# Scatter search algorithms for the single row facility layout problem

Ravi Kothari
Diptesh Ghosh

**W.P. No. 2012-04-01**
April 2012



**INDIAN INSTITUTE OF MANAGEMENT**
**AHMEDABAD – 380015**
**INDIA**

# Scatter search algorithms for the single row facility layout problem

Ravi Kothari
Diptesh Ghosh

**Abstract**

The single row facility layout problem (SRFLP) is the problem of arranging facilities with given lengths on a line, with the objective of minimizing the weighted sum of the distances between all pairs of facilities. The problem is NP-hard and research has focused on heuristics to solve large instances of the problem. In this paper we present four scatter search algorithms to solve large sized SRFLP instances. Our computational experiments show that these algorithms generate better solutions to 26 of the 43 large sized benchmark SRFLP instances than were previously known in the literature. In the other 17 instances they output the best solutions previously known in the literature.
Keywords: Facilities planning and design; Single Row Facility Layout, Scatter search

## 1 Introduction

In the single row facility layout problem (SRFLP), we are given a set $F = \{1, 2, \ldots, n\}$ of $n > 2$ facilities, the length $l_j$ of each facility $j \in F$, and weights $c_{ij}$ for each pair $(i, j)$ of facilities, $i, j \in F$, $i \neq j$. The objective of the problem is to find a permutation $\Pi$ of facilities in $F$ that minimizes the total cost given by the expression

$$z(\Pi) = \sum_{1 \leq i < j \leq n} c_{ij} d_{ij}$$

where $d_{ij}$ is the distance between the centroids of the facilities $i$ and $j$ when arranged according the permutation $\Pi$. The cardinality of $F$ is called the size of the problem. The objective of the SRFLP is to arrange the facilities in $F$ on a line so as to minimize the weighted sum of the distances between all pairs of facilities. This problem is known to be NP-hard (Beghin-Picavet and Hansen 1982). The SRFLP was first proposed in Simmons (1969) and since then it has been used to design the arrangements of rooms in hospitals, departments in office buildings or in supermarkets (Simmons 1969), to arrange machines in flexible manufacturing systems (Heragu and Kusiak 1988), to assign files to disk cylinders in computer storage, and to design warehouse layouts (Picard and Queyranne 1981).

Several formulations of the SRFLP has been proposed in the literature (see Kothari and Ghosh 2011, for a comprehensive review) and researchers have used many exact and approximate approaches to solve the problem. Exact approaches include branch and bound (Simmons 1969), mathematical programming (Love and Wong 1976, Heragu and Kusiak 1991, Amaral 2006; 2008), cutting planes (Amaral 2009), dynamic programming (Picard and Queyranne 1981, Kouvelis and Chiang 1996), branch and cut (Amaral and Letchford 2012), and semidefinite programming (Anjos et al. 2005, Anjos and Vannelli 2008, Anjos and Yen 2009, Hungerländer and Rendl 2011). These methods have been able to obtain optimal solutions to SRFLP instances with up to 42 facilities. Researchers have focused on approximate approaches or heuristics to solve larger sized SRFLP instances. Heuristics are of two types, construction and improvement. Construction heuristics for the SRFLP have been proposed in Heragu and Kusiak (1988), Kumar et al. (1995), and Braglia (1997). However, improvement heuristics have superseded these construction heuristics in the literature and have yielded the best known solutions for large sized SRFLP instances. Most improvement heuristics for the SRFLP are metaheuristics which are either single solution approaches or population approaches. Single solution based heuristics include simulated annealing (Romero and Sánchez-Flores 1990, Kouvelis and Chiang 1992, Heragu and Alfa 1992) and tabu search (Samarghandi and Eshghi 2010), while the population heuristics for SRFLP include ant colony optimization (Solimanpur et al. 2005), scatter search (Kumar et al. 2008), particle swarm optimization (Samarghandi et al. 2010), and genetic algorithm (Datta et al. 2011). Among these the genetic algorithm in Datta et al. (2011) yield best results for benchmark SRFLP instances of large sizes.

Scatter search (see Glover 1977) have recently been shown to yield promising results for solving various non-linear and combinatorial optimization problems Glover (1998). Although it is a population based heuristic, it is significantly different from genetic algorithms. Similarities and differences between scatter search and genetic algorithms have been previously discussed in Glover (1994; 1995). To the best of our knowledge, the only study which has applied scatter search algorithm to the SRFLP is Kumar et al. (2008). It deals with SRFLP instances of size 30 only. We have not encountered any scatter search algorithm for larger sized SRFLP instances.

In this paper we present scatter search algorithms for solving large sized SRFLP instances. Our paper is organized as follows. In Section 2 we describe a generic scatter search algorithm for the SRFLP. We then describe our specific scatter search algorithms and present results of our computational experiments with these algorithms in Section 3. We conclude the paper in Section 4 with a summary of the work.

# 2 Scatter search for the SRFLP

Scatter search for a SRFLP is an evolutionary technique which orients its exploration relative to a set of permutations, called the reference set. The reference set typically consists of good permutations obtained by other methods. The criterion of "good" may refer not only to the cost of the permutation, but also to certain other properties that the permutations or a set of permutations may possess. Most scatter search algorithms for combinatorial optimization problems use the scatter search template in Glover (1998) as a reference for building the algorithm. We also draw ideas from the same template in our scatter search algorithms to solve large sized SRFLP instances. As per the template, our scatter search algorithms consist of five steps which we describe below.

## Diversification generation method

This element of scatter search determines the quality of permutations in the reference set and ensures diversification in the search process. We use the concept of deviation distances (see Sörensen 2007) to measure the diversification in a set of permutations. In a SRFLP a permutation is identical to the permutation obtained by reversing the positions of all the facilities in the permutation. So we define the distance between two permutations $\Pi_1$ and $\Pi_2$ as the minimum of the deviation distance between $\Pi_1$ and $\Pi_2$ and the deviation distance between $\Pi_1$ and the permutation obtained by reversing $\Pi_2$. Using this distance measure, we develop two methods, DIV-1 and DIV-2, to generate a diversified set of good permutations which serves as the initial population for our scatter search algorithms. Both these methods require an initial seed permutation of facilities in $F$ as an input for generating the initial population.

In the DIV-1 method, starting from the initial seed permutation $\Pi = (1, 2, \ldots, n)$, we generate a large set of permutations with cardinality $U\_Size$ by randomly interchanging facilities in $\Pi$. We then choose $E\_Size$ lowest cost permutations from the set to form a set of elite permutations. We then generate the initial population of size $P\_Size$ by choosing permutations from the elite set in a way such that the minimum distance between any two permutations in the population is as high as possible. The parameters $U\_Size$, $E\_Size$, and $P\_Size$ are specified by the user.

In the DIV-2 method, the initial seed permutation $\Pi$ is generated using Theorem 1 in Samarghandi and Eshghi (2010). A distance value $H\_Dist$ is also taken as an input. We first include $\Pi$ in the initial population. We then generate permutations by randomly interchanging facilities in $\Pi$, and include these permutations in the initial population only if the minimum distance between the newly generated permutation and each of the permutations in the initial population is at least $H\_Dist$. The method stops when we have the required number of permutations in the initial population.

## Improvement method

The improvement method tries to improve upon the permutations in the initial population and the permutations obtained by the solution combination method at later stages of the algorithm. In our scatter search algorithms we use local search iterations with an insertion neighborhood to improve the permutations. An insertion neighbor of a permutation is obtained by removing a facility from its position and introducing it at another position in the permutation. A local search iteration searches all the insertion neighbors of a given permutation and returns the best insertion neighbor of the permutation.

## Reference set update method

The reference set update method is used to build and update a set of permutations known as the reference set. The reference set consists of $B\_Size$ good permutations those which are best and diverse obtained during the run of the algorithm. The value of $B\_Size$ is typically not more than 20. At the start of the algorithm the user inputs two parameters $B_1$ and $B_2$ such that $B_1+B_2 = B\_Size$. The reference set is created from the set of permutations obtained by applying the improvement method on each permutation in the initial population. The improved permutations are then sorted in non-decreasing order of their costs and the first $B_1$ and last $B_2$ permutations in the sorted list are selected to be the members of the reference set. Hence $B_1$ permutations have low objective function values and $B_2$ permutations improve the diversity in the reference set.

In the later stages of the scatter search algorithm, the permutations in the reference set are combined using the subset generation method and solution combination method, and then improved using the improvement method to generate new permutations which may replace some of the existing members of the reference set. A new permutation replaces the highest cost permutation in the reference set if its cost is lower than it. When no new permutations get added to the reference set during a reference set update, we say that the reference set has converged and the lowest cost permutation in the reference set is output as a solution to the problem.

## Subset generation method

The subset generation method produces subsets of permutations in the reference set. These subsets are used to generate new permutations using the solution combination method. Outlines of various subset generation methods is available in Glover (1998). In our algorithms we generate all the subsets of the reference set with cardinality 2. We restrict ourselves to subsets of size 2 since scatter search algorithms with subsets of higher sizes are computationally impractical for large SRFLP instances.

## Solution combination method

The solution combination method combines the permutations in the subsets generated by the subset generation method into a new permutation. The new permutation thus generated is then subjected to the improvement method till it converges to a local optimum. The locally optimal permutation is finally either added to the reference set or discarded based on its cost. In our scatter search algorithms we use two combination methods, one of which is purely deterministic in implementation and the other one which has a random component.

The first combination method is the alternating combination method. It creates a new permutation by alternately selecting facilities from the first permutation and second permutations in the subset, omitting the facilities that have already been located in the new permutation. The operation is shown in Figure 1 with the shaded cells indicating the facilities selected for inclusion in the new permutation at the corresponding positions.

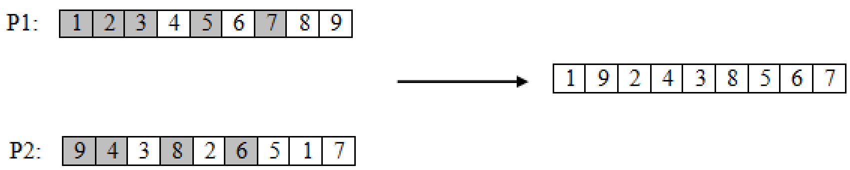


Figure 1: Alternating combination between permutations P1 and P2

The second combination method is the partially matched combination method. It is known as the PMX crossover mechanism in genetic algorithms, and is the most widely used combination operator for permutation problems. This combination method works as follows. For notational convenience we denote the two permutations in subset by P1 and P2, and the combined permutation as P3.

Step 1 Select a random segment (combination segment) of facilities from P1 and copy it to same location in P3. This constitutes the Step 1 as shown in Figure 2.

Step 2 Starting from the first combination point (i.e., the first facility in the combination segment in Step 1) look for facilities in that segment of P2 which have not been copied to P3. For each of these facilities $i$ in P2 look in P3 to see which facility $j$ has been copied in its place from P1 and place the facility $i$ in P3 in the same position as that occupied by facility $j$ in P2. This is Step 2 in figure 2. Note that if the position occupied by facility $j$ in P2 has already been occupied in P3 by facility $k$, then place facility $i$ in P3 in the same position as that occupied by facility $k$ in P2. Perform this step for every facility in the combination segment.

Step 3 Having dealt with the facilities from the combination segment, the rest of the locations in P3 are filled with facilities from P2 in order to ultimately obtain a complete permutation as shown in Step 3 in figure 2.

Having described the components of scatter search we are now in a position to describe a generic scatter search algorithm for the SRFLP. It starts by generating an initial population of permutations using a diversification generation method. It then uses an improvement method to improve the permutations in the initial population and builds a reference set using the reference set update method. The algorithm then performs iterations consisting of three steps till the reference set converges. In the first step it creates subsets of the reference set using a subset generation method. In the second step, for each of the subsets, it combines the permutations in the subset using a solution combination method to create one permutation corresponding to each subset. Each of these permutations is then improved using the improvement method. In the third step it tries to

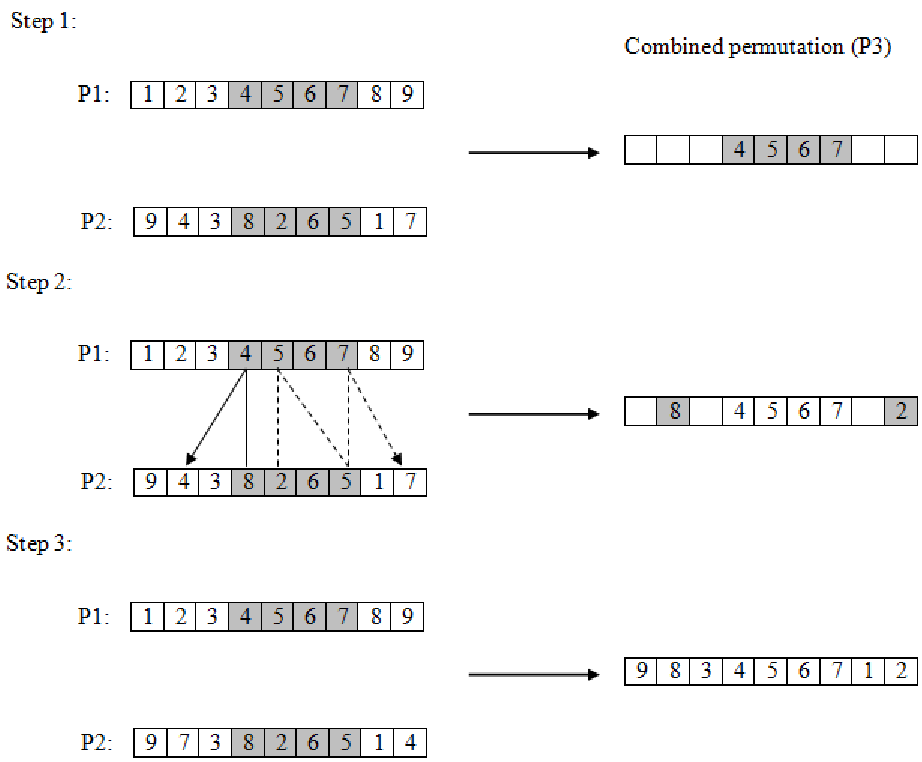


Figure 2: Partially matched combination between P1 and P2

update the reference set using these improved permutations. A permutation enters the reference set by replacing the highest cost permutation in the reference set if the cost of the permutation is lower than the cost of at least one permutation already present in the reference set. If none of the improved permutations enter the reference set in the third step, then the reference set is said to have converged and the scatter search algorithm terminates after reporting the lowest cost permutation in the reference set.

In our computational experiments described in the next section we create four scatter search implementations by specifying the methods used in the generic scatter search algorithm.

# 3 Computational experiments

We performed computational experiments with scatter search algorithms to compare its performance with other methods available in the literature to solve large sized SRFLP instances. We developed four versions of scatter search. All the versions used the same improvement method, reference set update method, and subset generation method but differed in diversification generation methods and subset generation methods. The details of the diversification generation method and solution combination method for the four algorithms are given in Table 1.

We coded these algorithms in C and performed our experiments on a personal computer with four Intel Core i5-2500 3.30GHz processors, 4GB RAM, running Ubuntu Linux 11.10. Based on initial experiments we set $U\_Size = 1000$, $E\_Size = 500$, and $P\_Size = 100$. For DIV-2 the value of $H\_Dist$ was set to $n^2/4$ where $n$ is the size of the instance. Following usual practice we set the size of the reference set $B\_Size$ at 20, with $B_1 = B_2 = 10$.

Table 1: The four scatter search algorithms used in experiments

| Algorithm | Diversification generation Method | Solution combination Method |
|---|---|---|
| SS-1A | DIV-1 | Alternating combination |
| SS-1P | DIV-1 | Partially matched combination |
| SS-2A | DIV-2 | Alternating combination |
| SS-2P | DIV-2 | Partially matched combination |

We experimented with three sets of benchmark SRFLP instances. The first set is due to Anjos et al. (2005) and consists of four groups of five instances each of size 60, 70, 75, and 80. We refer to these as the Anjos instances. These instances have been widely experimented with in the published literature (see, e.g., Datta et al. 2011, Hungerländer and Rendl 2011, Samarghandi and Eshghi 2010). The second set of instances are QAP-based `sko` instances. This set consists of four groups of five instances each of size 64, 72, 81, and 100. They were initially proposed in Anjos and Yen (2009) and have been experimented with in the literature in Anjos and Yen (2009), Amaral and Letchford (2012), Hungerländer and Rendl (2011). We refer to them as the `sko` instances. The third set is due to Amaral and Letchford (2012) and consists of three instances of size 110 each. We refer to these as the Amaral instances.

There are several published studies on the SRFLP which have reported results on the Anjos instances. Among them Samarghandi and Eshghi (2010) uses tabu search, Datta et al. (2011) uses a genetic algorithm, and Hungerländer and Rendl (2011) uses a SDP-relaxation based approach to provide upper bounds on the cost of optimal solutions. Among these, the results reported in Samarghandi and Eshghi (2010) have been superseded by those reported in Datta et al. (2011) and Hungerländer and Rendl (2011). Apart from the published literature, the Anjos instances have been experimented with using tabu search with 2-opt and insertion neighborhoods (Kothari and Ghosh 2012c), Lin-Kernighan based neighborhood search (Kothari and Ghosh 2012b), and genetic algorithm (Kothari and Ghosh 2012a). The results from these three studies supersede the results in the published literature. In Table 2 we present the costs of the best permutations obtained by the four scatter search algorithms and compare them with the results available in the literature. The first column of the table provides the name of the instance, and the second column provides its size. The third and fourth columns of the table present the costs of the best permutations obtained in the published and unpublished literature Kothari and Ghosh (2012c;b;a). The results in the third column have a superscript of either 'd' or 'h'. The letter 'd' indicates that the cost reported was obtained in Datta et al. (2011) while the letter 'h' indicates that it was obtained in Hungerländer and Rendl (2011). The last four columns report the costs of the best permutations obtained by the four scatter search algorithms that we present in this paper.

The results in Table 2 show that all the four scatter search algorithms generate results that are competitive with the unpublished literature. They equal the best costs in the published literature for 15 of the 20 instances and generate better permutations in the other five instances. The costs reported in these five instances were however already obtained in the unpublished literature. The permutations obtained in the five instances where scatter search generated permutations better than those reported in the published literature are presented in the appendix to the paper.

In Table 3 we report the execution times required by the four scatter search algorithms on the Anjos instances. We also report the execution times from Datta et al. (2011) and Hungerländer and Rendl (2011) for comparison. Note that the machines used in Datta et al. (2011) and Hungerländer and Rendl (2011) had different specifications than the one that we use for our experiments. The first two columns in the table present details about the instances, the third and fourth columns present times reported in Datta et al. (2011) and Hungerländer and Rendl (2011) respectively, and the last four columns gives the times required by the four scatter search algorithms.

Table 2: Comparison on solution costs to Anjos instances

| Instance | Size | Best costs in the literature Published[a] | Unpublished[b] | SS-1A | SS-1P | SS-2A | SS-2P |
|---|---|---|---|---|---|---|---|
| Anjos_60_01 | 60 | 1477834.0[dh] | 1477834.0 | 1477834.0 | 1477834.0 | 1477834.0 | 1477834.0 |
| Anjos_60_02 | 60 | 841776.0[h] | 841776.0 | 841776.0 | 841776.0 | 841776.0 | 841776.0 |
| Anjos_60_03 | 60 | 648337.5[dh] | 648337.5 | 648337.5 | 648337.5 | 648337.5 | 648337.5 |
| Anjos_60_04 | 60 | 398406.0[h] | 398406.0 | 398406.0 | 398406.0 | 398406.0 | 398406.0 |
| Anjos_60_05 | 60 | 318805.0[dh] | 318805.0 | 318805.0 | 318805.0 | 318805.0 | 318805.0 |
| Anjos_70_01 | 70 | 1528560.0[dh] | 1528537.0 | 1528537.0 | 1528537.0 | 1528537.0 | 1528537.0 |
| Anjos_70_02 | 70 | 1441028.0[dh] | 1441028.0 | 1441028.0 | 1441028.0 | 1441028.0 | 1441028.0 |
| Anjos_70_03 | 70 | 1518993.5[dh] | 1518993.5 | 1518993.5 | 1518993.5 | 1518993.5 | 1518993.5 |
| Anjos_70_04 | 70 | 968796.0[d] | 968796.0 | 968796.0 | 968796.0 | 968796.0 | 968796.0 |
| Anjos_70_05 | 70 | 4218002.5[h] | 4218002.5 | 4218002.5 | 4218002.5 | 4218002.5 | 4218002.5 |
| Anjos_75_01 | 75 | 2393456.5[d] | 2393456.5 | 2393456.5 | 2393456.5 | 2393456.5 | 2393456.5 |
| Anjos_75_02 | 75 | 4321190.0[d] | 4321190.0 | 4321190.0 | 4321190.0 | 4321190.0 | 4321190.0 |
| Anjos_75_03 | 75 | 1248537.0[d] | 1248423.0 | 1248423.0 | 1248423.0 | 1248423.0 | 1248423.0 |
| Anjos_75_04 | 75 | 3941845.5[d] | 3941816.5 | 3941816.5 | 3941816.5 | 3941816.5 | 3941816.5 |
| Anjos_75_05 | 75 | 1791408.0[d] | 1791408.0 | 1791408.0 | 1791408.0 | 1791408.0 | 1791408.0 |
| Anjos_80_01 | 80 | 2069097.5[d] | 2069097.5 | 2069097.5 | 2069097.5 | 2069097.5 | 2069097.5 |
| Anjos_80_02 | 80 | 1921177.0[d] | 1921136.0 | 1921136.0 | 1921136.0 | 1921136.0 | 1921136.0 |
| Anjos_80_03 | 80 | 3251368.0[d] | 3251368.0 | 3251368.0 | 3251368.0 | 3251368.0 | 3251368.0 |
| Anjos_80_04 | 80 | 3746515.0[d] | 3746515.0 | 3746515.0 | 3746515.0 | 3746515.0 | 3746515.0 |
| Anjos_80_05 | 80 | 1588901.0[d] | 1588885.0 | 1588885.0 | 1588885.0 | 1588885.0 | 1588885.0 |

a: Best costs published in the literature. The letters ‘d’ and ‘h’ refer to the costs reported in Datta et al. (2011) and upper bounds reported in Hungerländer and Rendl (2011) respectively.

b: Best costs reported in Kothari and Ghosh (2012c;b;a).

From Table 3 we observe that the times required by the four scatter search algorithms are slightly higher than those reported in Datta et al. (2011). This is due to the fact that the subset generation method in scatter search is exhaustive in nature and hence requires much longer time than the crossover operation for genetic algorithms. The convergence of scatter search is however much faster; for the Anjos instances, scatter search always converged in less than 5 iterations. The execution times required by the scatter search algorithms is clearly much less than those reported in Hungerländer and Rendl (2011).

In the published literature, results of computational experiments with the `sko` instances have been reported in Amaral and Letchford (2012), Anjos and Yen (2009), and Hungerländer and Rendl (2011). Amaral and Letchford (2012) uses a polyhedral approach while Anjos and Yen (2009) and Hungerländer and Rendl (2011) use SDP-relaxation based approaches. All three report upper bounds to the costs of optimal solutions. The results in Amaral and Letchford (2012) supersede those in Anjos and Yen (2009) and Hungerländer and Rendl (2011). In the unpublished literature, good quality permutations for the `sko` instances have been reported using tabu search with 2-opt and insertion neighborhoods (Kothari and Ghosh 2012c), local search using Lin-Kernighan neighborhoods (Kothari and Ghosh 2012b) and using a genetic algorithm (Kothari and Ghosh 2012a). These results are competitive with the results in the published literature. In Table 4 we compare the costs of the permutations output by the four scatter search algorithms with those known from the literature. The first and second columns of the table present details about the instances, the third column presents the costs of the best permutations known from the published literature, and the fourth column presents the costs of the best permutations known from the published literature. Each cost value in the fourth column has a superscript of ‘g’, ‘l’ or ‘t’ to indicate whether the best cost has been reported in Kothari and Ghosh (2012a), Kothari and Ghosh (2012b), and Kothari and

Table 3: Comparison on execution times (in seconds) for Anjos instances

| Instance | Size | DA&F[a] | H&R[b] | SS-1A | SS-1P | SS-2A | SS-2P |
|---|---|---|---|---|---|---|---|
| Anjos_60_01 | 60 | 19.54 | 103169.00 | 55.41 | 46.73 | 53.25 | 41.15 |
| Anjos_60_02 | 60 | 22.34 | 111126.00 | 52.07 | 41.54 | 51.43 | 47.04 |
| Anjos_60_03 | 60 | 68.81 | 85021.00 | 49.12 | 50.40 | 74.61 | 64.71 |
| Anjos_60_04 | 60 | 20.71 | 95439.00 | 69.38 | 53.52 | 70.22 | 52.74 |
| Anjos_60_05 | 60 | 26.41 | 99106.00 | 48.78 | 52.35 | 61.36 | 47.77 |
| Anjos_70_01 | 70 | 64.83 | 96094.00 | 94.45 | 82.61 | 122.24 | 91.01 |
| Anjos_70_02 | 70 | 77.49 | 94287.00 | 108.87 | 88.16 | 114.53 | 96.62 |
| Anjos_70_03 | 70 | 68.26 | 94514.00 | 83.65 | 83.17 | 117.31 | 99.83 |
| Anjos_70_04 | 70 | 100.59 | 98928.00 | 145.01 | 102.26 | 139.60 | 95.31 |
| Anjos_70_05 | 70 | 60.48 | 101765.00 | 95.44 | 81.59 | 112.91 | 79.93 |
| Anjos_75_01 | 75 | 125.26 | 136673.00 | 136.76 | 127.66 | 151.92 | 119.47 |
| Anjos_75_02 | 75 | 128.95 | 142118.00 | 171.40 | 118.01 | 163.11 | 136.90 |
| Anjos_75_03 | 75 | 157.95 | 138066.00 | 195.62 | 143.69 | 155.72 | 136.57 |
| Anjos_75_04 | 75 | 119.92 | 139378.00 | 120.14 | 125.37 | 166.94 | 122.04 |
| Anjos_75_05 | 75 | 101.67 | 148237.00 | 156.94 | 138.71 | 134.89 | 120.80 |
| Anjos_80_01 | 80 | 75.41 | 210289.00 | 158.81 | 162.59 | 167.66 | 172.45 |
| Anjos_80_02 | 80 | 68.75 | 211635.00 | 225.85 | 174.80 | 233.71 | 166.69 |
| Anjos_80_03 | 80 | 85.90 | 209839.00 | 185.16 | 181.84 | 166.86 | 146.98 |
| Anjos_80_04 | 80 | 77.81 | 211847.00 | 201.84 | 148.41 | 209.13 | 181.59 |
| Anjos_80_05 | 80 | 196.51 | 210630.00 | 197.26 | 199.79 | 189.51 | 205.12 |

a: Execution times published in Datta et al. (2011)
b: Execution times reported in Hungerländer and Rendl (2011)

Ghosh (2012c) respectively. The last four columns present the costs of the permutations output by the four scatter search algorithms.

The results in Table 4 clearly demonstrate the superiority of the scatter search algorithms over the previously known methods for the `sko` instances. In 18 of the 20 instances, the results from the scatter search algorithms are superior to those known in the published literature. For these 18 instances we report the permutations from scatter search in the appendix to the paper. In 13 among the `sko` instances, they are superior to the results known even in the unpublished literature. There is only one instance, namely `sko`_64_02, in which the SS-1A algorithm outputs a permutation which is worse than in the published literature. For all other instances the scatter search algorithms have either matched or superseded the best known results in the literature. We did not observe the dominance of any of the four scatter search algorithms over the other three for these instances.

We report the execution times required by the four scatter search algorithms on the `sko` instances in Table 5. The first two columns in the table describe the instance, and the remaining four columns report the execution times required by the four scatter search algorithms. We do not report the times from Amaral and Letchford (2012) for these instances since it only reported a run-time limit of one day for these instances. The table shows that the four scatter search algorithms required reasonable execution times for these instances. We observed that all the four scatter search algorithms converged in less than 5 iterations for the `sko` instances.

Results for the Amaral instances have been reported in Amaral and Letchford (2012) in the published literature, and in Kothari and Ghosh (2012a) in the unpublished literature. The results in Kothari and Ghosh (2012a) are better for the first two instances, while the result in Amaral and Letchford (2012) is better for the third instance. In Table 6 we report the results obtained from the four scatter search algorithms for these instances. The first two columns in table describe the instance, the third and fourth columns report the results from Amaral and Letchford (2012) and Kothari and Ghosh (2012a), and the other four columns report the costs of permutations output by

Table 4: Comparison on solution costs to sko instances

| Instance | Size | Best costs from the literature A&L[a] | Unpublished[b] | SS-1A | SS-1P | SS-2A | SS-2P |
|---|---|---|---|---|---|---|---|
| sko_64_01 | 64 | 96930.0 | 96915.0[t] | 96884.0 | 96883.0 | 96884.0 | 96890.0 |
| sko_64_02 | 64 | 634332.5 | 634332.5[g] | 634338.5 | 634332.5 | 634332.5 | 634332.5 |
| sko_64_03 | 64 | 414356.5 | 414323.5[lg] | 414323.5 | 414323.5 | 414323.5 | 414323.5 |
| sko_64_04 | 64 | 297358.0 | 297205.0[l] | 297129.0 | 297129.0 | 297137.0 | 297129.0 |
| sko_64_05 | 64 | 501922.5 | 501922.5[tlg] | 501922.5 | 501922.5 | 501922.5 | 501922.5 |
| sko_72_01 | 72 | 139174.0 | 139150.0[lg] | 139153.0 | 139150.0 | 139150.0 | 139150.0 |
| sko_72_02 | 72 | 712261.0 | 712005.0[l] | 711998.0 | 711998.0 | 711998.0 | 711998.0 |
| sko_72_03 | 72 | 1054184.5 | 1054110.5[tlg] | 1054141.5 | 1054110.5 | 1054110.5 | 1054110.5 |
| sko_72_04 | 72 | 920693.5 | 919590.5[g] | 919586.5 | 919586.5 | 919586.5 | 919586.5 |
| sko_72_05 | 72 | 428305.5 | 428228.5[g] | 428226.5 | 428226.5 | 428228.5 | 428228.5 |
| sko_81_01 | 81 | 205475.0 | 205145.0[t] | 205112.0 | 205106.0 | 205120.0 | 205112.0 |
| sko_81_02 | 81 | 523021.5 | 521391.5[lg] | 521391.5 | 521391.5 | 521391.5 | 521391.5 |
| sko_81_03 | 81 | 970920.0 | 970862.0[l] | 970796.0 | 970796.0 | 970796.0 | 970796.0 |
| sko_81_04 | 81 | 2032634.0 | 2031803.0[g] | 2031803.0 | 2031803.0 | 2031803.0 | 2031803.0 |
| sko_81_05 | 81 | 1303756.0 | 1302733.0[g] | 1302711.0 | 1302711.0 | 1302711.0 | 1302711.0 |
| sko_100_01 | 100 | 378584.0 | 378378.0[g] | 378249.0 | 378234.0 | 378259.0 | 378234.0 |
| sko_100_02 | 100 | 2076714.5 | 2076023.5[t] | 2076008.5 | 2076008.5 | 2076008.5 | 2076008.5 |
| sko_100_03 | 100 | 16177226.5 | 16148818.0[l] | 16145598.0 | 16145614.0 | 16145598.0 | 16149444.0 |
| sko_100_04 | 100 | 3237111.0 | 3232740.0[l] | 3232522.0 | 3232531.0 | 3232522.0 | 3232522.0 |
| sko_100_05 | 100 | 1034922.5 | 1033338.5[t] | 1033085.5 | 1033080.5 | 1033085.5 | 1033080.5 |

a: Upper bounds reported in Amaral and Letchford (2012).
b: Best costs reported in the unpublished literature. The letters 'g', 'l' and 't' indicate whether the best cost has been reported in Kothari and Ghosh (2012a), Kothari and Ghosh (2012b), and Kothari and Ghosh (2012c) respectively.

the scatter search algorithms. We see from the table that all the scatter search algorithms output permutations that are superior to those known from the literature, both published and unpublished. For these instances, the results from SS-1A, SS-2A, and SS-2P are identical, and those from SS-1P is uniformly worse than the other three algorithms. We report the best permutations obtained by scatter search on these instances in the appendix to the paper.

Table 7 presents the execution times required by the four scatter search algorithms on the Amaral instances. The first two columns in the table describe the instance, and the remaining four columns report the execution times required by the four scatter search algorithms. We do not report the times from Amaral and Letchford (2012) since that paper only reported a run-time limit of 2.5 days for these instances. We see that all scatter search algorithms required reasonably low execution times for these problems. All scatter search algorithms converged in less than 10 iterations.

Based on results from our computational experiments, we conclude that the four scatter search algorithms presented in this paper are superior to all other algorithms for the SRFLP available in the literature and are recommended for solving large sized SRFLP instances.

# 4 Summary and future research

In this paper we have presented four scatter search algorithms for the single row facility layout problem (SRFLP). Our algorithms inherit the basic structure from the scatter search template presented in Glover (1998). Each of our algorithms use a diversification generation method to generate

Table 5: Comparison on execution times (in seconds) for sko instances

| Instance | Size | SS-1A | SS-1P | SS-2A | SS-2P |
|---|---|---|---|---|---|
| sko_64_01 | 64 | 104.44 | 82.80 | 113.68 | 92.98 |
| sko_64_02 | 64 | 92.62 | 68.31 | 96.57 | 88.52 |
| sko_64_03 | 64 | 73.16 | 80.33 | 94.54 | 67.59 |
| sko_64_04 | 64 | 89.61 | 91.05 | 86.03 | 84.92 |
| sko_64_05 | 64 | 85.05 | 75.86 | 66.69 | 77.51 |
| sko_72_01 | 72 | 143.78 | 115.21 | 161.73 | 150.07 |
| sko_72_02 | 72 | 151.93 | 192.12 | 164.51 | 226.29 |
| sko_72_03 | 72 | 124.10 | 114.16 | 124.71 | 112.02 |
| sko_72_04 | 72 | 158.49 | 135.46 | 188.20 | 132.00 |
| sko_72_05 | 72 | 152.25 | 116.50 | 291.69 | 172.41 |
| sko_81_01 | 81 | 423.73 | 311.63 | 548.55 | 273.92 |
| sko_81_02 | 81 | 213.56 | 226.91 | 250.67 | 227.27 |
| sko_81_03 | 81 | 217.01 | 241.50 | 336.89 | 263.23 |
| sko_81_04 | 81 | 215.28 | 193.98 | 225.60 | 236.82 |
| sko_81_05 | 81 | 316.77 | 287.22 | 243.86 | 257.06 |
| sko_100_01 | 100 | 1039.38 | 877.12 | 874.60 | 789.64 |
| sko_100_02 | 100 | 796.18 | 526.09 | 747.75 | 468.01 |
| sko_100_03 | 100 | 670.34 | 599.92 | 851.42 | 529.58 |
| sko_100_04 | 100 | 456.26 | 592.00 | 651.25 | 532.43 |
| sko_100_05 | 100 | 1068.02 | 591.75 | 1159.18 | 619.13 |

Table 6: Comparison on solution costs to Amaral instances

| | | Best costs from the literature | | | | | |
|---|---|---|---|---|---|---|---|
| Instance | Size | A&L[a] | K&G[b] | SS-1A | SS-1P | SS-2A | SS-2P |
| Amaral_1 | 110 | 144331884.5 | 144302160.0 | 144296768.0 | 144297440.0 | 144296768.0 | 144296768.0 |
| Amaral_2 | 110 | 86065390.0 | 86056632.0 | 86050112.0 | 86050208.0 | 86050112.0 | 86050112.0 |
| Amaral_3 | 110 | 2234803.5 | 2234825.5 | 2234743.5 | 2234798.5 | 2234743.5 | 2234743.5 |

a: Upper bounds published in Amaral and Letchford (2012).
b: Best costs reported in Kothari and Ghosh (2012a).

Table 7: Comparison on execution times (in seconds) for Amaral instances

| Instance | Size | SS-1A | SS-1P | SS-2A | SS-2P |
|---|---|---|---|---|---|
| Amaral_1 | 110 | 1090.24 | 975.11 | 944.01 | 777.15 |
| Amaral_2 | 110 | 788.97 | 680.10 | 905.64 | 774.89 |
| Amaral_3 | 110 | 698.25 | 811.08 | 739.04 | 611.40 |

an initial population containing good quality and diverse permutations, an improvement method to improve the permutations in the algorithm, a reference set update method to build and maintain a reference set of good quality permutations, and a subset generation method and a solution combination method to update the reference set with new permutations. We present two diversification generation methods called DIV-1 and DIV-2 and two solution combination methods, called the alternating combination method and partially matched combination method, whose combinations yield four scatter search algorithms called SS-1A, SS-1P, SS-2A, and SS-2P.

We perform computational experiments to compare the performance of the four scatter search algorithms with the best known results in the literature for three sets of benchmark instances comprising of 43 large sized SRFLP instances. Our computational experience indicates that all the four scatter search algorithms are superior to all other algorithms for the SRFLP that are known in the literature. Compared to the results reported in the published literature, among the 43 benchmark instances, they matched the best known solutions in 17 instances and obtain better solutions in the other 26 instances within reasonable execution times. If we include the unpublished literature in our comparison, the scatter search algorithms obtained better solutions in 21 instances and matched the best results in the other 22. We therefore feel that scatter search algorithms are algorithms of choice for solving large sized SRFLP instances.

## References

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

Amaral, A. R. S. (2008). An Exact Approach to the One-Dimensional Facility Layout Problem. *Operations Research*, 56(4):1026–1033.

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

Amaral, A. R. S. and Letchford, A. N. (2012). A polyhedral approach to the single row facility layout problem. Available at www.lancs.ac.uk/staff/letchfoa/articles/SRFLP-rev.pdf.

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

Anjos, M. F. and Vannelli, A. (2008). Computing Globally Optimal Solutions for Single-Row Layout Problems Using Semidefinite Programming and Cutting Planes. *INFORMS Journal on Computing*, 20(4):611–617.

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

Beghin-Picavet, M. and Hansen, P. (1982). Deux problèmes daffectation non linéaires. *RAIRO, Recherche Opérationnelle*, 16(3):263–276.

Braglia, M. (1997). Heuristics for single-row layout problems in flexible manufacturing systems. *Production Planning & Control*, 8(6):558–567.

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

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166.

Glover, F. (1994). Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). *Discrete Applied Mathematics*, 49:231 – 255.

Glover, F. (1995). Scatter search and star-paths: beyond the genetic metaphor. *OR Spectrum*, 17:125–137. 10.1007/BF01719256.

Glover, F. (1998). A template for scatter search and path relinking. *Lecture notes in computer science*, 1363:13–54.

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

Heragu, S. S. and Kusiak, A. (1988). Machine Layout Problem in Flexible Manufacturing Systems. *Operations Research*, 36(2):258–268.

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

Hungerländer, P. and Rendl, F. (Unpublished results, 2011). A computational study for the single-row facility layout problem. Available at www.optimization-online.org/DB_FILE/2011/05/3029.pdf.

Kothari, R. and Ghosh, D. (2011). The single row facility layout problem: State of the art (w.p. no. 2011-12-02). Ahmedabad, India: IIM Ahmedabad, Production & Quantitative Methods. Available at www.iimahd.ernet.in/assets/snippets/workingpaperpdf/7736113342011-12-02.pdf.

Kothari, R. and Ghosh, D. (2012a). A competitive genetic algorithm for single row facility layout (w.p. no. 2012-03-01). Ahmedabad, India: IIM Ahmedabad, Production & Quantitative Methods. Available at www.optimization-online.org/DB_HTML/2012/02/3369.html.

Kothari, R. and Ghosh, D. (2012b). A Lin-Kernighan heuristic for single row facility layout (w.p. no. 2012-01-04). Ahmedabad, India: IIM Ahmedabad, Production & Quantitative Methods. Available at www.optimization-online.org/DB_HTML/2012/01/3315.html.

Kothari, R. and Ghosh, D. (2012c). Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods (w.p. no. 2012-01-03). Ahmedabad, India: IIM Ahmedabad, Production & Quantitative Methods. Available at www.optimization-online.org/DB_HTML/2012/01/3314.html.

Kouvelis, P. and Chiang, W.-C. (1992). A simulated annealing procedure for single row layout problems in flexible manufacturing systems. *International Journal of Production Research*, 30(4):717–732.

Kouvelis, P. and Chiang, W.-C. (1996). Optimal and Heuristic Procedures for Row Layout Problems in Automated Manufacturing Systems. *Journal of the Operational Research Society*, 47(6):803–816.

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

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

Love, R. F. and Wong, J. Y. (1976). On solving a one-dimensional space allocation problem with integer programming. *INFOR*, 14(2):139–144.

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

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

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

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

Simmons, D. M. (1969). One-Dimensional Space Allocation: An Ordering Algorithm. *Operations Research*, 17(5):812–826.

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

Sörensen, K. (2007). Distance measures based on the edit distance for permutation-type representations. *Journal of Heuristics*, 13:35–47. 10.1007/s10732-006-9001-3.

# Appendix

We provide details of the permutations for the instances in which we have improved the best permutation known in the literature. Note that the facilities are numbered from 1 through $n$ where $n$ is the problem size.

| Instance | Size | Cost | Permutation |
|---|---|---|---|
| Anjos-70-01 | 70 | 1528537.0 | 53 47 65 40 2 28 62 22 15 8 32 9 63 31 69 51 68 1 4 16 64 61 41 38 56 67 70 44 10 26 14 19 33 42 49 5 30 36 23 55 60 13 18 21 24 27 54 11 12 58 6 59 52 7 20 66 3 34 45 46 25 43 48 17 29 57 39 35 37 50 |
| Anjos-75-03 | 75 | 1248423.0 | 47 69 42 10 19 33 15 17 43 51 41 46 29 23 68 26 60 4 39 74 64 61 56 20 36 12 27 13 48 71 11 65 57 5 67 45 21 28 35 24 9 75 58 73 40 7 32 6 49 52 59 34 3 16 62 31 30 44 37 2 38 66 70 18 72 8 25 55 53 63 14 1 54 50 22 |
| Anjos-75-04 | 75 | 3941816.0 | 36 60 5 14 15 50 7 75 10 42 62 37 8 70 30 47 22 57 20 41 29 40 33 39 46 12 3 64 35 65 16 52 28 53 44 73 34 18 24 45 13 32 1 67 2 19 55 48 56 63 66 26 23 58 59 54 43 71 4 31 11 74 61 51 6 25 27 68 69 38 72 49 9 17 21 |
| Anjos-80-02 | 80 | 1921136.0 | 11 66 61 45 48 43 49 63 69 51 80 60 78 64 52 31 71 37 79 7 70 30 73 76 59 19 68 26 13 75 42 62 44 50 2 65 8 34 40 9 36 39 38 4 20 53 3 67 6 21 35 25 23 28 27 72 1 77 33 16 29 41 54 12 47 74 32 56 15 46 14 58 5 55 17 10 24 18 57 22 |
| Anjos-80-05 | 80 | 1588885.0 | 2 7 47 52 38 27 61 73 21 67 68 10 37 74 11 22 1 70 8 31 19 9 76 33 36 62 30 34 79 75 54 44 6 43 53 35 71 66 55 65 59 3 32 39 77 18 4 80 20 45 40 63 23 46 50 57 5 56 15 24 28 26 64 14 51 16 29 12 72 42 49 13 78 69 60 58 41 48 17 25 |
| sko-64-01 | 64 | 96883.0 | 31 57 35 18 60 63 2 56 25 50 37 17 26 59 7 1 45 33 44 21 14 8 58 9 22 24 29 52 23 13 49 10 16 53 42 4 64 19 20 62 55 48 38 6 47 51 39 40 5 30 43 28 11 54 32 46 3 34 41 61 27 12 36 15 |
| sko-64-03 | 64 | 414323.5 | 15 12 9 61 56 41 42 49 13 29 4 52 22 23 16 46 36 51 64 55 21 27 31 3 44 14 58 57 24 53 10 25 63 43 18 47 30 35 17 38 34 45 1 39 5 60 26 28 40 11 54 2 8 33 37 19 32 48 20 7 50 59 62 6 |
| sko-64-04 | 64 | 297129.0 | 15 59 57 53 11 54 32 41 30 8 19 10 33 56 25 17 2 38 50 26 34 37 1 7 45 47 35 44 58 13 49 21 24 4 64 39 62 5 16 23 12 29 55 22 9 28 48 14 40 61 52 18 6 3 51 46 20 63 27 43 42 36 60 31 |
| sko-72-01 | 72 | 139150.0 | 34 3 24 54 45 11 5 25 69 72 62 4 43 66 20 58 19 15 41 48 40 42 68 51 36 57 55 44 71 6 9 16 39 32 63 37 13 50 33 49 46 59 38 7 1 29 61 23 52 65 47 26 28 22 14 2 21 67 17 30 27 70 10 56 8 35 18 60 64 31 53 12 |
| sko-72-02 | 72 | 711988.0 | 34 5 58 41 45 11 62 64 20 9 40 71 17 1 19 4 66 54 6 16 72 61 15 36 69 68 57 26 44 55 24 67 39 33 29 25 48 42 51 8 50 31 21 49 7 30 27 2 38 46 60 22 35 28 10 70 52 47 65 53 3 43 63 37 32 13 59 23 14 56 18 12 |
| sko-72-03 | 72 | 1054110.5 | 58 7 72 15 5 3 2 49 38 51 41 55 20 71 34 1 4 36 29 44 57 40 45 11 32 66 19 68 24 42 52 69 25 12 37 43 6 53 47 35 16 27 54 33 48 23 64 22 65 63 13 46 50 8 21 62 61 59 70 10 30 26 28 9 67 17 39 18 60 56 14 31 |
| sko-72-04 | 72 | 919586.5 | 12 3 56 64 24 72 45 50 36 41 20 11 65 61 15 4 66 43 62 63 37 58 34 69 25 6 53 18 5 19 39 9 68 67 16 54 48 51 42 40 44 57 55 31 29 38 60 46 23 49 22 14 35 33 71 70 13 10 30 7 27 28 1 47 52 26 21 2 32 59 17 8 |

| Instance | Size | Cost | Permutation |
|---|---|---|---|
| sko-72-05 | 72 | 428226.5 | 51 29 34 40 71 9 44 23 1 7 33 38 55 16 41 58 59 46 65 6 19 15 20 14 49 66 68 61 36 4 57 45 5 69 25 11 24 26 13 43 32 47 52 42 62 63 22 50 37 21 12 2 35 72 17 31 67 64 60 39 48 30 28 54 27 70 10 18 56 3 53 8 |
| sko-81-01 | 81 | 205106.0 | 66 74 3 61 30 25 50 9 48 56 67 65 33 69 23 17 4 6 41 21 81 14 15 19 38 49 31 7 39 36 26 70 35 42 22 55 11 57 78 44 2 45 52 32 68 13 62 1 5 27 58 20 18 73 43 59 24 51 37 63 12 79 75 80 77 71 72 29 64 46 28 10 40 34 76 16 54 53 60 47 8 |
| sko-81-02 | 81 | 521391.5 | 22 55 38 49 25 35 81 14 52 48 56 6 15 61 21 3 69 17 33 66 74 70 41 50 23 67 36 30 65 42 4 11 9 34 26 31 19 44 2 45 12 32 80 29 79 5 58 1 78 51 46 13 68 37 73 59 40 71 75 53 43 24 18 20 39 10 27 63 7 62 54 72 47 16 76 57 64 60 77 28 8 |
| sko-81-03 | 81 | 970796.0 | 47 77 53 16 44 54 10 34 40 76 60 59 63 20 37 51 13 18 64 68 57 21 27 62 24 78 43 7 49 22 45 52 58 69 75 73 1 8 71 32 46 56 4 67 42 23 17 50 65 41 66 81 26 36 80 19 12 33 38 70 3 25 61 2 14 15 31 9 5 6 48 72 28 11 55 29 79 74 39 35 30 |
| sko-81-04 | 81 | 2031803.0 | 66 74 8 25 26 65 69 23 51 3 14 81 20 50 2 30 61 21 12 28 6 44 9 11 19 17 15 70 48 33 46 39 72 40 34 32 49 60 4 67 37 56 78 42 54 10 18 35 36 55 1 45 52 27 62 58 75 64 47 63 73 31 13 68 57 76 77 24 53 38 16 43 59 7 41 5 29 22 79 71 80 |
| sko-81-05 | 81 | 1302711.0 | 66 74 56 48 52 36 6 15 8 9 5 3 55 72 61 68 27 2 45 7 62 39 78 70 57 31 13 16 44 21 17 50 1 46 11 32 81 28 41 14 53 71 49 10 25 20 58 67 24 51 59 77 69 73 33 23 76 37 63 12 18 19 43 60 42 26 35 80 30 65 75 47 38 29 64 22 4 79 54 34 40 |
| sko-100-01 | 100 | 378234.0 | 3 35 44 29 17 21 7 76 53 45 12 36 50 41 48 61 23 49 2 70 82 92 38 72 81 64 90 66 47 51 46 26 100 69 20 40 14 43 99 94 67 85 30 73 24 89 8 68 93 96 56 4 98 42 19 59 9 84 87 86 62 52 22 16 31 34 75 80 54 55 13 27 10 60 28 57 32 63 5 88 77 25 58 74 95 11 6 15 33 91 79 39 83 71 37 1 97 18 78 65 |
| sko-100-02 | 100 | 2076008.5 | 78 91 97 58 27 5 39 24 42 65 10 93 18 73 88 25 32 11 95 60 64 9 85 30 96 62 54 63 71 83 13 75 84 79 22 34 87 74 37 57 6 80 16 89 15 28 31 55 56 4 23 82 1 19 86 44 90 66 47 98 2 21 52 51 49 92 45 67 35 46 20 69 70 100 59 14 43 81 8 68 40 26 7 72 38 94 50 48 76 36 41 77 61 12 53 3 99 17 29 33 |
| sko-100-03 | 100 | 16145598.0 | 44 78 97 35 39 89 24 80 54 21 5 37 71 82 91 31 63 27 13 83 75 11 16 14 9 62 65 73 30 85 99 96 12 93 56 95 10 4 55 74 25 1 18 84 79 57 6 32 77 20 88 34 81 22 15 17 42 98 26 59 23 64 46 7 52 51 67 50 94 29 48 36 38 72 19 86 87 68 66 90 2 43 69 70 41 47 100 61 40 92 53 49 45 76 60 33 8 28 58 3 |
| sko-100-04 | 100 | 3232522.0 | 49 42 39 79 71 25 32 93 97 18 5 94 52 68 8 98 83 9 16 88 22 33 43 21 27 75 80 24 60 67 86 28 31 74 19 89 54 15 1 56 96 65 4 91 85 55 13 11 78 63 57 62 37 77 59 81 61 50 92 48 90 100 38 46 26 82 69 53 35 72 66 70 36 51 10 40 20 30 47 73 14 41 87 34 64 95 44 29 23 12 17 99 2 76 58 45 84 6 3 7 |
| sko-100-05 | 100 | 1033080.5 | 78 90 55 63 25 13 58 6 30 84 60 4 65 96 74 28 85 54 75 10 80 24 27 31 15 95 37 71 97 11 83 39 42 89 16 9 91 18 57 79 5 88 22 43 33 77 1 32 93 56 81 34 73 64 87 14 2 20 67 99 46 41 26 53 94 68 40 36 82 66 49 70 44 8 100 35 21 7 47 98 29 61 59 51 69 19 62 23 12 17 38 92 52 72 86 45 48 50 3 76 |
| Amaral-110-01 | 110 | 144296768.0 | 8 80 53 83 98 6 49 17 54 100 57 48 2 44 52 29 94 106 41 23 18 34 30 107 78 13 70 32 31 12 105 89 81 61 59 38 45 66 68 85 37 27 103 101 104 72 36 82 10 58 87 63 21 60 96 95 20 51 102 47 25 35 43 42 88 99 97 46 90 26 50 91 71 28 5 64 16 74 77 15 56 84 69 1 93 39 24 22 55 79 7 14 92 108 4 33 110 86 3 109 9 11 65 76 67 75 73 62 19 40 |

| Instance | Size | Cost | Permutation |
|---|---|---|---|
| Amaral-110-02 | 110 | 86050112.0 | 63 61 0 12 32 10 103 68 73 24 53 15 4 56 85 95 46 54 33 79 13 57 2 69 100 90 74 47 50 8 45 98 55 25 14 87 38 75 76 83 106 1 29 72 6 92 27 21 41 34 91 107 22 99 70 67 3 89 84 93 51 5 71 77 82 81 59 23 48 9 96 16 18 30 62 52 42 65 44 37 60 28 40 19 26 88 43 104 94 80 78 109 49 35 105 58 108 101 97 36 64 102 17 20 66 86 31 39 11 7 |
| Amaral-110-03 | 110 | 2234743.3 | 83 19 60 50 26 91 37 32 67 44 33 98 7 24 54 97 34 2 62 79 43 87 21 3 74 29 25 106 10 75 99 22 51 55 13 108 49 5 109 30 92 102 4 68 58 35 71 104 14 56 81 77 36 12 84 78 63 31 27 48 69 42 20 18 38 47 66 76 64 46 100 80 1 59 53 11 65 96 52 95 86 28 73 8 40 17 82 0 45 41 88 107 89 103 105 85 15 72 70 57 39 93 94 101 16 6 90 9 23 61 |