Computers & Industrial Engineering 103 (2017) 1–16

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

Single row facility layout using multi-start simulated annealing

Gintaras Palubeckis

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

ARTICLE INFO

Article history:
Received 20 April 2016
Received in revised form 31 August 2016
Accepted 29 September 2016
Available online 6 October 2016

Keywords:
Combinatorial optimization
Heuristics
Facility layout
Simulated annealing

ABSTRACT

In the single row facility layout problem (SRFLP), we are given a set of n facilities, their lengths, and a flow cost matrix. The problem asks to arrange the facilities along a straight line so as to minimize the sum of the products of the flow costs and center-to-center distances between facilities. We develop a multi-start simulated annealing (MSA) algorithm for solving this problem. The algorithm employs two move types, pairwise interchanges of facilities and insertions. We propose $O(n)$-time procedures for computing gains of both types of moves. They make our algorithm very fast compared to the traditional approaches, which are based on a straightforward procedure for obtaining the move gain in $O(n^2)$ time. When the temperature during the cooling process drops to a certain level, the computation is accelerated with the use of an additional technique which is reminiscent of a local search algorithm (it takes $O(1)$ time to compute the move gain and $O(n^2)$ time to perform the required calculations when a move is accepted). We report computational results for SRFLP instances of size up to 1000 facilities. The results demonstrate the superiority of the MSA algorithm over the state-of-the-art methods. The source code implementing MSA is made publicly available as a benchmark for future comparisons.

1. Introduction

The problem studied in this paper belongs to the vast family of problems whose solution space is composed of permutations on a set of elements of some category. In our case, the elements are the facilities, and the problem we aim to address is the *single row facility layout problem* (SRFLP for short). An instance of the SRFLP is given by the number of facilities n, their lengths $L_1, \ldots, L_n$, and a symmetric $n \times n$ matrix $W = (w_{ij})$ whose entry w_{ij} represents the cost of the flow of material between facilities i and j. The problem asks to arrange the facilities along a straight line so as to minimize the sum of the products of the flow costs and center-to-center distances between facilities. Let Π be the set of all permutations of $I = \{1, \ldots, n\}$, which is defined as the set of all vectors $(p(1), p(2), \ldots, p(n))$ such that $p(k) \in I$, $k = 1, \ldots, n$, and $p(k) \neq p(l), k = 1, \ldots, n-1$, $l = k+1, \ldots, n$. Then, mathematically, the SRFLP can be expressed as:

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

where $p(k)$ and $p(l)$ are the facilities in the kth and lth positions of permutation p, respectively, and $d_{p(k)p(l)}$ is the distance between the centroids of facilities $p(k)$ and $p(l)$. Assuming $k < l$, the distance is defined as follows:

$$d_{p(k)p(l)} = L_{p(k)}/2 + \sum_{k<m<l} L_{p(m)} + L_{p(l)}/2. \tag{2}$$

The formulation (1) and (2) models the problem of obtaining a good linear layout of machines within manufacturing cells (Heragu & Kusiak, 1988). The SRFLP also arises in other contexts such as the arrangement of rooms on one side of a corridor in supermarkets, hospitals and office buildings (Simmons, 1969), the arrangement of books on a shelf in a library (Picard & Queyranne, 1981), and design of warehouse layouts (Picard & Queyranne, 1981).

There is a vast literature on algorithms for the SRFLP, both exact and heuristic. Exact methods for solving the SRFLP include branch-and-bound (Simmons, 1969), dynamic programming (Picard & Queyranne, 1981), mixed-integer linear programming (Amaral, 2006, 2008a; Heragu & Kusiak, 1991; Love & Wong, 1976), cutting plane algorithm (Amaral, 2009a), branch-and-cut (Amaral & Letchford, 2013), and semidefinite programming approaches (Anjos & Vannelli, 2008; Hungerländer & Rendl, 2013). Polyhedral studies for the SRFLP have been done by Sanjeevi and Kianfar (2010) and Amaral and Letchford (2013). To deal with larger-scale SRFLP instances, many heuristic and metaheuristic-based solution techniques have been proposed. Among them, we find constructive procedures (Djellab & Gourgand, 2001; Kumar, Hadjinicola, & Lin, 1995), simulated annealing (de Alvarenga,

E-mail address: gintaras.palubeckis@ktu.lt

http://dx.doi.org/10.1016/j.cie.2016.09.026
0360-8352/

Negreiros-Gomes, & Mestria, 2000; Heragu & Alfa, 1992; Romero & Sánchez-Flores, 1990), tabu search (de Alvarenga et al., 2000; Kothari & Ghosh, 2013a; Samarghandi & Eshghi, 2010), ant algorithm (Solimanpur, Vrat, & Shankar, 2005), particle swarm optimization (PSO) technique (Samarghandi, Taabayan, & Jahantigh, 2010), PSO and ant colony optimization hybrid (Teo & Ponnambalam, 2008), enhanced local search (Amaral, 2008b), genetic algorithms (Datta, Amaral, & Figueira, 2011; Ficko, Brezocnik, & Balic, 2004; Kothari & Ghosh, 2014a; Ozcelik, 2012), Lin-Kernighan heuristic (Kothari & Ghosh, 2013b), path relinking algorithms (Kothari & Ghosh, 2012b), hybrid estimation of distribution algorithm (Ou-Yang & Utamima, 2013), imperialist competitive method (Lian, Zhang, Gao, & Shao, 2011), and scatter search implementations (Kothari & Ghosh, 2014b; Kumar, Asokan, Kumanan, & Varma, 2008). A slightly more detailed look at the existing algorithms for the SRFLP as well as an adaptation of the variable neighborhood search (VNS) metaheuristic to this problem were given in (Palubeckis, 2015b). A notable recent study addressing the SRFLP is that by Guan and Lin (2016). Their algorithm, dubbed VNSACO, combines the ideas of VNS with the principles of ant colony optimization (ACO). The VNS component of the algorithm is responsible for search intensification, whereas the ACO component is used as a mechanism for search diversification. The algorithm employs a local search procedure which uses three neighborhood structures and adopts the first-improvement strategy. Guan and Lin compared VNSACO with three state-of-the-art algorithms for the SRFLP: hybrid genetic algorithm of Ozcelik (2012), insertion-based tabu search method of Kothari and Ghosh (2013a) and genetic algorithm of the same authors (Kothari & Ghosh, 2014a). Through computational experiments, it was found that VNSACO performed better than the other algorithms included in the comparison.

In this paper, we are interested in using a simulated annealing (SA) approach to address the SRFLP. As mentioned earlier, there are a few SA algorithms developed for this problem. Historically the first such algorithms were proposed by Romero and Sánchez-Flores (1990). They presented two methods based on the simulated annealing paradigm. One of them uses an equilibrium test to decide whether the temperature must be decreased. The test compares the last formed set of n accepted solutions against the set of n accepted solutions formed just before. The second method does not employ an equilibrium test. Instead, it updates the temperature every $3n$ iterations using a rather complex formula. To generate a sequence of solutions, each of the methods can use either a pairwise interchange strategy or an insertion strategy. The authors reported computational results for SRFLP instances with problem size ranging from $n = 20$ to $n = 60$. The results show that the insertion strategy is the better of the two strategies. They also indicate that the second method is slightly superior to the first one. Heragu and Alfa (1992) proposed an SA algorithm hybridized with the modified penalty algorithm presented by Heragu and Kusiak (1991). The latter algorithm was used to generate a good initial solution for SA. The annealing part of the hybrid algorithm proceeds by performing random pairwise interchanges of facilities. At each temperature level, the duration of the search is controlled by two parameters: the maximum number of interchanges and the maximum number of new solutions to be accepted. Heragu and Alfa reported computational results for SRFLP instances with up to 30 facilities. They also applied their hybrid SA technique to a multi-row layout problem in which all the facilities have the same length. More recently, de Alvarenga et al. (2000) proposed another SA algorithm for the SRFLP. As it is quite common in SA implementations, the temperature is updated with iterations in the outer loop of their algorithm. In the inner loop, all $n(n-1)/2$ solutions in the pairwise interchange neighborhood of the current solution are examined. It is supposed that the equilibrium state is reached at the end of this loop, and the temperature can be reduced to the next lower level. In the algorithm, randomness is used only in the acceptance criterion for an interchange move. The authors presented computational results of their SA algorithm on a set of small-sized problem instances from the literature.

A special case of the SRFLP is the single row equidistant facility layout problem (SREFLP) which occurs when all the facilities are of the same length. Unlike SRFLP, the SREFLP is also a special case of the well-studied quadratic assignment problem. Some recent papers on the SREFLP are those by Amaral (2009b), Hungerländer (2014), and Palubeckis (2012, 2015a).

For a more detailed overview of single row facility layout, there are several surveys available. The reader is referred to Anjos and Liers (2012), Kothari and Ghosh (2012a), and Hungerländer and Rendl (2013) and the most recent survey by Keller and Buscher (2015).

The present research is motivated by the need for an algorithm that could be used to find high-quality solutions for large-scale SRFLP instances in a reasonable amount of time. For small and medium size problem instances, good results were obtained using the VNS algorithm of Palubeckis (2015b). However, it was observed that, when the number of facilities exceeded 300, the performance of this algorithm began to decrease. This led us to a search for alternative approaches. One of the attempts was to revive the simulated annealing technique for the SRFLP. Our choice of SA was inspired by the great success of this technique for some other optimization problems on permutations, e.g., the minimum linear arrangement problem (Rodriguez-Tello, Hao, & Torres-Jimenez, 2008) and the bandwidth minimization problem on graphs (Torres-Jimenez et al., 2015). The purpose of this work is to show that the SA method is significant in solving SRFLP in the sense that, when carefully implemented, it is able to demonstrate good performance on large-scale instances of the problem. The most expensive operation in SA algorithms is computing the difference between the objective function values of the current solution and candidate solution. In the case of SRFLP, this operation can be accomplished using the expression for the function F in (1). However, the time complexity of this approach is $O(n^2)$. Our main contribution is innovative procedures for computing the differences between the objective function values of neighboring solutions. We develop two such procedures, one for pairwise interchange neighborhood structure and another for insertion neighborhood structure. Each of them runs in $O(n)$ time. We also apply another method which allows evaluating a neighbor of the current solution with time complexity of $O(1)$. However, this method uses auxiliary matrices which need to be updated every time an interchange or insertion move is executed. Since this update is quite expensive to perform, the method is effective only at lower temperature levels when the move rejection probability is sufficiently high. The developed SA algorithm was embedded in a multi-start scheme and tested on five sets of SRFLP instances whose size goes up to 1000 facilities. Numerical results show the excellent performance of our algorithm.

The remainder of this paper is arranged as follows. In the next section, we present our approach to computing gains of both the interchange and insertion moves. In Section 3, we propose a multi-start simulated annealing algorithm for the SRFLP. In Section 4, we report the results of computational experiments. Concluding remarks are given in Section 5.

2. Computing move gains

Simulated annealing is a metaheuristic method that has been used to find near-optimal solutions for various optimization problems. At the heart of this method is the following expression,

which calculates the acceptance probability (Černý, 1985; Kirkpatrick, Gelatt, & Vecchi, 1983):

$$\exp(-(F(p')-F(p))/T) \tag{3}$$

where F is an objective function, p is the current solution, p' is a solution in a neighborhood of p, and T is the temperature value. In our case, F is the objective function of the SRFLP as given by (1) and p, p' are permutations on the set of facilities.

An important issue in the design of SA algorithms is the choice of the neighborhood structure. Our SA implementation is based on two types of neighborhoods. One of them is the pairwise interchange neighborhood $N_2(p)$, which, for $p \in \Pi$, is defined as the set of all possible permutations that can be obtained from p by swapping positions of two facilities in the permutation p. That is, $N_2(p) = \{p' \in \Pi | p'$ and p differ on exactly two entries$\}$. As an alternative, we employ the insertion neighborhood structure $N_1(p), p \in \Pi$. For $p \in \Pi$, the set $N_1(p)$ consists of all permutations that can be obtained from p by removing a facility from its current position in p and inserting it at a different position. Formally, $N_1(p) = \{p' \in \Pi |$ there exist k, $l \in I$, $k \neq l$, such that $p'(l) = p(k)$, $p'(i) = p(i-1)$, $i = l+1, \ldots, k$, if $l < k$, $p'(i) = p(i+1)$, $i = k, \ldots, l-1$, if $l > k$, and $p'(i) = p(i)$ for the remaining $i \in I\}$, where $I = \{1, \ldots, n\}$ as before. A local transformation applied to a permutation p in order to obtain a neighboring solution $p' \in N_i(p)$, $i \in \{1, 2\}$, is called a move. The difference between solution values $\delta = F(p') - F(p)$ is called the gain of the move. The computation of gain values is of utmost importance for the efficiency of the SA algorithms for the SRFLP. The gains are used in the exponent (3) expressing the probability of accepting worsening moves. Throughout the search process, the value of the expression (3) is calculated a large number of times. A straightforward procedure for obtaining the move gain δ is to compute, using (1), the objective function value $F(p')$ for the permutation p' and take the difference between $F(p')$ and the cost of the current solution, $F(p)$, which is available from the previous iteration of the algorithm. The time complexity of this procedure is $O(n^2)$. To our knowledge, this approach to computing move gains has been used in earlier implementations of the simulated annealing technique for the SRFLP (de Alvarenga et al., 2000; Heragu & Alfa, 1992; Romero & Sánchez-Flores, 1990).

In this section, we present a method for calculation of gain values in linear time. The start point in the development of the new technique is the following formulation of the SRFLP proposed by Palubeckis (2015b):

$$\min_{p \in \Pi} F(p) = \sum_{m=1}^{n-1} c_m(\lambda_{p(m)} + \lambda_{p(m+1)}), \tag{4}$$

where c_m is the sum of the flow costs between the first m facilities and the remaining $n-m$ facilities, i.e., $c_m = \sum_{k=1}^{m}\sum_{l=m+1}^{n} w_{p(k)p(l)}$, and $\lambda_{p(i)}$, $i \in I$, is the half-length of the facility $p(i)$, i.e., $\lambda_{p(i)} = L_{p(i)}/2$. The vector $C = (c_1, \ldots, c_{n-1})$ is called the *cut vector* and its entries are called *cut values*. The latter are computed recursively using the following equation:

$$c_m = c_{m-1} + e_{p(m)}, \quad m = 1, \ldots, n-1, \tag{5}$$

where $c_0 = 0$ and $E = (e_1, \ldots, e_n)$ is the vector indexed by facilities whose rth entry e_r, assuming that $r = p(m)$, is given by

$$e_r = \sum_{i=m+1}^{n} w_{rp(i)} - \sum_{i=1}^{m-1} w_{rp(i)}, \quad r \in I. \tag{6}$$

Both vectors C and E are dependent on the permutation p. However, in order to lighten notations, we skip writing this dependency explicitly. Moreover, to make the formulas for the gain more compact, we will use two $n \times n$ matrices computed from the lengths of facilities: (λ^-_{rs}) with entries $\lambda^-_{rs} = \lambda_r - \lambda_s$ and (λ^+_{rs}) with entries $\lambda^+_{rs} = \lambda_r + \lambda_s$. We also assume as a convention that $c_n = 0$.

2.1. Pairwise interchanges

First, we consider the case of pairwise interchange moves. Specifically, suppose that the permutation $p' \in N_2(p)$ is obtained by swapping positions of the facilities r and s in the permutation p. Let the gain of this move be denoted by $\delta(r, s)$. Assume without loss of generality that r appears to the left of s in p. An example of layouts defined by permutation p and permutation p' obtained from p by interchanging facilities 4 and 8 is shown in Fig. 1. The next statement provides formulas to calculate the value of $\delta(r, s)$.

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

$$\delta(r,s) = (c_{k-1} - c_l)\lambda^-_{sr} + c'_k\lambda^+_{sp(k+1)} - c_k\lambda^+_{rp(k+1)} + \sum_{m=k+1}^{l-2}(c'_m - c_m)\lambda^+_{p(m)p(m+1)} + c'_{l-1}\lambda^+_{p(l-1)r} - c_{l-1}\lambda^+_{p(l-1)s}, \tag{7}$$

where

$$c'_k = c_{k-1} + e_s + 2\sum_{i=k}^{l-1} w_{sp(i)}, \tag{8}$$

$$c'_m = c'_{m-1} + e_{p(m)} + 2(w_{rp(m)} - w_{sp(m)}), \quad m = k+1, \ldots, l-1, \tag{9}$$

and, for $l = k+1$,

$$\delta(r,s) = (c_{k-1} - c_l)\lambda^-_{sr} + (c_{k-1} + e_s + 2w_{rs} - c_k)\lambda^+_{rs}. \tag{10}$$

Proof. Let us denote the mth term in the right-hand side of (4) by $F_m(p)$. Thus $F(p) = \sum_{m=1}^{n-1} F_m(p)$. It is not hard to see that $\delta(r,s) = F(p') - F(p) = \sum_{m=k-1}^{l}(F_m(p') - F_m(p))$ (in the example of Fig. 1, we have $p(i) = i$ for all facilities i, $k = 4$, $l = 8$, and $\delta(4,8) = F(p') - F(p) = \sum_{m=3}^{8}(F_m(p') - F_m(p))$). Evidently, the cut values c_{k-1} and c_l remain unchanged when moving from p to p'. Therefore, we can write

$$F_{k-1}(p') - F_{k-1}(p) = c_{k-1}(\lambda_s - \lambda_r) = c_{k-1}\lambda^-_{sr}, \tag{11}$$

$$F_l(p') - F_l(p) = c_l(\lambda_r - \lambda_s) = -c_l\lambda^-_{sr}. \tag{12}$$

To continue, let c'_m, $m = 1, \ldots, n-1$, denote the cut values for the solution p'. Observe that $c'_k, \ldots, c'_{l-1}$, in general, differ from $c_k, \ldots, c_{l-1}$ (in Fig. 1, such values are $c'_4, \ldots, c'_7$). Suppose first that $l > k+1$. We distinguish between the following three cases.

(i) $m = k$. Using definition of F_m, we get

$$\begin{aligned} F_k(p') - F_k(p) &= c'_k(\lambda_s + \lambda_{p(k+1)}) - c_k(\lambda_r + \lambda_{p(k+1)}) \\ &= c'_k\lambda^+_{sp(k+1)} - c_k\lambda^+_{rp(k+1)}. \end{aligned} \tag{13}$$

(ii) $m \in [k+1, \ldots, l-2]$. In this case

$$\begin{aligned} F_m(p') - F_m(p) &= (c'_m - c_m)(\lambda_{p(m)} + \lambda_{p(m+1)}) \\ &= (c'_m - c_m)\lambda^+_{p(m)p(m+1)}. \end{aligned} \tag{14}$$

(iii) $m = l-1$. Similarly as in case (i) we have

$$F_{l-1}(p') - F_{l-1}(p) = c'_{l-1}\lambda^+_{p(l-1)r} - c_{l-1}\lambda^+_{p(l-1)s}. \tag{15}$$

Collecting Eqs. (11)–(15), we obtain (7).

To show (8), let e'_s be the sth entry of the vector E with respect to the permutation p'. By the definition of the E entries, we have that $e_s = \sum_{i=l+1}^{n} w_{sp(i)} - \sum_{i=1}^{l-1} w_{sp(i)}$ and

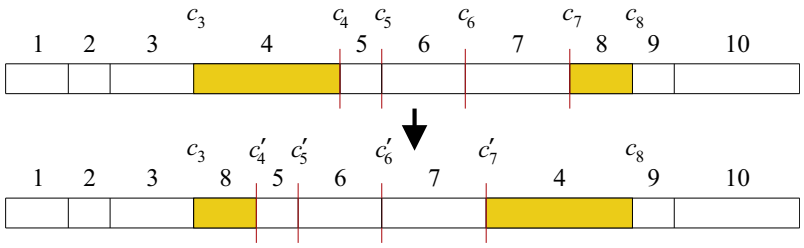

Fig. 1. Pairwise interchange of facilities 4 and 8 (the updated cut values c_m, $m = 4,\ldots,7$, are marked by placing a vertical line between the mth and $(m+1)$th facilities).

$$e'_s = \sum_{i=k+1}^{n} w_{sp'(i)} - \sum_{i=1}^{k-1} w_{sp'(i)} = e_s + \sum_{i=k+1}^{l} w_{sp'(i)} + \sum_{i=k}^{l-1} w_{sp(i)}$$
$$= e_s + 2\sum_{i=k}^{l-1} w_{sp(i)}$$

(see Fig. 1, in which $s = 8$, $k = 4$ and $l = 8$). Now, since $c'_{k-1} = c_{k-1}$ and $s = p'(k)$, it follows that (8) is the analog of (5) for the permutation p'. The same is true for Eq. (9) because $e'_{p'(m)} = e_{p(m)} + 2(w_{rp(m)} - w_{sp(m)})$ for every $m \in \{k+1,\ldots,l-1\}$. Thus, the validity of (8) and (9) is demonstrated. To finish the proof, suppose that $l = k+1$. In this case, the difference $F_m(p') - F_m(p)$ for $m = k-1$ and $m = l = k+1$ is given by (11) and, respectively, (12). For $m = k$, we have

$$F_k(p') - F_k(p) = (c'_k - c_k)\lambda^+_{rs}$$
$$= (c_{k-1} + e_s + 2w_{rs} - c_k)\lambda^+_{rs}. \quad (16)$$

Adding (11), (12) and (16) together, we arrive at (10). □

An iteration of the interchange-based SA algorithm includes computing the gain δ and, if the move is accepted, updating the vectors E and C. The following statement shows that this can be done efficiently.

Proposition 2. *The time complexity of an iteration of the SA algorithm based on the use of Eqs. (7)–(10) is $O(n)$.*

Proof. First, notice that, in the case of $l > k+1$, the sum in (8) has to be calculated only once. Therefore, the cut values for the permutation p' can be obtained in linear time. It is clear that computing the gain $\delta(r,s)$ by using (7) takes asymptotically the same number of operations. If positions of the facilities r and s are swapped, then the entries $e_{p'(m)}$, $m = k,\ldots,l-1$, of the vector E are replaced by $c'_m - c'_{m-1}$, which is in accordance with (5), the cut values c_m, $m = k,\ldots,l-1$, are replaced by c'_m, and the entry e_r of E is updated by subtracting $2\sum_{i=k}^{l-1} w_{rp'(i)}$ from it. The time complexity of these rearrangements is $O(n)$. □

Palubeckis (2015b) has shown that, using special matrices, the gain $\delta(r,s)$ can be computed in constant time. However, the complexity of updating these matrices is $O(n^2)$. So this method is applicable only in the cases where the acceptance probability, as given by (3), is relatively low. Similarly, as it was done for the SREFLP in (Palubeckis, 2015a), in this paper we will combine both methods. For high temperatures, we will use Proposition 1 and, for low temperatures, we will take advantage of the following formula (Palubeckis, 2015b):

$$\delta(r,s) = (c_l^- - c_k^- + 2w_{rs})d_{rs} + 2(b_{r,l-1} + b_{s,k+1}) + \lambda^-_{rs}(c_l^+ - c_k^+), \quad (17)$$

where $c_m^- = c_m - c_{m-1}$, $c_m^+ = c_m + c_{m-1}$, $m \in I$, $D = (d_{rs})$ is the distance matrix with entries given by Eq. (2), and $B = (b_{uj})$ is an $n \times n$ matrix whose entry corresponding to facility $u = p(m)$ and permutation position j is defined as follows

$$b_{uj} = \begin{cases} a_{uj}\lambda^+_{up(j)} + \sum_{i=j+1}^{m-1} a_{ui}\lambda^+_{p(i-1)p(i)} & \text{if } j < m \\ a_{uj}\lambda^+_{up(j)} + \sum_{i=m+1}^{j-1} a_{ui}\lambda^+_{p(i)p(i+1)} & \text{if } j > m \\ 0 & \text{if } j = m, \end{cases}$$

where the "a" coefficients are entries of the matrix $A = (a_{uj})$ computed for the permutation p:

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

After swapping positions of the facilities r and s in the permutation p, the matrices A, B and D can be updated in $O(n^2)$ and the vectors $C, C^- = (c_m^-)$ and $C^+ = (c_m^+)$ in $O(n)$ time. The detailed procedures for updating A, B, D and C along with its derivatives C^- and C^+ can be found in the paper by Palubeckis (2015b).

2.2. Insertions

Given $p \in \Pi$, let $p' \in N_1(p)$ be obtained from the permutation p by removing the facility $r = p(k)$ from position k and inserting it at position l. Such a move is illustrated in Fig. 2. Let the gain of the above defined insertion move be denoted by $\delta'(r,l)$. Assume that $|k-l| > 1$, since otherwise, for $k \neq l$, the move reduces to interchanging adjacent facilities r and $p(l)$. An efficient way to compute $\delta'(r,l)$ is provided by the following formulas.

Proposition 3. *Let $p \in \Pi$, $k,l \in I$, $|k-l| > 1$, and $r = p(k)$. Then, for $k < l$,*

$$\delta'(r,l) = c_{k-1}\lambda^-_{p(k+1)r} - c_k\lambda^+_{p(k+1)r} + \sum_{m=k+1}^{l-1}(c'_{m-1} - c_m)\lambda^+_{p(m)p(m+1)}$$
$$+ c'_{l-1}\lambda^+_{rp(l)} + c_l\lambda^-_{rp(l)}, \quad (18)$$

where $c'_m, m = k,\ldots,l-1$, are obtained from the recurrence

$$c'_m = c'_{m-1} + e_{p(m+1)} + 2w_{rp(m+1)} \quad (19)$$

with the initial condition $c'_{k-1} = c_{k-1}$, and for $k > l$,

$$\delta'(r,l) = c_k\lambda^-_{p(k-1)r} - c_{k-1}\lambda^+_{p(k-1)r} + \sum_{m=l}^{k-2}(c'_{m+1} - c_m)\lambda^+_{p(m)p(m+1)}$$
$$+ c'_l\lambda^+_{rp(l)} + c_{l-1}\lambda^-_{rp(l)}, \quad (20)$$

where c'_m, $m = k-1, k-2,\ldots,l$, are computed from the recurrence

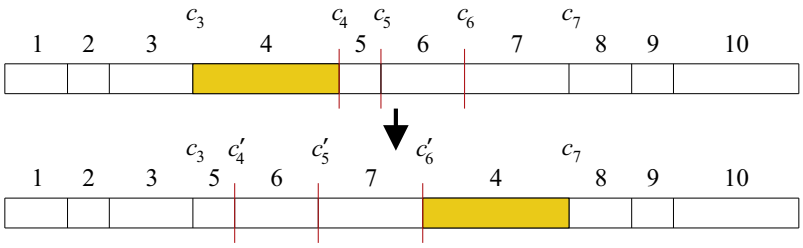

Fig. 2. Relocating facility 4 from position 4 to position 7 (the updated cut values c_m, $m = 4, 5, 6$, are marked by placing a vertical line between the mth and $(m+1)$th facilities).

$$c'_m = c'_{m+1} - e_{p(m)} + 2w_{rp(m)} \tag{21}$$

using the initial condition $c'_k = c_k$.

Proof. First suppose that $k < l$. Adopting notations from the proof of Proposition 1, we can write

$$\begin{aligned}\delta'(r,l) &= \sum_{m=k-1}^{l} (F_m(p') - F_m(p)) \\ &= F_{k-1}(p') - F_{k-1}(p) - F_k(p) + \sum_{m=k+1}^{l-1} (F_{m-1}(p') - F_m(p)) \\ &\quad + F_{l-1}(p') + F_l(p') - F_l(p). \end{aligned} \tag{22}$$

The above expression is obtained taking into account the fact that the terms $F_{k+1}(p), \ldots, F_{l-1}(p)$ can be mapped, respectively, to the terms $F_k(p'), \ldots, F_{l-2}(p')$, and this mapping has the property that the cut values in $F_m(p)$, $m \in \{k+1, \ldots, l-1\}$, and in $F_{m-1}(p')$ are multiplied by the same factor, $\lambda^+_{p(m)p(m+1)}$ (in Fig. 2, the mapping is from $F_5(p)$ to $F_4(p')$ and from $F_6(p)$ to $F_5(p')$). The sum in (22) corresponding to this mapping can be rewritten as $\sum_{m=k+1}^{l-1}(c'_{m-1} - c_m)\lambda^+_{p(m)p(m+1)}$. The first two terms in (22) can be replaced by $c_{k-1}\lambda^-_{p(k+1)r}$ and the last two terms by $c_l\lambda^-_{rp(l)}$. Substituting expressions for $F_k(p)$ and $F_{l-1}(p')$ into (22), we arrive at (18). Eq. (19) follows from (5) and the fact that the updated entry of E for the facility $p(m+1)$, $m \in \{k, \ldots, l-1\}$, is equal to $e'_{p'(m)} = e_{p(m+1)} + 2w_{rp(m+1)}$.

For $k > l$, the analog of (22) is

$$\begin{aligned}\delta'(r,l) &= F_k(p') - F_k(p) - F_{k-1}(p) + \sum_{m=l}^{k-2} (F_{m+1}(p') - F_m(p)) \\ &\quad + F_l(p') + F_{l-1}(p') - F_{l-1}(p). \end{aligned} \tag{23}$$

It is easy to see that (23) can be transformed to (20). Finally, (21) is obtained by combining the equations $c'_{m+1} = c'_m + e'_{p'(m+1)}$ and $e'_{p'(m+1)} = e_{p(m)} - 2w_{rp(m)}$, $m \in \{l, \ldots, k-1\}$. □

As remarked before, in the case of $|k-l| = 1$, an insertion move is identical to a pairwise interchange move. Thus $\delta'(r,l) = \delta(r, p(l))$ if $k = l-1$, and $\delta'(r,l) = \delta(p(l), r)$ if $k = l+1$.

Let us assume that $|k-l| > 1$. When the facility r is relocated from position k to position l in the permutation p, the vectors C and E need to be updated. This is straightforward for C since the new cut values can be saved for later use at the gain calculation step (where these values are computed recursively by applying either Eq. (19) or (21) depending on the sign of $k-l$). If $k < l$, each entry $e_{p(m)}$ of E with $m \in \{k+1, \ldots, l\}$ is increased by adding $2w_{rp(m)}$ to it. If $k > l$, then the entries $e_{p(m)}$, $m = l, \ldots, k-1$, are updated. In this case, $2w_{rp(m)}$ is subtracted from $e_{p(m)}$. The entry e_r is decreased by $2\sum_{m=k+1}^{l} w_{rp(m)}$ if $k < l$ and increased by $2\sum_{m=l}^{k-1} w_{rp(m)}$ if $k > l$. Note that all these operations are fast. The following result is similar to Proposition 2 and should be clear.

Proposition 4. *The time complexity of an iteration of the SA algorithm based on the use of Eqs. (18)–(21) is* $O(n)$.

At the expense of increased time complexity of performing a move whenever it is accepted, the gain $\delta'(r,l)$ can be computed in constant time. If $k < l$, then the following formula can be used (Palubeckis, 2015b):

$$\delta'(r,l) = 2b_{rl} - c^-_k(d_{p(k+1)p(l)} + \lambda^+_{p(k+1)p(l)}) + L_r(c_l - c_k), \tag{24}$$

where b_{rl}, $d_{p(k+1)p(l)}$ and c^-_k are the entries of the matrices B, D and vector C^-, respectively (see Section 2.1 for their definitions). If $k > l$, then the expression for $\delta'(r,l)$ becomes

$$\delta'(r,l) = 2b_{rl} + c^-_k(d_{p(l)p(k-1)} + \lambda^+_{p(l)p(k-1)}) + L_r(c_{l-1} - c_{k-1}). \tag{25}$$

As alluded to in Section 2.1, the update of the matrices B and D after performing a move requires $O(n^2)$ time. Therefore, the gain computation method based on Eqs. (24) and (25) is useful only when the cooling process reaches low temperature range at which a relatively small number of solutions are accepted.

It is noteworthy that the methods we have described in this section use auxiliary data (matrices and vectors) that are common to both move types, pairwise interchanges and insertions. This allows us to develop an SA algorithm which can change the move type from iteration to iteration.

3. The algorithm

Our implementation of the SA method is a multi-start algorithm, in which starting solutions for SA are generated randomly. We refer to this version of the simulated annealing metaheuristic for the SRFLP as the MSA algorithm. The parameters of MSA are retained unchanged from one restart to the next. For better readability, the description of MSA is organized in a top-down manner. The main procedure of the algorithm is presented in Fig. 3. The core of the algorithm is two nested "for" loops. As it is common in SA implementations, the outer loop, called the cooling schedule, controls the temperature, while the inner loop iterates a fixed number of times, $\bar{z}$, for a given temperature level. The temperature is decreased by a cooling factor α whose value usually lies in the interval $[0.9, 0.995]$. The minimum (final) temperature of the cooling process, $T_{\min}$, is typically set to a positive number very close to zero. In MSA, we fixed $T_{\min}$ at 0.0001. To compute the maximum temperature $T_{\max}$, we take the initial permutation p generated at the start of MSA and randomly draw a sample N' of permutations in the neighborhood $N_2(p)$ of p. We set $T_{\max}$ to $\max_{p' \in N'}|F(p') - F(p)|$. The size of the sample in our experiments was fixed at 5000. Another key parameter for MSA is the length $\bar{z}$ of the inner "for" loop. The normal practice is to relate it to the size of the problem instance simply by setting $\bar{z}$ to $\bar{z}_0 n$, where $\bar{z}_0$ is a predefined positive constant. A discussion on the choice of specific values of $\bar{z}_0$ and α for MSA is deferred to the next section. At the beginning of an MSA iteration, the move type is selected at random. The type decision is made between pairwise interchange

```
// Input to MSA includes parameters α, z̄, T_min, γ, P
Randomly generate a permutation p ∈ Π
p* := p
F* := F(p)
Compute T_max
β̄ := ⌊(log(T_min) − log(T_max))/ log α⌋
while stop condition is not met do
  f := F(p)
  T := T_max
  for β = 1, ..., β̄ do
    for z = 1, ..., z̄ do
      Randomly draw a number ζ from the uniform distribution on [0, 1]
      if ζ ⩽ P then δ :=generate_interchange(p, β, β̄, z, γ, r, s)
      else δ :=generate_insertion(p, β, β̄, z, γ, r, l)
      end if
      if δ ⩽ 0 then accept:=true
      else
        Randomly draw a number ξ from the uniform distribution on [0, 1]
        if ξ ⩽ exp(−δ/T) then accept:=true
        else accept:=false
        end if
      end if
      if accept then
        f := f + δ
        if ζ ⩽ P then p :=perform_interchange(p, r, s, β, β̄, γ)
        else p :=perform_insertion(p, r, l, β, β̄, γ)
        end if
        if f < F* then
          p* := p
          F* := f
        end if
      end if
    end for
    T := αT
  end for
  Randomly generate a permutation p ∈ Π
end while
// Output from MSA: solution p* of value F*
```

Fig. 3. Main procedure of MSA.

and insertion with probability P and, respectively, $1 - P$. If $0 < P < 1$, then the moves of both types are employed. Depending on the choice, either `generate_interchange` or `generate_insertion` procedure is executed. Whichever is selected, the returned gain value δ, if positive, is used to calculate the acceptance probability given by expression (3). The input to the above procedures includes parameter γ which controls the use of the gain computation methods described in the previous section. Upon acceptance of the move, either `perform_interchange` or `perform_insertion` is applied, depending on the value of the probability parameter P. The SA code is wrapped in a "while" loop to form the main part of the program implementing the MSA algorithm. Naturally, a condition to stop this loop should be specified. We performed computational experiments using a maximum CPU time limit as a termination condition.

Fig. 4 presents the pseudo-code of the procedure `generate_interchange`. The parameter γ, called a *switch parameter*, is used to split the sequence of temperature levels represented by the values of the variable β into two subsequences. The algorithm employs the move gain computation technique based on Proposi-

```
generate_interchange(p, β, β̄, z, γ, r, s)
Select a pair of facilities r, s at random
// Let k and l be their corresponding positions in p
// Assume w.l.o.g. that k < l
if β ⩽ γβ̄ then
  if β = 1 and z = 1 then
    Initialize vectors E and C        // Eqs. (5) and (6)
  end if
  if l > k + 1 then Compute δ by Eq. (7)
  else Compute δ by Eq. (10)
  end if
else
  if β = ⌊γβ̄⌋ + 1 and z = 1 then
    Initialize matrices B, D and vectors C (if ⌊γβ̄⌋ = 0), C⁻ and C⁺
  end if
  Compute δ by Eq. (17)
end if
return δ
```

Fig. 4. Randomly generating an interchange move.

tion 1 for the range of smaller β values $1, \ldots, \lfloor \gamma\bar{\beta} \rfloor$ and switches the mode of calculating move gains to using Eq. (17) for the rest of β values $\lfloor \gamma\bar{\beta} \rfloor + 1, \ldots, \bar{\beta}$. Thus the operation of SA is subdivided into two phases. The first iteration of each phase starts with β equal to the smallest value in the corresponding range and the SA inner loop index z equal to 1. At this moment, the algorithm has to initialize data that are needed to compute the move gain. These data are the vectors E and C in the first phase and matrices B, D and vectors C, C^- and C^+ in the second phase.

Fig. 5 shows the pseudo-code of the procedure `generate_insertion`. It bears some resemblance to `generate_interchange` given in Fig. 4. The major difference is that now the cases of $k < l$ and $k > l$ are processed separately. Notice that if the positions k and l are adjacent in the permutation p, then Eq. (10) of Proposition 1 is used.

The pseudo-code of the procedure `perform_interchange` is depicted in Fig. 6. We remark that updating the vectors E and C is discussed in Section 2 and the way of updating the matrices B and D is described in (Palubeckis, 2015b). The pseudo-code of the procedure `perform_insertion` is very similar to that of `perform_interchange`. The only difference is that swapping facilities r and s is replaced by the operation of moving the facility r to position l in permutation p.

Suppose that MSA is launched as SA without restarts. Let t_γ denote the number of times `accept` is set to the true value when the outer loop counter β is in the range $\lfloor \gamma\bar{\beta} \rfloor + 1$ to $\bar{\beta}$. Then the time complexity of the nested "for" loops of MSA can be evaluated as $O(\gamma\bar{\beta}\bar{z}n + (1-\gamma)\bar{\beta}\bar{z} + t_\gamma n^2 + n^2) = O(\bar{\beta}\bar{z}(\gamma(n-1)+1) + n^2(t_\gamma + 1))$. The term n^2 in the left-hand side of this equation is for the time to initialize the vector E and/or matrices B and D. In particular, if $\gamma = 1$, which means that only methods based on Propositions 1 and 3 are used, then the above expression becomes $O(\bar{\beta}\bar{z}n + n^2) = O(\bar{\beta}\bar{z}_0 n^2) = O(\bar{\beta}n^2)$ (since $\bar{z}_0$ is a constant). If SA employs a straightforward procedure for obtaining the move gain, that is, computes the gain using (1), then the computational complexity increases to $O(\bar{\beta}n^3)$.

4. Numerical results

In this section, we report on the performance of our approach on several sets of SRFLP instances. The goal of experimentation is to demonstrate that, by utilizing a linear-time procedure for computation of move gains, the obtained implementation of SA is a competitive algorithm for solving the SRFLP. We compare this implementation against the variable neighborhood search algorithm proposed by Palubeckis (2015b). In that paper, this algorithm was named VNS-LS3. Here we will refer to it by the same name. One may wonder whether VNS-LS3 can be strengthened using the results of Section 2. The answer, however, is negative. VNS-LS3 is based on the full exploration of the insertion neighborhood $N_1(p)$. The cardinality of this neighborhood is $n(n-1)$. The gain of moving from the current solution p to its neighbor, using Proposition 3, can be computed in $O(n)$ time. Thus, the time complexity of exploration of $N_1(p)$, using this approach, is $O(n^3)$. The VNS-LS3 algorithm applies a different, very fast technique which performs only $O(n^2)$ operations for this task.

4.1. Experimental setup

The described algorithm has been coded in the C++ programming language, and the tests have been carried out on a PC with an Intel Core 2 Duo CPU running at 3.0 GHz. The same computer was used to run VNS-LS3 in (Palubeckis, 2015b). We note that VNS-LS3 was written in C++ too.

As a testbed for investigating the algorithms, we considered three sets of benchmark instances commonly used in the literature

```
generate_insertion(p, β, β̄, z, γ, r, l)
Select a facility r and its new position l at random
// Let k be the current position of r in p
if β ⩽ γβ̄ then
  if β = 1 and z = 1 then
    Initialize vectors E and C        // Eqs. (5) and (6)
  end if
  if k < l then
    if l > k + 1 then Compute δ′ by Eq. (18)
    else Compute δ′ = δ(r, p(l)) by Eq. (10)
    end if
  else        // the case of k > l
    if k > l + 1 then Compute δ′ by Eq. (20)
    else Compute δ′ = δ(p(l), r) by Eq. (10)
    end if
  end if
else
  if β = ⌊γβ̄⌋ + 1 and z = 1 then
    Initialize matrices B, D and vectors C (if ⌊γβ̄⌋ = 0), C⁻ and C⁺
  end if
  if k < l then Compute δ′ by Eq. (24)
  else Compute δ′ by Eq. (25)
  end if
end if
return δ′
```

Fig. 5. Randomly generating an insertion move.

```
perform_interchange(p, r, s, β, β̄, γ)
Swap positions of facilities r and s in permutation p
if β ⩽ γβ̄ then
  Update vectors E and C
else
  Update matrices B, D and vectors C, C⁻ and C⁺
end if
return updated permutation p
```

Fig. 6. Performing an interchange move.

and, in addition, two sets of larger-scale instances of our own. Specifically, we conducted computational experiments with MSA and VNS-LS3 on the following datasets:

(a) instances introduced by Anjos, Kennings, and Vannelli (2005) whose size ranges from 60 to 80 facilities.
(b) quadratic assignment problem-based *sko* instances (Skorin-Kapov, 1990) tailored for the SRFLP by Anjos and Yen (2009) (with $n \in \{64, 72, 81, 100\}$).
(c) three instances of size 110 taken from Amaral and Letchford (2011).
(d) the dataset introduced by Palubeckis (2015b) (it consists of 20 instances with n ranging from 110 to 300).
(e) a series of new SRFLP instances ranging in size from 310 to 1000 facilities.

The datasets (a) and (b) have been widely used to evaluate the performance of algorithms for solving the SRFLP, including those proposed in Anjos and Yen (2009), Ozcelik (2012), Hungerländer and Rendl (2013), Kothari and Ghosh (2013a, 2013b, 2014a, 2014b), Palubeckis (2015b), and Guan and Lin (2016). Many of them (Guan & Lin, 2016; Hungerländer & Rendl, 2013; Kothari & Ghosh, 2013a, 2013b, 2014a, 2014b; Ozcelik, 2012; Palubeckis, 2015b) have also been tested on the instances introduced by Amaral and Letchford (2011). Each of the datasets (d) and (e) is composed of randomly generated SRFLP instances. The lengths of facilities and the off-diagonal entries of the flow cost matrix in these instances are integer numbers drawn uniformly at random from the intervals $[1, 10]$ and $[0, 10]$, respectively.

In the main experiments, we run both MSA and VNS-LS3 10 times on each SRFLP instance in the selected datasets. The performance of the algorithms is measured in terms of the objective

function value of the best solution out of 10 runs as well as the average objective function value of 10 solutions. As it is argued by Birattari and Dorigo (2007), generally, the second of these metrics is more meaningful than the first one.

4.2. The choice of parameters

We performed a few preliminary experiments with MSA to find good parameter settings. Toward this goal, our first step was to determine a suitable value for the cooling factor α. As it is well known, this parameter should be close to 1. We varied α from 0.91 to 0.99 in increments of 0.01. While running MSA, other parameters were kept fixed at the following values: $\bar{z}_0 = 100$, $\gamma = 0.5$, and $P = 0$ (these values were found to be acceptable during the first stage of experimentation). Using the same SRFLP generator as for the datasets (d) and (e), we created a training set consisting of 10 instances with n ranging from 210 to 300, and run MSA once on each of them. We have found that the algorithm is not sensitive to the cooling factor α over the range of its values tested. One may guess that, for larger α in this range, more iterations of a single SA run are performed, but for smaller α, more restarts of SA are executed. We decided to fix α at 0.95. Another experiment was conducted in order to evaluate the influence of the parameter $\bar{z}_0$ on the solution quality. We examined three values of $\bar{z}_0$: 50, 100 and 200. The results of MSA were of very similar quality for $\bar{z}_0 = 50$ and $\bar{z}_0 = 100$ and slightly worse for $\bar{z}_0 = 200$. For further experiments, we elected to set $\bar{z}_0$ to 100.

Our next step was to investigate the effect of using various values of the switch parameter γ on the speed of the algorithm. We restricted ourselves to considering three variants of MSA (and SA) differing in the value of the probability parameter P : $P = 1$, $P = 0$, and $P = 0.5$. In the rest of this paper, we will refer to these variants as MSA_1, MSA_0 and MSA_0.5, respectively, in the case of multi-start simulated annealing, and as SA_1, SA_0 and SA_0.5, respectively, in the case of conventional simulated annealing. We applied SA_1, SA_0 and SA_0.5 to four instances from the training set. The outcome of this experiment is presented in Tables 1–3. The first column of each of them contains the instance names. The integer in the name gives the number of facilities. The second column displays, for each instance, the objective function value of a solution found by the corresponding configuration of SA. The remaining columns provide the running times of the tested algorithm for various values of the switch parameter γ.

Inspection of Tables 1–3 reveals that all three variants of SA take the shortest time when the parameter γ is set to 0.5. If $\gamma = 1$, which means that the method of move gain computation is entirely based on Propositions 1 and 3, the running time increases by around 50%. At the other end, if $\gamma = 0$, then the performance of SA, as expected, is very poor. In this case, a large amount of time is spent executing interchange or insertion moves at high temperatures during annealing. Another observation is that the insertion-based SA algorithm, SA_0, is noticeably faster (for $\gamma \geqslant 0.5$) than the interchange-based SA algorithm, SA_1. The next subsections are devoted to presenting our main empirical results. They are obtained when applying our algorithm with γ fixed at 0.5.

4.3. Comparison with the traditional SA implementation

Let us denote by SA2 a modification of the SA algorithm in which the move gains are computed directly by using the objective function expression (1). We refer to the variations of this modification with $P = 1$, $P = 0$ and $P = 0.5$ as SA2_1, SA2_0 and SA2_0.5, respectively. In Table 4, we compare the computational times of SA and SA2 on four representative instances selected from the (d) dataset. As can be seen, time savings from SA versus SA2 are spectacular. Notice that the speedup achieved by SA grows with the size of the problem. For example, for 300-facility instance, SA_0 is more than 250 times faster than SA2_0. Thus, even when the SA algorithm is run in a multi-start framework, the computational times can be kept within reasonable limits.

4.4. Computational results for the benchmark instances of Anjos et al. (2005) ($60 \leqslant n \leqslant 80$)

With P fixed, the code implementing MSA requires only one parameter to be supplied — a maximum CPU time limit imposed on a run of the algorithm. For the Anjos et al. instances, initially, we set this limit to 10 s and then, in an additional experiment, increased it to 60 s.

The results of solving SRFLP instances from the (a) dataset are summarized in Tables 5–7. The number of facilities is encoded in the instance names listed in the first column. The second column of Table 5 shows the best known values for the benchmark instances of Anjos et al. (2005). The third column provides the gap, G_{best}, of the value of the best solution out of 10 runs (in parentheses, the gap, G_{av}, of the average value of 10 solutions) found by MSA_1 to the value displayed in the second column. The fourth column gives the standard deviation of solution values for MSA_1. Next, Table 5 presents the same kind of statistics for the other two variants of the MSA technique. The ninth column of the table reports the results obtained with the VNS-LS3 algorithm of Palubeckis (2015b). Like MSA, it was run 10 times with a limit of 10 s per run. The last column shows the performance of the algorithm proposed by Guan and Lin (2016). We included this algorithm in the comparison because it has been compared favorably with other state-of-the-art methods from the literature (see Guan & Lin (2016) for details). Guan and Lin conducted experiments with VNSACO on a notebook PC equipped with Intel Core i3-2350M 2.30 GHz CPU. The average running time of VNSACO for Anjos et al. instances was 38.99 s. The bottom row of Table 5 shows the results averaged over all 20 problem instances. From the table, we notice that all the tested approaches were able to find the best solution (with objective function value given in the second column) for each problem instance in the dataset. Another observation is that VNS-LS3 reaches the best known values more stably than other algorithms involved in the comparison. It can also be seen that MSA_0, MSA_0.5 and VNSACO performed very similarly, and MSA_1 yielded the worst results.

To see how the MSA algorithm performs on the Anjos et al. instances given more time, we conducted another experiment. We increased the cutoff time for a run to 60 s. The results

Table 1
Dependence of the running time (in s) of SA_1 on the switch parameter γ.

Instance	Value	γ										
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
t150	12,483,617	122.9	44.0	12.0	5.2	4.0	4.2	4.5	4.9	5.3	5.7	6.1
t200	27571471.5	305.3	106.3	27.3	10.4	7.2	7.2	7.8	8.6	9.4	10.1	10.9
t250	57696560.5	617.2	210.7	52.1	17.8	11.4	11.1	12.2	13.4	14.7	15.9	17.2
t300	90896942.5	1154.2	397.9	95.5	30.7	17.8	16.6	18.1	19.9	21.8	23.6	25.5

Table 2
Dependence of the running time (in s) of SA_0 on the switch parameter γ.

Instance	Value	γ										
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
t150	12,479,154	147.8	58.9	15.8	5.5	3.5	3.4	3.6	3.8	4.1	4.3	4.5
t200	27584148.5	366.9	142.8	35.9	10.7	5.8	5.4	5.8	6.2	6.7	7.2	7.7
t250	57695981.5	739.6	283.2	70.2	19.3	9.3	8.2	8.8	9.5	10.3	11.1	11.8
t300	90877683.5	1379.4	533.3	130.9	34.8	14.9	12.0	12.7	13.8	14.9	16.0	17.1

Table 3
Dependence of the running time (in s) of SA_0.5 on the switch parameter γ.

Instance	Value	γ										
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
t150	12,483,264	135.5	51.4	14.0	5.4	3.8	3.8	4.1	4.5	4.8	5.1	5.4
t200	27579801.5	336.3	124.7	31.7	10.7	6.6	6.4	6.9	7.5	8.1	8.7	9.3
t250	57700043.5	678.9	247.6	61.7	18.9	10.5	9.7	10.6	11.6	12.6	13.6	14.6
t300	90886480.5	1266.2	465.4	113.4	32.5	16.2	14.2	15.3	16.8	18.3	19.8	21.4

Table 4
Comparison of running time (in s) between SA and SA2.

Instance	SA_1	SA_0	SA_0.5	SA2_1	SA2_0	SA2_0.5
p150	4.3	3.4	3.9	372.7	365.8	368.7
p200	7.1	5.5	6.4	886.7	887.9	887.1
p250	11.2	8.5	9.9	1767.1	1781.9	1775.1
p300	16.2	11.9	13.8	3086.9	3103.5	3095.8

Table 5
Comparison of the objective function values for the benchmark instances of Anjos et al. (2005): best (G_{best}) and average (G_{av}) solution gaps to the best known result and the standard deviation σ.

Instance	Best known value	MSA_1		MSA_0		MSA_0.5		VNS-LS3	VNSACO[d]
		$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$	σ	$G_{best} = G_{av} = \sigma$	$G_{best}(G_{av})$
Anjos-60-1	1,477,834[c]	0(1.8)	2.7	0(0)	0	0(0)	0	0	0(0)
Anjos-60-2	841,776[b]	0(13.8)	28.1	0(9.6)	28.8	0(0)	0	0	0(0)
Anjos-60-3	648337.5[c]	0(59.9)	133.5	0(0)	0	0(0)	0	0	0(136.8)
Anjos-60-4	398,406[b]	0(0)	0	0(0)	0	0(0)	0	0	0(15.5)
Anjos-60-5	318,805[c]	0(0)	0	0(0)	0	0(0)	0	0	0(0)
Anjos-70-1	1,528,537[b]	0(20.1)	30.7	0(0)	0	0(0)	0	0	0(0)
Anjos-70-2	1,441,028[c]	0(48.7)	146.1	0(0)	0	0(0)	0	0	0(0)
Anjos-70-3	1518993.5[c]	0(0)	0	0(0)	0	0(0)	0	0	0(0)
Anjos-70-4	968,796[a]	0(200.0)	220.7	0(0)	0	0(0)	0	0	0(0)
Anjos-70-5	4218002.5[b]	0(0)	0	0(0)	0	0(0)	0	0	0(0)
Anjos-75-1	2393456.5[a]	0(0)	0	0(0)	0	0(0)	0	0	0(0)
Anjos-75-2	4,321,190[c]	0(33.9)	50.5	0(11.1)	20.6	0(8.9)	20.4	0	0(21.1)
Anjos-75-3	1,248,423[b]	0(1094.5)	742.4	0(68.4)	55.8	0(45.6)	55.8	0	0(34.2)
Anjos-75-4	3941816.5[b]	0(0)	0	0(0)	0	0(0)	0	0	0(0)
Anjos-75-5	1,791,408[c]	0(81.4)	244.2	0(0)	0	0(0)	0	0	0(0)
Anjos-80-1	2069097.5[c]	0(57.3)	67.8	0(0)	0	0(0)	0	0	0(0)
Anjos-80-2	1,921,136[b]	0(15.2)	30.4	0(0)	0	0(0)	0	0	0(0)
Anjos-80-3	3,251,368[a]	0(0)	0	0(0)	0	0(0)	0	0	0(0)
Anjos-80-4	3,746,515[c]	0(32.0)	64.0	0(0)	0	0(0)	0	0	0(0)
Anjos-80-5	1,588,885[b]	0(67.9)	140.9	0(0)	0	0(0)	0	0	0(0)
Average		0(86.3)	95.1	0(4.5)	5.3	0(2.7)	3.8	0	0(10.4)

[a] Indicates the best known value reported in Datta et al. (2011).
[b] Indicates the best known value reported in Lian et al. (2011).
[c] Indicates the best known value reported in Samarghandi and Eshghi (2010).
[d] This column is obtained from the data presented in Guan and Lin (2016).

are reported in Table 6. It has columns for solution gaps, standard deviation, and the average running time, t, to the best solution in a run. We notice that one minute of CPU time was sufficient for MSA_0.5 to reach the best known result in all runs for all instances in the dataset. Both MSA_1 and MSA_0 were unable to do this only for one problem instance (Anjos-75-3). We also observe that MSA_1 is slower than MSA_0 and MSA_0.5. The average running time for the latter two MSA variants is less than 3 s.

Table 7 illustrates the improvement to solution quality with increasing time limit for the two Anjos et al. instances, one with $n = 75$ and the other with $n = 80$. Its entries in columns 3–9 give solution gaps G_{best} and G_{av}. We can see that 2 s per run are sufficient to reach $G_{best} = 0$ by all algorithms except MSA_1 for both problem instances. In the case of MSA_1, performing 5-s runs appeared to be enough for the same purpose. The time to reach $G_{av} = 0$ varies between 10 and 150 s for Anjos-75-3 and between 2 and 60 s for Anjos-80-5, depending on the algorithm.

Table 6
Results of running MSA variants on the benchmark instances of Anjos et al. (2005) for 60 s: best (G_{best}) and average (G_{av}) solution gaps to the best known result, standard deviation σ, and the average running time (t given in seconds) to the best solution in a run.

Instance	MSA_1			MSA_0			MSA_0.5	
	$G_{best}(G_{av})$	σ	t	$G_{best}(G_{av})$	σ	t	$G_{best} = G_{av} = \sigma$	t
Anjos-60-1	0(0)	0	5.6	0(0)	0	0.5	0	0.6
Anjos-60-2	0(0)	0	8.6	0(0)	0	3.7	0	1.1
Anjos-60-3	0(0)	0	5.7	0(0)	0	2.4	0	2.8
Anjos-60-4	0(0)	0	2.1	0(0)	0	0.9	0	1.7
Anjos-60-5	0(0)	0	1.0	0(0)	0	0.5	0	0.5
Anjos-70-1	0(0)	0	8.1	0(0)	0	2.5	0	4.0
Anjos-70-2	0(0)	0	4.7	0(0)	0	1.5	0	1.5
Anjos-70-3	0(0)	0	3.9	0(0)	0	0.6	0	0.8
Anjos-70-4	0(0)	0	11.9	0(0)	0	2.7	0	1.7
Anjos-70-5	0(0)	0	3.0	0(0)	0	1.6	0	0.9
Anjos-75-1	0(0)	0	3.4	0(0)	0	0.8	0	1.0
Anjos-75-2	0(0)	0	16.7	0(0)	0	9.0	0	4.9
Anjos-75-3	0(22.8)	45.6	23.0	0(11.4)	34.2	12.0	0	10.8
Anjos-75-4	0(0)	0	2.9	0(0)	0	1.2	0	1.3
Anjos-75-5	0(0)	0	4.8	0(0)	0	1.9	0	1.8
Anjos-80-1	0(0)	0	13.8	0(0)	0	2.6	0	1.5
Anjos-80-2	0(0)	0	7.0	0(0)	0	1.2	0	1.2
Anjos-80-3	0(0)	0	2.1	0(0)	0	1.1	0	0.9
Anjos-80-4	0(0)	0	5.9	0(0)	0	1.8	0	1.4
Anjos-80-5	0(0)	0	11.0	0(0)	0	2.9	0	2.6
Average	0(1.1)	2.3	7.3	0(0.6)	1.7	2.6	0	2.1

Table 7
Variation of solution quality with execution time for Anjos-75-3 and Anjos-80-5 instances: best and average (in parentheses) solution gaps to the best known result.

Instance	Algorithm	Time limit (s)						
		2	5	10	30	60	100	150
Anjos-75-3	MSA_1	114(2374.6)	0(1493.4)	0(1094.5)	0(192.4)	0(22.8)	0(11.4)	0(0)
Anjos-75-3	MSA_0	0(1010.3)	0(550.6)	0(68.4)	0(11.4)	0(11.4)	0(0)	
Anjos-75-3	MSA_0.5	0(953.7)	0(392.4)	0(45.6)	0(0)			
Anjos-75-3	VNS-LS3	0(22.8)	0(11.4)	0(0)				
Anjos-80-5	MSA_1	16(1570.6)	0(261.1)	0(67.9)	0(1.6)	0(0)		
Anjos-80-5	MSA_0	0(86.2)	0(3.2)	0(0)				
Anjos-80-5	MSA_0.5	0(392.6)	0(1.6)	0(0)				
Anjos-80-5	VNS-LS3	0(0)						

Table 8
Comparison of the objective function values for the adapted *sko* instances (Anjos & Yen, 2009): best (G_{best}) and average (G_{av}) solution gaps to the best known result and the standard deviation σ.

Instance	Best known value	MSA_1		MSA_0		MSA_0.5		VNS-LS3[g]		VNSACO[h]
		$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$
sko64-1	96,881[d]	0(0.2)	0.6	0(0.3)	0.5	0(0.3)	0.5	0(0)	0	0(29.6)
sko64-2	634332.5[a]	0(0)	0	0(1.8)	2.7	0(21.8)	65.4	0(0)	0	0(56.7)
sko64-3	414323.5[c]	0(0)	0	0(0)	0	0(0)	0	0(0)	0	0(458.9)
sko64-4	297,129[d]	0(70.7)	76.7	0(35.8)	71.6	0(23.9)	53.9	0(0)	0	0(114.1)
sko64-5	501922.5[a]	0(0)	0	0(0)	0	0(0)	0	0(0)	0	0(89.7)
sko72-1	139,150[c]	0(0.6)	1.2	0(1.2)	1.5	0(0.3)	0.9	0(0)	0	0(4.1)
sko72-2	711,998[d]	0(192.1)	204.7	0(56.0)	104.7	0(113.7)	139.0	0(0)	0	0(322.3)
sko72-3	1054110.5[b]	0(118.0)	170.4	0(7.1)	14.3	0(80.6)	145.7	0(0)	0	0(199.1)
sko72-4	919586.5[d]	0(16.3)	13.7	0(2.9)	8.7	0(0)	0	0(0)	0	0(15.2)
sko72-5	428226.5[d]	0(26.1)	77.6	0(0)	0	0(0)	0	0(0)	0	0(512.0)
sko81-1	205,106[e]	0(77.2)	79.1	0(3.9)	3.8	0(10.2)	18.3	0(0)	0	0(143.8)
sko81-2	521391.5[c]	0(1.4)	3.0	0(0)	0	0(0)	0	0(0)	0	0(241.4)
sko81-3	970,796[d]	0(107.4)	183.0	0(0)	0	0(0)	0	0(0)	0	0(810.7)
sko81-4	2,031,803[d]	0(21.2)	41.7	0(0)	0	0(31.2)	47.7	0(0)	0	0(972.8)
sko81-5	1,302,711[d]	0(69.2)	133.6	0(0)	0	0(0)	0	0(0)	0	0(1161.8)
sko100-1	378,234[f]	1(5.1)	2.9	0(0)	0	0(0.4)	0.9	0(0)	0	0(100.0)
sko100-2	2076008.5[f]	0(169.4)	483.3	0(0.4)	1.2	0(7.9)	16.9	0(0)	0	0(473.0)
sko100-3	16145614.5[f]	0(5812.2)	6410.5	0(793.5)	1588.0	0(0)	0	0(0)	0	0(5262.3)
sko100-4	3,232,522[f]	0(1208.9)	3336.1	0(4.3)	12.9	0(12.9)	19.7	0(0)	0	0(1996.5)
sko100-5	1033080.5[f]	0(98.5)	112.3	0(1.0)	2.0	0(2.0)	2.4	0(72.0)	110.0	0(558.2)
Average		0.1(399.7)	566.5	0(45.4)	90.6	0(15.3)	25.6	0(3.6)	5.5	0(676.1)

[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).
[g] Results are taken from Palubeckis (2015b).
[h] This column is obtained from the data presented in Guan and Lin (2016).

Table 9
Results of running MSA variants on the adapted *sko* instances (Anjos & Yen, 2009) for 60 s: best (G_{best}) and average (G_{av}) solution gaps to the best known result, standard deviation σ, and the average running time (t given in seconds) to the best solution in a run.

Instance	MSA_1			MSA_0			MSA_0.5		
	$G_{best}(G_{av})$	σ	t	$G_{best}(G_{av})$	σ	t	$G_{best}(G_{av})$	σ	t
sko64-1	0(0)	0	4.8	0(0)	0	6.9	0(0)	0	7.3
sko64-2	0(0)	0	4.2	0(0)	0	7.7	0(0)	0	3.8
sko64-3	0(0)	0	2.4	0(0)	0	1.1	0(0)	0	0.9
sko64-4	0(1.6)	3.2	24.7	0(0)	0	5.9	0(0)	0	12.4
sko64-5	0(0)	0	0.7	0(0)	0	0.9	0(0)	0	0.8
sko72-1	0(0)	0	6.3	0(0)	0	7.4	0(0)	0	5.1
sko72-2	0(0)	0	19.3	0(0)	0	10.1	0(1.0)	3.0	14.2
sko72-3	0(0)	0	14.7	0(0)	0	6.2	0(0)	0	12.5
sko72-4	0(0)	0	13.9	0(0)	0	4.2	0(0)	0	4.1
sko72-5	0(0)	0	6.3	0(0)	0	2.0	0(0)	0	2.0
sko81-1	0(6.6)	9.5	22.3	0(0.6)	0.9	12.7	0(0)	0	23.7
sko81-2	0(0)	0	6.7	0(0)	0	3.7	0(0)	0	3.3
sko81-3	0(0)	0	13.6	0(0)	0	2.4	0(0)	0	1.5
sko81-4	0(0)	0	11.2	0(0)	0	3.6	0(0)	0	6.8
sko81-5	0(0)	0	11.3	0(0)	0	1.0	0(0)	0	1.2
sko100-1	0(0.7)	0.5	31.3	0(0)	0	3.4	0(0)	0	5.1
sko100-2	0(3.0)	8.0	14.0	0(0)	0	4.0	0(0)	0	6.0
sko100-3	0(23.4)	70.2	17.3	0(0)	0	6.5	0(0)	0	4.7
sko100-4	0(0.4)	0.8	26.8	0(0)	0	5.6	0(0)	0	7.1
sko100-5	0(0)	0	23.9	0(0)	0	9.7	0(0)	0	11.9
Average	0(1.8)	4.6	13.8	0(0.0)	0.0	5.3	0(0.1)	0.2	6.7

Table 10
Variation of solution quality with execution time for sko72-2, sko81-1 and sko100-5 instances: best and average (in parentheses) solution gaps to the best known result.

Instance	Algorithm	Time limit (s)						
		2	5	10	30	60	100	200
sko72-2	MSA_1	0(344.7)	0(275.3)	0(192.1)	0(23.5)	0(0)		
sko72-2	MSA_0	0(393.8)	0(157.2)	0(56.0)	0(0)			
sko72-2	MSA_0.5	10(614.9)	0(221.1)	0(113.7)	0(2.0)	0(1.0)	0(0)	
sko72-2	VNS-LS3	0(0)						
sko81-1	MSA_1	87(287.6)	3(117.2)	0(77.2)	0(12.5)	0(6.6)	0(2.0)	0(0)
sko81-1	MSA_0	4(168.2)	0(65.5)	0(3.9)	0(0.8)	0(0.6)	0(0)	
sko81-1	MSA_0.5	6(208.3)	0(74.7)	0(10.2)	0(1.0)	0(0)		
sko81-1	VNS-LS3	0(55.8)	0(37.2)	0(0)				
sko100-5	MSA_1	0(1096.3)	0(411.6)	0(98.5)	0(13.1)	0(0)		
sko100-5	MSA_0	0(435.2)	0(29.7)	0(1.0)	0(0.5)	0(0)		
sko100-5	MSA_0.5	0(400.4)	0(5.8)	0(2.0)	0(0.5)	0(0)		
sko100-5	VNS-LS3	0(303.6)	0(96.0)	0(72.0)	0(72.0)	0(48.0)	0(24.0)	0(0)

Table 11
Comparison of the objective function values for the benchmark instances of Amaral and Letchford (2011): best (G_{best}) and average (G_{av}) solution gaps to the best known result and the standard deviation σ.

Instance	Best known value[a]	MSA_1		MSA_0	MSA_0.5	VNS-LS3[b]	VNSACO[c]
		$G_{best}(G_{av})$	σ	$G_{best} = G_{av} = \sigma$	$G_{best} = G_{av} = \sigma$	$G_{best} = G_{av} = \sigma$	$G_{best}(G_{av})$
SRFLP-110-1	144296664.5	0(110.4)	118.2	0	0	0	0(54993.3)
SRFLP-110-2	86,050,037	0(20.8)	42.0	0	0	0	0(16226.9)
SRFLP-110-3	2234743.5	0(0)	0	0	0	0	0(0)
Average		0(43.7)	53.4	0	0	0	0(23740.1)

[a] Best known value reported by Ozcelik (2012).
[b] Results are taken from Palubeckis (2015b).
[c] This column is obtained from the data presented in Guan and Lin (2016).

Table 12
Comparison of the average running time (in s) to the best solution in a run for the benchmark instances of Amaral and Letchford (2011).

Instance	MSA_1	MSA_0	MSA_0.5	VNS-LS3
SRFLP-110-1	50.0	8.1	14.6	9.6
SRFLP-110-2	29.9	10.3	10.4	4.7
SRFLP-110-3	9.3	1.7	1.9	1.2
Average	29.7	6.7	9.0	5.2

Table 13
Comparison of the objective function values for instances of Palubeckis (2015b): best (G_{best}) and average (G_{av}) solution gaps to the best objective value and the standard deviation σ.

Instance	Best value[a]	MSA_0		MSA_0.5		VNS-LS3[a]	
		$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$	σ
p110	4,435,868	0(0)	0	0(0)	0	0(0)	0
p120	6,282,721	0(0)	0	0(0)	0	0(0)	0
p130	7880929.5	0(0)	0	0(0)	0	0(0)	0
p140	9,257,162	0(0)	0	0(0)	0	0(0)	0
p150	10624389.5	0(13.6)	40.8	0(56.8)	76.9	0(322.0)	322.0
p160	14,873,277	0(4.0)	8.8	0(19.4)	38.8	0(0)	0
p170	16,630,187	0(0)	0	0(1.4)	2.8	0(3680.5)	3680.5
p180	18746031.5	0(0.2)	0.4	0(0.1)	0.3	0(0)	0
p190	24,453,272	0(6.7)	13.4	0(702.7)	723.2	0(481.5)	735.5
p200	27482649.5	0(584.0)	1747.7	0(9.2)	11.0	0(5916.6)	8499.6
p210	29512000.5	0(0)	0	0(0)	0	0(242.3)	726.9
p220	37351850.5	0(0)	0	0(0)	0	0(618.6)	1855.8
p230	46,744,886	0(0)	0	0(0)	0	0(0)	0
p240	46,717,781	0(0)	0	0(0.6)	1.8	0(34.8)	53.2
p250	54526293.5	0(9.0)	18.2	0(7.7)	19.9	0(4577.0)	3851.5
p260	63300360.5	0(346.0)	509.7	0(675.1)	549.2	0(13012.8)	11173.6
p270	68960438.5	0(0.3)	0.9	0(0)	0	0(0)	0
p280	73,845,821	0(14.1)	42.3	0(14.5)	43.5	0(23596.5)	13997.7
p290	86255267.5	0(177.9)	163.8	14(779.5)	1428.0	0(4446.1)	7419.5
p300	95,937,735	0(117.8)	238.7	0(0)	0	0(10459.9)	19335.3
Average		0(63.7)	139.2	0.7(113.4)	144.8	0(3369.4)	3582.6

[a] Best values and VNS-LS3 results are taken from Palubeckis (2015b).

Table 14
Comparison of the average running time (in seconds) to the best solution in a run for instances of Palubeckis (2015b).

Instance	MSA_0	MSA_0.5	VNS-LS3
p110	3.5	6.5	1.1
p120	12.0	5.2	1.6
p130	13.7	14.5	2.8
p140	21.7	26.1	10.9
p150	53.4	42.1	17.9
p160	36.9	27.8	16.9
p170	44.7	25.5	28.6
p180	23.0	26.7	7.0
p190	47.5	33.5	31.0
p200	44.8	43.4	25.9
p210	73.7	60.9	117.4
p220	136.5	117.4	80.9
p230	15.7	32.9	49.4
p240	143.8	217.7	83.3
p250	314.1	280.2	163.3
p260	436.1	275.5	236.3
p270	153.8	135.7	116.8
p280	177.1	172.3	183.4
p290	219.5	291.2	221.3
p300	346.3	261.8	212.7
Average	115.9	104.8	80.4

4.5. Computational results for the sko instances (Anjos & Yen, 2009) ($n \in \{64, 72, 81, 100\}$)

In Table 8, we computationally compare the algorithms for the *sko* instances. The cutoff time for a run of MSA variants and VNS-LS3 was set to 10 s. The information in the table is organized in the same manner as in Table 5. Again, we identify VNS-LS3 as the best-performing method. We also see that MSA_0 and MSA_0.5 usually find solutions that are not far from the best known results. We notice that both MSA_1 and VNSACO perform significantly worse than the other three algorithms.

Like in the case of the Anjos et al. dataset, we extended our experiments with MSA by increasing the maximum time limit per run from 10 to 60 s. The results are summarized in Table 9. We observe that MSA_1 was unable to find the best solution in all runs for 6 *sko* instances whereas each of MSA_0 and MSA_0.5 failed for only one instance. The average running time of the latter two MSA variants is less than 7 s. The slowest variant is MSA_1. It takes twice more time than MSA_0.5. Comparing solution times, we find that the most difficult instance for our algorithm in the *sko* dataset is sko81-1.

In Table 10, we give details regarding the solution quality versus time limit for a small subset of the *sko* instances. We have selected sko72-2, sko81-1 and sko100-5. Each of them appeared to be rather difficult to solve by one of the three best algorithms (MSA_0, MSA_0.5 and VNS-LS3). We can see that the algorithms are capable of reaching the best solution in at least one run of duration 5 s in all cases except for MSA_1 applied to sko81-1. However, finding the best solution in all 10 runs takes up to 200 s per run.

4.6. Computational results for the benchmark instances of Amaral and Letchford (2011) ($n = 110$)

In Table 11, we assess the performance of the considered methods on the instances in the (c) dataset. The maximum CPU time

Table 15
Variation of solution quality with execution time for p200 and p290 instances: best and average (in parentheses) solution gaps to the best objective value.

Instance	Algorithm	Time limit (s)					
		50	100	300	600	1200	1800
p200	MSA_0	0(862.8)	0(584.0)	0(0.2)	0(0.1)	0(0)	
p200	MSA_0.5	0(2367.6)	0(9.2)	0(0.8)	0(0.7)	0(0)	
p200	VNS-LS3	0(6581.5)	0(5916.6)	0(548.2)	0(0)		
p290	MSA_0	43(14196.7)	0(3691.3)	0(261.3)	0(177.9)	0(177.9)	0(120.7)
p290	MSA_0.5	495(17349.0)	325(6659.3)	210(1561.7)	14(779.5)	0(188.0)	0(157.5)
p290	VNS-LS3	208(28243.7)	0(17003.1)	0(6917.6)	0(4446.1)	0(4434.4)	0(1908.8)

Table 16
Comparison of the objective function values for larger-scale SRFLP instances: best (G_{best}) and average (G_{av}) solution gaps to the best objective value and the standard deviation σ.

Instance	Best value	MSA_0		MSA_0.5		VNS-LS3	
		$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$	σ	$G_{best}(G_{av})$	σ
p310	105,754,955	0(41.1)	100.9	0(68.2)	136.4	0(13353.4)	17669.7
p320	119522881.5	0(104.3)	197.1	0(185.5)	242.4	0(13811.6)	13338.1
p330	124891823.5	0(0)	0	0(0)	0	0(13784.4)	28689.2
p340	129796777.5	0(8.2)	16.2	0(18.8)	46.8	0(18129.7)	12708.7
p350	149,594,388	0(1.6)	2.9	0(0.8)	2.4	0(850.6)	1217.9
p360	141,122,187	0(15.2)	16.4	0(945.0)	1747.4	7552(18499.0)	10736.4
p370	174663159.5	0(0)	0	0(0)	0	0(27322.7)	18539.3
p380	189452403.5	0(6106.4)	7581.8	5(10075.2)	6727.4	0(55065.2)	40208.0
p390	208,688,557	0(1.8)	2.7	0(1.6)	2.5	0(25119.8)	58074.3
p400	213768810.5	0(137.8)	405.4	0(372.6)	574.0	0(86822.1)	70000.1
p410	243,494,269	0(130.7)	250.0	0(315.3)	302.2	771(139188.6)	58223.4
p420	270756527.5	0(12.2)	9.7	0(8.0)	11.0	0(61250.8)	67133.8
p430	286334521.5	50(10169.3)	6545.3	5426(11591.7)	5945.7	0(51128.8)	40579.3
p440	301067264.5	0(11.8)	17.8	0(22.2)	23.5	0(51841.0)	69322.9
p450	324,488,485	0(2505.0)	6301.6	0(678.6)	1837.9	0(58852.1)	43175.5
p460	314,884,659	0(23.5)	31.0	0(1506.2)	2287.2	19,046(121696.6)	81326.7
p470	379,529,990	43(5275.6)	3748.4	0(4997.4)	3641.0	4571(83188.3)	72517.9
p480	366,821,075	5(33.8)	26.0	0(453.6)	887.2	64,279(128487.8)	54662.9
p490	413901954.5	24(120.5)	108.6	0(218.7)	360.9	18,216(178850.7)	139882.6
p500	465570835.5	12(7897.4)	9294.1	0(12228.2)	8777.8	27,797(171775.2)	71813.7
p550	587090450.5	0(495.4)	1371.7	12(1784.9)	3760.3	0(156073.4)	109625.5
p600	801567664.5	27(15066.6)	9027.7	5696(21370.7)	9781.3	64,091(205450.0)	128615.1
p650	927,512,834	52(16701.7)	23277.0	0(9438.4)	14634.2	111,697(507097.6)	238047.4
p700	1,158,462,340	110(31815.3)	44511.8	123(58104.8)	60841.9	220,828(701776.0)	253314.3
p750	1438408860.5	84(8082.4)	7095.2	124(19131.5)	12840.1	717,965(984477.5)	212670.7
p800	1,861,593,391	198(55572.0)	27313.4	6433(74295.5)	36803.1	610,509(1208417.9)	471036.3
p850	2,126,675,923	0(80540.5)	92376.0	18(125397.4)	126021.2	884,039(1832777.7)	596218.2
p900	2,600,124,305	1993(86203.3)	51133.4	214(110839.4)	62645.7	1,221,813(2025800.6)	570647.6
p950	2993153592.5	0(42993.6)	78621.7	18,932(192666.8)	119746.0	1,640,934(2979178.8)	782731.3
p1000	3,426,647,267	0(120817.5)	197536.2	110,592(282866.5)	173755.0	3,524,866(4200597.8)	546123.7
Average		86.6(16362.8)	18897.3	4919.2(31319.4)	21812.7	304632.5(537355.5)	162628.3

limit for each run of our algorithms for these instances was set to 100 s. The results of VNSACO are extracted from the paper of Guan and Lin (2016). The authors reported that the average execution time of VNSACO on this dataset was 330.32 s (on a different computer than we used). It can be observed from Table 11 that all three instances were found to be relatively easy for MSA_0, MSA_0.5 and VNS-LS3. The third variant of our SA implementation, MSA_1, once again fails to match the performance of the other two MSA variants. We remark that MSA_1 needed 200 and, respectively, 125 s to reach $G_{av} = 0$ for the first two instances in Table 11. We can also see from the table that the performance of VNSACO is inferior to that of both VNS-LS3 and MSA algorithms.

The results for average running time of the tested methods are shown in Table 12. We notice that VNS-LS3 and MSA_0 take the least time, and MSA_1 is the slowest of the four algorithms. We find that for VNS-LS3, MSA_0 and MSA_0.5 the average CPU time for reaching the best result is much less than the cutoff time of 100 s.

4.7. Computational results for instances of Palubeckis (2015b) ($110 \leqslant n \leqslant 300$)

Based on the results of the previous subsections, we have selected the two best performing MSA variants, namely, MSA_0 and MSA_0.5, for further experimentation. We compared these variants against the VNS-LS3 algorithm. The results for the (d) dataset are summarized in Table 13. Its structure is the same as that of Tables 5, 8 and 11. The integer following "p" in the name of an instance indicates the number of facilities. The maximum CPU time limit imposed on a run of an algorithm was set to 100 s for $n \leqslant 200$ and 600 s for $n > 200$. We note that the gap between the value of the best solution out of 10 runs and the value

Table 17
Comparison of the average running time (in s) to the best solution in a run for larger-scale SRFLP instances.

Instance	MSA_0	MSA_0.5	VNS-LS3
p310	561.1	661.9	386.3
p320	624.6	608.5	400.0
p330	303.0	297.3	365.7
p340	481.8	713.9	423.4
p350	570.0	277.0	559.9
p360	629.1	498.3	593.8
p370	455.8	227.1	717.9
p380	835.9	492.5	428.4
p390	544.6	392.8	749.1
p400	482.3	744.3	429.3
p410	870.2	717.3	754.3
p420	459.7	699.2	955.0
p430	935.5	800.3	796.2
p440	894.3	608.6	1067.4
p450	1260.7	900.3	1317.6
p460	733.5	743.0	1197.2
p470	956.8	638.4	892.0
p480	1059.2	881.6	1106.9
p490	753.4	1105.8	1063.9
p500	991.4	876.6	1254.3
p550	1031.7	1925.2	2271.3
p600	1926.0	1481.8	3128.6
p650	1693.5	2213.7	3132.7
p700	1830.8	1825.8	3054.9
p750	1694.1	1259.8	3468.4
p800	2203.7	2000.7	3503.9
p850	1728.4	2700.5	3537.3
p900	1393.9	2198.4	3460.1
p950	1522.8	2270.3	3445.8
p1000	1718.8	1847.6	3569.9
Average	1038.2	1086.9	1601.0

Table 18
Variation of solution quality with execution time for p500 and p1000 instances: best and average (in parentheses) solution gaps to the best objective value.

Instance	Algorithm	Time limit (s)					
		300	600	1200	1800	2400	3600
p500	MSA_0	14(31334.4)	14(15329.5)	14(10222.5)	12(7897.4)	4(3773.3)	4(3307.8)
p500	MSA_0.5	18,264(40103.0)	5029(22938.7)	5029(18641.4)	0(12228.2)	0(8788.1)	0(7185.4)
p500	VNS-LS3	257,283(477723.9)	139,837(344855.6)	27,797(236570.1)	27,797(171775.2)	13,986(151141.7)	13,986(119750.8)
p1000	MSA_0	46,942(991496.8)	46,942(553855.9)	2278(382632.6)	2278(204432.3)	0(144658.0)	0(120817.5)
p1000	MSA_0.5	140,535(917063.9)	140,535(825328.0)	140,535(652596.7)	140,535(392713.6)	140,535(357972.8)	110,592(282866.5)
p1000	VNS-LS3	4,482,938 (5075815.8)	4,370,158 (4940706.7)	4,015,273 (4723840.3)	3,654,730 (4523764.1)	3,615,922 (4410539.7)	3,524,866 (4200597.8)

given in the second column is zero for each algorithm and each instance except for MSA_0.5 applied to p290. In this case, the gap is equal to 14. Table 13 discloses that MSA provides a significantly better performance than the VNS-LS3 method. The quality of solutions produced by two configurations of MSA is very similar, with MSA_0 having a slight edge. One might expect MSA_0 to provide a more significant advantage over MSA_0.5 because it uses a faster SA variant (see columns for SA_0 and SA_0.5 in Table 4). The number of restarts of SA_0 within MSA_0 (averaged over all runs and all instances in Table 13) is 51.3. In the case of MSA_0.5, the average number of restarts of SA_0.5 is 44.1, which is 14.0% less in comparison with MSA_0. We guess, however, that the combined use of insertion and interchange moves in SA_0.5 leads to a better search diversification. Therefore, MSA_0.5 is able to compensate to some extent for losses due to a reduced number of SA restarts.

Table 14 presents the results for running time, averaged over 10 runs for each problem instance. We can see that VNS-LS3 stops improving on the best solution in a run earlier than both variants of the MSA algorithm. More precisely, VNS-LS3 took less time than MSA_0 as well as MSA_0.5 for 16 problem instances out of 20.

In Table 15, we show how the solution quality depends on the running time for a couple of instances of the (d) dataset. We find that, for p200, 50 (respectively, 1200) seconds per run are sufficient to reach $G_{\text{best}} = 0$ (respectively, $G_{\text{av}} = 0$) by all tested algorithms. The instance p290 is more difficult to solve. The best solution was obtained in 100 s per run by MSA_0 and VNS-LS3 and in 1200s per run by MSA_0.5. However, even after increasing the cutoff time to half an hour the average solution gap remained positive. Nevertheless, we can see that this gap for the MSA variants is much smaller than for the VNS-LS3 algorithm.

4.8. Computational results for new SRFLP instances ($310 \leqslant n \leqslant 1000$)

In our final experiment, we applied our algorithms to even larger problem instances than those in the (d) dataset. The results obtained for these instances (dataset (e)) are reported in Table 16. In order to have a reference point as a basis of comparison we performed one long run of MSA_0 for each instance in the (e) series. The cutoff time for a run was 10 h for $n \leqslant 500$ and 20 h for $n > 500$. The objective function values of the solutions found by MSA_0 are listed in the second column. The remaining data in the table are obtained when the algorithms are run within the following time limits: 1200 s for $n \leqslant 400$, 1800 s for $400 < n \leqslant 500$, and 3600 s for $n > 500$. The solution gaps in each row of the table are computed with respect to the reference value for the corresponding instance.

From the table, we can conclude that MSA_0 is the best and MSA_0.5 is the second best in the experiment with the (e) dataset. The average number of restarts of SA_0 within MSA_0 (respectively, SA_0.5 within MSA_0.5) was 56.8 (respectively, 47.4). This 16.5% reduction in the number of SA restarts for MSA_0.5 is larger than in the case of the (d) dataset with instances of smaller sizes. Because of this fact as well as possibly less gain from using both types of moves for large n, the performance of MSA_0.5 relative to MSA_0 in Table 16 is substantially worse than in Table 13. Nevertheless, we observe that MSA_0.5, like MSA_0, was successful in finding the best solutions in each run for two instances, p330 and p370. We can also see that with increasing n, VNS-LS3 fails to compete with the MSA algorithm. In particular, the average solution gap observed with VNS-LS3 is over 30 times larger than that obtained with MSA_0. By analyzing the results in Table 16, we find that for five instances in the dataset (p600, p700, p750, p800 and p900) none of the approaches was able to match the reference value.

In Table 17, we compare the average running times of the MSA variants and VNS-LS3 algorithm. Considering the (e) dataset as a whole, both MSA_0 and MSA_0.5 are found to be faster than VNS-LS3. However, for a subset of smaller-scale instances (with $n \leqslant 500$), the running times of all algorithms are rather similar. On the other side, for the 10 largest instances (with $n > 500$), a clear advantage in speed of MSA over VNS-LS3 can be observed.

Table 18 shows the dependence of solution quality on the maximum time limit for two instances in the dataset, p500 and p1000. The results suggest the unequivocal superiority of MSA over VNS-LS3 for these problem instances. It can be noticed that the solutions produced by MSA_0 in 300 s runs are much better than the solutions produced by VNS-LS3 in one-hour runs.

In closing this section, we should mention that the C++ code implementing the MSA algorithm for the SRFLP is made publicly available as a benchmark for future comparisons, see (Palubeckis, 2016).

5. Conclusions

In this paper, we have developed a multi-start simulated annealing algorithm for solving the single row facility layout problem. The algorithm can be configured to run in one of the following three modes: (i) with only pairwise interchanges of facilities allowed; (ii) with only insertion moves enabled; and (iii) using both the interchange and insertion moves in a combined manner. We propose $O(n)$-time procedures for computing gains of both types of moves. They make our SA method very fast compared to the traditional approaches. When the temperature during the cooling process drops to a certain level, the algorithm switches to the second phase which is reminiscent of a local search technique — it computes the move gain in time $O(1)$, but however takes $O(n^2)$ time to perform the required calculations when a move (interchange or insertion) is accepted.

We carried out computational experiments on five sets of SRFLP instances containing a total of 93 test problems. The results indicate that, for the largest problem instances, configuration (ii) of MSA shows the best performance among the above listed three configurations. We experimentally compared our SA implementation with the variable neighborhood search algorithm from the literature. We conclude that for large SRFLP instances (exceeding 200 facilities) both configurations (ii) and (iii) of MSA visibly outper-

form VNS in terms of solution quality. The superiority of MSA becomes more pronounced as the problem size increases. Basically, the size of problem instances for which good solutions can be obtained in a reasonable amount of time using the presented MSA algorithm reaches 1000 of facilities.

In this paper, we addressed the most frequently considered formulation of the SRFLP in which all the clearances between adjacent facilities are equal to zero. However, the proposed algorithm can be extended with a little effort to deal with a version of the SRFLP where clearances between facilities are arbitrary nonnegative numbers. In particular, the equations of Propositions 1 and 3 can easily be generalized to apply to this type of the problem.

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

Amaral, A. R. S. (2008b). Enhanced local search applied to the single-row facility layout problem. In *XL SBPO–Simpósio Brasileiro de Pesquisa Operacional* (pp. 1638–1647). João Pessoa, Brazil <http://www.din.uem.br/sbpo/sbpo2008/pdf/arq0026.pdf> Accessed 15.07.16.

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

Amaral, A. R. S. (2009b). A mixed 0–1 linear programming formulation for the exact solution of the minimum linear arrangement problem. *Optimization Letters, 3*(4), 513–520.

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* (Vol. 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.

Birattari, M., & Dorigo, M. (2007). How to assess and report the performance of a stochastic algorithm on a benchmark problem: Mean or best result on a number of runs? *Optimization Letters, 1*(3), 309–311.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications, 45*(1), 41–51.

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.

Guan, J., & Lin, G. (2016). Hybridizing variable neighborhood search with ant colony optimization for solving the single row facility layout problem. *European Journal of Operational Research, 248*(3), 899–909.

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.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

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

Kothari, R., & Ghosh, D. (2012b). *Path relinking for single row facility layout. Working Paper No. 2012-05-01*. India: Indian Institute of Management, Ahmedabad.

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.

Kumar, K. R., Hadjinicola, G. C., & Lin, T.-L. (1995). A heuristic procedure for the single-row facility layout problem. *European Journal of Operational Research, 87*(1), 65–73.

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, Los Angeles, CA, USA* (pp. 578–586).

Love, R. F., & Wong, J. Y. (1976). On solving a one-dimensional space allocation problem with integer programming. *INFOR, 14*(2), 139–143.

Ou-Yang, C., & Utamima, A. (2013). Hybrid estimation of distribution algorithm for solving single row facility layout problem. *Computers and Industrial Engineering, 66*(1), 95–103.

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. (2015a). Fast simulated annealing for single-row equidistant facility layout. *Applied Mathematics and Computation, 263*, 287–301.

Palubeckis, G. (2015b). Fast local search for single row facility layout. *European Journal of Operational Research, 246*(3), 800–814.

Palubeckis, G. (2016). Single row facility layout. Accessed 14.04.16 <http://www.personalas.ktu.lt/~ginpalu/srflp.html>.

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

Rodriguez-Tello, E., Hao, J.-K., & Torres-Jimenez, J. (2008). An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers and Operations Research, 35*(10), 3331–3346.

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.

Sanjeevi, S., & Kianfar, K. (2010). A polyhedral study of triplet formulation for single row facility layout problem. *Discrete Applied Mathematics, 158*(16), 1861–1867.

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. In *Proceedings of the 4th IEEE conference on automation science and engineering* (pp. 597–602). Key Bridge Marriott, Washington, DC, USA: IEEE Computer Society.

Torres-Jimenez, J., Izquierdo-Marquez, I., Garcia-Robledo, A., Gonzalez-Gomez, A., Bernal, J., & Kacker, R. N. (2015). A dual representation simulated annealing algorithm for the bandwidth minimization problem on graphs. *Information Sciences, 303*, 33–49.