OR Spectrum (2012) 34:1–21
DOI 10.1007/s00291-010-0204-5

REGULAR ARTICLE

A branch-and-bound algorithm for the single-row equidistant facility layout problem

Gintaras Palubeckis

Published online: 7 March 2010

Abstract In this paper, we deal with the single-row equidistant facility layout problem (SREFLP), which asks 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 branch-and-bound algorithm for solving this problem. The lower bound is computed first by performing transformation of the flow matrix and then applying the well-known Gilmore–Lawler bounding technique. The algorithm also incorporates a dominance test which allows to drastically reduce redundancy in the search process. The test is based on the use of a tabu search procedure designed to solve the SREFLP. We provide computational results for problem instances of size up to 35 facilities. For a number of instances, the optimal value of the objective function appeared to be smaller than the best value reported in the literature.

Keywords Facility layout · Quadratic assignment problem · Branch-and-bound algorithm · Tabu search

1 Introduction

Facility layout problems are a family of design problems involving the physical arrangement of a given set of facilities within a given configuration. One of the members of this family is the *single-row equidistant facility layout problem* (SREFLP for short), which can be formulated as follows. Given n facilities and an $n \times n$ matrix $C = (c_{ij})$ where c_{ij} is the (nonnegative) flow between facilities i and j, the aim is to

G. Palubeckis (✉)
Department of Practical Informatics, Kaunas University of Technology,
Studentu 50-408, 51368 Kaunas, Lithuania
e-mail: gintaras@soften.ktu.lt

find a one-to-one assignment of facilities to n locations equally spaced along a straight line so as to minimize the total assignment cost, which is the sum of the products of the flows and distances between facilities:

$$\min_{p\in\Pi} F(p) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} d_{p(i)p(j)}, \tag{1}$$

where Π is the set of all permutations of $\{1, \ldots, n\}$, and $d_{p(i)p(j)}$ is the distance between locations $p(i)$ and $p(j)$. In the SREFLP, it can be assumed that the locations are points on the x-axis with coordinates $1, 2, \ldots, n$. Then

$$d_{p(i)p(j)} = |p(i) - p(j)|. \tag{2}$$

The problem defined by (1) is the quadratic assignment problem (QAP) formulated by Koopmans and Beckmann (1957). Thus, the SREFLP defined by (1) and (2) is a special case of the QAP.

The model (1) and (2) arises in a variety of settings. Yu and Sarker (2003) applied this model to design a flowline in a manufacturing system. The problem consists of assigning machine-cells to equally spaced locations along a linear material handling track. The goal is to find an assignment that minimizes the total inter-cell flow costs. Picard and Queyranne (1981) considered the problem of locating rooms along a corridor within, for example, a hospital department. Given the number of trips per unit of time between each pair of rooms, the objective is to minimize the total traveled distance. When all room lengths are equal, this problem reduces to the SREFLP. Bhasker and Sahni (1987) used (1) and (2) to model the problem of arranging circuit components on a straight line so as to minimize the total wire length needed to realize the inter component nets. A set of nets typically is represented by a hypergraph. However, frequently the hypergraph is replaced by an edge-weighted graph and an instance of (1) and (2) is obtained.

Since the SREFLP is a special case of the QAP, any algorithm for the latter can also be used for the former. However, studying special cases of a more general problem may lead to the discovery of more efficient algorithms tailored to solve these special cases. In the area of quadratic assignment, a good example is the algorithm of Christofides and Benavent (1989) for the Tree QAP. This problem is a special case of the QAP where the nonzero flows between facilities form a tree. Another example is the grey pattern problem formulated in the context of the QAP by Taillard (1995). A branch-and-bound algorithm for this problem was developed by Drezner (2006). A classical approach for solving the SREFLP is to use the dynamic programming technique. Such an algorithm was proposed by Karp and Held (1967) and extended to the case of facilities with varying lengths by Picard and Queyranne (1981). The time complexity of the dynamic programming algorithm for the single-row facility layout problems is $O(n2^n)$ (Karp and Held 1967; Picard and Queyranne 1981). However, this algorithm requires very large memory for storing the partial solutions and, therefore, is impractical for larger n. There are several papers which present computational experiments with the dynamic programming algorithm for the layout problems that are

similar to the SREFLP. In particular, Kouvelis et al. (1995) used this algorithm to solve instances of the row layout problem of size up to 20. Öncan and Altınel (2008) applied the dynamic programming method for the balanced unidirectional cyclic layout problem. They were able to solve problem instances of size up to 20 workstations on the platform with 1 GB RAM. The computational results suggest that their implementation of dynamic programming runs out of memory long before running out of time. Therefore, it is important to focus on developing alternative approaches to the exact solution of the single-row facility layout problems, for example, branch-and-bound and branch-and-cut algorithms.

When the number of facilities in a SREFLP instance is large, the only reasonable option is to apply heuristic algorithms. Sarker et al. (1998) described an algorithm, called the depth-first insertion heuristic, which is suitable for solving large instances of the SREFLP. This algorithm starts with a feasible solution and iteratively tries to improve it. At each iteration, the algorithm evaluates all assignments obtained by placing the selected facility in each of the other $n - 1$ locations. To make the target location available for this facility, some of the other facilities are moved to the neighbouring locations. Later, Yu and Sarker (2003) proposed another algorithm, called DDH (Directional Decomposition Heuristic), which is based on a similar reasoning as the algorithm of Sarker et al. (1998) but is computationally more efficient. As already alluded to, the SREFLP can also be solved using algorithms developed for the QAP. To find near-optimal solutions to the QAP, many heuristic algorithms were proposed. For an overview of such algorithms, we refer the reader to a recent survey by Loiola et al. (2007).

Besides the QAP, there are other combinatorial optimization problems somehow related to the SREFLP. One of them is the single-row facility layout problem where the facilities (for example, machines) are geometrically represented by rectangles of fixed height and varying widths. A feasible solution is an arrangement of the facilities next to each other along a horizontal line. The objective function is to minimize the sum of the products of the flows and distances between facilities. Unlike the SREFLP, this problem is not a special case of the QAP. An exact solution method for the single-row facility layout problem was recently presented by Amaral (2006). Yet another related problem, called the minimum linear arrangement (MinLA) problem, is to find a one-to-one mapping of the vertices of a given undirected n-vertex graph onto n equally spaced points on a line. The objective is to minimize the sum of edge lengths. Typically, this problem arises in application domains, for example, in graph drawing, where the number of vertices is very large. Specific exact algorithms were developed to solve the MinLA problem on restricted classes of graphs. For example, such an algorithm for trees was proposed by Chung (1988). A summary of the most significant theoretical and algorithmic developments in the area of MinLA can be found in a survey by Díaz et al. (2002).

In this paper, we present a branch-and-bound algorithm for solving the SREFLP. One of our motivations was to solve optimally the SREFLP instances from the literature (Obata 1979; Wang and Sarker 2002; Yu and Sarker 2003), which, so far, were treated only heuristically. We were able to find optimal solutions for instances of size up to 35 facilities. The algorithm employs a dominance test which allows to discard a large portion of the partial solutions. This test provides an assessment of

a partial solution by performing a short run of a tabu search procedure developed for the SREFLP. The lower bounds on the objective function are computed using an algorithm based on a method described by Palubeckis (1988). This algorithm first performs transformation of the flow matrix and then applies the Gilmore–Lawler bound (Gilmore 1962; Lawler 1963) to the modified problem instance. The Gilmore–Lawler bound computation procedure is known to be very fast. Therefore, despite the fact that a number of stronger lower bounds exist, the Gilmore–Lawler bound still retains its practical importance. For example, using this bound, the well-known Steinberg wiring problem was solved (see Anstreicher 2003 for details). In order to have a good solution at the root node of the search tree, our approach incorporates an iterated tabu search (ITS) technique for the SREFLP. We have chosen this technique because it has shown good performance for other optimization problems with the quadratic objective function—the unconstrained binary quadratic optimization problem (Palubeckis 2006) and the maximum diversity problem (Palubeckis 2007). The branch-and-bound algorithm applies the ITS procedure once before starting the enumeration process. For all problems solved by the branch-and-bound algorithm, the solution found by this procedure was proved to be optimal.

The remainder of this paper is structured as follows. In Sect. 2, we develop an algorithm for lower bound computation. In Sect. 3, we present a branch-and-bound algorithm for solving the SREFLP. A brief description of our implementation of the tabu search method for this problem is given in Sect. 4. Our computational experience with the proposed algorithms is reported in Sect. 5. Finally, Sect. 6 concludes the paper.

2 A lower bound

The purpose of this section is to present a lower bound computation algorithm which is used within a branch-and-bound framework for solving the SREFLP. Before describing the algorithm, we introduce some assumptions, definitions and notations. We can assume w.l.o.g. that C is a matrix with all entries below the main diagonal equal to zero (upper triangular). Indeed, if this is not the case, we can replace c_{ij} by $c_{ij} + c_{ji}$ for each positive c_{ji}, $i < j$, and set c_{ji} to zero. To simplify notations, for an upper triangular flow matrix C, we assume that the order of the subscripts of c_{ij}, $i \neq j$, is not significant, i.e., c_{ij} and c_{ji} represent the same entry—that which is located above the main diagonal of C. We also assume that $c_{ii} = 0$, $i = 1, \ldots, n$. Given a flow matrix C, we can associate with it an edge-weighted complete graph $G(C)$ with vertex set $V = \{1, \ldots, n\}$ and edge weights c_{ij}, $i, j \in V$, $i < j$. In the other direction, if we have an edge-weighted complete graph G with vertex set $U \subseteq V$ and edge weights w_{ij}, $i, j \in U$, $i < j$, then we can construct an $n \times n$ upper triangular matrix $W(G) = (w_{ij})$, where w_{ij} is equal to the weight of the edge (i, j) of G if $i, j \in U$, $i < j$, and is equal to zero, otherwise. An edge-weighted graph with all nonzero weights equal to either 1 or -1 is called *signed*. For a flow matrix C, we denote by $F^*(C)$ the optimal value of the SREFLP instance (1), (2) defined by C.

The bound computation algorithm consists of two phases. In the first phase, by performing some transformations, the original flow matrix C is replaced by a flow

matrix C' for which $F^*(C') \leqslant F^*(C)$. In the second phase, the Gilmore–Lawler bound is applied to the pair of matrices C' and $D = (d_{ij})$. The idea to use transformations of the flow matrix is not new. It was suggested by Palubeckis (1988) in the context of the QAP. Other successful applications of the flow matrix transformation method for obtaining lower bounds for the QAP were reported by Chakrapani and Skorin-Kapov (1994) and by Karisch and Rendl (1995). In this section, we describe a specific algorithm for performing transformation of the flow matrix in the case of the single-row equidistant facility layout problem. The rationale behind the algorithm is to iteratively apply a matrix decomposition operation as defined in the formulation of the following obvious fact.

Proposition 1 *If* $C = C_1 + C_2$, *then*

$$F^*(C) \geqslant F^*(C_1) + F^*(C_2). \tag{3}$$

A simple way to perform such a decomposition is to take one of the matrices, say C_2, having only a few nonzero entries. In such a case, the problem instance defined by the matrix C_2 is of very small size. Indeed, we can eliminate each of the facilities for which all entries both in the corresponding row and column are zero. Since only a few facilities are left, the optimal value $F^*(C_2)$ of the objective function of the resulting problem instance can be found quite easily. If $F^*(C_2)$ is known, then, in accordance with (3), the computation of the lower bound on $F^*(C)$ is reduced to the computation of the lower bound on $F^*(C_1)$, where $C_1 = C - C_2$. With a matrix C_2 as discussed above, we can associate an edge-weighted graph $G' = (V', E')$ with vertices corresponding to facilities that were not eliminated. The weights of the edges of G' are taken from the matrix C_2. It is clear that the reasoning can be reversed. We may assume that we are given a small edge-weighted complete graph $G' = (V', E')$ with vertex set $V' \subset V = \{1, \ldots, n\}$. Then we can apply Proposition 1 with respect to the matrix $C_2 = \alpha W(G')$, where α is a positive constant. The reason for introducing the factor α is that we use signed graphs in the transformation process, which implies that all the entries of the matrix $W(G')$ are 1, 0, or -1. By multiplying them by α we get a matrix C_2 that can be subtracted from C. A necessary requirement for α selection is that all the entries of the resulting matrix $C_1 = C - C_2$ should be nonnegative. A formula for the α calculation we have used is given below (see (6)). Now suppose that G' has a simple structure and $F^*(C_2) = \alpha F^*(W(G'))$ is known. Then, as already remarked, we arrive at the problem of finding a lower bound on $F^*(C_1)$. Clearly, the right-hand side of (3) is equal to $\min_{p\in\Pi}(\sum_{i=1}^n \sum_{j=1}^n c_{ij} d_{p(i)p(j)} - \alpha \sum_{(i,j)\in E'} w_{ij} d_{p(i)p(j)}) + F^*(C_2)$, where w_{ij}, $(i, j) \in E'$, denote the weights of the edges of G'. We can rewrite the above expression as $\min_{p\in\Pi}(\sum_{i=1}^n \sum_{j=1}^n c_{ij} d_{p(i)p(j)} + \alpha(F^*(W(G')) - \sum_{(i,j)\in E'} w_{ij} d_{p(i)p(j)}))$. Thus we see that the lower bound on $F^*(C)$ is obtained by adding $\Lambda(G') = F^*(W(G')) - \sum_{(i,j)\in E'} w_{ij} d_{p(i)p(j)}$ multiplied by α to the right-hand side of (1). Since this difference is nonpositive for each $p \in \Pi$, we get the following inequality

$$\sum_{(i,j)\in E'} w_{ij} d_{p(i)p(j)} \geqslant F^*(W(G')), \tag{4}$$

which is valid for all $p \in \Pi$. The lower bound computation algorithm we present in this section is centered around the use of inequalities of type (4). The algorithm iteratively modifies the objective function of the problem by adding $\alpha\Lambda(G')$ to it.

One of the simplest inequalities of type (4) is induced by a signed triangle, that is, a graph T with vertices i, j, k, positive edges (i, j), (j, k) and negative edge (i, k). Obviously, $p = (p(i) = 1, p(j) = 2, p(k) = 3)$ is an optimal permutation for T, and $F^*(W(T)) = 0$. Hence, for $G' = T$, (4) becomes the well-known triangle inequality on the distance metric

$$d_{p(i)p(j)} + d_{p(j)p(k)} \geqslant d_{p(i)p(k)}. \tag{5}$$

This inequality can be generalized by taking signed graphs $H_{st} = (V_s \cup V_t, E_{st})$ with the set of positive edges $E_{st}^+ = \{(i, j) \mid i \in V_s, j \in V_t\}$ and the set of negative edges $E_{st}^- = \{(i, j) \mid i, j \in V_s, i < j\} \cup \{(i, j) \mid i, j \in V_t, i < j\}$. In Palubeckis (1997), such graphs of small order were used in a lower bound computation algorithm for a special case of the QAP where one of the matrices is composed of rectilinear distances between points in Euclidean space. Evidently, the above defined triangle T is precisely $H_{2,1}$ with $V_2 = \{i, k\}$ and $V_1 = \{j\}$. The algorithm to be described in this section also manipulates two other such graphs with small value of $s + t$, namely, $H_{2,2}$ and $H_{3,2}$. The right-hand side in (4) for these graphs is given by the following statement.

Proposition 2 *For* $H_{2,2}$ *(respectively,* $H_{3,2}$*),* $F^*(W(H_{2,2})) = 2$ *(respectively,* $F^*(W(H_{3,2})) = 0$*).*

Proof An optimal layout of $H_{2,2} = (\{i, j\} \cup \{k, l\}, E_{2,2})$ is defined by the permutation $p = (p(i) = 1, p(j) = 3, p(k) = 2, p(l) = 4)$. The corresponding optimal value equals 2. Analogously, for $H_{3,2} = (\{i, j, q\} \cup \{k, l\}, E_{3,2})$, an optimal permutation (of value 0) is $p = (p(i) = 1, p(j) = 3, p(q) = 5, p(k) = 2, p(l) = 4)$.

In a specific implementation of the algorithm, we used only the above-listed three graphs—signed triangles, $H_{2,2}$ and $H_{3,2}$. For this reason, we do not consider in this paper other graphs for which the inequalities of type (4) could be exploited. For example, complete graphs can be mentioned as possible candidates to derive such inequalities. However, our computational experiments did not prove the usefulness of complete graphs in obtaining lower bounds for the SREFLP. Therefore, we decided to abandon the intention of processing such graphs, even the smallest ones like triangles.

The developed lower bounding algorithm applies the decomposition principle iteratively by progressively selecting subgraphs of $G(C)$, each of which is matched to a signed graph in the set $H = \{H_{3,2}, H_{2,2}, H_{2,1}\}$. Some edges of $G(C)$ are assigned one of two colors: red or green. Where ρ is a positive constant, this coloring is defined as follows. An edge (i, j) is red if $c_{ij} < \rho$. Similarly, an edge (k, l) is green if $c_{kl} > \rho$. The edges of weight ρ remain uncolored. At each iteration, the algorithm tries to find a fully colored complete subgraph of $G(C)$ which is isomorphic to a graph $H_{st} \in H$ in the following sense: under an appropriate bijection θ between vertex sets, its green edges are mapped to positive edges of H_{st} and its red edges are mapped to negative edges of H_{st}. The selected subgraph is then used to decompose the flow matrix C

into C_1 and C_2 as stated in Proposition 1 and the discussion below it. The factor α is calculated as follows:

$$\alpha = \min\left(\min_{(i,j)\in E_{st}^{+}} c_{\varphi(i)\varphi(j)} - \rho, \rho - \min_{(i,j)\in E_{st}^{-}} c_{\varphi(i)\varphi(j)} \right), \tag{6}$$

where φ is the inverse of the bijection θ. Such a choice of α guarantees that each edge of the selected subgraph either retains in $G(C_1)$ the same color as it was in $G(C)$ or becomes uncolored. For an edge (i, j), the latter case occurs only when $\alpha = |c_{ij} - \rho|$.

Next, we describe an efficient algorithm for computing lower bounds on $F(p)$. We consider a general situation where $m \geqslant 0$ facilities are already assigned to the first m locations (for each such facility, the corresponding component of p is positive). Let $V_{\text{assign}}(p) = \{i \in V | p(i) > 0\}$, $V_{\text{rem}}(p) = V \backslash V_{\text{assign}}(p)$. We may assume that $m = |V_{\text{assign}}(p)| < n - 1$ since if $m = n - 1$, the location for the last facility is uniquely determined and no bound is needed. The input to the algorithm includes the following three lists: the lists $L_{3,2}$ and $L_{2,2}$ of subgraphs of $G(C)$ isomorphic to $H_{3,2}$ and, respectively, $H_{2,2}$ in the above-defined sense and the list R consisting of those red edges (i, k) of $G(C)$ for which there exists at least one vertex $j \in V$ such that the edges (i, j) and (j, k) are colored green. The algorithm also requires the value of ρ to be supplied. We will postpone the question of the choice of ρ until later in this section. When searching for suitable triangles, we construct for each edge $(i, k) \in R$, $\{i, k\} \cap V_{\text{rem}}(p) \neq \emptyset$, a set Z_{ik} of vertices j such that (i, j) and (j, k) are colored green and one of the following conditions is satisfied: (1) $p(j) = 0$; (2) $p(j) \neq 0$ and either $0 < p(i) < p(j)$ or $0 < p(k) < p(j)$. The algorithm, named LB (Lower Bound), can be described as follows.

1. Set $S := 0$, $Q := 0$. Introduce a matrix $C' = (c'_{ij})$ with entries $c'_{ij} := c_{ij}$, $i, j = 1, \ldots, n$.
2. For each subgraph $G_{3,2} = (V_3 \cup V_2, E_{3,2})$ in the list $L_{3,2}$, perform the following operations.
   2.1. Check whether $V_3 \cup V_2 \subseteq V_{\text{rem}}(p)$ and each edge in $E_{3,2}$, considered as an edge of $G(C')$, has a color assigned. If so, then proceed to 2.2. Otherwise, examine the next subgraph in the list $L_{3,2}$ or, if the end of the list is reached, go to 3.
   2.2. Calculate α according to (6).
   2.3. For each $(i, j) \in E_{3,2}$, set $c'_{ij} := c'_{ij} - \alpha$ if the edge (i, j) is colored green, and $c'_{ij} := c'_{ij} + \alpha$ if (i, j) is colored red. Remove color from (i, j) if $c'_{ij} = \rho$.
3. For each subgraph in the list $L_{2,2}$, perform the same operations as in 2. Additionally, if the subgraph satisfies an analogue of the condition stated in 2.1, then set $Q := Q + 2\alpha$.
4. Build the set Z_{ik} (considering colors with respect to C') for each red edge $(i, k) \in R$ for which $\{i, k\} \cap V_{\text{rem}}(p)$ is nonempty. Form a subset R' of R by taking only edges with positive $|Z_{ik}|$. If R' is empty, then go to 8.
5. Select an edge $(i, k) \in R'$ for which $|Z_{ik}|$ is smallest. Then select a vertex $j \in Z_{ik}$ such that $\min(c'_{ij}, c'_{jk}) \geqslant \min(c'_{il}, c'_{lk})$ for each $l \in Z_{ik}$.

6. Calculate α according to (6) for the triangle (i, j, k).
7. Set $c'_{ij} := c'_{ij} - \alpha$, $c'_{jk} := c'_{jk} - \alpha$, $c'_{ik} := c'_{ik} + \alpha$. If $j \in V_{\text{assign}}(p)$, then add $\alpha(p(j) - u)$ to S, where $u = p(i)$ if $i \in V_{\text{assign}}(p)$, and $u = p(k)$ if $k \in V_{\text{assign}}(p)$ (according to the definition of Z_{ik} and the fact that $\{i, k\} \cap V_{\text{rem}}(p) \neq \emptyset$, exactly one of i and k belongs to $V_{\text{assign}}(p)$). Update the coloring of the triangle (i, j, k). Also, update the set Z_{qr} for each $(q, r) \in R'$. Shrink the set R' by removing all edges (q, r) for which Z_{qr} is empty. If $|R'| > 0$, then return to 5. Otherwise proceed to 8.
8. Compute $F_+(p) = \sum_{i \in V_{\text{assign}}(p)} (m + 1 - p(i)) \sum_{j \in V_{\text{rem}}(p)} c'_{ij}$.
9. Compute the Gilmore–Lawler bound $B_{\text{GL}}(p)$ for C' and $n - m$ vertices in $V_{\text{rem}}(p)$.
10. Return with the lower bound $B(p) = F_{\text{partial}}(p) + F_+(p) + B_{\text{GL}}(p) + Q - S$, where $F_{\text{partial}}(p)$ is the value of (1) calculated for the partial layout p (by putting zero into (1) for each $d_{p(i)p(j)}$ such that either $p(i) = 0$ or $p(j) = 0$).

Clearly, for the whole problem (when $p(i) = 0$ for each $i \in V$), $B(p) = B_{\text{GL}}(p) + Q$. The term Q in the formula for $B(p)$ is due to the fact that $F^*(W(H_{2,2})) = 2$ (as stated in Proposition 2). The term S is included to correct the value of $F_{\text{partial}}(p)$ which is calculated with respect to the matrix C, and not with respect to C', as it should be the case. The sum $F_+(p)$ is a lower bound on the increase of the objective function value due to flows between the already assigned facilities and the rest of the facilities. This bound is calculated by placing each unassigned facility at the leftmost free location (its coordinate is $m+1$). The increase of $c'_{ij} d_{p(i)p(j)}$, $i \in V_{\text{assign}}(p)$, $j \in V_{\text{rem}}(p)$, for locations with coordinates greater than $m + 1$ is taken into account when constructing an $(n-m) \times (n-m)$ assignment matrix. Basically, the term $2(k - m - 1) \sum_{i \in V_{\text{assign}}(p)} c'_{ij}$ is added to the entry in this matrix in the intersection of the row corresponding to facility j and the column corresponding to location k, $k = m + 2, \ldots, n$. The factor of 2 appears in the above expression because $B_{\text{GL}}(p)$ is obtained by taking half the optimal value of the linear assignment problem (if $i, j \in V_{\text{rem}}(p)$, then c'_{ij} is used to form the matrix rows for facilities i and j; if $i \in V_{\text{assign}}(p)$, $j \in V_{\text{rem}}(p)$, then c'_{ij} is used just once—to form the row for facility j only). More details on the Gilmore–Lawler bound can be found in a number of publications. For example, application of this bound to the QAP is nicely described by Pardalos and Crouse (1989).

In the presented algorithm, we require each vertex of each selected subgraph G_{st}, $s \in \{3, 2\}$, $t = 2$, to belong to the subgraph of $G(C')$ corresponding to the set of unassigned facilities. This requirement is relaxed for triangles—one or even two vertices of a triangle isomorphic to $H_{2,1}$ may belong to $V_{\text{assign}}(p)$. The use of triangles is restricted by the definition of Z_{ik}, $(i, k) \in R'$. Such a strategy was revealed as being quite good by performing some preliminary experiments. We also notice that the convergence of the algorithm is guaranteed by the choice of α. Indeed, in Step 7, color is removed from at least one edge of the selected triangle, making it infeasible to be chosen in Step 5 once again.

The bound can be improved slightly by using a solution of high quality, possibly optimal, to guide the selection of the third vertex of the triangle, that is, $j \in Z_{ik}$ in Step 5 of LB. Actually, given such a solution p^*, Step 4 and the loop consisting of Steps 5 to 7 can be executed twice. In the first stage, the set Z_{ik} is replaced by a subset $Z'_{ik} \subseteq Z_{ik}$ containing only those vertices $j \in Z_{ik}$ for which $\min(p^*(i), p^*(k)) <$

$p^*(j) < \max(p^*(i), p^*(k))$. With such a strategy, we have the equality in (5) for p^*, and therefore there are good chances for p^* to remain close-to-optimal solution also for the transformed problem instance defined by the matrix C_1 (see (3) where, in the case of triangles, $F^*(C_2) = 0$). In the second stage, Steps 4 to 7, as described in LB, are executed. Typically, most of the triangles are selected in the first stage and a much smaller number in the second one. We will refer to this modification of LB as LB*. It may be noted, however, that the usefulness of LB* as a bounding technique in branch-and-bound algorithms is somewhat questionable. To apply LB*, we first have to find a good assignment of all unassigned facilities to locations $m+1, \ldots, n$, where $m = |V_{\text{assign}}(p)|$ as before. This means that each invocation of LB* is preceded by a call to some heuristic algorithm for the SREFLP. Our experiments showed that the cumulative time of all such calls is not compensated for by a decrease in branching and bounding time. Certainly, using LB*, a smaller search tree is grown than in the case of using less tight bounds delivered by LB.

In a particular case where no facility is placed, the algorithm LB as well as LB* can be simplified slightly. Specifically, the variable S can be eliminated since its value stays at zero. Also, the query in Step 2.1 and the definition of Z_{ik} become simpler. Furthermore, Step 8 can be dropped. As a result, the lower bound is just the sum of $B_{\text{GL}}(p)$ and Q.

Now we return to the question of how to find a suitable value of ρ. Our approach is based on performing several trial runs of a variant of LB* with different ρ. The procedure accepts two parameters: initial value $\bar{c}$ for ρ and step δ by which ρ is lowered at each iteration. It consists of the following three steps.

1. Set $B^* := -1$, $r := \bar{c}$.
2. Apply a lower bound computation algorithm to the initial problem (1). Let B_r be the value returned by this algorithm.
3. Check whether $B_r > B^*$. If so, then set $B^* := B_r$, $\rho := r$, $r := r - \delta$, and go to 2. If not, then return with the current value of ρ.

The described procedure is used only once, namely, at the initialization phase of the branch-and-bound algorithm. We run it with $\bar{c} = \max(1, \lfloor c_{\text{aver}} \rfloor)$, $\delta = \max(1, \lfloor c_{\text{aver}}/\mu \rfloor)$, where $c_{\text{aver}} = (\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij})/(n(n-1)/2)$, μ is an upper bound on the number of repetitions of Steps 2 and 3. In Step 2 of the above procedure, we used LB* with Steps 2 and 3 removed (i.e., without considering $L_{3,2}$ and $L_{2,2}$).

Example. Consider the flow matrix of SREFLP instance O-8 taken from the Obata's data set (see Appendix A of Yu and Sarker 2003):

$$\begin{bmatrix} - & 1 & 5 & 5 & 7 & 8 & 3 & 4 \\ 1 & - & 1 & 5 & 7 & 8 & 1 & 6 \\ 5 & 1 & - & 4 & 10 & 1 & 3 & 8 \\ 5 & 5 & 4 & - & 7 & 7 & 6 & 10 \\ 7 & 7 & 10 & 7 & - & 2 & 10 & 10 \\ 8 & 8 & 1 & 7 & 2 & - & 7 & 8 \\ 3 & 1 & 3 & 6 & 10 & 7 & - & 2 \\ 4 & 6 & 8 & 10 & 10 & 8 & 2 & - \end{bmatrix}.$$

Since this matrix is symmetric, we first compute the lower bound for its upper triangle and then multiply this bound by two. The application of the procedure for determining the value of ρ yields $\rho = 4$. Steps 2 and 3 of LB*(≡ LB) process subgraphs with vertex sets $\{1, 2, 7\} \cup \{5, 6\}$ and $\{1, 2\} \cup \{5, 6\}$, respectively. In each case, $\alpha = 1$. Step 3 also gives $Q = 2$. Step 4 and the loop consisting of Steps 5 to 7 are executed twice. Initially, the vertex sets Z'_{ik}, $(i, k) \in R$, are used. They are formed and maintained using the layout $p^* = (p^*(1) = 3, p^*(2) = 1, p^*(3) = 7, p^*(4) = 4, p^*(5) = 6, p^*(6) = 2, p^*(7) = 8, p^*(8) = 5)$. In Step 5, the following triangles are selected (the first and the last vertices define a red edge): $(1, 6, 2)(\alpha = 1)$, $(7, 5, 8)(\alpha = 2)$, $(2, 8, 3)(\alpha = 2)$, $(2, 5, 3)(\alpha = 1)$, $(2, 6, 7)(\alpha = 1)$, $(2, 4, 7)(\alpha = 1)$, $(3, 8, 6)(\alpha = 2)$, $(3, 1, 6)(\alpha = 1)$. Afterwards, Steps 4 to 7 are executed using the vertex sets Z_{ik}, $(i, k) \in R$. Step 5 constructs the triangle $(3, 5, 7)$ with $\alpha = 1$. One can easily check that the resulting flow matrix is as given below:

$$\begin{bmatrix} - & 4 & 4 & 5 & 5 & 4 & 4 & 4 \\ & - & 4 & 4 & 4 & 4 & 4 & 4 \\ & & - & 4 & 8 & 4 & 4 & 4 \\ & & & - & 7 & 7 & 5 & 10 \\ & & & & - & 4 & 6 & 8 \\ & & & & & - & 5 & 6 \\ & & & & & & - & 4 \\ & & & & & & & - \end{bmatrix}.$$

The assignment matrix is as follows:

$$\begin{bmatrix} 115 & 90 & 74 & 66 & 66 & 74 & 90 & 115 \\ 112 & 88 & 72 & 64 & 64 & 72 & 88 & 112 \\ 116 & 92 & 76 & 68 & 68 & 76 & 92 & 116 \\ 142 & 110 & 92 & 84 & 84 & 92 & 110 & 142 \\ 146 & 112 & 93 & 85 & 85 & 93 & 112 & 146 \\ 122 & 95 & 79 & 71 & 71 & 79 & 95 & 122 \\ 119 & 93 & 77 & 69 & 69 & 77 & 93 & 119 \\ 132 & 102 & 86 & 78 & 78 & 86 & 102 & 132 \end{bmatrix}.$$

An optimal permutation for the linear assignment problem is $p' = (p'(1) = 2, p'(2) = 8, p'(3) = 1, p'(4) = 6, p'(5) = 3, p'(6) = 4, p'(7) = 7, p'(8) = 5)$. The optimal value equals 745. Hence $B_{\mathrm{GL}}(p) = \lceil 745/2 \rceil = 373$ and $B(p) = 2(B_{\mathrm{GL}}(p) + Q) = 750$ (the factor of two is present here because we have restricted ourselves to considering only the upper triangle of the matrix C). The classical Gilmore–Lawler bound is equal to 676. The optimal value for O-8 is 784. Thus, by performing simple and computationally very cheap operations we were able to get a lower bound which is much closer to the optimum.

3 A branch-and-bound algorithm

The algorithm we describe in this section starts with an empty layout and generates a sequence of partial layouts until the currently best solution is proved to be optimal. This process defines a search tree with nodes representing partial layouts.

To discard portions of the search tree, three tests are used. One of them is bounding test, which invokes LB and compares the returned value against the value of the best solution found during the search. Another test for pruning nodes is a dominance test. Suppose $p(i) > 0$, $i \in I(p) \subset I = \{1, \ldots, n\}$, $|I(p)| = m$, and $p(i) = 0$, $i \in \bar{I}(p) = I \backslash I(p)$. In other words, $I(p)$ stands for the set of already assigned facilities. It is assumed that $p(i) \in \{1, \ldots, m\}$ for each $i \in I(p)$. In order to apply the dominance test we replace the set $\bar{I}(p)$ by a macro facility q. In this way, we obtain an instance of the SREFLP with the set of facilities $I(p) \cup \{q\}$, the set of locations $\{1, \ldots, m+1\}$ and flows c_{ij}, $i, j \in I(p)$, $c_{iq} = \sum_{j \in \bar{I}(p)} c_{ij}$, $i \in I(p)$. The initial layout for this instance is given by the vector p_0 with components $p_0(i) = p(i)$, $i \in I(p)$, $p_0(q) = m+1$. We assume that the facility q is fixed in a sense that it cannot be moved from the location $m+1$ to any location to the left. Under this assumption, we apply a heuristic algorithm to this instance of the SREFLP. If the heuristic finds a solution that is better than the initial layout p_0, then the partial layout p is classified as dominated. The same verdict is issued also in the case where the heuristic finds an equally good solution as p_0 and this new solution p' is lexicographically smaller than p_0, that is, the following condition is satisfied: there exists $l < m$ such that, for $k = 1, \ldots, l-1$, $i = j$ whenever $p'(i) = p_0(j) = k$, and $i < j$ whenever $p'(i) = p_0(j) = l$. Clearly, there is no need to create a node for such p in the search tree. It may be noted that the heuristic algorithm can exit immediately (with answer "dominated") when nonoptimality of the initial layout p_0 is proved. As a heuristic, we use a tabu search algorithm to be briefly described in the next section. Our choice of tabu search was influenced by the fact that this technique, unlike evolutionary and some other methods, is capable to find solutions of good quality in a very short amount of time. When applying the dominance test, we tune the algorithm to execute a very small number of iterations.

The third test is called a symmetry test. It rests on the fact that, for a solution p to (1), (2), $F(p) = F(\tilde{p})$, where $\tilde{p}$ is obtained from p by assigning facilities in a reverse order: $\tilde{p}(i) = n - p(i) + 1$, $i = 1, \ldots, n$. This property can be utilized to constrain exploration of the solution space. The symmetry test is applied with respect to some selected facility, say, $y \in I$. The test discards a partial layout p if and only if the cardinality of the set $I(p)$ is at least $\lceil n/2 \rceil$ and $p(y) = 0$. In the role of y we use a facility i for which the sum $\sum_{j=1}^{n} c_{ij}$ is smallest.

The developed algorithm for the SREFLP is based on the branch-and-bound technique. To present the algorithm more clearly, we divide its description into two parts: the main procedure, named B&B (Branch-and-Bound), and an auxiliary procedure SELECT_FACILITY. The main procedure goes as follows.

B&B

1. Apply a heuristic algorithm to (1). Let p^* be the solution produced by this algorithm. Set $F^* := F(p^*)$.

2. Compute ρ.
3. Build lists $L_{3,2}$, $L_{2,2}$ and R. Choose facility y.
4. Create the root node of the search tree. Attach an empty set U to it. Set $h := 0$ (h stands for the search tree level), $p(i) := 0$, $i \in I$.
5. Set $m := h$.
6. Apply SELECT_FACILITY(m). It returns h and, possibly, some unassigned facility; in such a case, let it be denoted by j. If $h < 0$, then stop with the optimal solution p^* of value F^*. Otherwise, check whether $h < m$. If so, then return to 5. If not, then proceed to 7.
7. Create a new node with empty U. Assign facility j to location $m + 1$ by setting $p(j) := m + 1$. Increment h by one and go to 5.

As can be seen from the above description, the algorithm starts with four initialization steps. In Step 1 of the algorithm, we have chosen to use an iterated tabu search method as a heuristic for getting an initial solution to (1). We defer the description of this method to the next section. Steps 2 and 3 prepare the required data for LB used in the bounding test. Step 4 starts exploration of the solution space. To each node of the search tree a set U is associated. This set is introduced to contain facilities that are promising candidates for assignment to the leftmost unoccupied location. Upon creation of a node, the set U is empty. The facilities are included in this set while executing SELECT_FACILITY procedure. This procedure accepts as input a partial layout with m facilities and, provided U is empty, evaluates all unassigned facilities as possible candidates to location $m + 1$. It can be described as follows.

SELECT_FACILITY(m)

1. Check whether the set U attached to the current node is empty. If so, then proceed to 2. Otherwise go to 4.
2. If $m = 0$, then set $U := I$ and go to 4. If $m = n - 1$, then go to 3. Otherwise, for each $i \in \bar{I}(p) = \{i \in I \mid p(i) = 0\}$, perform the following operations. Temporarily assign i to the location $m + 1$. Apply the symmetry test, the dominance test and, if $m \geqslant m_0$, also the bounding test to the resulting partial layout. If it passes all the tests, then include i into U. After processing all facilities in $\bar{I}(p)$, go to 4.
3. Evaluate the layout obtained. If it is better than the best layout found so far, then update p^* and F^*.
4. If the set U is nonempty and contains at least one unmarked facility, then mark one of them arbitrarily and return with it as well as with $h = m$. Otherwise, check whether $m > 0$. If so, then remove the facility from the location m. In both cases, return with $h = m - 1$.

Step 1 of SELECT_FACILITY basically checks whether, for the current node, the procedure is entered for the first time. If the answer is in the affirmative, then the following three cases are processed separately: (1) the node is the root of the tree; (2) the node is a leaf; (3) the node corresponds to a partial layout with $m \in [1, n - 2]$. In the first case, each facility is considered as a possible candidate for placement in the first position. In the second case, the only remaining facility is assigned to the last location and thus the complete layout is obtained. In the third case, attempts are made to assign $i \in \bar{I}(p)$ to location $m + 1$. During the run of the algorithm, new nodes will

be created only for those resulting partial layouts which pass the tests defined earlier in this section. In order to make the algorithm computationally more efficient the bounding test is applied only when the number of already assigned facilities is greater than or equal to some specified integer m_0. Of course, the value of this parameter should be very small.

When SELECT_FACILITY is called for the same node once again, essentially only Step 4 is executed. In this case, one or more facilities in U are found marked as already used to extend the current layout and, hence, will not be selected. As the next candidate for assignment to the location $m+1$, any unmarked facility is chosen. If no such facility exists, then backtracking occurs. This operation amounts to removal of the facility from the location m (for $j \in I$ such that $p(j) = m$, setting $p(j) := 0$) and lowering the level of the search tree. The backtracking operation is also performed when the node is processed for the first time, but Step 4 is reached with empty U.

4 Iterated tabu search

In this section, we briefly describe an implementation of the tabu search method for the SREFLP. We preferred to use this method because of its simplicity as well as its ability to find good quality solutions in a short amount of time. Successful applications of the tabu search technique to a more general problem, the QAP, include Battiti and Tecchiolli (1994), Skorin-Kapov (1990) and Taillard (1991), among others.

As is clear from the previous section, the tabu search metaheuristic is applied in two contexts: the construction of the initial solution at the root node and search for a better solution than the current partial layout while performing the dominance test. It should be obvious that the power of the bounding test, and thus computation time, significantly depends on the value of the best solution found throughout the search process. Therefore, it is important to produce a high quality solution early in search, best of all, at the root node (Step 1 of B&B). For this reason, we decided to develop an iterated tabu search (ITS) algorithm for the SREFLP and use it in the initialization step of the branch-and-bound method. The ITS algorithm alternates between two phases: solution perturbation and tabu search. The presence of the perturbation phase makes the tabu search algorithm much smarter. In the description of ITS given below, $\Delta(p, i, j)$ denotes the increase in the value of the objective function F obtained by exchanging facilities i and j in solution p. The formula for $\Delta(p, i, j)$ basically is the same as the one used in the case of the QAP. The formula for that case can be found in a number of sources, for example, in Burkard and Rendl (1984). Obviously, if $\Delta(p, i, j) < 0$, then the exchange of i and j in the permutation p leads to an improved solution. A formal description of the algorithm is given as follows (b, γ_1 and γ_2 are parameters to be discussed below).

ITS

1. Randomly generate an assignment of facilities to locations $p = (p(1), \ldots, p(n))$. Set $p^* := p$. Compute $\Delta(p, i, j)$ for each pair $i, j \in \{1, \ldots, n\}, i < j$.
2. Apply TS(p, p^*).

3. Check if stopping criterion is met. If so, then stop with the solution p^* of value $F(p^*)$. Otherwise proceed to 4.
4. Set $q := 0$, $I := \{1, \ldots, n\}$. Randomly draw an even integer γ from the interval $[\gamma_1, \gamma_2]$ (or just set $\gamma := \gamma_2$ if $\gamma_2 \leqslant \gamma_1$).
5. Form a set K of $b' = \min(b, |I|(|I| - 1)/2)$ pairs (i, j), $i, j \in I$, $i < j$, such that $\Delta(p, i, j) \leqslant \Delta(p, r, s)$ for each $(i, j) \in K$ and each $(r, s) \notin K$, $r, s \in I$, $r < s$ (in other words, pick the b' smallest values $\Delta(p, i, j)$ among those with facilities i, j in I).
6. Randomly select $(k, l) \in K$. Remove both k and l from I. Exchange facilities k and l by setting $\eta := p(k)$, $p(k) := p(l)$, $p(l) := \eta$. Update $\Delta(p, i, j)$ for each pair $i, j \in \{1, \ldots, n\}$, $i < j$. Set $q := q + 2$. If $q < \gamma$, then return to 5. Otherwise go to 2.

In the above description, p and p^* stand for the current and, respectively, best found solutions. Initially, both p and p^* are assigned a random permutation. Step 1 also initializes $\Delta(p, i, j)$ for all pairs of facilities. Step 2 invokes the tabu search procedure TS. It returns updated p and possibly p^*. Step 3 requires a stopping criterion to be specified. It may be any, for example, an upper bound on the number of calls to TS or a stopping rule based on the CPU clock. In our experiments, we adopted the latter alternative. Steps 4 to 6 of ITS generate a new starting permutation for tabu search. In the loop defined by Steps 5 and 6, $\gamma/2$ pairwise exchanges of facilities are performed. The value of γ belongs to $[\gamma_1, \gamma_2]$ where γ_1 is a positive constant, γ_2 is the largest even integer less than or equal to $n\bar{\gamma}$ and $\bar{\gamma}$ is a tuning factor for controlling the degree of search diversification. Step 5 builds a set of pairs (i, j), $i, j \in I$, with the largest value of $\Delta(p, i, j)$. The cardinality of this set is bounded above by the parameter b whose value is chosen experimentally.

As it follows from the description of ITS, the current best solution p^* is updated in Step 2 by calling the tabu search procedure TS. Our implementation of TS given below is based on the general guidelines provided by Glover (1989).

TS(p, p^*)

1. Set $a := 0$, $\tilde{F} := F(p)$, $T_i := 0$, $i = 1, \ldots, n$.
2. Set $\beta := \infty$, $r := 0$.
3. For $i, j = 1, \ldots, n$, $i < j$, do
   3.1. Increment a by 1.
   3.2. If $\tilde{F} + \Delta(p, i, j) < F(p^*)$, then set $\beta := \Delta(p, i, j)$, $k := i$, $l := j$, $r := 1$ and go to 4.
   3.3. If $T_i = 0$, $T_j = 0$ and $\Delta(p, i, j) < \beta$, then set $\beta := \Delta(p, i, j)$, $k := i$, $l := j$.
4. Exchange facilities k and l. Set $\tilde{F} := \tilde{F} + \beta$. Update $\Delta(p, i, j)$ for each pair $i, j \in \{1, \ldots, n\}$, $i < j$. If $r = 0$, then go to 6. Otherwise proceed to 5.
5. Apply a local search procedure to p. Let p also denote the permutation returned by it. Set $p^* := p$, $\tilde{F} := F(p)$.
6. If a is greater than or equal to a predetermined upper limit $\bar{a}$, then return. Otherwise, decrement by one each positive T_i, $i \in \{1, \ldots, n\}$. Set $T_k := \tau$, $T_l := \tau$ and go to 2.

The input to TS includes the tabu tenure value τ and the maximum number of iterations $\bar{a}$. We take $\bar{a} = \lambda n(n-1)/2$, where λ is a scaling factor used to control the execution time of TS. In the description provided above, T_i, $i = 1, \ldots, n$, denote tabu values, which are initialized in Step 1 and updated in Step 6. The local search procedure used in Step 5 of TS is a straightforward 2-opt local improvement method.

In another context, namely within the dominance test, we use a simplified version of TS. This version is applied to special instances of the SREFLP defined at the beginning of Sect. 3. It should be assumed that for such instances, in the description of TS, n is replaced by m. The purpose of the simplified TS is merely to find a solution p' that is better than the solution $p = p^* = p_0$ submitted to TS. No further improvement of p' is needed. Therefore, Step 5 can be dropped. If the condition in Step 3.2 is satisfied, then TS stops prematurely and the dominance test gives a FAIL verdict (the partial layout inducing p_0 is dominated). Another change is to relax the condition in Step 3.3. We require at least one, and not necessarily both, of T_i and T_j to be equal to zero. This modification is introduced to deal with situations where m is very small. In such situations, the condition $T_i = T_j = 0$ may prevent most or even all pairs of facilities from being selected in Step 3.3. By relaxing this condition, we increase the level of intensification in the search process. TS can also be modified to examine solutions of value exactly $F(p_0)$. If such a solution p' is found and, moreover, this solution is lexicographically smaller than p_0 (see definition in Sect. 3), then the search is stopped and the dominance test returns the answer "dominated". In our implementation of the dominance test, we apply this rule only at the stage of computing initial values for $\Delta(p, i, j)$. Actually, the test immediately terminates if either $\Delta(p_0, i, j) < 0$ or $\Delta(p_0, i, j) = 0$ and the solution p' obtained by exchanging facilities i and j is lexicographically smaller than p_0, i.e., $i < j$ and $p_0(i) > p_0(j)$. We end this section by noting that, in the case of the dominance test, the values for the tabu search parameters τ and $\bar{a}$ should be smaller than in the case where TS is applied within ITS. These values may vary depending on the number of assigned facilities.

5 Computational results

The main purpose of experimentation was to evaluate the performance of the developed branch-and-bound algorithm as well as to show that many of the benchmark instances of the SREFLP from the literature can be solved exactly, and not only heuristically approached.

The described algorithm was coded in the C programming language and run on a Pentium M 1733 MHz notebook. As a testbed for investigating the algorithm, the following four sets of problem instances were considered: instances introduced by Obata (1979) (they are also listed in Appendix A of Yu and Sarker 2003), instances defined by Sarker (1989) (the matrix generating all instances in the series can be found in Wang and Sarker 2002), instances by Yu and Sarker (2003), and instances by Nugent et al. (1968). The last of these sets consists of instances of the QAP, therefore, we use only their flow matrices and ignore distance matrices. These instances can be taken from the Quadratic Assignment Problem Library (QAPLIB).

Table 1 Performance of B&B on the Obata instances

Instance	Optimum	Time	LB*	GL bound
O-5	150	1	146	142
O-6	292	1	274	264
O-7	472	1	450	422
O-8	784	1	750	676
O-9	1,032	1	994	908
O-10	1,402	1	1,340	1,204
O-15	5,134	2	4,854	4,310
O-20	12,924	37	12, 092	10,492

Some preliminary testing has been conducted using several problem instances of smaller size. The obtained results helped us select the appropriate values of the algorithm's parameters. For the ρ calculation, SELECT_FACILITY and ITS, the parameter settings are as follows: $\mu = 50$, $m_0 = 3$, $\gamma_1 = 10$, $\bar{\gamma} = 0.5$, $b = 500$. As for TS, we distinguish between two cases. When TS is used within ITS, we set τ to $\min(10, n/4)$ and λ to 40. Meanwhile, when TS is used within the dominance test, these values depend on the number m of positive components of the vector p: $\tau = 3$ if $m \geqslant 10$, $\tau = 2$ if $5 \leqslant m < 10$, $\tau = 1$ if $m < 5$, $\lambda = 10$ if $m \geqslant 7$, $\lambda = 5$ if $m < 7$. It should be noted that we apply TS in the context of the dominance test only when $m \geqslant 4$. Otherwise the test reduces to examining $\Delta(p, i, j), i, j \in I(p)$. A computer program implementing the algorithm also requires the CPU time limit for a run of ITS to be specified. We set this limit to 1 s if $n \leqslant 30$ and to 5 s if $n > 30$.

The main results of our experiments are summarized in Tables 1, 2, 3 and 4. The first column of Table 1 gives the names identifying the problem instances. The number of facilities in an instance is included into its name. The second and third columns display, for each Obata instance, the optimal value and, respectively, the time taken to solve that instance. The time in all the tables of the paper is reported in the form *hours*:*minutes*:*seconds* or *minutes*:*seconds* or *seconds*. The last two columns of Table 1 list the lower bounds on the objective function—LB* described in Sect. 2 and the Gilmore–Lawler (GL) bound, respectively. The meaning of the first five columns of Tables 2, 3 and 4 is the same as that of the columns of Table 1. In Table 2, the column under heading F_{DDH} displays, for each instance, the objective function value of a solution found by the directional decomposition heuristic (DDH) proposed by Yu and Sarker (2003). The last column gives the best known values for the Sarker instances. The entries of these columns are extracted from Table 1 of Yu and Sarker (2003). From that paper, also the last column of Table 3 is reproduced (see Table 3 and Appendix B of Yu and Sarker 2003). Finally, the last column of Table 4 is taken from Table 2 of Sarker et al. (1998). It should be noted that, for a number of problem instances we have tested, optimal solutions were found and their optimality has been proved in earlier studies. Such instances are O-5, . . ., O-10, N-12, S-12 (see Sarker et al. 1998) and Y-6, . . ., Y-12 (see Yu and Sarker 2003).

As it can be seen from Tables 1, 2, 3 and 4, each instance of size $n \leqslant 20$ was solved in less than 1 min. However, the algorithm needs much more time to solve the problem instances of size 25 and larger. Particularly in the case of Y-35, proving the

Table 2 Performance of B&B on the Sarker instances

Instance	Optimum	Time	LB*	GL bound	F_{DDH}	F_{best}
S-12	4,431	1	4,312	4,122	4,431	4,431
S-13	**5,897**	1	5,730	5,432	5,933	5,919
S-14	7,316	1	7,108	6,748	7,326	7,316
S-15	8,942	2	8,702	8,267	8,942	8,942
S-16	11,019	3	10,686	10,176	11,019	11,019
S-17	**13,172**	5	12,759	12,069	13,282	13,173
S-18	15,699	8	15,224	14,378	15,699	15,699
S-19	**18,700**	22	18,115	17,071	18,704	18,704
S-20	21,825	55	21,102	19,832	21,828	21,825
S-21	24,891	1:41	24,059	22,632	24,891	24,891
S-22	**28,607**	3:48	27,543	25,994	28,644	28,614
S-23	33,046	8:12	31,823	29,982	33,046	33,046
S-24	37,498	13:41	36,238	34,015	37,498	37,498
S-25	42,349	36:21	40,856	38,355	42,349	42,349

Optimal values in bold indicate cases where B&B improved upon the best solutions reported in the literature

Table 3 Performance of B&B on the Yu–Sarker instances

Instance	Optimum	Time	LB*	GL bound	F_{DDH}
Y-6	1,372	1	1,368	1,358	1,372
Y-7	1,801	1	1,786	1,752	1,801
Y-8	2,302	1	2,278	2,235	2,302
Y-9	2,808	1	2,781	2,735	2,808
Y-10	3,508	1	3,437	3,280	3,529
Y-11	4,022	1	3,970	3,858	4,032
Y-12	4,793	1	4,684	4,486	4,793
Y-13	5,471	1	5,329	5,089	5,471
Y-14	**6,445**	1	6,299	5,986	6,455
Y-15	**7,359**	2	7,201	6,884	7,423
Y-20	12,185	23	11,670	10,963	12,185
Y-25	**20,357**	22:38	19,488	17,903	20,359
Y-30	27,673	16:17:07	26,372	23,433	27,673
Y-35	**38,194**	459:08:51	36,007	31,750	38,201

Optimal values in bold indicate cases where B&B improved upon the best solutions reported in the literature

Table 4 Performance of B&B on the Nugent et al. instances

Instance	Optimum	Time	LB*	GL bound	F_{best}
N-12	1,000	1	878	706	1,000
N-15	2,186	2	1,866	1,508	2,186
N-20	**5,642**	41	4,796	3,674	5,666
N-25	9,236	24:03	7,714	5,454	–
N-30	**16,494**	12:34:18	13,670	9,266	16,698

Optimal values in bold indicate cases where B&B improved upon the best solutions reported in the literature

Table 5 Size of the search tree and the relative performance of the tests

Instance	#nodes	Dom. test	LB test	Sym. test
O-20	16,977	68.81[a]	30.37[b]	0.82[c]
S-20	32,317	71.94	25.89	2.17
S-25	622,095	76.88	22.41	0.71
Y-20	9,898	66.27	33.04	0.69
Y-25	352,765	75.34	23.93	0.73
Y-30	8,840,011	78.21	21.33	0.46
N-20	23,118	70.61	27.51	1.88
N-25	403,089	76.92	22.68	0.40
N-30	6,728,266	79.47	20.22	0.31

[a] Percentage of partial layouts fathomed due to dominance test
[b] Percentage of partial layouts fathomed due to bounding test
[c] Percentage of partial layouts fathomed due to symmetry test

optimality of the solution produced by ITS took more than 19 days. It is interesting to note that optimal solutions were found by ITS for all instances on which B&B was executed. Hence, the goal of the branching and bounding process was just to prove their optimality. The time (1 s) allotted to ITS was longer than the time (at most 0.7 s) taken by this process for each problem instance of size 15 or less. For 10 instances, the optimal solutions delivered by B&B are better than the best solutions reported in the literature. Such instances are indicated in Tables 2, 3 and 4 by displaying the optimal value in boldface. As we can see from Table 3, the DDH algorithm also failed to solve optimally both Y-10 and Y-11. However, the optimal solutions for these instances were obtained by Yu and Sarker (2003) using a complete enumeration approach. Comparing the results in the fourth and fifth columns of the tables, we can conclude that the lower bound for the SREFLP described in this paper is significantly stronger than the classical Gilmore–Lawler bound. It can be checked that LB* is closer to the optimal value than to the GL bound for all instances in the tables except the smallest two in Table 1—O-5 and O-6.

In Table 5, we give the number of nodes in the search tree (second column) and evaluate the effectiveness of tests used to fathom partial layouts. The results are reported for nine SREFLP instances of size ranging from 20 to 30. The last three columns of the table present $100N_{\text{dom}}/N$, $100N_{\text{LB}}/N$ and $100N_{\text{sym}}/N$, where N_{dom} (respectively, N_{LB} and N_{sym}) is the number of partial layouts that are fathomed due to dominance test (respectively, due to bounding test and due to symmetry) in Step 2 of SELECT_FACILITY, and $N = N_{\text{dom}} + N_{\text{LB}} + N_{\text{sym}}$. By analyzing the results in Table 5, we find that N_{dom} is 2–4 times larger than N_{LB}. However, it should not be forgotten that the dominance test is applied before the bounding test. In other words, the lower bound is calculated only for those partial solutions which have passed the dominance test. Certainly, exchanging the order of these two tests would increase N_{LB} and correspondingly decrease N_{dom} (since N is order-independent). However, the dominance test is faster than the bounding test, therefore, we gave some priority to the former. Generally, we notice that all the tests are rather fast and, therefore, the large number of nodes in the search tree, for example for both 30-facility instances, does not lead to extremely long computation times.

We also tested two variations of the basic branch-and-bound algorithm for the SREFLP. The first one is B&B with the dominance test removed. The second variation

Table 6 Alternative solution strategies

Instance	Without dominance test		LB replaced by GL bound	
	#nodes	Time	#nodes	Time
O-20	3,790,282	3:30:47	89,616	1:18
S-20	14,082,729	9:52:42	144,010	1:40
S-25	–	–	4,858,593	1:40:06
Y-20	1,293,124	1:04:31	44,998	49
Y-25	–	–	3,319,866	1:16:47
Y-30	–	–	148,390,064	89:19:36
N-20	5,312,912	4:14:46	86,668	1:10
N-25	–	–	2,743,842	1:07:32
N-30	–	–	81,447,624	56:06:47

Table 7 Results for the largest instances in the Yu–Sarker data set

Instance	F_{ITS}	Time	LB*	GL bound	F_{DDH}
Y-40	47,561	1.7	43,891	38,402	47,717
Y-45	62,890	0.2	58,551	49,790	63,132
Y-50	83,127	1.3	76,520	64,529	83,150
Y-60	112,055	7.0	102,828	83,980	112,170

is derived from B&B by replacing LB with the Gilmore–Lawler bound. The computational results are summarized in Table 6, where for each variation two columns are given—the first column in each pair lists the number of nodes in the search tree and the second one reports the solution time. A maximum time limit of 100 h was enforced for all runs. Table 6 shows that the presence of the dominance test is a decisive factor for a good performance of the algorithm. Without using this test, the largest problem instance solved to optimality was S-21. This modification of the algorithm did not terminate within the allotted time limit for S-22–S-25, Y-25, Y-30, N-25 and N-30. The solution time for each other instance of size $n \geqslant 20$ exceeded 1 h (see the third column). Comparing Tables 5 and 6, we see that the algorithm variation based on the Gilmore–Lawler bound builds much larger search trees than the main version of B&B. More importantly, this variation is significantly slower than the main version, especially when the number of facilities increases. In particular, the increase in solution time is approximately 450 and 350% for the two largest instances in Table 6—Y-30 and N-30, respectively.

The final computational experiment has been conducted on the four largest instances in the Yu–Sarker data set. In this experiment, we only run the iterated tabu search algorithm once and computed the lower bounds. The imposed time limit was 5 s for Y-40, Y-45, Y-50 and 10 s for Y-60. The results are shown in Table 7. The column under heading F_{ITS} contains, for each instance, the value of the solution produced by ITS. The third column gives the time taken by ITS to first find a solution that is best in the run. The meaning of the remaining columns is the same as in Table 3 where the numerical results for smaller instances in the Yu–Sarker data set are reported. The values in

the last column were obtained by Yu and Sarker (2003) and are extracted from Table 3 in their paper. We see from Table 7 that ITS was able to find better solutions than those produced by the DDH algorithm for all four problem instances. However, DDH was capable of providing good solutions with much less computational effort. The computation times for DDH reported by Yu and Sarker (2003) are comparable with those in Table 7, but Yu and Sarker carried out their experiments on a Pentium II Xeon 450 MHz PC, which is slower than the computer we have used.

6 Conclusions

In this paper we have developed a branch-and-bound algorithm for the single-row equidistant facility layout problem. The algorithm uses a lower bound which is reasonably tight, in particular, significantly tighter than the well-known Gilmore–Lawler bound. It is very important that the proposed bound is not expensive to compute. The algorithm also incorporates a dominance test which allows to drastically reduce redundancy in the search process. The test is performed by applying a tabu search procedure designed to solve the SREFLP. Using the developed algorithm, we were able to solve problem instances of size up to 35 facilities. One observation from the experiments is that the optimal values for a number of SREFLP instances are smaller than the best known values reported in the literature. The approach presented in this paper is an example of how metaheuristics can be used in concert with exact methods to effectively produce optimal solutions to the problem of interest.

References

Amaral ARS (2006) On the exact solution of a facility layout problem. Eur J Oper Res 173:508–518
Anstreicher KM (2003) Recent advances in the solution of quadratic assignment problems. Math Program 97:27–42
Battiti R, Tecchiolli G (1994) The reactive tabu search. ORSA J Comput 6:126–140
Bhasker J, Sahni S (1987) Optimal linear arrangement of circuit components. J VLSI Comput Syst 2:87–109
Burkard RE, Rendl F (1984) A thermodynamically motivated simulation procedure for combinatorial optimization problems. Eur J Oper Res 17:169–174
Chakrapani J, Skorin-Kapov J (1994) A constructive method for improving lower bounds for a class of quadratic assignment problems. Oper Res 42:837–845
Christofides N, Benavent E (1989) An exact algorithm for the quadratic assignment problem on a tree. Oper Res 37:760–768
Chung FRK (1988) Labelings of graphs. In: Beineke LW, Wilson RJ (eds) Selected topics in graph theory, vol 3, pp 151–168. Academic Press, San Diego
Díaz J, Petit J, Serna M (2002) A survey of graph layout problems. ACM Comput Surv 34:313–356
Drezner Z (2006) Finding a cluster of points and the grey pattern quadratic assignment problem. OR Spectrum 28:417–436
Gilmore PC (1962) Optimal and suboptimal algorithms for the quadratic assignment problem. J SIAM 10:305–313
Glover F (1989) Tabu search—part I. ORSA J Comput 1:190–206
Karisch SE, Rendl F (1995) Lower bounds for the quadratic assignment problem via triangle decompositions. Math Program 71:137–151
Karp RM, Held M (1967) Finite-state processes and dynamic programming. SIAM J Appl Math 15:693–718
Koopmans TC, Beckmann M (1957) Assignment problems and the location of economic activities. Econometrica 25:53–76

Kouvelis P, Chiang W, Yu G (1995) Optimal algorithms for row layout problems in automated manufacturing systems. IIE Trans 27:99–104
Lawler EL (1963) The quadratic assignment problem. Manage Sci 9:586–599
Loiola EM, de Abreu NMM, Boaventura-Netto PO, Hahn P, Querido T (2007) A survey for the quadratic assignment problem. Eur J Oper Res 176:657–690
Nugent CE, Vollmann TE, Ruml J (1968) An experimental comparison of techniques for the assignment of facilities to locations. Oper Res 16:150–173
Obata T (1979) Quadratic assignment problem: evaluation of exact and heuristic algorithms. Dissertation, Rensselaer Polytechnic Institute, Troy, NY
Öncan T, Altınel İK (2008) Exact solution procedures for the balanced unidirectional cyclic layout problem. Eur J Oper Res 189:609–623
Palubeckis GS (1988) A generator of test quadratic assignment problems with known optimal solution. USSR Comp Math Math Phys 28:97–98 [Translated from: Zh Vychisl Mat Mat Fiz 28:1740–1743]
Palubeckis G (1997) The use of special graphs for obtaining lower bounds in the geometric quadratic assignment problem. Informatica 8:377–400
Palubeckis G (2006) Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica 17:279–296
Palubeckis G (2007) Iterated tabu search for the maximum diversity problem. Appl Math Comput 189: 371–383
Pardalos PM, Crouse JV (1989) A parallel algorithm for the quadratic assignment problem. In: Proceedings of the 1989 ACM/IEEE Conference on Supercomputing, pp 351–360. ACM Press, New York
Picard J-C, Queyranne M (1981) On the one-dimensional space allocation problem. Oper Res 29:371–391
Sarker BR (1989) The amoebic matrix and one-dimensional machine location problems. Dissertation, Department of Industrial Engineering, Texas A&M University, College Station, TX
Sarker BR, Wilhelm WE, Hogg GL (1998) One-dimensional machine location problems in a multi-product flowline with equidistant locations. Eur J Oper Res 105:401–426
Skorin-Kapov J (1990) Tabu search applied to the quadratic assignment problem. ORSA J Comput 2:33–45
Taillard E (1991) Robust taboo search for the quadratic assignment problem. Parallel Comput 17:443–455
Taillard ÉD (1995) Comparison of iterative searches for the quadratic assignment problem. Location Sci 3:87–105
Wang S, Sarker BR (2002) Locating cells with bottleneck machines in cellular manufacturing systems. Int J Prod Res 40:403–424
Yu J, Sarker BR (2003) Directional decomposition heuristic for a linear machine-cell location problem. Eur J Oper Res 149:142–184