INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD • INDIA

Research and Publications

An Exponential Neighborhood Local Search Algorithm for the Single Row Facility Location Problem

Diptesh Ghosh

W.P. No. 2011-08-01
August 2011

The main objective of the Working Paper series of IIMA is to help faculty members, research staff, and doctoral students to speedily share their research findings with professional colleagues and to test out their research findings at the pre-publication stage.

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD – 380015
INDIA

An Exponential Neighborhood Local Search Algorithm for the Single Row Facility Location Problem

Diptesh Ghosh

Abstract

In this work we present a local search algorithm for the single row facility location problem. In contrast to other local search algorithms for the problem, our algorithm uses an exponential neighborhood structure. Our computations indicate that our local search algorithm generates solutions to benchmark instances of the problem whose costs are on average within 2% of costs of optimal solutions within reasonable execution time.

1 Introduction

The single row facility location problem (SRFLP) is defined in the following manner. We are given a set $F = \{1, 2, \ldots, n\}$ of n facilities. Each facility $i \in F$ has a length l_i. We are also given a transmission intensity value c_{ij} for each pair $\{i,\ j\}$ of facilities $i, j \in F$. The objective of the SRFLP is to arrange the facilities in a line, such that the sum of the transmission costs between all pairs of facilities in F is minimized. The transmission cost between a given pair of facilities is the product of the corresponding transmission intensity and the distance between the centroids of the two facilities. Hence a solution to the SRFLP is a permutation of the facilities in F. The size of a SRFLP instance is the cardinality of the set F. The SRFLP finds practical applications in physical arrangements of facilities (Simmons, 1969), in computer systems (Picard and Queyranne, 1981) and in automated guided vehicle systems (Heragu and Kusiak, 1988). The problem is known to be NP-hard (Beghin-Picavet and Hansen, 1982).

Love and Wong (1976) proposed the first integer programming formulation for the SRFLP. Subsequently Heragu and Kusiak (1991) and Amaral (2006) proposed other integer programming formulations for the problem. Solution methods that have been used to obtain optimal solutions to the SRFLP include branch and bound methods (Simmons, 1969, 1971), dynamic programming (Picard and Queyranne, 1981), mixed integer linear programming (Amaral, 2006, 2008; Heragu and Kusiak, 1991; Love and Wong, 1976), and semi-definite programming (Anjos et al., 2005; Anjos and Vannelli, 2008; Anjos and Yen, 2009; Hungerländer and Rendl, 2011). Most of these methods are suitable for SRFLP instances with 40 facilities or less, but are prohibitively expensive for larger problems. Lower bounds for the problem have also been reported in the literature (see, e.g. Anjos et al., 2005; Amaral, 2009; Anjos and Yen, 2009; Hungerländer and Rendl, 2011). Amaral and Letchford (2011) provides a comprehensive polyhedral study of the SRFLP, deriving five exponential sized classes of valid inequalities with conditions under which they become facet-inducing. It also provides a branch and cut algorithm for the SRFLP. Hungerländer and Rendl (2011) presents a thorough computational study of the SRFLP which includes improved lower and upper bounds to the problem.

Since the SRFLP is NP-hard, the literature on the problem also includes heuristic approaches to solve large sized SRFLP instances. Kumar et al. (1995) reports a greedy heuristic for the problem. Interestingly, the heuristic does not take the lengths of the facilities into account during its execution. Among improvement heuristics, the methods reported in the literature include simulated annealing

(Heragu and Alfa, 1992), tabu search (Samarghandi and Eshghi, 2010), genetic algorithm (Datta et al., 2011), scatter search (Kumar et al., 2008), ant colony algorithm (Solimanpur et al., 2005), and particle swarm algorithm (Samarghandi et al., 2010). Among heuristics, the genetic algorithm described in Datta et al. (2011) generates the best solutions for benchmark SRFLP instances.

In this work, we propose an exponential neighborhood structure for the SRFLP. We also provide a heuristic to search this neighborhood to obtain good quality solutions. We use this neighborhood structure in a simple local search algorithm and apply the algorithm on large sized benchmark instances of the SRFLP with sizes ranging from 60 to 100. The remainder of the paper is organized as follows. In Section 2 we describe the local search algorithm that we present in this work, and in Section 3 we present the results of our computational experiments with this algorithm. We conclude the paper in Section 4 with a summary of our contribution and directions for future work.

2 The Local Search Algorithm

A generic local search algorithm starts with an initial solution as its input, and then proceeds to iteratively improve the solution by searching among solutions genereated using a pre-defined set of operations (called moves) until it reaches a stage in which it cannot improve the solution further. It then outputs the improved solution and terminates. The set of moves used by local search defines a neighborhood structure in the space of all feasible solutions. The performance of a local search algorithm critically depends on the neighborhood it searches, and the initial solution which it takes as input. The neighborhood which is commonly used in the literature to solve SRFLP instances is the 2-opt neighborhood (see, e.g., Samarghandi and Eshghi, 2010). In this neighborhood, a move is an operation which swaps the positions of two facilities in a given solution. All $n(n+1)/2$ solutions that are generated from a given solution by swapping the positions of two facilities form the neighborhood of the given solution.

In our algorithm we propose an exponential neighborhood called the k-FIX neighborhood, where k is pre-specified. The set of solutions which are neighbors to a solution, say Π, in this neighborhood is generated using a two step process. In the first step, we create a list of all $n!/(n-k)!$ subsets F_s of F with cardinality k. In the second step, for each of the subsets F_s that we created in the first step, we generate all solutions in which the positions of the k elements in F_s relative to each other are the same as that in Π. Notice that the sets of solutions obtained in the second step from any two different subets obtained in the first step have solutions in common. All the solutions that are generated in the second step form the neighborhood of the solution Π.

Clearly, the size of the k-FIX neighborhood is exponential in the size of the instance. Hence in any reasonable sized SRFLP instance, it is impractical to evaluate all neighbors in order to identify the best solution in the neighborhood . In this paper, for every subset F_s that we generate in the first step, we use a greedy algorithm, which we call GREEDY(Π, F_s) to find a good quality solution among those generated from F_s in the second step. GREEDY(Π, F_s) is different from the algorithm proposed in Kumar et al. (1995) in that it takes the lengths of the facilities into account. We then choose the best among the $n!/(n-k)!$ solutions thus generated as an approximation of the best solution in the neighborhood of Π. We describe GREEDY(Π, F_s) below. In the description we use the term "partial solution" to denote a permutation of facilities in a subset of F.

Let us suppose without loss of generality that we want a good quality neighboring solution to the solution $\Pi = (\pi_1, \pi_2, \ldots, \pi_n)$ using a subset $F_s = \{\pi_1, \pi_2, \ldots, \pi_k\}$ generated in the first step. The greedy algorithm starts with a partial solution in which the relative positions of π_1 through π_k are unchanged, i.e., it starts with the partial solution $(\pi_1, \pi_2, \ldots, \pi_k)$. For each of the elements π_j, $j = k+1, \ldots, n$, the algorithm computes a weight $w_j = \sum_{i=1}^{n} c_{ij}$ and orders the facilities in non-increasing order of these weights. It then performs $n-k$ iterations. At the beginning of the r-th iteration, the partial solution has $k+(r-1)$ facilities. During the iteration, the algorithm chooses

the r-th facility in the ordering and inserts it in the best of all possible $k+r$ positions in the partial solution. Thus at the end of the iteration, the cardinality of the partial solution increases by 1. The greedy algorithm ends when it has inserted all the $n-k$ facilities in $F \setminus F_s$.

The choice of initial solutions is also important for a local search algorithm. In our local search algorithm we use the GREEDY$(\cdot, \emptyset)$ to obtain an initial solution.

Given these details, Algorithm LOCALSEARCH formally describes the local search algorithm that we use to solve SRFLP instances. In the description we use $f(\Pi)$ to denote the cost of the solution Π.

ALGORITHM LOCALSEARCH

Input: A SRFLP instance with a set F of n; an integer k.

Output: A solution to the instance, i.e., a permutation Π of facilities in F.

Steps:

Step 1: Set $\Pi \longleftarrow$ GREEDY$(\cdot, \emptyset)$, $c \longleftarrow f(\Pi)$.

Step 2: Choose $S_{r\star} \subseteq F$ with $|F_{r\star}| = k$, for which

$$f_{r\star} = \text{GREEDY}(\Pi, F_{r\star}) \leq \min_{F_r \subseteq F, |F_r|=k} \{\text{GREEDY}(\Pi, F_r)\}.$$

Step 3: If $f_{r\star} < f(\Pi)$, set $\Pi \longleftarrow$ GREEDY$(\Pi, F_{r\star})$ and go to Step 2. Else output Π and terminate.

---

3 Computational Experience

We coded the LOCALSEARCH algorithm presented in Section 2 in C and executed it on a Linux machine with a 2.2GHz processor and 4GB RAM. This computer has speed characteristics similar to the one used in Hungerländer and Rendl (2011)[1]. We chose a 3-FIX neighborhood for our experiments, since larger values of k in a k-FIX neighborhood require very large amounts of execution time. Since most of the heuristics generated optimal solutions when the problem sizes were small, we used problem instances with sizes ranging from 60 to 100 for our computational experiments. The instances were Anjos instances which have regularly been used in computational experiments for the SRFLP and the `sko` instances which have been used in Hungerländer and Rendl (2011).

Tables 1 and 2 presents the results of our computational experiments on the Anjos instances and the `sko` instances respectively. The results are compared with the results presented in Hungerländer and Rendl (2011).

Notice that the objective value of optimal solutions to the problems that we experimented with are not known, but lower bounds to these values are available (Hungerländer and Rendl, 2011). If we define the suboptimality value for solutions obtained by a heuristic as the excess of the ratio of objective value of the solution output by the heuristic to the objective value of an optimal solution over unity, then the solutions obtained by LOCALSEARCH have suboptimality values of at most 0.68% on average for the Anjos instances, and of at most 2.2% on average for the `sko` instances. The suboptimality values of the upper bounds reported in Hungerländer and Rendl (2011) are better. However, a fair comparison will take the execution times into account; LOCALSEARCH requires 1.83% of the execution time required by the algorithm in Hungerländer and Rendl (2011) on average for the Anjos instances and 2.24% of the execution time on average for the `sko` instances. Given

[1] see `http://www.cpubenchmark.net/midlow_range_cpus.html` for exact speed comparison.

Table 1: Results from LOCALSEARCH on the Anjos benchmark problem instances

		Hungerländer and Rendl (2011)			LOCALSEARCH	
Instance	Size	Lower Bound	Upper Bound	Time	Objective	Time
Anjos_60_01	60	1477295.0	1477834.0	103169	1479941.0	682
Anjos_60_02	60	841559.0	841776.0	111126	844892.0	687
Anjos_60_03	60	647283.5	648337.5	85021	650563.5	680
Anjos_60_04	60	398095.0	398406.0	95439	401202.0	678
Anjos_60_05	60	318801.0	318805.0	99106	320381.0	684
Anjos_70_01	70	1526359.0	1528560.0	96094	1536600.0	1959
Anjos_70_02	70	1439122.0	1441028.0	94287	1446222.0	1957
Anjos_70_03	70	1517803.5	1518993.5	94514	1531658.5	1976
Anjos_70_04	70	967316.0	969150.0	98928	976197.0	1979
Anjos_70_05	70	4213774.5	4218002.5	101765	4236723.5	1977
Anjos_75_01	75	2387590.5	2393600.5	136673	2406772.5	3151
Anjos_75_02	75	4309185.0	4322492.0	142118	4329962.0	3163
Anjos_75_03	75	1243136.0	1249251.0	138066	1253995.0	3189
Anjos_75_04	75	3936460.5	3941845.5	139378	3953651.5	3124
Anjos_75_05	75	1786154.0	1791469.0	148237	1805235.0	3189
Anjos_80_01	80	2063346.5	2070391.5	210289	2083649.5	4968
Anjos_80_02	80	1918945.0	1921202.0	211635	1933450.0	4894
Anjos_80_03	80	3245254.0	3251413.0	209839	3263348.0	4994
Anjos_80_04	80	3739657.0	3747829.0	211847	3763970.0	4984
Anjos_80_05	80	1585491.0	1590847.0	210630	1602981.0	4964

Table 2: Results from LOCALSEARCH on the `sko` benchmark problem instances

		Hungerländer and Rendl (2011)			LOCALSEARCH	
Instance	Size	Lower Bound	Upper Bound	Time	Objective	Time
sko64_01	64	96607.0	97194.0	76179	98707.0	1070
sko64_02	64	633694.5	634332.5	115080	641910.5	1063
sko64_03	64	413079.5	414384.5	126675	421714.5	1070
sko64_04	64	295423.0	298155.0	105809	301558.0	1070
sko64_05	64	501342.5	502063.5	119158	511693.5	1068
sko72_01	72	138885.0	139231.0	106399	141812.0	2383
sko72_02	72	707643.0	715611.0	106841	723070.0	2400
sko72_03	72	1048930.5	1061762.5	117527	1068710.5	2394
sko72_04	72	916229.5	924019.5	122308	933273.5	2293
sko72_05	72	426224.5	430288.5	113983	435754.5	2382
sko81_01	81	203424.0	207063.0	189850	208538.0	5434
sko81_02	81	518711.5	526157.5	215888	529954.5	5416
sko81_03	81	962886.0	979281.0	209860	981498.0	5358
sko81_04	81	2019058.0	2035569.0	206509	2066083.0	5357
sko81_05	81	1293905.0	1311166.0	212368	1321658.0	5421
sko100_01	100	375999.0	380562.0	690441	387863.0	23585
sko100_02	100	2056997.5	2084924.5	726412	2106591.5	23547
sko100_03	100	15987840.5	16216076.5	765534	16334235.0	23601
sko100_04	100	3200643.0	3263493.0	735279	3281817.0	23576
sko100_05	100	1021584.5	1040929.5	725367	1049581.5	23631

that the execution times required by the algorithm in Hungerländer and Rendl (2011) are prohibitive (e.g., for `sko` instances with 100 facilities the algorithm in Hungerländer and Rendl (2011) requires approximately 202.38 hours on average, while LOCALSEARCH requires 6.55 hours on average) and also given that LOCALSEARCH is a much easier algorithm to implement, LOCALSEARCH seems to be a more practical algorithm to obtain near-optimal solutions to large SRFLP instances.

4 Summary and Future Research Directions

In this work we present a local search algorithm for the single row facility location problem (SRFLP). Our algorithm uses a k-FIX neighborhood that is exponential in the size of the problem. To the best of our knowledge, this is the first work that uses exponential neighborhoods in an algorithm to solve the SRFLP; all other algorithms are based on local search use a 2-opt neighborhood in which the positions of two of the facilities are interchanged to generate a neighbor. Our computational experiments show that the local search algorithm that we propose using a 3-FIX neighborhood can generate solutions within 2 to 3% of the optimum in reasonable time.

There are several directions in which the work presented here can be extended. We provide three immediate extensions here. First, note that the algorithm we present here is a simple local search algorithm which terminates at the first local optimum that it encounters. It is easy to extend such an algorithm into a more sophisticated algorithm like tabu search, which is likely to generate better quality solutions, although the execution time that it will require will be more. Secondly, the search through the k-FIX neighborhood of a solution can be made more efficient through careful book-keeping. Third, it will be interesting to use different starting solutions and/or implement LOCALSEARCH in a multistart mode to see if its performance can be improved significantly.

References

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

Amaral, A.R.S., 2008. An exact approach for the one-dimensional facility layout problem. Operations Research 56, 1026–1033.

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

Amaral, A.R.S., Letchford, A.N., 2011. A Polyhedral Approach to the Single Row Facility Layout Problem. Available at `http://www.optimization-online.org/DB_FILE/2008/03/1931.pdf`.

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

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

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

Beghin-Picavet, M., Hansen, P., 1982. Deux problemes d?affectation non linaires. RAIRO, Recherche Oprationelle 16, 263–276.

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

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

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

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

Hungerländer, P., Rendl, F., 2011. A Computational Study for the Single-Row Facility Layout Problem. Available at `www.optimization-online.org/DB_FILE/2011/05/3029.pdf`.

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

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

Love, R.F., Wong, J.Y., 1976. On solving a single row space allocation problem with integer programming. INFOR 14, 139–143.

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

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

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

Simmons, D.M., 1969. Single row space allocation: An ordering algorithm. Operations Research 17, 812–826.

Simmons, D.M., 1971. A further note on one-dimensional space allocation. Operations Research 17, 249.

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