Applied Mathematics and Computation 263 (2015) 287–301

Contents lists available at ScienceDirect

Applied Mathematics and Computation

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

Fast simulated annealing for single-row equidistant facility layout

Gintaras Palubeckis*

Faculty of Informatics, Kaunas University of Technology, 51368 Kaunas, Lithuania

ARTICLE INFO

Keywords:
Combinatorial optimization
Single-row facility layout
Quadratic assignment
Simulated annealing

ABSTRACT

Given n facilities and a flow matrix, the single-row equidistant facility layout problem (SREFLP) is to find a one-to-one assignment of n facilities to n locations equally spaced along a straight line so as to minimize the sum of the products of the flows and distances between facilities. We develop a simulated annealing (SA) algorithm for solving this problem. The algorithm provides a possibility to employ either merely pairwise interchanges of facilities or merely insertion moves or both of them. It incorporates an innovative method for computing gains of both types of moves. Experimental analysis shows that this method is significantly faster than traditional approaches. To further speed up SA, we propose a two-mode technique when for high temperatures, at each iteration, only the required gain is calculated and, for lower temperatures, the gains of all possible moves are maintained from iteration to iteration. We experimentally compare SA against the iterated tabu search (ITS) algorithm from the literature. Computational results are reported for SREFLP instances with up to 300 facilities. The results show that the performance of our SA implementation is dramatically better than that of the ITS heuristic.

1. Introduction

The *single-row equidistant facility layout problem* (SREFLP for short) is an important member of a wide set of problems whose solutions are permutations. It can be stated as follows. Suppose that there are n facilities and, in addition, there is an $n \times n$ symmetric matrix, $W = (w_{ij})$, whose entry w_{ij} represents the flow of material between facilities i and j. The SREFLP is to find a one-to-one assignment of n facilities to n locations equally spaced along a straight line so as to minimize the sum of the products of the flows and distances between facilities. It can be assumed without loss of generality that the locations are points on a horizontal line with x-coordinates $1, 2, \ldots, n$. Then, mathematically, the SREFLP can be expressed as:

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

where Π is 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 $p(i) \in I$, $i = 1, \ldots, n$, and $p(i) \neq p(j)$, $i = 1, \ldots, n-1$, $j = i+1, \ldots, n$. Thus $p(i)$, $i \in I$, is the facility assigned to location i.

The applicability of the SREFLP has been reported in several settings. Yu and Sarker [44] considered this problem in the context of designing a cellular manufacturing system. In this application, the goal is to assign machine-cells to locations so that the total

* Tel.: +370 37 300373.
E-mail address: gintaras@soften.ktu.lt, gintaras.palubeckis@ktu.lt

http://dx.doi.org/10.1016/j.amc.2015.04.073
0096-3003/

inter-cell flow costs would be minimized. An important condition is that the locations must be equally spaced in a linear layout. Sarker et al. [37] investigated a similar problem arising in flexible manufacturing. Its objective is to minimize the total machine-to-machine material transportation cost when machines are assigned to locations along a linear material handling track. Picard and Queyranne [32] addressed the problem of designing the physical arrangement of rooms within a department. In this case, the matrix W represents the traffic, i.e. its entry w_{ij} equals the number of trips per unit of time between room i and room j. The quality of a layout is measured by the total travel distance, which is defined as the sum of the door-to-door distances weighted by the traffic between the rooms. When all room lengths are equal, the problem can be modeled as the SREFLP. Cheng [11] and Bhasker and Sahni [8] applied the model (1) in the area of physical design automation of electronic systems. They considered a linear placement problem, which asks to arrange circuit components on a straight line so as to minimize the objective function of the sum of wiring lengths. Other applications of the SREFLP are found in sheet metal fabrication [4], printed circuit board and disk drive assembly [12], and assigning flights to gates in an airport terminal [40].

The SREFLP, as defined by (1), is a special case of the quadratic assignment problem (QAP) formulated by Koopmans and Beckmann [23]. In recent years, a number of high-performing algorithms for the QAP have been proposed, including those presented in [1,6,7,14,15,20,21,39,42]. For a survey of less recent algorithms for the QAP, the reader is referred to Loiola et al. [27]. A good account of results on the QAP can also be found in the monograph by Burkard et al. [9]. However, the algorithms for the SREFLP generally differ from those for the QAP, because they take advantage of the fact that the locations for facilities are equally spaced along a straight line. Therefore, the development of specific algorithms for the SREFLP is an important line of research. There are a few more combinatorial optimization problems related to the SREFLP. One of them is the minimum linear arrangement (MinLA) problem, which asks to find a labeling of the vertices of a given undirected n-vertex graph by consecutive integers from 1 to n such that the sum of edge lengths is minimized, where the length of an edge is the absolute difference of the labels on its end vertices. The graphs arising in applications of the MinLA, for example, in graph drawing tend to be rather sparse. The most successful heuristic techniques for the MinLA problem include the multilevel algorithm based on the algebraic multigrid scheme by Safro et al. [35] and the two-stage simulated annealing algorithm by Rodriguez-Tello et al. [33]. An overview of results in the area of MinLA is presented in a survey by Díaz et al. [13]. Another related problem is the single-row facility layout problem (SRFLP) in which the facilities, represented by rectangles, are allowed to have different lengths. Given the matrix of flows, the problem is to find an arrangement of the facilities next to each other along a horizontal line such that the weighted sum of the center-to-center distances between all pairs of facilities is minimized. Unlike the SREFLP, this problem is not a special case of the QAP. The state of the art on single-row facility layout is surveyed in the papers by Kothari and Ghosh [24] and Hungerländer and Rendl [18]. A natural generalization of the SRFLP is the corridor allocation problem (CAP) considered in [2,3]. This problem asks to find an arrangement of the facilities on both sides of the corridor leaving no space between two adjacent facilities. The objective function of the CAP is essentially the same as that of the SRFLP.

A number of approaches, both exact and heuristic, for the SREFLP have been proposed in the literature. Karp and Held [22] developed an exact algorithm based on the dynamic programming technique. The time complexity of this algorithm is $O(n2^n)$ [22,32]. However, the algorithm can only handle problem instances of moderate size because it requires a very large memory space to store the partial solutions [26]. A branch-and-bound algorithm for solving the SREFLP was presented in [30]. Using it, a number of benchmark instances with up to 35 facilities were solved to optimality. Recently, Hungerländer [17] proposed a semidefinite programming approach for the SREFLP. The author developed an algorithm that was able to solve instances of size up to 40.

It is well known that the SREFLP given by (1) is NP-hard in its general form. Thus, SREFLP instances of larger size can be tackled only using heuristic optimization methods. Suryanarayanan et al. [40] developed a heuristic algorithm based on the multidimensional scaling technique. The solution obtained by applying this technique was further improved using a pairwise interchange method. The authors reported computational results for problem instances of size up to 40 facilities. Sarker et al. [37] proposed an algorithm called the depth-first insertion heuristic (DIH). This heuristic starts with a feasible solution (permutation of facilities) and iteratively tries to improve it. At each iteration, DIH examines all assignments obtained by removing the selected facility from its current position in the permutation and inserting it at each of the other $n-1$ positions. While performing this operation, some of the other facilities are shifted to the neighboring locations. As remarked in [37], the time complexity of DIH is $O(n^4)$. Yu and Sarker [44] presented another algorithm for the SREFLP, called directional decomposition heuristic (DDH). This algorithm bears some similarity to DIH but is computationally more efficient. In [30], an iterated tabu search (ITS) algorithm for solving the SREFLP was proposed. A tabu search procedure lies at the core of a dominance test incorporated into a branch-and-bound algorithm described in [30]. As the results of numerical experiments provided in that paper show, ITS performs considerably better than the DDH method.

The primary motivation of this paper is to investigate computationally the applicability of the simulated annealing (SA) metaheuristic to the single-row equidistant facility layout problem. We consider two move types, pairwise interchanges of facilities and insertions (when a facility is removed from its current position and inserted at a different position in the permutation). We propose an original method to compute the move gains, that is, the differences between the objective function values of the new and current solutions. Interestingly, this method operates on the same auxiliary data for both move types. Another innovative aspect of the work presented in this paper is the use of the two modes of calculating move gains. For high temperatures, we adopt the traditional approach when, at each iteration of the SA algorithm, first a pair of facilities (respectively, a facility and its new position) are selected and then the gain is computed in $O(n)$ (respectively, $O(n^2)$) operations. For lower temperatures, our strategy is to maintain, throughout the cooling process, special matrices and vectors that allow computing the gain of any pairwise interchange or insertion move in $O(1)$ time. The speedup provided by this approach is significant. We have implemented

the SA algorithm as a multi-start procedure, in which starting solutions are generated randomly. We experimentally compare the developed algorithm against the ITS method proposed in [30]. We note that the comparisons of simulated annealing algorithms with tabu search procedures for the quadratic assignment problem were presented in [19,31]. One of the SA implementations in [19] is SA with restarts. The cooling process is restarted by resetting the temperature to its initial value. Simulated annealing and tabu search approaches for the CAP are compared in [2]. Experimental results show that, for this problem, the SA algorithm is superior to the tabu search technique.

The remainder of this paper is arranged as follows. In the next section, we present a general scheme of the simulated annealing adaptation for solving the SREFLP. In Section 3, we propose a novel method for computing move gains and describe two implementations of generic simulated annealing. Section 4 is devoted to an experimental analysis and comparisons of algorithms. Concluding remarks are given in Section 5.

2. General scheme

The simulated annealing metaheuristic is a well-known general-purpose optimization method exploiting an analogy between the annealing of solids and the search for an optimal solution in an optimization problem. In physics, annealing refers to a thermal process in which a solid is first heated up to a very high temperature and then slowly cooled down until the minimum energy level is reached. The simulated annealing technique is designed to mimic this process. It generates a sequence of solutions whose objective function values converge to the optimum value. An important feature of this technique is that it accepts deteriorations of the objective function to a limited extent. This prevents the search from getting trapped in local optima at early stages.

Since the literature on simulated annealing is abundant, it is not one of the purposes of this section to provide a detailed description of this metaheuristic. Instead, we restrict ourselves to presenting the steps of the algorithm for the SREFLP and giving some comments, in particular on the simulated annealing parameters. We call the algorithm GSA (Generic Simulated Annealing). Two implementations of the GSA framework are described in the next section. The algorithm can be stated as follows.

Generic Simulated Annealing (GSA)

1. Randomly generate a permutation $p \in \Pi$. Initialize p^* with p and F^* with $F(p)$.
2. Compute the initial temperature $T_{\max}$. Set $\bar{v} := \lfloor (\log(T_{\min}) - \log(T_{\max}))/\log\alpha \rfloor$.
3. Initialize f with $F(p)$ and T with $T_{\max}$.
4. For $v = 1, \ldots, \bar{v}$ do the following:
   4.1. For $z = 1, \ldots, \bar{z}$ do the following:
      4.1.1. Randomly select a solution p' in a neighborhood of p. Compute $h = F(p') - F(p) = F(p') - f$. If $h \le 0$, then go to 4.1.3. Otherwise go to 4.1.2.
      4.1.2. Randomly draw a number ξ from the uniform distribution on [0, 1]. If $\xi \le \exp(-h/T)$, then proceed to 4.1.3; else repeat 4.1.1–4.1.3 for the next z value, if any.
      4.1.3. Replace p with p' and, if auxiliary data are used in h calculation, update these data. Set $f := f + h$. If $f < F^*$, then set $p^* := p'$ and $F^* := f$.
   4.2. If GSA is run in a multi-start fashion, then go to 4.3. Otherwise go to 4.4.
   4.3. Check if the termination condition is satisfied. If so, then stop with the solution p^* of value F^*. If not, then proceed to 4.4.
   4.4. Reduce the temperature by setting $T := \alpha T$.
5. If GSA is run in a multi-start fashion, then randomly generate a new starting permutation p and return to 3. Otherwise, stop with the solution p^* of value F^*.

The variable $T_{\max}$ in the above algorithm stands for the initial temperature. To compute $T_{\max}$, we take the solution p of Step 1 as a basis and generate a sample of permutations, each obtained from p by interchanging two randomly selected facilities. We set $T_{\max}$ to the largest absolute value of $F(p') - F(p)$ over permutations p' in this sample. The size of the sample in our experiments was fixed at 5000. The parameter $T_{\min}$ of GSA is the final temperature of the cooling process. Typically, $T_{\min}$ is set to a positive number very close to zero. The temperature is decreased by a cooling factor α as cooling progresses. The value of α should be taken close to 1, usually between 0.9 and 0.995. Another core parameter of the algorithm is the number of moves, $\bar{z}$, performed at a temperature level. A reasonable choice is to take $\bar{z} = \bar{z}_0 n$, where $\bar{z}_0$ is a predefined positive constant, for example, 100.

One may notice that the outer loop index v and the inner loop index z are present in the description of GSA but are nowhere used in the computations. We expose these variables here because they play an important role in the implementations of GSA described in the next section. There we will also define the neighborhood structures used in the algorithm and provide an elaboration of Steps 4.1.1 and 4.1.3 of GSA.

As a final remark, we note that GSA can be run both as a conventional simulated annealing algorithm terminating upon reaching the lower bound on the temperature and as a multi-start method repeating the cooling process with the same initial temperature and different starting permutation. We capitalize on this possibility in Section 4 where experimental results are presented. When applying GSA in a multi-start regime, the termination condition used in our computational experiments was a maximum CPU time limit for a run.

3. Computing move gains

In this section, we will discuss the ways in which $h = F(p') - F(p)$ in Step 4.1.1 of GSA can be computed. The difference h between the objective function values of the new and current solutions is called the move gain. The two main move types in the case of the SREFLP are pairwise interchanges of facilities and insertions. The pairwise interchange neighborhood of $p \in \Pi$, denoted by $N_2(p)$, 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, $N_2(p) = \{p' \in \Pi \,|\, p' \text{ and } p \text{ differ on exactly two entries}\}$. The insertion neighborhood of $p \in \Pi$, denoted by $N_1(p)$, 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. To state this more formally, let $I = \{1, \ldots, n\}$. Then $N_1(p) = \{p' \in \Pi \,|\, \text{there exist } k, l \in I, k \neq l, \text{ such that } p'(l) = p(k), p'(i) = p(i-1), i = l+1, \ldots, k, \text{ if } l < k, p'(i) = p(i+1), i = k, \ldots, l-1, \text{ if } l > k, \text{ and } p'(i) = p(i) \text{ for the remaining } i \in I\}$.

3.1. Pairwise interchanges

For $p \in \Pi$, suppose that $p' \in N_2(p)$ is obtained from p by interchanging facilities r and s. Let us denote the gain of this move by $\Delta(p, r, s) = F(p') - F(p)$. To provide a formula for $\Delta(p, r, s)$, we need a couple of notations. We will write $\bar{p}$ to refer to the inverse permutation of p. Thus, if $p(k) = r$, then $\bar{p}(r) = k$. We will use $D = (d_{ij})$ to denote the distance matrix of the problem with entries $d_{ij} = |i - j|$ for $i, j = 1, \ldots, n$. With these notations, the interchange move gain can be computed as follows:

$$\Delta(p, r, s) = \sum_{m=1, m\neq k, l}^{n} (w_{rp(m)} - w_{sp(m)})(d_{lm} - d_{km}) \tag{2}$$

where $k = \bar{p}(r)$ and $l = \bar{p}(s)$. This formula is well known in the literature on quadratic assignment and related problems (see, for example, [10]).

Suppose that (2) is used to get $h = \Delta(p, r, s)$ in Step 4.1.1 of GSA. Then the total complexity of h calculations in GSA without restarts is $O(\bar{v}\bar{z}n)$. An alternative approach is to initialize and maintain a matrix of gains throughout the cooling process. Let us denote this matrix by $H = (\Delta(p, i, j))$. Its entries are computed like in (2):

$$\Delta(p, i, j) = \sum_{m=1, m\neq \bar{p}(i), \bar{p}(j)}^{n} (w_{jp(m)} - w_{ip(m)})(d_{\bar{p}(j)m} - d_{\bar{p}(i)m}), \quad i, j = 1, \ldots, n. \tag{3}$$

After swapping positions of facilities r and s, the matrix H should be updated. For $i, j \in I\backslash\{r, s\}$, $\Delta(p', i, j)$ can be computed in constant time using the following expression (known from [41] and other papers):

$$\Delta(p', i, j) = \Delta(p, i, j) + (w_{ir} - w_{is} + w_{js} - w_{jr})(d_{\bar{p}(i)k} - d_{\bar{p}(i)l} + d_{\bar{p}(j)l} - d_{\bar{p}(j)k}). \tag{4}$$

Upon setting $p := p'$, $\Delta(p', i, j)$ becomes an entry of the updated matrix H. For pairs (i, r) and (i, s), $i \in I\backslash\{r, s\}$, it is reasonable to compute $\Delta(p, i, r)$ and $\Delta(p, i, s)$ anew using (3). Because facilities r and s have been interchanged, this formula must be applied with $\bar{p}(r) = l$ and $\bar{p}(s) = k$. The final operation is to invert the sign of both $\Delta(p, r, s)$ and $\Delta(p, s, r)$. It is easy to see that the matrix H can be initialized in time $O(n^3)$ and updated, after a pairwise interchange move, in time $O(n^2)$. Suppose that the matrix H corresponding to the current solution p exists at the beginning of Step 4.1.1. Then the gain h is an entry of H, and this step performs only a constant number of operations. Thus the accumulated complexity of all executions of Steps 4.1.1 and 4.1.3 of GSA without restarts is $O(\bar{v}\bar{z} + tn^2 + n^3)$, where t is the number of times Step 4.1.3 is performed. It is expected that this alternative approach is slower than the traditional method based on applying (2) in Step 4.1.1 of GSA. The idea advocated in this paper is to combine both methods. Let $t(v)$ denote the number of times Step 4.1.3 is executed during the v-th iteration of the outer "for" loop in Step 4 of GSA. We have observed that $t(v)$ for larger values of v, which correspond to lower temperatures, is very small in comparison to $\bar{z}$ (which in our algorithm is equal to $100n$). Motivated by this observation, we introduce a parameter $\gamma \in [0, 1]$ and combine the two described methods as follows: we calculate the interchange gain $h = \Delta(p, r, s)$ by (2) for $v = 1, \ldots, \lfloor \gamma \bar{v} \rfloor$, and use the gain matrix H for $v = \lfloor \gamma \bar{v} \rfloor + 1, \ldots, \bar{v}$. The accumulated time complexity of Steps 4.1.1 and 4.1.3 of GSA in this approach is $O(\gamma \bar{v}\bar{z}n + (1-\gamma)\bar{v}\bar{z} + t_\gamma n^2 + (1 - \lfloor \gamma \rfloor)n^3) = O(\bar{v}\bar{z}(\gamma(n-1)+1) + t_\gamma n^2 + (1 - \lfloor \gamma \rfloor)n^3)$, where $t_\gamma = \sum_{v=\lfloor \gamma \bar{v} \rfloor + 1}^{\bar{v}} t(v)$ (for notational simplicity, we assume here and in the rest of this section that a sum is zero if the lower limit of the summation index is greater than the upper limit). We call γ a *switch parameter* because it switches the mode of calculating move gains.

We have been doing some preliminary computational experiments with GSA and have found that updating the gain matrix H in Step 4.1.3 is quite expensive. Next, we will describe another technique for getting $h = \Delta(p, r, s)$ in Step 4.1.1 of GSA in constant time. This technique uses two matrices, A and B, with rows corresponding to facilities and columns corresponding to permutation positions. Each of these matrices depends on permutations. Therefore, to define them, we fix $p \in \Pi$. Consider facility u and denote $\bar{p}(u)$ by m. In other words, suppose that $p(m) = u$. The entry a_{ui} of the matrix $A = (a_{ui})$ represents the total flow of material between facility u and all facilities to the left (if $i < m$) or right (if $i > m$) of u up to and including $p(i)$:

$$a_{ui} = \begin{cases} \sum_{q=i}^{m-1} w_{up(q)} & \text{if} \quad i < m \\ \sum_{q=m+1}^{i} w_{up(q)} & \text{if} \quad i > m \\ 0 & \text{if} \quad i = m. \end{cases} \tag{5}$$

The entry b_{uj} of the matrix $B = (b_{uj})$ is defined similarly, with the difference being that now entries of A instead of flows are summed up:

$$b_{uj} = \begin{cases} \sum_{i=j}^{m-1} a_{ui} & \text{if} \quad j < m \\ \sum_{i=m+1}^{j} a_{ui} & \text{if} \quad j > m \\ 0 & \text{if} \quad j = m. \end{cases} \tag{6}$$

Notice that in both cases the sum is performed over the same set of indices. Clearly, b_{uj} can be expressed in terms of flows. For example, if $j < m$, then $b_{uj} = \sum_{i=j}^{m-1} w_{up(i)}(i - j + 1)$. However, in order to recursively calculate the entries of each row of B we prefer to use (5) and (6).

Along with matrices A and B, we also need a couple of vectors. The first of them, called *cut vector*, is denoted by $C = (c_1, \ldots, c_{n-1})$. Like A and B, the vector C is associated with a permutation of facilities. Given $p \in \Pi$, its component c_i, $i \in \{1, \ldots, n-1\}$, is equal to the total flow of material between the first i facilities in p and the remaining facilities in p. Formally, $c_i = \sum_{k=1}^{i} \sum_{l=i+1}^{n} w_{p(k)p(l)}$. From this definition, it follows that the components of the cut vector C, called *cut values*, can be computed using the following recurrence:

$$c_i = c_{i-1} - \sum_{k=1}^{i-1} w_{p(i)p(k)} + \sum_{l=i+1}^{n} w_{p(i)p(l)}, \quad i = 1, \ldots, n-1, \tag{7}$$

with the initial condition $c_0 = 0$. It is not hard to see that, using C, the objective function in (1) can be rewritten as

$$F(p) = \sum_{i=1}^{n-1} c_i. \tag{8}$$

Continuing to assume $c_0 = 0$, suppose also that $c_n = 0$. The second vector, $C^- = (c_1^-, \ldots, c_n^-)$, is derived from C:

$$c_i^- = c_i - c_{i-1}, \, i = 1, \ldots, n. \tag{9}$$

From (7) and the definition of the matrix A, we can write

$$c_i^- = a_{p(i)n} - a_{p(i)1}. \tag{10}$$

Now we can present the expression for $\Delta(p, r, s)$.

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

$$\Delta(p, r, s) = (c_l^- - c_k^- + 2w_{rs})d_{kl} + 2(b_{r,l-1} + b_{s,k+1}). \tag{11}$$

If $l = k + 1$, then $\Delta(p, r, s)$ reduces to $c_l^- - c_k^- + 2w_{rs}$.

Proof. Let p' denote the permutation obtained from p by interchanging facilities r and s. Let $C' = (c'_1, \ldots, c'_{n-1})$ stand for the cut vector corresponding to the solution p'. Since $c'_i = c_i$ for $i = 1, \ldots, k-1$ and $i = l, \ldots, n-1$, it follows from (8) that

$$\Delta(p, r, s) = F(p') - F(p) = \sum_{i=k}^{l-1} (c'_i - c_i). \tag{12}$$

For $i \in \{k, \ldots, l-1\}$, the i-th cut value for p' is

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

Using (10), we get

$$c'_i - c_i = c_l^- - c_k^- + 2w_{rs} + 2a_{ri} + 2a_{s,i+1}. \tag{13}$$

Observe that $\sum_{i=k}^{l-1} a_{ri} = \sum_{i=k+1}^{l-1} a_{ri} = b_{r,l-1}$. Similarly, $\sum_{i=k}^{l-1} a_{s,i+1} = \sum_{i=k+1}^{l} a_{si} = \sum_{i=k+1}^{l-1} a_{si} = b_{s,k+1}$. Substituting (13) into (12) and taking into account the fact that $l - k = d_{kl}$ we arrive at (11). If $l = k + 1$, then $b_{r,l-1} = 0$, $b_{s,k+1} = 0$, $d_{kl} = 1$, and the right-hand side of (11) becomes equal to $c_l^- - c_k^- + 2w_{rs}$. □

To complete a description of our method for gain calculation, we must say how initialization and updating of the matrices A and B are performed. For both purposes, we use the following equations, which are obtained directly from the definition of A and B:

$$a_{ui} = a_{u,i-1} + w_{up(i)}, \quad i > m, \tag{14}$$

$$a_{ui} = a_{u,i+1} + w_{up(i)}, \quad i < m, \tag{15}$$

$$b_{ui} = b_{u,i-1} + a_{ui}, \quad i > m, \tag{16}$$

$$b_{ui} = b_{u,i+1} + a_{ui}, \quad i < m. \tag{17}$$

To compute the cut vector C, we apply the following variation of (10):

$$c_i = c_{i-1} + a_{p(i)n} - a_{p(i)1}, \quad i \in \{1, \ldots, n-1\}. \tag{18}$$

The time complexity of the gain calculation using (11), data (i.e., A, B, C and C^-) initialization, and data updating is $O(1)$, $O(n^2)$ and $O(n^2)$, respectively. Indeed, for example, the row of the matrix A for facility u with $\bar{p}(u) = m$ can be formed recursively first by setting $a_{um} = 0$ and then using Eqs. (14) and (15). This clearly takes $O(n)$ time. Thus, the whole matrix A can be computed by performing $O(n^2)$ operations. We remind that the time complexity of initializing the matrix H is $O(n^3)$. Meanwhile, the complexity of data updating in the case of the method based on the matrix H is the same as in our new technique.

An implementation of GSA, relying on Eq. (11), will be described in Section 3.3. Specific procedures for initialization and updating of the matrices A, B and vectors C, C^- will also be given there.

3.2. Insertions

Given $p \in \Pi$, let $p' \in N_1(p)$ be obtained from the permutation p by relocating facility r from position $k = \bar{p}(r)$ to position l. This operation also touches facilities in the segment of p from position l to position $k-1$ if $l < k$ and from position $k+1$ to position l if $l > k$. In the former case, they are shifted to the right by one position and in the latter case, they are shifted to the left. We denote the difference between objective function values $F(p')$ and $F(p)$ (the insertion gain) by $\delta(p, r, l)$. The straightforward method to calculate the insertion gain is based on the following equations. If $l < k$, then

$$\delta(p, r, l) = \sum_{m=1, m\neq k}^{n} w_{rp(m)}\rho_m + \sum_{j=l}^{k-1}\left(\sum_{m=1}^{l-1} w_{p(j)p(m)} - \sum_{m=k+1}^{n} w_{p(j)p(m)}\right) \tag{19}$$

where

$$\rho_m = \begin{cases} d_{l,m+1} - d_{km} & \text{if } l \leqslant m < k \\ d_{lm} - d_{km} & \text{otherwise.} \end{cases}$$

If $l > k$, then

$$\delta(p, r, l) = \sum_{m=1, m\neq k}^{n} w_{rp(m)}\rho'_m + \sum_{j=k+1}^{l}\left(\sum_{m=l+1}^{n} w_{p(j)p(m)} - \sum_{m=1}^{k-1} w_{p(j)p(m)}\right) \tag{20}$$

where

$$\rho'_m = \begin{cases} d_{l,m-1} - d_{km} & \text{if} \quad k < m \leqslant l \\ d_{lm} - d_{km} & \text{otherwise.} \end{cases}$$

The first term in (19) as well as in (20) is the contribution to the insertion gain from the facility r if moved to its new position. The second term there represents the change of the objective function value that occurs as a result of shifting facilities $p(l), \ldots, p(k-1)$ right if $l < k$ or $p(k+1), \ldots, p(l)$ left if $l > k$. It is clear that the computation of the second term and consequently $\delta(p, r, l)$ takes $O(n^2)$ time. Thus the insertion move is much more expensive than the pairwise interchange move. The complexity of all executions of Steps 4.1.1 and 4.1.3 of insertion-based GSA without restarts is $O(\bar{\nu}\bar{z}n^2)$.

Our alternative method for insertion gain calculation is centered on applying the matrices A, B and vectors C, C^-, precisely as in the case of interchange moves. As we can see from the following result, given B and C, the gain can be computed in constant time.

Proposition 2. *For $p \in \Pi$, $k, l \in \{1, \ldots, n\}$, $k \neq l$, and $r = p(k)$,*

$$\delta(p, r, l) = \begin{cases} c_{l-1} - c_{k-1} + c_k^- d_{kl} + 2b_{rl} & \text{if} \quad l < k \\ c_l - c_k - c_k^- d_{kl} + 2b_{rl} & \text{if} \quad l > k. \end{cases} \tag{21}$$

Proof. Let p' be the permutation obtained from p by removing the facility r from position k and inserting it at position l. First consider the case of $l > k$. Then, using (8), we get

$$\delta(p, r, l) = F(p') - F(p) = \sum_{i=k}^{l-1}(c'_i - c_i), \tag{22}$$

where c'_i, $i = k, \ldots, l-1$, are the cut values for the permutation p'. From the definition of p', for $i \in \{k, \ldots, l-1\}$, we have

$$c'_i = c_i + \sum_{q=1}^{k-1} w_{rp(q)} - \left(\sum_{q=k+1}^{n} w_{rp(q)} - \sum_{q=k+1}^{i} w_{rp(q)} - w_{rp(i+1)} \right) + \sum_{q=k+1}^{i} w_{rp(q)} + \sum_{q=i+2}^{n} w_{p(i+1)p(q)} - \left(\sum_{q=1}^{i} w_{p(i+1)p(q)} - w_{rp(i+1)} \right)$$
$$= c_i + a_{r1} - a_{rn} + 2a_{ri} + a_{p(i+1)n} - a_{p(i+1)1} + 2w_{rp(i+1)}.$$

Making use of (10) and the fact that $a_{ri} + w_{rp(i+1)} = a_{r,i+1}$ gives

$$c'_i - c_i = c^-_{i+1} - c^-_k + 2a_{r,i+1}. \tag{23}$$

Substituting (23) into (22) we obtain three sums $S_1 = \sum_{i=k}^{l-1} c^-_{i+1}$, $S_2 = \sum_{i=k}^{l-1} c^-_k$ and $S_3 = \sum_{i=k}^{l-1} a_{r,i+1}$. It is easy to see that $S_1 = \sum_{i=k}^{l-1}(c_{i+1} - c_i) = c_l - c_k$ and $S_2 = c^-_k d_{kl}$. By making the change of the index variable $i' = i + 1$, we rewrite the third sum as $S_3 = \sum_{i'=k+1}^{l} a_{ri'}$. So, according to (6), $S_3 = b_{rl}$. Putting the sums S_1, S_2 and S_3 together, we prove the case $l > k$ of (21).

Now suppose that $l < k$. The idea of the proof is to apply the formula we have derived for the case where $l > k$. For permutation $p \in \Pi$, let $\hat{p}$ be its reverse defined by $\hat{p}(i) = p(n - i + 1)$, $i = 1, \ldots, n$. We denote by $\hat{c}_i$, $i = 1, \ldots, n-1$, and $\hat{c}^-_i$, $i = 1, \ldots, n$, the components of the cut vector C and, respectively, vector C^- for the solution $\hat{p}$. It is clear that $F(\hat{p}) = F(p)$ and $F(\hat{p}') = F(p')$, where $\hat{p}'$ is the reverse of p'. The facility r is placed in position $n - k + 1$ in the permutation $\hat{p}$. Its new position $n - l + 1$ is to the right of its current position, so the second expression in (21) is applicable. Taking the above facts into account, we can write

$$\delta(p, r, l) = \delta(\hat{p}, r, n - l + 1) = \hat{c}_{n-l+1} - \hat{c}_{n-k+1} - \hat{c}^-_{n-k+1} d_{n-k+1,n-l+1} + 2\hat{b}_{r,n-l+1},$$

where $\hat{b}_{r,n-l+1}$ is the $(r, n-l+1)$-th entry of the matrix B defined with respect to $\hat{p}$. Since $\hat{c}_{n-l+1} = c_{l-1}$, $\hat{c}_{n-k+1} = c_{k-1}$, $\hat{c}^-_{n-k+1} = \hat{c}_{n-k+1} - \hat{c}_{n-k} = c_{k-1} - c_k = -c^-_k$, $d_{n-k+1,n-l+1} = d_{kl}$ and $\hat{b}_{r,n-l+1} = b_{rl}$ it follows that $\delta(p, r, l) = c_{l-1} - c_{k-1} + c^-_k d_{kl} + 2b_{rl}$. The proof is complete. □

We notice that the expression in (21) for $l < k$ can be derived similarly as it has been done in the case of $l > k$. We, however, preferred to use a different technique that takes advantage of the fact that the pair $p, \hat{p}$ represents essentially a single solution to the problem.

The complexity estimates for the method we have just described are the same as those for the interchange gain calculation method based on matrices A, B and vectors C, C^-. Indeed, given B and C, the complexity of computing the gain δ by (21) is $O(1)$. Using (14)–(17), the matrices A and B can be initialized in $O(n^2)$ time. After performing insertion operation, these matrices can be re-initialized using the same approach. However, in the next subsection we will present a slightly faster algorithm, yet having the same quadratic worst-case complexity. The time taken to initialize and update (by (18)) the vector C as well as C^- is linear in n. Thus the bottleneck of the approach is processing the matrices A and B.

3.3. Implementations of generic simulated annealing

Our first implementation of GSA rests on the use of Propositions 1 and 2. A nice feature of this approach is that for gain calculation it employs data (matrices A, B and vectors C, C^-) that are common to both move types, pairwise interchanges and insertions. However, as outlined in Section 3.1, we use Propositions 1 and 2 only when the outer loop index v in Step 4 of GSA is in the range between $\lfloor \gamma \bar{v} \rfloor + 1$ and $\bar{v}$. Here γ and $\bar{v}$ are the parameters of the algorithm. If $v \in \{1, 2, \ldots, \lfloor \gamma \bar{v} \rfloor\}$, then the gain is computed either by (2) or by (19) (or (20)) depending on which move, interchange or insertion, is selected. The move type at each iteration is chosen at random. To speed up computation of Δ and δ values, it is useful to set up the distance matrix D in advance instead of calculating its entries each time when they are needed. Step 4.1.1 in the first implementation of GSA takes the following form:

4.1.1. Randomly draw a number ζ from the uniform distribution on $[0, 1]$. If $\zeta \le P$, then set the case indicator K to 1 and proceed to 4.1.1.1. Otherwise, set $K := 0$ and go to 4.1.1.2.
  4.1.1.1. Randomly select a pair of facilities r, s. If $v \leqslant \gamma \bar{v}$, then compute $h = \Delta(p, r, s)$ using (2) and go to 4.1.1.3. Otherwise, check whether $v = \lfloor \gamma \bar{v} \rfloor + 1$ and $z = 1$. If so, then call the procedure InitData to initialize the matrices A, B and vectors C, C^-. If not, then go directly to h calculation. In both cases, assuming w.l.o.g. that $k = \tilde{p}(r) < \tilde{p}(s) = l$, set h to $\Delta(p, r, s)$ computed by (11) and go to 4.1.1.3.
  4.1.1.2. Randomly select a facility r and its new position $l \ne \tilde{p}(r)$ in the permutation p. If $v \leqslant \gamma \bar{v}$, then compute $h = \delta(p, r, l)$ by (19) or (20) depending on whether $l < k$ or $l > k$ and go to 4.1.1.3. Otherwise, check whether $v = \lfloor \gamma \bar{v} \rfloor + 1$ and $z = 1$. If so, then call the procedure InitData to initialize the matrices A, B and vectors C, C^-. If not, then go directly to h calculation. In both cases, set h to $\delta(p, r, l)$ computed by (21) and proceed to 4.1.1.3.
  4.1.1.3. If $h \le 0$, then go to 4.1.3. Otherwise go to 4.1.2.

In the algorithm, P stands for the probability to select a pairwise interchange move. If $0 < P < 1$, then both neighborhood structures $N_1(p)$, $p \in \Pi$, and $N_2(p)$, $p \in \Pi$, are employed in GSA. The binary variable K denotes the type of move: its value of 1 means that an interchange will be chosen, and a value of 0 indicates that an insertion move will be randomly selected. The case indicator K is used in a variant of Step 4.1.3 of GSA, which will be given below. The variables v and z are loop counters introduced

in the description of GSA (see Section 2). As soon as v reaches $\lfloor \gamma \bar{v} \rfloor + 1$, the procedure InitData is invoked. Provided $\gamma < 1$, InitData is executed once per restart of the simulated annealing schedule. This procedure consists of the following steps.

InitData(p)

1. For $u = 1, \ldots, n$ do the following:
   1.1. Set $a_{um} := 0$, $b_{um} := 0$, where $m = \bar{p}(u)$.
   1.2. For $i = m + 1, \ldots, n$, compute a_{ui} by (14) and b_{ui} by (16).
   1.3. For $i = m - 1, m - 2, \ldots, 1$, compute a_{ui} by (15) and b_{ui} by (17).
2. Set $c_0 := 0$ and $c_n := 0$. For $i = 1, \ldots, n - 1$, compute c_i by (18) and c_i^- by (9). Set $c_n^- := -c_{n-1}$.

In Step 1 of InitData, the matrices A and B are built using Eqs. (14)–(17). In Step 2, the vectors C and C^- are initialized. Obviously, the time complexity of the procedure is determined by Step 1 and is $O(n^2)$.

The presented version of Step 4.1.1 of GSA requires specific implementation of Step 4.1.3. A description of this step is as follows:

4.1.3. If $K = 1$, then go to 4.1.3.1; else go to 4.1.3.2.
   4.1.3.1. Replace p with the solution p' obtained from p by interchanging the facilities r and s. If $v > \gamma \bar{v}$, then apply the procedure UpdateDataInterchange to update the matrices A, B and vectors C, C^-. Go to 4.1.3.3.
   4.1.3.2. Replace p with the solution p' obtained from p by removing the facility r from position $\bar{p}(r)$ and inserting it at position l. If $v > \gamma \bar{v}$, then apply the procedure UpdateDataInsertion to update the matrices A, B and vectors C, C^-.
   4.1.3.3. Set $f := f + h$. If $f < F^*$, then set $p^* := p'$ and $F^* := f$.

If the loop counter v exceeds $\gamma \bar{v}$, then Step 4.1.3 makes a call to either procedure UpdateDataInterchange or procedure UpdateDataInsertion, depending on the value of the case indicator K. The first of them is executed right after performing a pairwise interchange move. The parameter p of the procedure represents an updated permutation. Other two parameters, k and l, specify positions of the facilities s and, respectively, r in p. The procedure can be described as follows.

UpdateDataInterchange(p, k, l)

1. For $u = 1, \ldots, n$ do the following:
   1.1. Set $m := \bar{p}(u)$.
   1.2. If $m < k$, then compute a_{ui}, $i = k, \ldots, l - 1$, by (14) and b_{ui}, $i = k, \ldots, n$, by (16). Go to 1.5.
   1.3. If $m > l$, then compute a_{ui}, $i = l, l - 1, \ldots, k + 1$, by (15) and b_{ui}, $i = l, l - 1, \ldots, 1$, by (17). Go to 1.5.
   1.4. If $k < m < l$, then set $i_1 := k$ and $i_2 := l$. Otherwise, set $i_1 := m - 1$, $i_2 := m + 1$, $a_{um} := 0$ and $b_{um} := 0$. For $i = i_1, i_1 - 1, \ldots, 1$, compute a_{ui} by (15) and b_{ui} by (17). For $i = i_2, \ldots, n$, compute a_{ui} by (14) and b_{ui} by (16).
   1.5. Repeat from 1.1 for the next value of u, if any.
2. For $i = k, \ldots, l - 1$, compute c_i by (18) and c_i^- by (9). Set $c_l^- := c_l - c_{l-1}$.

It can be noted that UpdateDataInterchange uses the same set of Eqs. (14)–(18) and (9) as InitData. The difference between the two procedures is that UpdateDataInterchange recalculates only a subset of entries of each of the matrices A and B and vectors C and C^-. Much of these data remain unchanged. Therefore, UpdateDataInterchange runs faster than InitData although its worst-case complexity is still $O(n^2)$.

The parameters of the procedure UpdateDataInsertion are the updated permutation p and positions k and l of the facility r in the permutation before and, respectively, after performing the insertion move. The procedure goes as follows.

UpdateDataInsertion(p, k, l)

1. Set $\lambda := \min(k, l)$, $\mu := \max(k, l)$.
2. For $u = 1, \ldots, n$ do the following:
   2.1. Set $m := \bar{p}(u)$.
   2.2. If $m < \lambda$, then compute a_{ui}, $i = \lambda, \ldots, \mu - 1$, by (14) and b_{ui}, $i = \lambda, \ldots, n$, by (16). Go to 2.6.
   2.3. If $m > \mu$, then compute a_{ui}, $i = \mu, \mu - 1, \ldots, \lambda + 1$, by (15) and b_{ui}, $i = \mu, \mu - 1, \ldots, 1$, by (17). Go to 2.6.
   2.4. If $m = l$ (equivalently, $u = r$), then set $a_{um} := 0$, $b_{um} := 0$. Then, for $i = l - 1, l - 2, \ldots, 1$, compute a_{ui} by (15) and b_{ui} by (17), and, for $i = l + 1, \ldots, n$, compute a_{ui} by (14) and b_{ui} by (16). Go to 2.6.
   2.5. If $k < l$, then set $i_1 := k - 1$, $i_2 := l$ and $a_{ui} := a_{u,i+1}$, $b_{ui} := b_{u,i+1}$ for $i = k, \ldots, l - 1$. Otherwise, set $i_1 := l$, $i_2 := k + 1$ and $a_{ui} := a_{u,i-1}$, $b_{ui} := b_{u,i-1}$ for $i = k, k - 1, \ldots, l + 1$. In both cases, for $i = i_1, i_1 - 1, \ldots, 1$, compute a_{ui} by (15) and b_{ui} by (17). Moreover, for $i = i_2, \ldots, n$, compute a_{ui} by (14) and b_{ui} by (16).
   2.6. Repeat from 2.1 for the next value of u, if any.
3. For $i = \lambda, \ldots, \mu - 1$, compute c_i by (18) and c_i^- by (9). Set $c_\mu^- := c_\mu - c_{\mu-1}$.

As we can see, the UpdateDataInsertion procedure is a little more complex than UpdateDataInterchange. In particular, the row of the matrix A as well as B for the facility r is initialized anew (Step 2.4). Also, the rows of these matrices for other facilities assigned to permutation positions in the range $\min(k, l)$ to $\max(k, l)$ are entirely updated in Step 2.5. However, some entries of the matrices are not recalculated but simply shifted either to the left (if $k < l$) or to the right (if $k > l$) by one position. The worst-case time complexity of the procedure UpdateDataInsertion is $O(n^2)$.

In what follows, we denote the single start variant of the described implementation of GSA by SA and the multi-start variant of this implementation by MSA. If $P = 1$, which means that only pairwise interchange moves are applied, the time complexity of Step 4 of SA is $O(\gamma\bar{v}\bar{z}n + (1-\gamma)\bar{v}\bar{z} + t_\gamma n^2 + (1 - \lfloor\gamma\rfloor)n^2) = O(\bar{v}\bar{z}(\gamma(n-1)+1) + (t_\gamma + 1 - \lfloor\gamma\rfloor)n^2)$, where t_γ is the number of times Step 4.1.3 is executed when the outer loop counter v is in the range $\lfloor\gamma\bar{v}\rfloor + 1$ to $\bar{v}$. If insertion moves are allowed, then the complexity of SA increases to $O(\bar{v}\bar{z}(\gamma(n^2-1)+1) + (t_\gamma + 1 - \lfloor\gamma\rfloor)n^2)$.

Another implementation of GSA, denoted by SA2, is based on the use of Eqs. (3) and (4). Step 4.1.1 in SA2 takes the following form:

4.1.1. Randomly select a pair of facilities r, s. If $v \leqslant \gamma\bar{v}$, then proceed to 4.1.1.1. Otherwise go to 4.1.1.2.
  4.1.1.1. Compute $h = \Delta(p, r, s)$ by (2). Go to 4.1.1.3.
  4.1.1.2. Check whether $v = \lfloor\gamma\bar{v}\rfloor + 1$ and $z = 1$. If so, then, using (3), compute $\Delta(p, i, j)$ for all pairs of facilities (i, j) (if not, then the matrix $H = (\Delta(p, i, j))$ already exists). In both cases, set $h := \Delta(p, r, s)$.
  4.1.1.3. If $h \leq 0$, then go to 4.1.3. Otherwise go to 4.1.2.

Again, as in case of SA, during the first $\lfloor\gamma\bar{v}\rfloor$ iterations, Eq. (2) to calculate the interchange move gain is used. When the outer loop index v exceeds $\lfloor\gamma\bar{v}\rfloor$, then the gain is taken from the matrix H. Notice that SA2 does not employ insertion moves. This is because it was our decision to base SA2 exclusively on known techniques and avoid using innovative methods relying on the matrices A, B and vectors C, C^-, which are well suited for the calculation of insertion gains. Put more simply, SA2 follows a traditional approach to maintaining move gains. Certainly our main implementation of GSA is SA, and SA2 is used as a convenient reference algorithm for comparison.

In Step 4.1.3, SA2 performs several operations. The most time-consuming of them is updating of the gain matrix $H = (\Delta(p, i, j))$. Formally, this step in SA2 becomes:

4.1.3. Replace p with the solution p' obtained from p by interchanging the facilities r and s. If $v > \gamma\bar{v}$, then update $\Delta(p, i, j)$ by (4) for all i, j both different from r and s, compute $\Delta(p, i, r)$ and $\Delta(p, i, s)$ by (3) for all i different from r and s, and invert the sign of both $\Delta(p, r, s)$ and $\Delta(p, s, r)$. Set $f := f + h$. If $f < F^*$, then set $p^* := p'$ and $F^* := f$.

As already remarked in Section 3.1, the time complexity of Step 4 of SA2 is $O(\bar{v}\bar{z}(\gamma(n-1)+1) + t_\gamma n^2 + (1 - \lfloor\gamma\rfloor)n^3)$. The last term in parentheses is due to initialization of the gain matrix H. Of course, this matrix is needed only when $\gamma < 1$.

We note that the algorithms described in this section bear some similarity with the simulated annealing algorithm proposed in [2]. Let us denote the latter by SA-CAP. Though Ahonen et al. [2] presented SA-CAP for solving the corridor allocation problem, it should be clear that with small changes this algorithm can be adapted to deal with the SRFLP and its special case the SREFLP. However, there are also several differences between our algorithms SA, MSA and SA2 on one side and SA-CAP on the other side. In particular, SA-CAP employs a local search (LS) procedure, which is invoked at the beginning of SA-CAP as well as at the end of each restart of the basic simulated annealing algorithm. Our implementations of the SA metaheuristic do not include a LS component. Another difference is that SA-CAP is restricted to perform relocations of a facility from one row necessarily to the end of the opposite one. Provided $P < 1$, in SA and MSA algorithms, all possible positions for relocation of a facility are allowed. Also, calculation of the initial temperature is different. For this task, SA-CAP uses the so-called reversed simulated annealing procedure (see [2]). Our algorithms follow a different strategy. To calculate the initial temperature, they generate a sample of random pairwise interchanges of facilities. We can also point out a couple of differences regarding the restart policy. First, initial solutions used to restart the cooling process in SA-CAP and MSA are different. In SA-CAP, it is a solution returned by the first call to the LS procedure and is the same for each restart. In the case of MSA, a new starting permutation is generated randomly at the beginning of each cooling cycle. Second, in the SA-CAP algorithm, the initial temperature is sequentially reduced from one restart to the next one. Unlike SA-CAP, MSA uses a fixed cooling schedule, that is, all cooling cycles start from the same initial temperature.

The SREFLP differs from the SRFLP in that the possible locations for facilities are equally spaced along a straight line. Equivalently, it can be thought that, in the case of SREFLP, all facilities have the same length. In our algorithms, this fact is exploited when calculating the interchange move gains. Using (2), this takes $O(n)$ time per pair of facilities. On the other hand, in order to compute such a gain, the algorithms for the SRFLP need to perform $O(n^2)$ operations (because $O(n)$ facilities change their location). Another place where we exploit the above-mentioned fact is manipulating the matrix of gains H in the SA2 algorithm. We are not aware of any method for fast construction of the gain matrix in the case of the SRFLP. Linear-time computation of gains in our specialized algorithms (instead of $O(n^2)$ time in SA-CAP) is the main reason why they should run faster on SREFLP instances than the SA-CAP algorithm developed for a more general problem. Of course, in SA-CAP, the time of gain calculation is further increased by the necessity to manipulate facility lengths.

4. Experimental analysis and comparisons

The primary purpose of experimentation was to show that the idea to save data for fast computation of move gains at lower temperatures during cooling was fruitful. Another purpose was to compare our novel method for maintaining move gains, which is based on the cut vector and special matrices, with a traditional approach. These methods are embedded into simulated annealing algorithms SA and SA2, respectively. Additionally, we compare better of these two implementations of the simulated annealing metaheuristic against iterated tabu search algorithm proposed in [30].

Table 1
Dependence of the running time (in seconds) of SA with $P = 1$ 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
p70-1	259839	3.8	1.9	1.1	1.0	1.1	1.2	1.3	1.3	1.5	1.5	1.6
p80-1	381870	5.2	2.4	1.4	1.2	1.3	1.4	1.5	1.7	1.8	1.9	2.1
p90-1	555767	7.1	3.2	1.7	1.5	1.6	1.7	1.9	2.0	2.2	2.4	2.5
p100-1	753130	8.6	3.7	1.9	1.7	1.8	2.0	2.2	2.4	2.6	2.8	3.0
p150	2592982	31.0	12.5	4.6	3.5	3.6	4.0	4.5	5.0	5.5	6.0	6.5
p200	6247830	68.9	25.2	8.2	5.7	5.9	6.7	7.5	8.5	9.4	10.3	11.2
p250	12213944	129.8	44.1	13.3	8.7	8.8	10.0	11.4	12.8	14.2	15.7	17.1
p300	21267772	211.5	67.8	19.4	12.1	12.1	13.8	15.8	17.9	20.1	22.1	24.2

Table 2
Dependence of the running time (in seconds) of SA2 on the switch parameter γ.

Instance	γ										
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
p70-1	27.4	11.8	4.0	1.7	1.2	1.2	1.3	1.4	1.5	1.5	1.6
p80-1	40.4	16.7	5.4	2.4	1.7	1.6	1.7	1.8	1.9	1.9	2.0
p90-1	57.8	23.3	7.5	3.1	2.0	1.9	2.0	2.1	2.2	2.4	2.5
p100-1	76.4	30.0	9.5	3.8	2.4	2.2	2.4	2.5	2.7	2.8	3.0
p150	316.5	128.8	36.9	12.3	6.0	5.1	5.3	5.6	5.9	6.1	6.4
p200	760.7	292.1	79.7	25.2	10.9	8.4	8.7	9.3	9.9	10.5	11.1
p250	1496.6	551.0	150.1	45.0	18.3	13.3	13.6	14.5	15.3	16.1	17.0
p300	2510.8	894.1	239.2	69.5	25.2	17.4	18.0	19.6	21.1	22.6	24.1

4.1. Experimental setup

The described algorithms have been coded in the C++ programming language. The same language was used in [30] to implement ITS. 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, we used two sets of SREFLP instances of our own and, in addition, several instances from the *sko* test set of Anjos and Yen [5]. Our first set consists of four subsets, each of cardinality 5. The size of the instances in the first to fourth subsets is 70, 80, 90 and 100, respectively. For each instance, the entries of the flow matrix are integer numbers chosen randomly from a uniform distribution between 0 and 10. The second test set is composed of 20 flow matrices of size $n = 110, 120, \ldots, 300$. They are generated in precisely the same way as those in the first set. The *sko* instances of Anjos and Yen [5] are used to evaluate the performance of various algorithms for solving the SRFLP. They were created from the QAP instances of Skorin-Kapov [38] by specifying the lengths of the facilities. In some instances, all the facilities have the length of 1. Thus, they can be treated as being instances of the SREFLP. We have used them in computational experiments with the MSA algorithm (see Section 4.3).

In the literature, one can find several other benchmark sets of SREFLP instances, including those introduced by Obata [29], Sarker [36,43], and Yu and Sarker [44]. However, the problem instances in these sets are of small size. It was found that they were easy for simulated annealing implementations. The majority of problems in the above-mentioned benchmark sets were solved to optimality by exact algorithms presented in [30] and [17]. The algorithm of Hungerländer [17] failed to solve only three largest instances (of size 45, 50 and 60) in the Yu–Sarker [44] data set. We run SA and SA2 on these instances. Both algorithms were able to attain the best known results with quite modest computational efforts. In particular, for 60-facility instance, SA with $P = 1$ produced the best result in all 10 runs taking an average per-run time of less than 10 s. One can conjecture that, for the three largest Yu–Sarker instances, the best known solutions reported in [30] are optimal. After preliminary investigations, we decided not to use benchmarks from [29,36,44] in our main experiments, because their size is too small for a meaningful comparison of algorithms.

The algorithms described in Section 3 have several parameters that control their performance. For many simulated annealing algorithms in the literature (see, for example, [16,19,28,34]), the number of moves to be attempted at each temperature level is set to $100n$, where n denotes the size of the problem instance. We adopted this choice in our experiments. We set $\bar{z}_0 = 100$ (and thus $\bar{z} = 100n$) in GSA and, consequently, in both SA and SA2. The final temperature of the cooling process, $T_{\min}$, should be taken close to zero but still positive [28]. We fixed $T_{\min}$ at 0.0001. As remarked in Section 2, the cooling factor α is typically chosen from the interval [0.9, 0.995]. Our preliminary experiments showed that setting α close to the middle of this interval resulted in reasonable performance of the tested algorithms. Therefore, we fixed α at 0.95. The remaining parameters of SA are P and γ. In the next subsection, we investigate computationally three configurations of SA differing in the value of the probability parameter P: with $P = 1$, which means that only pairwise interchanges of facilities are considered; with $P = 0$, in which case only insertions are allowed; and with $P = 0.5$, specifying that interchange moves and insertion moves are drawn with equal probability. In the next subsection, we also assess the performance of SA for various values of the switch parameter γ. We investigate there the influence of γ on the running time of the SA2 algorithm as well.

Table 3
Dependence of the running time (in seconds) of SA with $P = 0$ 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
p70-1	259768	5.1	3.1	2.3	2.5	3.1	3.7	4.3	4.9	5.5	6.1	6.7
p80-1	381878	7.0	4.1	3.0	3.5	4.2	5.1	6.0	6.8	7.7	8.6	9.5
p90-1	555674	9.6	5.5	4.1	4.7	5.8	7.1	8.3	9.5	10.8	12.1	13.3
p100-1	753390	11.9	6.6	5.1	6.1	7.7	9.3	11.0	12.6	14.3	16.0	17.7
p150	2592988	43.4	23.7	16.2	19.1	24.2	29.8	35.3	41.0	46.7	52.3	58.0
p200	6247432	97.5	50.8	35.5	42.7	54.9	68.1	81.0	94.4	107.3	120.6	134.0
p250	12214312	184.2	92.9	66.3	81.1	104.4	130.0	155.2	180.5	205.8	231.1	257.1
p300	21261524	303.0	148.7	110.0	136.8	177.7	220.0	262.8	306.8	350.8	393.6	437.6

Table 4
Dependence of the running time (in seconds) of SA with $P = 0.5$ 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
p70-1	259850	4.4	2.5	1.7	1.8	2.1	2.4	2.8	3.1	3.5	3.8	4.2
p80-1	381884	6.1	3.3	2.2	2.4	2.8	3.3	3.8	4.3	4.8	5.3	5.8
p90-1	555685	8.4	4.4	2.9	3.1	3.7	4.4	5.1	5.8	6.5	7.3	8.0
p100-1	753130	10.3	5.2	3.6	3.9	4.8	5.7	6.6	7.6	8.5	9.4	10.4
p150	2592725	37.2	18.1	10.4	11.3	13.9	17.0	19.9	23.0	26.1	29.2	32.3
p200	6247525	83.3	38.1	21.9	24.2	30.5	37.4	44.3	51.4	58.4	65.5	72.6
p250	12214469	157.2	68.7	39.9	45.0	56.7	70.2	83.4	96.8	110.2	123.5	137.3
p300	21256618	257.3	108.1	64.5	74.6	95.0	117.1	139.5	162.6	185.7	208.2	231.2

4.2. Computational results of conventional simulated annealing

In order to evaluate the performance of the algorithms in terms of computation times we conducted an experiment on a set of representative instances of various sizes. These instances are listed in the first column of Tables 1–4. The integer following "p" in the name of an instance indicates the number of facilities. Tables 1 and 2 report the results obtained by SA with $P = 1$ and SA2, respectively. The second column of Table 1 displays, for each instance, the objective function value of a solution found by SA configured to perform only interchange moves. These data are not shown in Table 2, because SA2 always produces the same solution as SA with $P = 1$. In fact, for the same starting permutation, this configuration of SA and SA2 generate precisely the same sequence of interchange moves. The remaining columns of Tables 1 and 2 provide the running times of SA and, respectively, SA2 for various values of the switch parameter γ.

From Table 1, we see that the best value of the switch parameter γ for interchange-based SA is 0.3. For $\gamma = 0.4$, the algorithm takes very similar running times. Notice that, for these γ values, SA is significantly faster than the variant of SA relying entirely on the use of Eq. (2) (in this case $\gamma = 1$). In particular, for the largest instance in Table 1, the speed up is equal to 2. We also observe that setting γ to 0 in SA is not a good choice. In other words, using our new gain calculation method based on Proposition 1 for high temperatures was, as expected, time consuming.

Inspection of Table 2 reveals that the best performance of SA2 is achieved when the switch parameter γ is set to 0.5. However, even the best configuration of SA2 runs slower than interchange-based SA with $\gamma = 0.3$. In Tables 1 and 2, the columns labelled "0.0" compare two methods for calculating interchange move gains. The method employed in SA uses matrices A, B and vectors C, C^-, and the method embedded in SA2 uses the gain matrix H. We see that the first of them is an order of magnitude faster than the second one. We think that one of the reasons for this is that the procedure UpdateDataInterchange(p, k, l) in SA updates only a subset of entries of each of the matrices A and B. The size of the subset in each case depends on the positions k and l of the facilities being swapped in the permutation p. Meanwhile, SA2 in Step 4.1.3 is forced to update the entire matrix H (we continue to assume that $\gamma = 0$). Another reason is that an entry of A or B can be updated by performing just a single addition operation, as it is seen from (14) to (17). On the other side, the main formula used for updating the entries of the gain matrix H (that is, Eq. (4)) involves several addition/subtraction and one multiplication operations. To compare the performance of SA and SA2, in Fig. 1 we plot the running time versus γ for the largest SREFLP instance in our test suite.

Table 3 reports the results of the insertion-based SA algorithm. The structure of this table is the same as that of Table 1. As can be seen, the best γ value for this variant of SA is 0.2. This value is smaller than in the case of SA configured (by setting $P = 1$) to perform pairwise interchanges of facilities. Comparing the results in the third and last columns of Table 3, we can conclude that the δ calculation method based on Proposition 2 is faster than the method of computing δ by Eqs. (19) and (20). However, the SA algorithm becomes more efficient when both methods are used in a combined manner. By analyzing the results in Tables 1 and 3, we find that SA with $P = 1$ and SA with $P = 0$ are on the same level in terms of solution quality. However, the first of these variants is superior to the second one when the computation time is taken into account. Note that the difference in the running times increases with the problem size. For example, for p300 instance, SA with $P = 1$ is nine times faster than SA with $P = 0$.

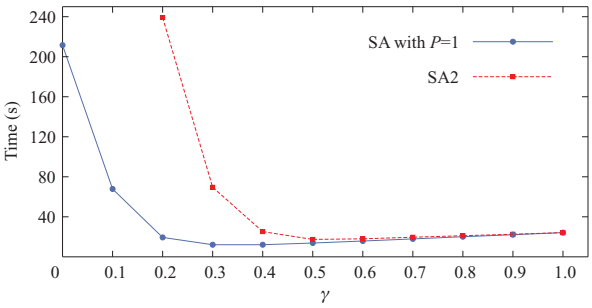

Fig. 1. Running times of SA and SA2 versus γ for p300 instance.

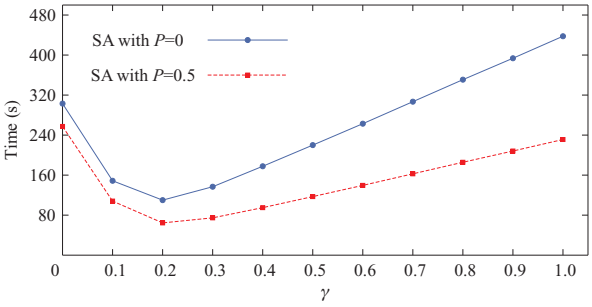

Fig. 2. Running times of two SA variants versus γ for p300 instance.

In Table 4, we present the results of SA with the probability parameter P set to 0.5. At each iteration of this variant of SA, both move types, pairwise interchanges and insertions, are equally probable. Looking at the table, we find that, like in the case of $P = 0$, the algorithm performs best when the switch parameter γ is 0.2. Comparing the data in Tables 1, 3 and 4, we can see that the quality of the solutions delivered by SA with $P = 0.5$ is similar to the quality of the solutions produced by SA when the parameter P is set to an extreme value (1 or 0). The results also show that SA with $P = 0.5$ takes less time than SA with $P = 0$ but much more time than SA with $P = 1$. The former of these comparisons is illustrated in Fig. 2. We plot the running time versus γ for the largest instance in our dataset.

From the experiment, we can make the following conclusions:

- an appropriate choice of the parameter γ enables us to reduce computation time significantly;
- interchange-based SA runs much faster than insertion-based SA;
- interchange-based SA is superior to the SA2 algorithm.

4.3. Computational results of multi-start simulated annealing

In this subsection, we report the results of an empirical evaluation of the MSA algorithm. Like in the previous subsection, we investigate three variants of the algorithm differing in the value of the probability parameter P. Our interest is in running MSA with $P = 1$, $P = 0$ and $P = 0.5$. Based on the results of our previous experiment, we fix the switch parameter γ at 0.3 if $P = 1$ and at 0.2 in the remaining two cases. We do not include a multi-start version of SA2 in our comparisons because SA2 is fully dominated by SA with $P = 1$. We compare MSA against iterated tabu search (ITS) algorithm proposed in [30]. We do not consider algorithms presented in older papers because they are outperformed by ITS [30]. Given their stochastic nature, we executed each variant of MSA as well as the ITS algorithm 10 times on each of the test problems. Maximum CPU time limits for a run were as follows: 10 s for $n \le 70$, 20 s for $70 < n \le 80$, 40 s for $80 < n \le 90$, 60 s for $n = 100$, 150 s for $n = 110, 120, \ldots, 150$, 600 s for $n = 160, 170, \ldots, 200$, 1200 s for $n = 210, 220, \ldots, 250$, and 1800 s for $n = 260, 270, \ldots, 300$.

The results of solving various SREFLP instances 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 *sko* instances. We refer the reader to Table 4 in [25], which provides sources where these values come from. The second column of Tables 6 and 7 contains, for each instance, the value of the best solution obtained from all runs of all versions of MSA. The third column in each table shows the gap of the value of the best solution out of 10 runs (in parentheses, the gap of the average value of 10 solutions) found by MSA with $P = 1$ to the value displayed in the second column. The remaining columns report these statistics for the other two configurations of MSA as well as the ITS algorithm. The results, averaged over the whole dataset, are presented in the bottom row of each table.

As can be seen from Table 5, each of the MSA variants and also ITS succeeded in obtaining the best known solutions in at least one run. Comparing the average gaps, we notice that, for the four largest *sko* instances, ITS performed considerably worse

Table 5
Performance of MSA and ITS on the SRFLP instances of Anjos and Yen [5] (we use only instances in which all facilities have the same length).

Instance	Best known value	Sol. difference (i.e., heuristic solution value − best value)			
		MSA			ITS
		$P = 1$	$P = 0$	$P = 0.5$	
sko42-1	25525	0 (0.0)	0 (0.0)	0 (0.0)	0 (0.0)
sko49-1	40967	0 (1.4)	0 (0.4)	0 (0.4)	0 (1.4)
sko56-1	64024	0 (0.6)	0 (0.0)	0 (0.0)	0 (0.6)
sko64-1	96881	0 (1.9)	0 (1.8)	0 (1.2)	0 (360.7)
sko72-1	139150	0 (0.9)	0 (0.8)	0 (0.6)	0 (238.0)
sko81-1	205106	0 (12.9)	0 (24.1)	0 (3.5)	0 (494.1)
sko100-1	378234	0 (1.5)	0 (0.0)	0 (0.0)	0 (3859.7)
Average		0 (2.7)	0 (3.9)	0 (0.8)	0 (707.8)

Table 6
Performance of MSA and ITS on smaller problem instances.

Instance	Best value	Sol. difference (i.e., heuristic solution value − best value)			
		MSA			ITS
		$P = 1$	$P = 0$	$P = 0.5$	
p70-1	259762	0 (17.3)	0 (46.0)	0 (12.5)	0 (264.3)
p70-2	255020	0 (1.8)	0 (7.3)	0 (1.2)	0 (852.4)
p70-3	258422	0 (6.0)	0 (0.0)	0 (0.0)	0 (1013.5)
p70-4	255454	0 (1.2)	0 (1.9)	0 (0.4)	0 (814.1)
p70-5	251495	0 (1.6)	0 (4.2)	0 (5.3)	0 (651.3)
p80-1	381870	0 (0.2)	0 (35.1)	0 (1.4)	0 (1537.1)
p80-2	381676	0 (0.0)	0 (0.7)	0 (0.0)	0 (698.1)
p80-3	390472	0 (0.2)	0 (0.3)	0 (0.1)	0 (585.5)
p80-4	384525	0 (0.1)	0 (0.0)	0 (25.6)	224 (782.3)
p80-5	371508	0 (0.0)	0 (7.1)	0 (0.0)	0 (855.0)
p90-1	555670	0 (2.2)	0 (2.9)	0 (2.0)	0 (1859.5)
p90-2	548080	0 (0.0)	0 (0.0)	0 (0.3)	202 (1488.8)
p90-3	559524	0 (1.4)	0 (4.2)	0 (2.5)	818 (1722.8)
p90-4	555915	0 (0.0)	0 (0.5)	0 (0.0)	590 (2062.7)
p90-5	554379	0 (9.8)	0 (11.5)	0 (8.7)	0 (827.2)
p100-1	753130	0 (0.3)	0 (0.0)	0 (0.0)	726 (2079.7)
p100-2	755019	0 (9.6)	0 (49.0)	0 (28.7)	0 (3301.5)
p100-3	751619	0 (0.6)	0 (0.0)	0 (0.2)	519 (3540.6)
p100-4	775911	0 (1.3)	0 (0.5)	0 (0.0)	0 (3153.2)
p100-5	757168	0 (4.0)	0 (21.0)	0 (19.8)	0 (3313.9)
Average		0 (2.9)	0 (9.6)	0 (5.4)	153.9 (1570.2)

than MSA. Looking at the results for MSA, we find that the average gaps for the considered values of the parameter P are quite similar. We remark that the dataset of Anjos and Yen [5] has been used in several studies to test the performance of various algorithms for the SRFLP. One of the most recent studies is that of Kothari and Ghosh [25]. The authors developed four variants of the scatter search algorithm for the SRFLP and presented numerical results for one of them, named SS-1P. This variant was able to find the best known solutions for all instances in Table 5 except sko64-1. It is, however, clear that comparison of MSA with SS-1P is not fair because MSA was designed to solve the SREFLP, which is a special case of the SRFLP. We just say that the running times of SS-1P reported in [25] (on a different computer than we used) are significantly longer than those of MSA. For example, for the largest problem instance in Table 5, sko100-1, the SS-1P algorithm took 877 s on a PC with four Intel Core i5-2,500 3.30 GHz processors (see Table 5 in [25]). Our MSA algorithm with either $P = 0$ or $P = 0.5$ obtained the best known solution for the same instance in each of the 10 runs, taking 60 s per run. The running time comparison of MSA with SA-CAP from [2] is not fair either. Though both these algorithms follow the simulated annealing paradigm, the second of them was developed for solving the CAP, which is a more general and harder problem than the SREFLP. Ahonen et al. [2] used *sko* dataset to test the performance of algorithms for the CAP. They reported results for a subset of *sko* instances of smaller size. The SA-CAP algorithm took 25.5 s, 47.7 s and 92.6 s for sko42-1, sko49-1 and sko56-1, respectively. We remind that the run-time limit for MSA on these instances was 10 s.

Table 6 shows the computational statistics of the MSA and ITS algorithms on our first dataset. We see that ITS is beaten by the simulated annealing implementations. All variants of MSA were able to achieve best results for all instances in the dataset. The ITS algorithm was not so successful — it failed to obtain solutions of the best quality for 6 instances out of 20. Comparing the average gaps, we see that the performance of all three MSA variants is similar, with interchange-based MSA having a slight

Table 7
Performance of MSA and ITS on larger problem instances.

Instance	Best value	Sol. difference (i.e., heuristic solution value – best value)			
		MSA			ITS
		$P = 1$	$P = 0$	$P = 0.5$	
p110	990726	0 (0.3)	0 (0.6)	0 (0.6)	4027 (5709.8)
p120	1302180	0 (0.0)	0 (1.9)	0 (0.8)	996 (5507.8)
p130	1684370	0 (26.9)	0 (98.7)	0 (55.9)	1347 (6666.6)
p140	2086569	0 (5.7)	0 (6.1)	0 (3.2)	2198 (9744.3)
p150	2592056	2 (102.2)	0 (151.5)	0 (158.5)	493 (12256.6)
p160	3171765	0 (0.0)	0 (0.0)	0 (0.0)	6765 (16013.7)
p170	3805796	0 (1.5)	0 (1.5)	0 (0.6)	4453 (18349.2)
p180	4515444	2 (464.7)	1 (414.5)	0 (236.0)	4231 (13541.8)
p190	5306929	0 (17.5)	0 (17.7)	0 (4.2)	12265 (25533.9)
p200	6246061	1 (24.6)	1 (17.5)	0 (7.4)	25050 (38720.0)
p210	7219037	1 (4.5)	1 (4.4)	0 (4.9)	24441 (49821.7)
p220	8318980	0 (127.7)	1 (1164.8)	0 (26.0)	17171 (34844.1)
p230	9461136	18 (68.1)	1 (33.8)	0 (11.3)	14890 (37723.3)
p240	10789206	4 (25.5)	2 (30.6)	0 (13.9)	24923 (43834.4)
p250	12211391	0 (665.9)	8 (591.1)	548 (838.6)	18238 (41138.8)
p260	13734035	4 (74.6)	4 (127.9)	0 (131.4)	45043 (66049.7)
p270	15483623	8 (61.8)	1 (31.4)	0 (16.8)	43479 (76788.5)
p280	17210898	12 (82.9)	0 (72.9)	2 (60.7)	50924 (78329.7)
p290	19188461	0 (67.0)	5 (76.5)	8 (67.2)	39098 (90825.0)
p300	21253031	49 (307.3)	236 (561.0)	0 (357.0)	60213 (99437.1)
Average		5.0 (106.4)	13.0 (170.2)	27.9 (99.7)	20012.2 (38541.8)

edge. We also observe that for 9 problem instances the best quality solutions were produced in all 10 runs of at least one variant of MSA. Notably, the largest number of zero average gaps (6) in Table 6 was obtained by MSA with $P = 0.5$. The average gaps of the solutions delivered by ITS to the best solutions are quite large. As the last row shows, they exceed the gaps of MSA with $P = 1$ by more than 500 times on the average.

In Table 7, we report the results of tested algorithms for instances in our second dataset. The main observation from the table is that MSA performs markedly better than the ITS algorithm. By contrasting the last column with the columns for MSA we can conclude that, for each instance, ITS failed to find a solution of good quality in each of 10 runs. Another observation on our results for larger problem instances is that, again, all MSA variants are almost equally good. The largest number of best solutions was obtained when setting the parameter P to 0.5. In this case, this number is equal to 17, whereas in the case of $P = 1$ (respectively, $P = 0$) only 10 (respectively, 9). We note that MSA with $P = 0.5$ performed poorly for only one instance, namely p250. This led to a strong increase in the average of gaps for this configuration of the MSA algorithm (last row in Table 7).

5. Conclusions

In this paper we have presented a simulated annealing algorithm for the single-row equidistant facility layout problem. The algorithm provides a possibility to employ either merely pairwise interchanges of facilities or merely insertion moves or both of them. It incorporates an innovative method for computing gains of both types of moves, pairwise interchanges and insertions. Experimental analysis shows that this method is significantly faster than traditional approaches. To further speed up simulated annealing, we propose a two-mode technique when for high temperatures, at each iteration, only the required gain is calculated and, for lower temperatures, the gains of all possible moves are maintained from iteration to iteration.

From the computational experiments, we conclude that executing of the insertion moves takes more time than executing of the pairwise interchange moves. However, from the results of multi-start SA, no significant superiority in terms of solution quality of any of these two alternative choices can be detected. In fact, the largest number of best solutions was obtained when both move types were used in concert.

We compared the developed algorithm against the iterated tabu search method from the literature. Experimental evaluations on three sets of SREFLP instances of size up to 300 show that the performance of our simulated annealing implementation is dramatically better than that of the ITS heuristic. The latter constantly failed to reach the best solutions of SA for all instances of size larger than 100.

There are a few of interesting avenues for further research. First, the applicability of the two-mode approach for computing move gains could be investigated when considering simulated annealing algorithms for other combinatorial optimization problems, especially those defined on the set of permutations. In this regard, our results go beyond the study of the single-row equidistant facility layout problem. Second, the proposed move gain computation method can be useful in developing other heuristic algorithms for the SREFLP. We think that the algorithms incorporating local search procedures are particularly good candidates in this respect. Finally, the possibility of developing a high-performing simulated annealing algorithm for the SRFLP would also be worth exploring.

References

[1] W.P. Adams, M. Guignard, P.M. Hahn, W.L. Hightower, A level-2 reformulation-linearization technique bound for the quadratic assignment problem, Eur. J. Oper. Res. 180 (3) (2007) 983–996.
[2] H. Ahonen, A.G. de Alvarenga, A.R.S. Amaral, Simulated annealing and tabu search approaches for the corridor allocation problem, Eur. J. Oper. Res. 232 (1) (2014) 221–233.
[3] A.R.S. Amaral, The corridor allocation problem, Comput. Oper. Res. 39 (12) (2012) 3325–3330.
[4] N.A.G. Aneke, A.S. Carrie, A design technique for the layout of multi-product flowlines, Int. J. Prod. Res. 24 (3) (1986) 471–481.
[5] M.F. Anjos, G. Yen, Provably near-optimal solutions for very large single-row facility layout problems, Optim. Methods Softw. 24 (4–5) (2009) 805–817.
[6] U. Benlic, J.-K. Hao, Breakout local search for the quadratic assignment problem, Appl. Math. Comput. 219 (9) (2013) 4800–4815.
[7] U. Benlic, J.-K. Hao, Memetic search for the quadratic assignment problem, Expert Syst. Appl. 42 (1) (2015) 584–595.
[8] J. Bhasker, S. Sahni, Optimal linear arrangement of circuit components, J. VLSI Comput. Syst. 2 (1–2) (1987) 87–109.
[9] R. Burkard, M. Dell'Amico, S. Martello, Assignment Problems, SIAM, 2009.
[10] R.E. Burkard, F. Rendl, A thermodynamically motivated simulation procedure for combinatorial optimization problems, Eur. J. Oper. Res. 17 (2) (1984) 169–174.
[11] C.K. Cheng, Linear placement algorithms and applications to VLSI design, Networks 17 (4) (1987) 439–464.
[12] W.-M. Chow, An analysis of automated storage and retrieval systems in manufacturing assembly lines, IIE Trans. 18 (2) (1986) 204–214.
[13] J. Díaz, J. Petit, M. Serna, A survey of graph layout problems, ACM Comput. Surv. 34 (3) (2002) 313–356.
[14] Z. Drezner, Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem, Comput. Oper. Res. 35 (3) (2008) 717–736.
[15] P.M. Hahn, Y.-R. Zhu, M. Guignard, W.L. Hightower, M.J. Saltzman, A level-3 reformulation-linearization technique-based bound for the quadratic assignment problem, INFORMS J. Comput. 24 (2) (2012) 202–209.
[16] S.S. Heragu, A.S. Alfa, Experimental analysis of simulated annealing based algorithms for the layout problem, Eur. J. Oper. Res. 57 (2) (1992) 190–202.
[17] P. Hungerländer, Single-row equidistant facility layout as a special case of single-row facility layout, Int. J. Prod. Res. 52 (5) (2014) 1257–1268.
[18] P. Hungerländer, F. Rendl, A computational study and survey of methods for the single-row facility layout problem, Comput. Optim. Appl. 55 (1) (2013) 1–20.
[19] M.S. Hussin, T. Stützle, Tabu search vs. simulated annealing as a function of the size of quadratic assignment problem instances, Comput. Oper. Res. 43 (2014) 286–291.
[20] T. James, C. Rego, F. Glover, A cooperative parallel tabu search algorithm for the quadratic assignment problem, Eur. J. Oper. Res. 195 (3) (2009) 810–826.
[21] T. James, C. Rego, F. Glover, Multistart tabu search and diversification strategies for the quadratic assignment problem, IEEE Trans. Syst. Man Cybern. A: Syst. Humans 39 (3) (2009) 579–596.
[22] R.M. Karp, M. Held, Finite-state processes and dynamic programming, SIAM J. Appl. Math. 15 (3) (1967) 693–718.
[23] T.C. Koopmans, M. Beckmann, Assignment problems and the location of economic activities, Econometrica 25 (1) (1957) 53–76.
[24] R. Kothari, D. Ghosh, The single row facility layout problem: state of the art, OPSEARCH 49 (4) (2012) 442–462.
[25] R. Kothari, D. Ghosh, A scatter search algorithm for the single row facility layout problem, J. Heuristics 20 (2) (2014) 125–142.
[26] P. Kouvelis, W.-C. Chiang, G. Yu, Optimal algorithms for row layout problems in automated manufacturing systems, IIE Trans. 27 (1) (1995) 99–104.
[27] E.M. Loiola, N.M.M. de Abreu, P.O. Boaventura-Netto, P. Hahn, T. Querido, A survey for the quadratic assignment problem, Eur. J. Oper. Res. 176 (2) (2007) 657–690.
[28] T.D. Mavridou, P.M. Pardalos, Simulated annealing and genetic algorithms for the facility layout problem: a survey, Comput. Optim. Appl. 7 (1) (1997) 111–126.
[29] T. Obata, Quadratic assignment problem: evaluation of exact and heuristic algorithms (Dissertation), Rensselaer Polytechnic Institute, Troy, NY, 1979.
[30] G. Palubeckis, A branch-and-bound algorithm for the single-row equidistant facility layout problem, OR Spectr. 34 (1) (2012) 1–21.
[31] G. Paul, Comparative performance of tabu search and simulated annealing heuristics for the quadratic assignment problem, Oper. Res. Lett. 38 (6) (2010) 577–581.
[32] J.-C. Picard, M. Queyranne, On the one-dimensional space allocation problem, Oper. Res. 29 (2) (1981) 371–391.
[33] E. Rodriguez-Tello, J.-K. Hao, J. Torres-Jimenez, An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem, Comput. Oper. Res. 35 (10) (2008) 3331–3346.
[34] R.A. Rutenbar, Simulated annealing algorithms: an overview, IEEE Circuits Devices Mag. 5 (1) (1989) 19–26.
[35] I. Safro, D. Ron, A. Brandt, Graph minimum linear arrangement by multilevel weighted edge contractions, J. Algorithms 60 (1) (2006) 24–41.
[36] B.R. Sarker, The amoebic matrix and one-dimensional machine location problems (Dissertation), Department of Industrial Engineering, Texas A&M University, College Station, TX, 1989.
[37] B.R. Sarker, W.E. Wilhelm, G.L. Hogg, One-dimensional machine location problems in a multi-product flowline with equidistant locations, Eur. J. Oper. Res. 105 (3) (1998) 401–426.
[38] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, ORSA J. Comput. 2 (1) (1990) 33–45.
[39] J. Sun, Q. Zhang, X. Yao, Meta-heuristic combining prior online and offline information for the quadratic assignment problem, IEEE Trans. Cybern. 44 (3) (2014) 429–444.
[40] J.K. Suryanarayanan, B.L. Golden, Q. Wang, A new heuristic for the linear placement problem, Comput. Oper. Res. 18 (3) (1991) 255–262.
[41] E. Taillard, Robust taboo search for the quadratic assignment problem, Parallel Comput. 17 (4–5) (1991) 443–455.
[42] U. Tosun, T. Dokeroglu, A. Cosar, A robust island parallel genetic algorithm for the quadratic assignment problem, Int. J. Prod. Res. 51 (14) (2013) 4117–4133.
[43] S. Wang, B.R. Sarker, Locating cells with bottleneck machines in cellular manufacturing systems, Int. J. Prod. Res. 40 (2) (2002) 403–424.
[44] J. Yu, B.R. Sarker, Directional decomposition heuristic for a linear machine-cell location problem, Eur. J. Oper. Res. 149 (1) (2003) 142–184.