# A Genetic Algorithm for the Capacity and Distance Constrained Plant Location Problem[1]

Matthias Lanzinger, Benjamin Biesinger, Bin Hu, Günther Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
matthias.lanzinger@tuwien.ac.at
{biesinger|hu|raidl}@ac.tuwien.ac.at

**Abstract**

In this work we consider the capacity and distance constrained plant location problem (CDCPLP). This problem is a type of location and routing problem, in which a set of customers has to be assigned cost efficiently to plants that can be opened at discrete locations. Customers are then served from the associated plants by round-trips of plant's vehicles. Since the capacity of the plant and the total maximum driving distance is limited the CDCPLP is a hard combinatorial optimization problem. We propose a genetic algorithm with an embedded local search based on an incomplete solution representation. Computational results show that the presented algorithm is able to quickly find optimal or near-optimal solutions and to outperform a previous state-of-the-art tabu search on many instances.

## 1 Introduction

Many different variants of location problems have been considered in the literature so far. This is due to the fact that choosing appropriate locations for opening facilities or plants is one of the most important decisions for logistic companies. In this work we consider one variant of such a problem which does not only include the choice of locations but also the assignment of customers to them. These customers are served by round trips from the opened plants with limited capacity, and are therefore assigned to vehicles located at individual plants. The vehicles can make several trips a day but the number of assigned customers is limited by the number of working hours for the drivers. This problem is called capacity and distance constrained plant location problem (CDCPLP) and has been introduced by Albareda-Sambola et al. [2], who also describe several applications of this problem. One example is the planning of rural health care centers where each patient has to be assigned to exactly one such center and one vehicle.

In the following we will formally define the CDCPLP. We are given a complete bipartite graph $G = (I, J, E)$ where $I = \{1, \ldots, n\}$ is the set of customers, $J = \{1, \ldots, m\}$ the set of locations and $E = I \times J$ the set of edges. These edges are weighted with the assignment costs such that $c_{ij}$, $\forall i \in I, j \in J$ denotes the costs for assigning customer $i$ to the location $j$. Each customer $i \in I$ has a demand $d_i$ which has to be fulfilled by exactly one opened plant $j \in J$. Opening a plant $j$ implies cost $f_j$. The total demand of the customers assigned to a plant $j$ must not exceed the plant's capacity $b_j$. Additionally, each plant can use an unlimited number of homogeneous vehicles. Each vehicle has a maximum travel distance of $l$ and usage costs $g$. The travel distance for a round trip from the plant $j \in J$ to customer $i \in I$ is given by $t_{ij}$ and the total travel distance of all customers assigned to each vehicle must not exceed $l$. Customers are serviced in full round trips, i.e., the vehicle returns back to the plant before continuing on to another customer. The aim of the problem is the following. Let $Y \subseteq J$ be the set of opened plants, $X \subset I \times Y$ the assignments for customers to plants and $K_j$ the minimum number of needed vehicles for plant $j$ based on the assigned customers. Then we want to minimize the total costs $\sum_{j \in Y} f_j + \sum_{(i,j) \in X} c_{ij} + g \sum_{j \in Y} K_j$.

The CDCPLP can be seen as a three-staged combinatorial optimization problem (COP), where each level is connected through the costs. When each of the levels is considered individually they all represent a well-known classical COP: On the first level, choosing plant locations can be modeled as a facility

[1]This work is supported by the Austrian Science Fund (FWF) under grant P24660-N23.

location problem [6]. The assignment of the customers to the opened plants corresponds to an assignment problem [15]. The last level, in which the number of vehicles is minimized is basically a bin packing problem [7]. However, due to their strong interconnection they cannot be solved individually and a specialized solution method has to be developed in order to be able to solve the CDCPLP efficiently.

Most of the previous work for this problem focuses on exact methods based on integer programming (IP) but in this work we suggest a genetic algorithm with a local search for heuristically obtaining solutions to larger CDCPLP instances. In Section 2 the related work is discussed. The genetic algorithm, the chosen solution representation and its operators as well as the local search are described in Section 3. Section 4 is dedicated to the results of comprehensive tests which are performed on a large benchmark set and their results are presented there. Finally, conclusions are drawn in Section 5, where also a prospect for possible future work is given.

## 2 Related Work

The CDCPLP was introduced by Albareda-Sambola et al. [2] who described an integer programming formulation as well as a more advanced bilevel model in which they solved the last level as up to $|J|$ independent bin packing problems using integer programming techniques as well. Additionally they proposed a nested tabu search which uses neighborhood structures on all three levels. In Section 4 we compare our approach to this tabu search. The authors were able so exactly solve small instances and showed that the tabu search is viable for larger instances. An approach similar to the above bilevel model is the work by Fazel-Zarandi and Beck [9]. They use logic-based Benders decomposition in which they combine integer programming for the master problem with constraint programming for the subproblems. In the subproblems they also solve up to $|J|$ bin packing problems and generate Benders cuts. It was shown that this approach was able to solve the smaller instances much faster than the algorithm by Albareda-Sambola. Another trial towards exactly solving the CDCPLP on larger instances was again taken by Albareda-Sambola et al. [3] by improving their natural formulation from [2]. Several variants were compared and their model yielding the best results used binary decision variables $y_{js}$ $\forall j \in J, s = \{0, \ldots, \overline{k}\}$ indicating whether a plant $j$ is opened and uses exactly $s$ vehicles; $\overline{k}$ is an upper bound on the number of vehicles. Additionally, valid inequalities based on knapsack and bin packing constraints were added to the model.

Clearly, similar problems are described in the literature more extensively. The single source capacitated plant location problem (SSCPLP) [5] is a special case of the CDCPLP in which a set of customers has to be served by exactly one opened plant. Consequently, the CDCPLP must also be NP-hard. The difficulty lies in the choice of the plant locations and the assignment of the customers but in contrast to the CDCPLP there are no vehicle costs and distance constraints at the plants. Several exact [5, 10] and heuristic [1, 4] solution methods have been described in the literature so far.

A more general extensive survey of other location routing problems is given by Nagy and Salhi [14] who described several articles until 2007. More recently, Prodhon and Prins [16] extended this survey and included articles published until the year 2013. They noticed a growing number of publications in the last years and classified them into ten groups.

## 3 Genetic Algorithm

The proposed algorithm is a steady-state genetic algorithm (GA) with a randomized greedy heuristic for initial solution generation and an additional local search operation that is applied occasionally. As mentioned in the introduction, throughout this section the CDCPLP is viewed as split in three levels: the location level, the assignment level and the bin packing level. Our solution representation is solely based on the assignment level and described in more detail in Section 3.1.

We made some natural assumptions for the structure of problem instances: First, we assume that the cost of opening locations dominates the cost of single assignments as well as utilization costs for a single vehicle. Second, we expect that the locations have enough capacity for multiple customers. A failure to

meet these conditions will likely decrease our algorithms' performance and would mean that a different way of modeling the problem may be more appropriate. However, we think that these assumptions are realistic for most practical scenarios. Based on these assumptions good solution candidates would then typically be those with few opened locations with high utilization ratios. This intuition served us as a guideline when designing the GA.

### 3.1 Solution Representation

In order to reduce the search space of this three-staged problem, we use an incomplete solution representation based on the assignment level. We map each customer $i \in I$ to a location $j \in J$ s.t. the solution vector $A = (a_1, a_2, \ldots, a_n)$ with $a_i \in J, \forall i = 1, \ldots, n$ stores all customer assignments, where $a_i$ is the location customer $i$ is assigned to. The assignments have to satisfy the capacity and distance constraints to be feasible solutions. In this representation the set of opened plants is stored implicitly, the customer assignments explicitly and the assignments to vehicles not at all. Therefore, for evaluating a solution candidate a subproblem has to be solved. This subproblem computes the number of required vehicles for each location and is modeled as several independent bin packing problems, which is further discussed in Section 3.6. By using this solution representation all the genetic operators and the local search neighborhood structure act on the assignment level.

### 3.2 Initial Solutions

The procedure for constructing candidate solutions for the initial population is outlined in Algorithm 1. A feasible new assignment is one that does not violate the capacity or distance constraints at that location. A locally best assignment from customer $i \in I$ to a location $j \in J$ is one that minimizes the costs $cost_{ij}$ which are defined as:

$$cost_{ij} = c_{ij} + \frac{t_{ij}\, g}{l} \tag{1}$$

The heuristic aims at producing solutions of the kind we discussed above: few opened locations with high utilization. Algorithm 1 starts from a state where no locations are opened and no assignments are set. Now, as long as there are customers that are not assigned, a previously unopened location $loc$, chosen randomly, is opened. As long as the location has capacity remaining we assign an unassigned customer $c$ with lowest $cost_{c,loc}$ to the location; ties are broken randomly. Once there are no unassigned customers left the construction is complete.

### 3.3 Mutation

Deciding on a mutation operation for this problem involves two key considerations. First, it is easy to make a solution infeasible by small changes. When a solution is already rather good most of its locations will not have much free capacity left. Thus, assigning an additional customer, swapping a customer with another one with higher demand and similar operations can easily lead to a violation of the capacity constraints.

Mutating on the assignment level may seem the natural choice considering the solution representation, e.g., choosing a random customer $i \in I$ that is assigned to some location $j \in J$ and change his assignment to a new random location $j' \in J \setminus \{j\}$. There are, of course, various refinements that could be applied to the selection of customer and new location. However, they all share a similar problem. If the mutation opens a new location the new solution will likely have a worse fitness than the original solution due to our assumption that locations are usually more expensive than assignments. If no new location is opened by the mutation, however, our operation stays very local. In combination this results in a mutation operation that is likely too weak to reliably escape local optima.

Another option is to mutate on the location level, i.e., changing the set of opened locations. Our approach is based on the algorithm used for initial solution generation in Section 3.2. Let $O$ be the set of opened locations. Then one randomly chosen location $l \in O$ is closed leaving us with a set $P = O \setminus \{l\}$

**Algorithm 1:** Greedy solution construction

```
1  a = (a_1, ..., a_n) ← empty assignment;
2  locations ← J;
3  unassigned ← I;
4  while unassigned ≠ ∅ do
5      select random loc ∈ locations;
6      locations ← locations \{loc};
7      while true do
8          F ← {i ∈ unassigned | a is feasible with a_i = loc added};
9          if F = ∅ then
10             break;
11         end
12         c ← i ∈ F with minimal cost_{i,loc};
13         a_c ← loc;
14         unassigned ← unassigned \{c};
15     end
16 end
17 return a;
```

of prioritized locations. Now at line 5 of Algorithm 1, instead of selecting any random location we first randomly choose from locations in $P$. Once all of those have been opened the algorithm continues as in the original version.

Computational experiments were conducted to compare the two options as well as a hybrid in which both are applied with certain probabilities. In these experiments pure location level mutation produced the best results, especially for larger instances. This indicates that finding a good set of opened locations has more effect on the fitness of a solution than finding good assignments inside a fixed set of locations.

### 3.4 Crossover

Similar issues as discussed for mutation also occur for crossover operations. For many common crossover approaches the operation can produce solutions where the capacity constraints are violated. An adaption to satisfy the constraints or a post-crossover repair procedure is required to sustain feasibility in the population. The extent of repairs, i.e., the amount of assignments to change and new locations to open, is hard to predict and limit.

In an attempt of compromise we chose to adapt a uniform crossover operation to respect capacities, see Algorithm 2. The algorithm randomly iterates over the parent solutions' assignments, adding them one after another to the offspring which starts off empty. When both candidates of the parent solutions would preserve the feasibility of the newly generated solution one of them is selected uniformly at random. If only one of them preserves feasibility then that assignment is chosen. Cases where no parental assignment can be feasible adopted are postponed.

After this first step the postponed customers are handled in the following way. For each of them we first check if it can be assigned to one of the already opened locations. It is then assigned to a first fitting location. If no opened location is feasible, a new random location is opened where the postponed customer can be assigned to.

It turned out that the average number of such postponed assignments varied between only 0.2 and 1 in our preliminary experiments, depending more on the specific instance than on the parameters $I$ and $J$. Note that distance constraints do not have to be considered here: the parent solutions are assumed to be feasible and thus their assignments already satisfy the distance constraints.

**Algorithm 2:** Adapted uniform crossover

**Input**: Two parent solutions $A$, $B$
**Output**: Offspring $S$

$S = (S_1, \ldots, S_n) \leftarrow$ empty assignment;
$postponed \leftarrow \{\}$;
**foreach** $i \in \{1, \ldots, n\}$ **do**
  can$_A \leftarrow$ is $S$ still feasible with $S_i = A_i$?
  can$_B \leftarrow$ is $S$ still feasible with $S_i = B_i$?
  **if** $can_A \wedge can_B$ **then**
    $S_i \leftarrow$ random uniform choice between $A_i$ and $B_i$;
  **else if** $can_A$ **then**
    $S_i \leftarrow A_i$;
  **else if** $can_B$ **then**
    $S_i \leftarrow B_i$;
  **else**
    $postponed \leftarrow postponed \cup \{i\}$;
  **end**
**end**
**for** $c \in postponed$ **do**
  $Open :=$ set of opened locations in $S$;
  $assigned \leftarrow false$;
  **for** $loc \in Open$ **do**
    **if** $S$ *still feasible with assignment* $S_c = loc$ **then**
      $S_c \leftarrow loc$;
      $assigned \leftarrow true$;
    **end**
  **end**
  **if** $\neg assigned$ **then**
    $S_c \leftarrow$ random feasible unopened location;
  **end**
**end**
**return** $S$;

### 3.5 Local Search

Since both, mutation and crossover, have the possibility to open new facilities, the purpose of local search is to optimize the assignments when the facility locations are assumed to be fixed. Therefore, a local search using a swap neighborhood structure on the assignment level is applied to newly generated solutions with a probability $P_{local}$.

Let $A = (a_1, \ldots, a_I)$ be the starting solution. The local search generates every feasible neighboring solution by swapping the value of some $a_i \in A$ with an $a_j \in A$ with $i \neq j$. Best improvement is used as step function, so a best neighboring solution is always accepted as incumbent. This procedure is iterated until no further improvement is achieved.

### 3.6 Solution Evaluation

The used solution representation has no direct information on how many vehicles are required per plant. To compute them, as required for the objective cost, a bin packing problem is solved for each opened location. In our case the bins correspond to vehicles capacitated by $l$ and the objects to be packed to the customers with their distances $t_{i,j}$ to the location.

Two approaches were considered for the bin packing problems: an exact implementation as an integer

program (IP) using CPLEX 12.6 akin to the natural formulation from [12] and a first fit decreasing heuristic (FFD). In the latter, the objects are packed in non-increasing weight order one after another. FFD has an asymptotic approximation guarantee of 11/9 [11].

Preliminary results indicated that the substantially higher computational effort for solving the bin packing problem via CPLEX clearly does not pay off the slightly better results. In practice, applying the exact computation as a polishing step to the final population proved to be more reasonable.

However, there are several more advanced other possibilities to solve bin packing problems described in the literature, which are not considered here including exact [17] and heuristic [8, 13, 18] methods.

# 4 Computational experiments

For the computational tests we used the same benchmark instances as Albareda-Sambola et al. [2, 3]. These instances are extensions of those found at their homepage[1] for the Single Source Capacitated Plant Location Problem (SSCPLP). The $i$-th instance is called $p_i$ corresponding to the respective source instance of the SSCPLP and the naming used in [2, 3]. The instance size is denoted as $|J| \times |I|$. The SSCPLP instances are extended by the missing parameters $l$, $g$, and $t_{ij}$ $\forall i \in I$, $j \in J$ and increase the plant capacities by a factor of 1.5 (for details see [2, 3]). For instances $p_1$ to $p_6$ the distances $t_{ij}$ from customer $i \in I$ to location $j \in J$ are generated in two different ways, one where $t_{ij}$ correlates to $c_{ij}$ and one uncorrelated variant. All other instances only use the correlated version. For the $l$ and $g$ parameters different combinations are considered, as shown in Table 1.

Table 1: Parameter pairs used in instances

| Label | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| $l$ | 40 | 40 | 50 | 50 | 100 | 100 |
| $g$ | 50 | 100 | 80 | 150 | 150 | 300 |

All benchmarks were run on a single core of an Intel Core i5-2500K, 3.3 GHz PC. Our steady-state GA uses tournament selection with a tournament size of 2, a population size of 120 and a 100% crossover probability. The number of mutations per generation is a Poisson-distributed random number with mean one. New solutions replace the worst existing solution in the population.

On all experiments the algorithm was set to terminate after 50000 generations without improvement. As suggested in Section 3.3, pure location level mutation was used in all of the following experiments. From preliminary results a local search probability $P_{local}$ of 10% was determined to be a robust choice and fixed to this value. The bin packing instances in the objective function are solved using the FFD heuristic but at termination the fitness of every individual of the final population is calculated exactly using the IP approach.

The results for the $10 \times 20$ instances are shown in Table 2. The optimal reference values are taken from the exact results in [2] and %-gaps to them are listed. Every combination of base instance $p_1$ to $p_6$ with every parameter pair of Table 1 is considered, resulting in 36 correlated as well as 36 uncorrelated $10 \times 20$ instances. The table gives the results grouped by the label of these pairs. To account for variance every instance is run 20 times. The *Best* columns reflect the outcome with minimal objective of these 20 runs while *Worst* reflects the outcome with maximal objective. *Average %-gap* is the average gap to the optimum over all instances of one parameter pair.

Even though exact bin packing is only used on the final population we see that a good number of $10 \times 20$ instances are solved optimally. The execution times for the single runs are all between 1 and 3 seconds and therefore much lower than the ones reported by Albareda-Sambola et al. [3] which range from 15 to 70 seconds on $10 \times 20$ instances (on an Intel Core2 P8400 at 2.26GHz). Fewer optimally solved instances are observed with the increase of the $l$ parameter, i.e., with an increase in bin size. The GA behaves slightly worse on the uncorrelated instances. We believe this may be attributed to the

[1] www-eio.upc.es/~elena/sscplp/

computation of the $cost_{i,j}$ values defined in Section 3. While the distance is part of the calculation the results suggest that there may be room for improvement here. In all our runs the results are within $4.3\%$ of the optimal value and found within 3 seconds, which shows the applicability of the GA when time is crucial.

Table 2: Result summary for the 10x20 instances

| | Optimally solved | | Average %-gap | |
|---|---|---|---|---|
| | Best | Worst | Best | Worst |
| **Correlated** | **27/36** | **16/36** | **0.27** | **1.14** |
| A | 5/6 | 4/6 | 0.06 | 0.47 |
| B | 5/6 | 2/6 | 0.38 | 1.23 |
| C | 5/6 | 4/6 | 0.07 | 0.84 |
| D | 5/6 | 3/6 | 0.05 | 1.01 |
| E | 4/6 | 2/6 | 0.50 | 1.29 |
| F | 3/6 | 1/6 | 0.59 | 1.98 |
| **Uncorrelated** | **22/36** | **12/36** | **0.33** | **0.93** |
| A | 5/6 | 3/6 | 0.20 | 0.64 |
| B | 4/6 | 2/6 | 0.22 | 0.86 |
| C | 5/6 | 3/6 | 0.24 | 0.32 |
| D | 3/6 | 2/6 | 0.34 | 0.70 |
| E | 3/6 | 1/6 | 0.41 | 0.95 |
| F | 2/6 | 1/6 | 0.57 | 2.11 |

Table 3 compares the GAs performance with the Tabu Search (TS) from [2] on larger instances. The results are given in %-gaps in reference to the values in the *Ref* column, which are the overall best (but not necessarily optimal) solution values identified in [2]. More recent work allows for solving at least some of these instances optimally [3, 9] but give no specific objective values, leaving us with these as reference points. Instances $p_7$ to $p_{17}$ are of size $15 \times 30$, $p_{18}$ to $p_{25}$ are $20 \times 40$. These instances are only considered using parameter pair E, i.e, utilization cost $g$ of 150 and maximum travel distance $l$ of 100. As with the smaller instances before, every instance is run 20 times. The results of our algorithm are shown under the *GA* heading. The column $\overline{\%\text{-}Gap}$ shows the average gap to *Ref* over all 20 runs. *StdDev* is the standard deviation of the percentage gap over all runs. Finally, *Time* is the average execution time. The data under the *Tabu Search* heading is taken from [2]. The column *%-Gap* here also represents the gap from the solution to *Ref* and *Time* the time of a single execution of the search. Note that TS was executed on an Intel Pentium IV 2.4 GHz, which is a significantly weaker hardware than our setup. Thus, the times given are not suitable for direct comparison and are included only for reference. Column $\Delta_{Gap}(GA, TS)$ shows $Gap_{GA} - Gap_{TS}$, the highlighted values are those were the GA results are better.

We observe that TS is better than our average gap on 3 instances and the GA is better in 16 out of 19 instances. If we compare the best gap of all runs the TS is only better in $p_9$ and $p_{25}$. While the direct comparison of runtimes is difficult, as noted before, the numerical results demonstrate that the GA produces very good results in a small amount of time.

## 5 Conclusions and Future Work

In this work we presented a genetic algorithm for the capacity and distance constrained plant location problem. This problem can be seen as a three-staged combinatorial optimization problem, for which we used an incomplete solution representation. The solution evaluation is based on solving several independent bin packing subproblems. By using a randomized greedy construction heuristic for generating initial solutions, location-level mutation and local search we are able to outperform an existing tabu search algorithm on most benchmark instances with short execution times. Compared to an exact approach on small instances, the GA often finds optimal solutions or solutions within only a few percent of optimality.

Table 3: Detailed results for large instances

| | | GA | | | Tabu Search | | |
|---|---|---|---|---|---|---|---|
| Instance | Ref | $\overline{\%\text{-Gap}}$ | StdDev | Time(s) | % Gap | Time(s) | $\Delta_{Gap}$(GA,TS) |
| $p_7$ | 3757 | -0.19 | 0.43 | 5 | 1.36 | 69 | **-1.55** |
| $p_8$ | 5680 | 0.42 | 1.16 | 6 | 2.11 | 66 | **-1.69** |
| $p_9$ | 2526 | 2.53 | 0.88 | 7 | 0.44 | 71 | 2.09 |
| $p_{10}$ | 12360 | 2.24 | 0.53 | 8 | 7.65 | 35 | **-5.41** |
| $p_{11}$ | 3150 | -0.32 | 1.56 | 6 | -1.43 | 54 | 1.11 |
| $p_{12}$ | 3375 | -0.09 | 0.99 | 5 | 2.49 | 61 | **-2.58** |
| $p_{13}$ | 3416 | 2.17 | 1.28 | 6 | 3.16 | 78 | **-0.99** |
| $p_{14}$ | 4343 | 4.21 | 2.46 | 6 | 6.33 | 73 | **-2.12** |
| $p_{15}$ | 5265 | 0.65 | 0.68 | 8 | 0.70 | 38 | **-0.05** |
| $p_{16}$ | 7072 | 1.50 | 0.48 | 9 | 2.49 | 35 | **-0.99** |
| $p_{17}$ | 6650 | -1.73 | 0.79 | 4 | 6.77 | 79 | **-8.50** |
| $p_{18}$ | 9486 | 3.25 | 1.06 | 18 | 6.26 | 85 | **-3.01** |
| $p_{19}$ | 11463 | -0.55 | 0.99 | 15 | 7.48 | 116 | **-8.03** |
| $p_{20}$ | 13650 | 1.88 | 0.40 | 24 | 4.96 | 151 | **-3.08** |
| $p_{21}$ | 5146 | 0.91 | 1.44 | 21 | 2.35 | 160 | **-1.44** |
| $p_{22}$ | 3474 | -1.18 | 1.53 | 12 | 10.91 | 165 | **-12.09** |
| $p_{23}$ | 5387 | -4.83 | 0.77 | 9 | 18.75 | 172 | **-23.58** |
| $p_{24}$ | 5351 | -2.19 | 0.82 | 9 | 6.20 | 171 | **-8.39** |
| $p_{25}$ | 6328 | 3.26 | 0.59 | 8 | 1.74 | 166 | 1.52 |

Future work will include a more elaborate solution evaluation method. The results presented in this work showed weaknesses in terms of computation time for an exact evaluation based on integer programming and the used greedy evaluation is not always accurate. An evaluation procedure based on constraint programming, as done by Fazel-Zarandi and Beck [9] or a combination of exact and heuristic techniques could benefit the algorithm a lot. Another direction of future research could be the usage of more neighborhood structures, especially also on the location level, for the underlying local search.

# References

[1] R. K. Ahuja, J. B. Orlin, S. Pallottino, M. P. Scaparra, and M. G. Scutellà. A multi-exchange heuristic for the single-source capacitated facility location problem. *Management Science*, 50(6):749–760, 2004.

[2] M. Albareda-Sambola, E. Fernández, and G. Laporte. The capacity and distance constrained plant location problem. *Computers & Operations Research*, 36(2):597–611, 2009.

[3] M. Albareda-Sambola, E. Fernández, and G. Laporte. A computational comparison of several models for the exact solution of the capacity and distance constrained plant location problem. *Computers & Operations Research*, 38(8):1109–1116, 2011.

[4] M. J. Cortinhal and M. E. Captivo. Genetic algorithms for the single source capacitated location problem. In *Metaheuristics: Computer Decision-Making*, volume 86 of *Applied Optimization*, pages 187–216. Springer US, 2004.

[5] J. A. Diaz and E. Fernández. A branch-and-price algorithm for the single source capacitated plant location problem. *Journal of the Operational Research Society*, 53(7):728–740, 2002.

[6] Z. Drezner. *Facility Location: A Survey of Applications and Methods*. Springer Series in Operations Research and Financial Engineering. Springer New York, 1995.

[7] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145 – 159, 1990.

[8] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2(1):5–30, 1996.

[9] M. M. Fazel-Zarandi and J. C. Beck. Using logic-based benders decomposition to solve the capacity-and distance-constrained plant location problem. *INFORMS J. on Computing*, 24(3):387–398, 2012.

[10] K. Holmberg, M. Rönnqvist, and D. Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544 – 559, 1999.

[11] D. S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314, 1974.

[12] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6(4):366–422, 1960.

[13] John Levine and Frederick Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7):705–716, 2004.

[14] Gábor Nagy and Saïd Salhi. Location-routing: Issues, models and methods. *European Journal of Operational Research*, 177(2):649–672, 2007.

[15] D. W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774 – 793, 2007.

[16] Caroline Prodhon and Christian Prins. A survey of recent research on location-routing problems. *European Journal of Operational Research*, 238(1):1–17, 2014.

[17] Armin Scholl, Robert Klein, and Christian Jürgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627–645, 1997.

[18] Kevin Sim and Emma Hart. Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 1549–1556, New York, NY, USA, 2013. ACM.