization for the single row facility layout problem. *Computers and Industrial Engineering, 58*(4), 529–534.

Simmons, D. M. (1969). One-dimensional space allocation: An ordering algorithm. *Operations Research, 17*(5), 812–826.

Skorin-Kapov, J. (1990). Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing, 2*(1), 33–45.

Solimanpur, M., Vrat, P., & Shankar, R. (2005). An ant algorithm for the single row layout problem in flexible manufacturing systems. *Computers and Operations Research, 32*(3), 583–598.

Teo, Y. T., & Ponnambalam, S. G. (2008). A hybrid ACO/PSO heuristic to solve single row layout problem. *Proceedings of the 4th IEEE conference on automation science and engineering* (pp. 597–602). Key Bridge Marriott, Washington, DC, USA: IEEE Computer Society.