Iterated Greedy Algorithms for the Maximal Covering Location Problem

Francisco J. Rodriguez[1], Christian Blum[2],
Manuel Lozano[1], and Carlos García-Martínez[3]

[1] Department of Computer Science and Artificial Intelligence,
University of Granada, Granada, Spain
[2] ALBCOM Research Group, Technical University of Catalonia, Barcelona, Spain
[3] Department of Computing and Numerical Analysis,
University of Córdoba, Córdoba, Spain
fjrodriguez@decsai.ugr.es, cblum@lsi.upc.edu,
lozano@decsai.ugr.es, cgarcia@uco.es

Abstract. The problem of allocating a set of facilities in order to maximise the sum of the demands of the covered clients is known as the maximal covering location problem. In this work we tackle this problem by means of iterated greedy algorithms. These algorithms iteratively refine a solution by partial destruction and reconstruction, using a greedy constructive procedure. Iterated greedy algorithms have been applied successfully to solve a considerable number of problems. With the aim of providing additional results and insights along this line of research, this paper proposes two new iterated greedy algorithms that incorporate two innovative components: a population of solutions optimised in parallel by the iterated greedy algorithm, and an improvement procedure that explores a large neighbourhood by means of an exact solver. The benefits of the proposal in comparison to a recently proposed decomposition heuristic and a standalone exact solver are experimentally shown.

Keywords: iterated greedy algorithm, large neighbourhood search, maximal covering location problem.

1 Introduction

The *maximal covering location problem* MCLP [4] considers a predefined number of facilities that have to be allocated such that the demand of the clients covered by these facilities—given a maximum service distance—is maximal. This problem has several real-world applications in different fields, including the planning of service locations such us health-care centres, fire stations, and emergency centres.

More precisely, let M be the set of m potential facility locations and N the set of n clients to be covered. $D(i, j)$ denotes the distance between each pair of nodes $i \in N$ and $j \in M$, U is the maximum service distance, and w_i is the demand of client i. Note that a client i is covered by a facility installed at location $j \in M$ iff $D(i, j) \leq U$. The objective is to maximise the sum of the demands of all the

J.-K. Hao and M. Middendorf (Eds.): EvoCOP 2012, LNCS 7245, pp. 172–181, 2012.

clients covered by any of the p installed facilities. The MCLP may be formulated as the following zero-one integer programming problem [4]:

$$\textbf{max } z = \sum_{i=1}^{n} w_i \cdot x_i \tag{1}$$

$$\textbf{subject to: } \sum_{j \in S_i} y_j \geq x_i \quad \text{for } i \in N \tag{2}$$

$$\sum_{j \in M} y_j = p \tag{3}$$

$$x_i \in \{0,1\} \quad \text{for } i \in N \tag{4}$$

$$y_j \in \{0,1\} \quad \text{for } j \in M \tag{5}$$

Hereby, x_i is a binary variable indicating whether client i is covered by a facility, y_j is a binary variable that attests whether location j has been chosen to install a facility, and S_i is the set composed by all potential facility locations that cover client i, that is, $S_i = \{j \in M : D(i,j) \leq U\}$.

1.1 Previous Work

The MCLP is an NP-hard problem [9] that has received quite some attention since it was presented, having resulted in a variety of proposals for tackling the problem. The latter include exact algorithms for relaxations of the problem [4,6,7,8], greedy heuristics [4], and several metaheuristics such as genetic algorithms [2,14], tabu search [14], and simulated annealing [14]). Recently, Senne et al. [13] presented a decomposition heuristic to perform a cluster partitioning, resulting in smaller subproblems (clusters) that can be solved independently by exact methods (LagClus). The results obtained by this approach are compared in terms of quality and computational time required with regards to those of a commercial solver (CPLEX [1]), indicating that the proposed decomposition approach can substantially reduce the time for providing good-enough solutions to large problem instances.

1.2 Our Contribution

In this work, we propose two iterated greedy (IG) algorithms [5,11] for solving the MCLP. IG algorithms, generally, try to iteratively refine a solution by removing elements from this solution by means of a destructive procedure and reconstructing the resulting partial solution using a greedy constructive procedure. The first one of the proposed IG variants extends the basic IG idea by considering a population of solutions that are improved in parallel by means of the standard IG procedure. The resulting algorithm is labelled *population-based iterated greedy* (PBIG). Second, we propose a hybrid algorithm that combines PBIG with an exact solver. In particular, CPLEX is employed for this purpose. The idea consists in completing the solutions provided by the destruction procedure of IG

by means of the application of CPLEX. The latter only optimises a predefined number of components of each solution, while the remaining components of the solution are fixed to the values of the partial solutions received as input. This strategy is known as *large neighbourhood search* (LNS) [3,12]. The basic idea is to combine the advantages of a large neighbourhood, which usually enhances the exploration of a local search method, with an exhaustive tree-search exploration which is faster than enumeration. Our second IG approach is labelled *population based iterated greedy with large neighbourhood search* (PBIG+LNS).

1.3 Paper Organization

The remainder of this paper is organized as follows. In Section 2, we present in detail the two proposed IG variants for the MCLP. In Section 3, we present a empirical study that compares the behaviour of the two proposed IG algorithms with regards to those of the most recent proposal from the literature, LagClus, and a standalone CPLEX procedure. Finally, in Section 4, we discuss conclusions and further work.

2 Proposed IG Variants for the MLCP

In this section, we describe the two proposed IG variants for the MLCP. First, let us focus on the PBIG scheme. It extends IG by working on a population of solutions which is managed in the style of evolution strategies. The resulting algorithm is outlined in Figure 1. It starts by initialising the population P with t solutions generated by a probabilistic greedy constructive procedure (as outlined below). Inside the main loop, each solution $s \in P$ is optimised by means of a destruction/re-construction procedure, generating a new population P_n of solutions. The destruction step consists in randomly removing n_d elements from the considered solution s, resulting in a partial solution s_d. This solution is reconstructed by means of the same probabilistic greedy constructive procedure that was used to generate the initial population. This step results in a (possibly new) complete solution s_c which is then added to P_n. After applying this process to all solutions from P, the new population P_n is added to P, resulting in a new set P of size $2 \cdot t$. The last step of each iteration consists in choosing the best t solutions from P for the population of the next iteration. The proposed algorithm iterates through these phases until a computation limit t_{max} is reached.

2.1 The Probabilistic Greedy Procedure

The probabilistic greedy constructive procedure used for the initialisation of the population and the re-construction of partial solutions works as follows. At each step, it considers placing a new facility in any of the locations that is not yet occupied by an already installed facility. For each of these options, it calculates the contribution to the objective function value, that is, the increase in the function value caused by the respective option. The two options which cause the

```
   Input: t_max, t, n_d, prob
   Output: s
 1 P ←GenerateInitialPopulation(t);
 2 while computation time limit t_max not reached do
 3 |   P_n ← ∅ ;
 4 |   foreach s ∈ P do
 5 |   |   s_d ← Destruction(s, n_d);
 6 |   |   if PBIG() then s_c ← Construction(s_d, prob) // PBIG ;
 7 |   |   else s_c ← LNS(s_d) // PBIG+LNS ;
 8 |   |   P_n ← P_n ∪ {s_c};
 9 |   end
10 |   P ← P ∪ P_n;
11 |   P ← SelectBestSolutions(P, t) ;
12 end
```

Fig. 1. PBIG and PBIG+LNS scheme

highest increase are identified. Finally, the best option is chosen with probability *prob*, which is an input parameter of the algorithm. Otherwise the second-best option is chosen. The procedure stops once p facilities are installed.

2.2 PBIG+LNS

As mentioned already in the introduction, PBIG+LNS modifies PBIG by replacing the constructive step with a large neighborhood search method applied to the solutions generated by the destructive step of PBIG (see line 7 of the algorithm from Figure 1). This procedure is performed by an exact solver (CPLEX), whereby the size of the neighbourhood is determined by fixing the components provided by the current partial solution s_d. In particular, a binary variable y_j (see the definition of the MCLP) is fixed to 1 if location j is selected as a facility in s_d. In the same way, a variable x_i is fixed to 1 if client i is covered by any of the fixed facility locations. This means that CPLEX will try to find an allocation for the n_d unallocated facilities. In this way, CPLEX—which is already very efficient for problem instances with a small number of clients and facilities [13]—can be used as a sub-ordinate procedure for tackling large-size problem instances. The complete pseudocode of the LNS method is shown in Figure 2.

3 Computational Experiments

This section describes the computational experiments performed to assess the performance of the two IG algorithms presented in the previous section. Both PBIG and PBIG+LNS were coded in Java and the tests were conducted on a computer with a 3.2 GHz Intel i7 processor with 12 GB of RAM running Fedora Linux V15.

```
   Input: s_c
   Output: s_c
1  Y_fixed ← ∅ ;
2  X_fixed ← ∅ ;
3  foreach j ∈ M do
4  |  if IsSelectedAsFacility(j, s_d) then Y_fixed ← Y_fixed ∪ {y_j} end;
5  end
6  foreach i ∈ N do
7  |  if IsCovered(i, s_d) then X_fixed ← X_fixed ∪ {x_i} end;
8  end
9  s_c = CPLEX(Y_fixed, X_fixed) // Neighbourhood restricted to the set of
   free binary variables x_i and y_j ;
```

Fig. 2. Procedure LNS() of PBIG+LNS

3.1 Problem Instances

We have employed two different sets of problem instances:

1. Real case instances for facility location problems in Sao Jose dos Campos, Brazil (SCJ instances). These instances are available for download at `http://www.lac.inpe.br/lorena/instancias.html`.
2. An instance which was created on the basis of instance PCB3038 available from the TSPLIB [10].

3.2 Tuning Experiments

In order to perform a fine-tuning of the two proposed IG algorithms we first conducted tuning experiments in order to find values for the following algorithm parameters:

1. **Population size** t**:** values from $\{1, 2, 10, 10, 50, 100\}$ were considered.
2. **Destruction size** n_d**:** For the percentage of elements dropped from the current solution during the destructive step values from $\{5\%, 10\%, 20\%, 50\%\}$ were considered.
3. **Degree of determinism for the solution construction** *prob***:** for the probability of accepting the option with the best contribution during the greedy constructive procedure values from $\{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$) were considered. Note that $prob = 1.0$ corresponds to a completely deterministic solution (re-)construction.

For each combination of values for the three parameters (full factorial design), we applied both PBIG (and PBIG+LNS) to a subset of the real-case instances. The computation time limits were chosen dependent on the number of clients of the instance (50 seconds for instances with 324, respectively 500, clients and 100 seconds for those with 818 clients). A rank-based analysis was applied to the results. The parameter combination with the best average rank over all

Table 1. Parameters values

Parameter	PBIG	PBIG+LNS
Population size (t)	100	20
Elements dropped (n_d)	20%	50%
Degree of determinism ($prob$)	0.8	0.8

testing instances is shown in Table 1. Although PBIG and PBIG+LNS share core funcionality, we have performed their parameter analysis separately. The tuning outcome shows that, interestingly, the two algorithm versions reach their best performance with quite different values for two out of the three parameters.

3.3 Experimental Results

In the following we describe the outcome of a comparative analysis between the results of PBIG and PBIG+LNS and those of LagClus, which is the currently best heuristic method, and the standalone CPLEX procedure. The reported results for LagClus and CPLEX are the ones provided in [13]. They were obtained on a computer with an Intel Core 2 Duo 2.0 GHz processor and 2.0 GB RAM, running Windows XP. Tables 2 and 3 show the results for the SCJ instances, considering a service distance of 150 and 200, respectively. For each instance—which is determined by a number of clients, n, and a number of facilities, p—we provide the optimal solution obtained by CPLEX and the $gap = 100 \cdot ((Result - Optimal)/Optimal)$ for each algorithm. Moreover, for each algorithm we show the computational time needed to obtain the corresponding result. The results reported for PBIG and PBIG+LNS correspond to the average over 10 independent applications to each problem instance. Concerning the computation time limits, we used the same ones as for the tuning procedure.

The results of the considered algorithms as shown in Tables 2 and 3 allow us to make the following observations:

- Comparing the results of PBIG and PBIG+LNS, we can observe that PBIG+LNS clearly outperforms PBIG. In addition, analysing the computational time used by the two algorithms, we can observe that PBIG+LNS reduces the computational time requirements with respect to PBIG, especially when larger problem instances are concerned. This indicates that the hybridisation with LNS seems to be a decisive element to improve not only the quality of the results but also for the reduction of the computation time requirements.
- Concerning the comparison of PBIG+LNS with LagClus, we can observe that PBIG+LNS exceeds or equals the results of LagClus for all instances with $U = 150$ and—with one exception—also for all instances with $U = 200$. It is noticeable that PBIG+LNS is able to obtain optimal solutions in all 10 runs in 37 out of 46 cases, which is indicated by an average gap of 0.000. This fact shows that—in addition to reaching high quality solutions—PBIG+LNS is characterized by a very stable behaviour. Concerning computation time,

Table 2. Results for the SJC instances, U=150

n	p	Optimal	LagClus	PBIG	PBIG+LNS
		Result (time s)	*Gap (time s)*	*Gap (time s)*	*Gap (time s)*
324	20	7302 (0.015)	**0.000** (2.543)	**0.000** (0.391)	**0.000** (0.381)
	30	9127 (0.047)	0.027 (24.650)	0.072 (9.633)	**0.000** (5,082)
	40	10443 (0.188)	0.108 (25.985)	**0.000** (26.553)	**0.000** (10.320)
	50	11397 (0.391)	0.138 (24.452)	0.005 (20.313)	**0.000** (11.164)
	60	11991 (0.235)	0.024 (44.514)	0.097 (26.896)	**0.000** (10.576)
	80	12152 (0.031)	**0.000** (8.876)	**0.000** (9.181)	**0.000** (3.732)
	108	12152 (0.016)	**0.000** (1.595)	**0.000** (0.761)	**0.000** (0.672)
500	40	13340 (0.047)	**0.000** (3.453)	0.247 (4.560)	**0.000** (3.857)
	50	14773 (0.047)	**0.000** (4.938)	0.093 (22.381)	**0.000** (6.529)
	60	15919 (0.063)	**0.000** (8.233)	0.092 (31.052)	**0.000** (12.073)
	70	16908 (0.031)	**0.000** (3.723)	0.002 (41.019)	**0.000** (8.339)
	80	17749 (0.015)	**0.000** (5.406)	0.038 (44.786)	**0.000** (10.938)
	100	18912 (0.109)	**0.000** (10.276)	0.310 (46.737)	**0.000** (24.574)
	130	19664 (0.297)	0.015 (30.827)	0.230 (47.735)	**0.000** (30.642)
	167	19707 (0.047)	0.003 (14.600)	**0.000** (1.057)	**0.000** (0.931)
818	80	23325 (0.140)	0.003 (45.564)	0.293 (94.376)	**0.000** (28.637)
	90	24455 (0.266)	0.041 (56.388)	0.348 (91.452)	**0.000** (37.726)
	100	25435 (0.344)	0.012 (87.279)	0.306 (94.742)	**0.000** (40.067)
	120	26982 (0.297)	0.015 (69.658)	0.446 (93.713)	**0.000** (33.781)
	140	28802 (0.359)	0.095 (52.966)	0.597 (94.131)	**0.002** (44.736)
	160	28699 (0.391)	0.107 (58.453)	0.612 (93.477)	**0.004** (74.202)
	200	29153 (0.234)	0.011 (61.531)	0.096 (92.225)	**0.000** (40.068)
	273	29168 (0.031)	**0.000** (3.343)	**0.000** (1.439)	**0.000** (1.271)

it seems that the requirements of LagClus and PBIG+LNS are of the same order of magnitude. All in all, this indicates that PBIG+LNS is a new state-of-the-art method for what concerns heuristics for the MCLP.

– For what concerns the comparison to CPLEX, we must observe that CPLEX is able to solve all problem instances in very little computation time. Therefore, PBIG+LNS must be considered inferior to CPLEX for the SJC instances.

Finally, in Table 4 we show the results of all analysed algorithms for the set of instances derived from the TSPLIB instance PCB3038. PBIG and PBIG+LNS consider a computation time limit of 1500 seconds for each run. It is important to highlight that the number of clients considered in this case is much higher than in the case of the SCJ instances, which imposes a more complicated environment for the analysed algorithms. In fact, as shown in Table 4, CPLEX is unable to confirm the optimality of solutions within the predefined time limit of 20000 seconds for instances with more than 18 facilities. These cases are marked by an asterisk. Therefore, large-size instances are the ones for which PBIG+LNS is an interesting alternative. Concerning the results of Table 4, we can conclude the following:

Table 3. Results for the SJC instances, U=200

n	p	Optimal	LagClus	PBIG	PBIG+LNS
		Result (time s)	*Gap (time s)*	*Gap (time s)*	*Gap (time s)*
324	20	9670 (0.172)	0.243 (19.293)	0.307 (9.633)	**0.000** (11.064)
	30	11737 (0.484)	0.060 (28.943)	0.098 (25.676)	**0.000** (12.580)
	40	12151 (0.094)	0.008 (31.066)	0.023 (35.910)	**0.004** (19.813)
	50	12152 (0.015)	**0.000** (9.926)	**0.000** (0.494)	**0.000** (0.469)
	60	12152 (0.047)	**0.000** (4.575)	**0.000** (0.620)	**0.000** (0.506)
	80	12152 (0.016)	**0.000** (3.670)	**0.000** (0.637)	**0.000** (0.663)
	108	12152 (0.031)	0.248 (11.343)	**0.000** (0.643)	**0.000** (0.763)
500	40	17077 (0.233)	0.387 (24.668)	0.255 (45.223)	**0.012** (25.679)
	50	18361 (0.109)	0.003 (39.109)	0.326 (43.199)	**0.000** (24.485)
	60	19153 (0.063)	**0.005** (52.639)	0.352 (47.504)	0.017 (31.036)
	70	19551 (1.078)	0.069 (43.946)	0.397 (46.511)	**0.006** (30.314)
	80	19703 (0.156)	0.008 (35.495)	0.124 (45.296)	**0.001** (22.716)
	100	19707 (0.078)	**0.000** (16.624)	**0.000** (1.050)	**0.000** (0.832)
	130	19707 (0.047)	**0.000** (1.986)	**0.000** (1.025)	**0.000** (1.017)
	167	19707 (0.016)	0.016 (22.379)	**0.000** (0.954)	**0.000** (1.026)
818	80	27945 (0.203)	0.069 (57.835)	0.858 (94.841)	**0.000** (51.136)
	90	28519 (1.141)	0.071 (114.145)	0.828 (95.977)	**0.013** (75.486)
	100	28910 (1.391)	0.036 (88.885)	0.801 (91.365)	**0.000** (59.345)
	120	29165 (1.234)	0.002 (55.710)	0.185 (90.345)	**0.001** (56.270)
	140	29168 (0.125)	**0.000** (11.643)	**0.000** (79.766)	**0.000** (10.313)
	160	29168 (0.062)	**0.000** (9.738)	**0.000** (1.087)	**0.000** (1.087)
	200	29168 (0.032)	**0.000** (5.762)	**0.000** (1.288)	**0.000** (1.276)
	273	29168 (0.031)	0.207 (24.689)	**0.000** (1.491)	**0.000** (1.513)

Table 4. Results for the TSPLIB instance PCB3038, U=400

n	p	Optimal	LagClus	PBIG	PBIG+LNS
		Result (time s)	*Gap (time s)*	*Gap (time s)*	*Gap (time s)*
3038	17	125320 (802)	0.205 (844)	0.992 (354)	**0.125** (357)
	18	130004 (10265)	0.372 (817)	0.481 (384)	**0.236** (556)
	19	134262* (20000)	0.382 (1483)	0.552 (573)	**0.265** (602)
	20	138028* (20000)	0.698 (1712)	0.165 (941)	**0.033** (518)
	21	141279* (20000)	0.024 (3117)	0.085 (913)	**0.000** (583)
	22	143809* (20000)	0.024 (6656)	7.155 (573)	**0.010** (602)

- PBIG+LNS is able to significantly improve over the gap of LagClus for all considered instances, while requiring even less computation time.
- PBIG+LNS generates solutions close to the ones of CPLEX, while reducing substantially—especially for instances with more than 17 facilities—the time needed to obtain high-quality solutions.

4 Conclusions and Future Work

In this paper, we have proposed two IG algorithms for the maximum covering location problem. The proposed algorithms add two novel components to the

basic IG technique. In the first place, PBIG incorporates a population of solutions evolved in parallel by means of the classic destruction/re-construction procedure of IG algorithms. In the second place, PBIG+LNS extends PBIG by incorporating an exact solver to complete the solutions generated by the destructive procedure of IG, following the ideas of large neighborhood search. The resulting hybrid algorithm, PBIG+LNS, has proved to be superior to a recently proposed decomposition heuristic. Moreover, in the case of large-scale instances, where the computation time requirements of CPLEX explode, PBIG+LNS arises as a tool of choice to face this kind of problems.

We believe that the IG frameworks presented in this paper are an interesting contribution, worthy of further study. We will mainly focus on the following avenues of possible research: (1) study of the behaviour of the proposed PBIG+LNS for what concerns new instances of the problem and (2) adapting the PBIG+LNS approach for its application to other challenging optimisation problems, especially when large-size instances are concerned.

Acknowledgements. This work was supported by grants TIN2007-66523 and TIN2008-05854 of the Spanish government and by grant P08-TIC-4173 of the Andalusian regional goverment. Moreover, Christian Blum acknowledges support from the *Ramón y Cajal* program of the Spanish Ministry of Science and Innovation.

References

1. IBM ILOG CPLEX optimizer (November 2011), `http://www-01.ibm.com/software/integration/optimization/cplexoptimizer/`
2. Arakaki, R.G.I., Lorena, L.A.N.: A constructive genetic algorithm for the maximal covering location problem. In: Proceedings of the 4th Metaheuristics International Conference (MIC 2001), pp. 13–17 (2001)
3. Blum, C., Puchinger, J., Raidl, G.R., Roli, A.: Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing 11(6), 4135–4151 (2011)
4. Church, R., Velle, C.R.: The maximal covering location problem. Papers in Regional Science 32(1), 101–118 (1974)
5. Culberson, J.C., Luo, F.: Exploring the k-colorable landscape with iterated greedy. Dimacs Series in Discrete Mathematics and Theoretical Computer Science, pp. 245–284. American Mathematical Society (1996)
6. Galvao, R.D., Espejo, L.G.A., Boffey, B.: A comparison of lagrangean and surrogate relaxations for the maximal covering location problem. European Journal of Operational Research 124(2), 377–389 (2000)
7. Galvao, R.D., ReVelle, C.: A lagrangean heuristic for the maximal covering location problem. European Journal of Operational Research 88(1), 114–123 (1996)
8. Lorena, L.A., Pereira, M.A.: A lagrangean/surrogate heuristic for the maximal covering location problem using hillsman's edition. International Journal of Industrial Engineering 9, 57–67 (2001)
9. Megiddo, N., Zemel, E., Hakimi, S.L.: The maximum coverage location problem. SIAM Journal on Algebraic and Discrete Methods 4(2), 253–261 (1983)

10. Reinelt, G.: The traveling salesman: computational solutions for TSP applications. Springer, Heidelberg (1994)
11. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177(3), 2033–2049 (2007)
12. Shaw, P.: Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: Maher, M., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)
13. Senne, E.L.F., Pereira, M.A., Lorena, L.A.N.: A decomposition heuristic for the maximal covering location problem. Advances in Operations Research 2010 (2010)
14. Xia, L., Xie, M., Xu, W., Shao, J., Yin, W., Dong, J.: An empirical comparison of five efficient heuristics for maximal covering location problems. In: IEEE/INFORMS International Conference on Service Operations, Logistics and Informatics (SOLI 2009), pp. 747–753 (2009)