A Simple Tabu Search for Warehouse Location

Laurent Michel and Pascal Van Hentenryck

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-02-05
April 2002

A Simple Tabu Search for Warehouse Location

Laurent Michel and Pascal Van Hentenryck
Department of Computer Science
Brown University
Box 1910
Providence, RI 02912

Abstract

The uncapacitated warehouse location problem (UWLP) has been studied heavily in mathematical programming but has received limited attention from the local search community. This paper presents an simple, yet robust and efficient, tabu search algorithm for the UWLP. The algorithm was evaluated on the standard OR Library benchmarks and on the `M*` instances which are very challenging for mathematical programming approaches. The benchmarks include instances whose sizes are 100×1000 and 500×500. Despite its simplicity, the algorithm finds optimal solutions to all benchmarks quickly and with very high frequencies. It also compares favorably with state-of-the-art genetic algorithms.

1 Introduction

Given a set of n warehouses and a set of m stores, the uncapacitated warehouse location problem (UWLP) consists of choosing a subset of warehouses that minimizes the fixed costs of the warehouses and the transportation costs from the warehouses to the stores. The UWLP has attracted considerable attention in mathematical programming. (See [5, 7] for surveys of some of these approaches.) Many specific branch and bound algorithms were developed, including dual and primal-dual approaches [6, 9]. Dual-based and primal-dual algorithms are very effective on the OR Library benchmarks for the UWLP [3]. However, they experience significant difficulties and exhibit exponential behaviour on the M* instances [12]. These instances, which model real situations, have a large number of sub-optimal solutions and few useless warehouses.

Interestingly, the UWLP has received little interest in the local search community. Reference [2] presents simulated annealing algorithms which produce high-quality solutions but they are quite expensive in computation times. Reference [1] presents a tabu-search algorithm which performs $5n$ random walks at each iteration and selects the best one as the neighbor to move to. Each of these iterations takes significant computing time, which limits the applicability of the algorithm.

Genetic algorithms have been shown to be very successful on the UWLP. In a series of papers spanning over several years (e.g., [10, 11, 12]) and culminating in [12], Kratica and al have shown that genetic algorithms find optimal solutions on the OR Library and the `M*` instances (whenever the optimal solutions are known) with high frequencies and very good efficiency. Their final algorithm uses clever implementation techniques such as caching and bit vectors to avoid recomputing the objective function which is quite costly on large-scale problems. Reference [12] also contains a detailed comparison with mathematical programming approaches and shows that the speed-up of the genetic algorithm over mathematical programming approaches increases exponentially with problem size on the `M*` instances.

This paper presents a simple tabu-search algorithm which performs amazingly well on the UWLP. The algorithm uses a linear neighborhood and essentially takes $O(m \log n)$ time per iteration. The algorithm finds optimal solutions on the OR Library and the `M*` instances (whenever the optimal solution is known) with high frequencies. It also outperforms the state-of-the-art genetic algorithm of Kratica and al, both in efficiency and robustness. The rest of the paper is organized as follows. Section 2 defines the UWLP and Section 3 presents the tabu-search algorithm. Section 4 reports the experimental results and Section 5 concludes the paper.

2 Uncapacitated Warehouse Location

We are given a set of n warehouses W and a set of m stores S. Each warehouse w has a fixed cost f_w and the transportation cost from warehouse w to store s is given by c_{ws}. The problem is to find a subset of warehouses and an assignment of warehouses to the stores to minimize the fixed and the transportation costs. Observe that, once the warehouses are selected, it suffices to assign the stores to their closest warehouse. As a consequence, the problem consists of finding a subset *Open* of warehouses minimizing the function

$$obj(Open) = \sum_{w \in Open} f_w + \sum_{s \in Stores} \min_{a \in Open} c_{as}.$$

3 The Tabu-Search Algorithm

We now describe the tabu-search algorithm. Since the only combinatorial part is the selection of warehouses, it is natural to represent a state in the local search by a vector

$y = \langle y_1, \ldots, y_n \rangle$ where y_w is 1 if warehouse w is open and 0 otherwise. In the following, we use the notation

$$Open(y) = \{w \in N \mid y_w = 1\}$$

to represent the warehouses that are opened in a state y.

3.1 Neighborhood

The neighborhood is extremely simple. It consists of simply flipping the status of a warehouse. Given a state y, the neighborhood $\mathcal{N}(y)$ of y is defined as

$$\mathcal{N}(y) = \{\mathit{flip}(y, w) \mid w \in W\}$$

where

$$\mathit{flip}(\langle y_1, \ldots, y_n \rangle, w) = \langle y_1, \ldots, y_{w-1}, !y_w, y_{w+1}, \ldots, y_n \rangle.$$

Obviously, $|\mathcal{N}(y)| = n$.

3.2 Tabu Search

The tabu-search algorithm uses a tabu list T which contains the set of warehouses that cannot be flipped. At each iteration, the algorithm considers the set of neighbors which are not tabu and whose gains are maximal. In other words, the set of non-tabu neighbors is defined as

$$\mathcal{N}^T(y) = \{flip(y, w) \mid w \in W \setminus T\}.$$

The best objective value of the neighbors is defined as

$$bestObj(y) = \max_{e \in \mathcal{N}^T(y)} obj(Open(e))$$

and the set of neighbors considered by the tabu-search is defined as

$$\mathcal{N}^*(y) = \{e \in \mathcal{N}^T(y) \mid obj(Open(e)) = bestObj(y)\}.$$

If the neighbors in $\mathcal{N}^*(y)$ do not degrade the current solution, the algorithm randomly moves to one of these neighbors by flipping, say, warehouse w. Warehouse w is then marked tabu for a number of iterations and the length of the tabu list is updated. Otherwise, if the neighbors in $\mathcal{N}^*(y)$ degrade the current solution, the algorithm performs a diversification. It randomly selects an open warehouse and closes it. This meta-heuristic is closely related to the method advocated in [4] but it exploits specific problem knowledge in the diversification phase, which is critical to achieve adequate performance and robustness. The algorithm can also be viewed as a degenerate form of variable-neighborhood search [8].

```
int nbStable = 0;
float best = obj;
S* = y;
int tLen = 10;
while (nbStable < 500) {
   float old = obj;
   if (bestGain(y) >= 0) {
      w = random(bestFlips(y));
      y_w = ! y_w;
      t[w] = it + tLen;
      if (obj < old && tLen > 2)
         tLen = tLen - 1;
      if (obj >= old && tLen < 10)
         tLen = tLen + 1;
      it = it + 1;
   } else {
      w = random(Open(y));
      y_w = 0;
   }
   if (obj < best) {
      best = obj;
      S* = y;
      nbStable = 0;
   } else
      nbStable = nbStable + 1;
}
```

Figure 1: A Tabu-Search Algorithm for Uncapacitated Warehouse Location.

Figure 1 describes the implementation of the tabu-search algorithm in more detail (including the parameters used in our implementation). The algorithm uses the set of best flips

$$bestFlips(y) = \{w \mid flip(y, w) \in \mathcal{N}^*(y)\}$$

and the best gain

$$bestGain(y) = obj(y) - bestObj(y)$$

to represent the neighbors and their objective value compactly. We will show how to maintain these values incrementally in the next two sections. The search iterates a number of steps. At each iteration, if the best gain is nonnegative, the algorithm randomly selects a warehouse in `bestFlips(y)` and flips its value. The tabu-list length is also adjusted using a standard scheme. When the gain is negative, the algorithm randomly selects an open warehouse and closes it. The algorithm terminates when the objective function has not improved for 500 iterations.

3.3 Data Structures

To achieve good performance in practice, it is critical to maintain the best flips and the best gain incrementally during the search. As a consequence, our implementation maintains a collection of data structures incrementally. For each store s, the algorithm maintains the closest warehouse and the cost of the closest and the second closest warehouse, i.e.,

$$a_s = \underset{w \in Open(y)}{\operatorname{argmin}} c_{ws} \tag{1}$$

$$b_s = \min_{w \in Open(y)} c_{ws} \tag{2}$$

$$d_s = \min_{w \in Open(y) \setminus \{w_{a_s}\}} c_{ws}. \tag{3}$$

The last two equations make it easy to compute the potential benefit of opening and closing a warehouse. The benefit g_w^- of closing warehouse w (assuming that w is opened) is given by

$$g_w^- = f_w - \sum_{s \in S_w} (d_s - b_s) \tag{4}$$

where S_w is the set of stores assigned to w, i.e.,

$$S_w = \{s \in S \mid \ a_s = w\}$$

The benefit g_w^+ of opening warehouse w (assuming that w is closed) is given by

$$g_w^+ = -f_w + \sum_{s \in S} e_{ws}. \tag{5}$$

where

$$e_{ws} = \max(0, b_s - c_{ws}) \tag{6}$$

The gain g_w of flipping warehouse w can then be expressed as

$$g_w = \begin{cases} g_w^- & \Leftrightarrow \quad w \in Open(y) \\ g_w^+ & \Leftrightarrow \quad \text{otherwise}\ . \end{cases}$$

The best gain is given by

$$bestGain = \max_{w \in W \setminus T} g_w \tag{7}$$

The best flips can now be defined as

$$bestFlips = \{w \in W \setminus T \ \mid \ g_w = bestGain\}. \tag{8}$$

3.4 Incremental Algorithms

We now discuss how Equations 1-8 can be maintained incrementally. In the following, we use a^0 to denote the value of a before the flip and a^1 to denote the value of a after the flip. Similar conventions are used for b and c.

Maintaining g^-: The values g^- are updated whenever values a_s, b_s and d_s are modified. More precisely, for every triplet $\langle a_s, b_s, d_s \rangle$ such that

$$a_s^1 \neq a_s^0 \vee b_s^0 \neq b_s^1 \vee d_s^0 \neq d_s^1,$$

the following update rules must be applied

$$\begin{array}{rcl} g^-_{a_s^0} & = & g^-_{a_s^0} + (b_s^0 - d_s^0) \\ g^-_{a_s^1} & = & g^-_{a_s^1} - (b_s^1 - d_s^1). \end{array}$$

Observe that a_s^0 may, or may not, be equal to a_s^1.

Maintaining g^+: The values g^+ can also be maintained incrementally. From Equations 5 and 6, these values must only be updated when the value b_s changes, since c_{ws} is a constant. We now show which warehouses to consider when a store s has its value b_s updated.

Consider store s and its value b_s. The key observation is to notice that $max(0, b_s - c_{ws})$ is a monotonically decreasing function for a permutation π_s of the vector c_{ws} (see Figure 2). The permutation is simply obtained by sorting the warehouses by increasing order of the costs c_{ws}. As a consequence, it is easy to determine which warehouses are affected by the change in b_s and to update the values e_{ws} (and hence the values g_w^+) accordingly.

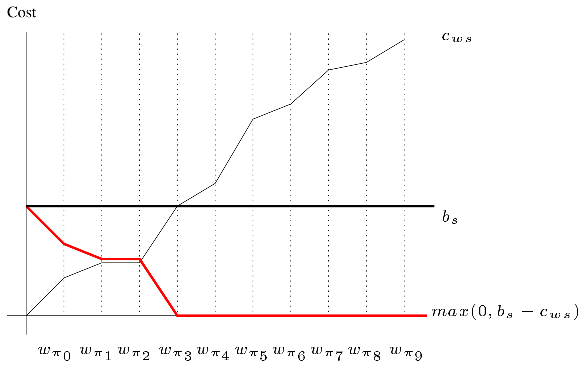

Figure 2: Updating g^+

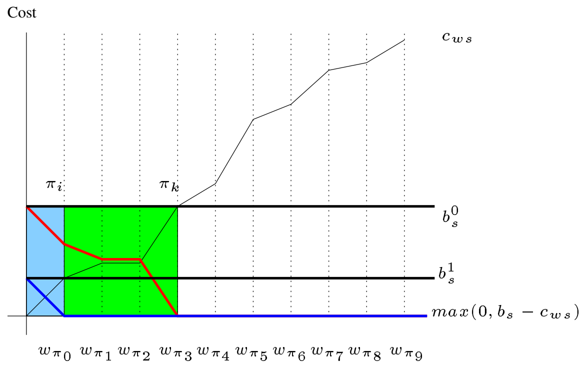

Figure 3: Opening a Warehouse.

Consider first the case where $b_s^1 \leq b_s^0$, which corresponds to opening a warehouse. Figure 3 depicts this situation. (The top red curve denotes $max(0, b_s^0 - c_{ws})$, while the bottom blue curve denotes $max(0, b_s^1 - c_{ws})$). It is sufficient to update g_w^+ only for those w such that $c_{ws} \leq b_s^0$. The other warehouses are unaffected. We obtain the following update rules:

$$\begin{array}{ll} g_w^+ = g_w^+ - (b_s^0 - b_s^1) & \forall \quad w : c_{ws} < b_s^1 \\ g_w^+ = g_w^+ - (b_s^0 - c_{ws}) & \forall \quad w : b_s^1 \leq c_{ws} < b_s^0. \end{array}$$

Consider now the case where $b_s^1 \geq b_s^0$, which corresponds to closing a warehouse. Figure 3 depicts this situation. (The bottom red curve denotes $max(0, b_s^0 - c_{ws})$, while the top blue curve denotes $max(0, b_s^1 - c_{ws})$). It is sufficient to update g_w^+ only for those w such that $c_{ws} \leq b_s^1$. The other warehouses are unaffected. We obtain the following update rules:

$$\begin{array}{ll} g_w^+ = g_w^+ + (b_s^1 - b_s^0) & \forall \quad w : c_{ws} < b_s^0 \\ g_w^+ = g_w^+ + (b_s^1 - c_{ws}) & \forall \quad w : b_s^0 \leq c_{ws} < b_s^1 \end{array}$$

Complexity Analysis: We analyze the complexity of maintaining Equations 1-8. Note that the entries of the arrays a, b, and d of Equations 1, 2, and 3 can be maintained with

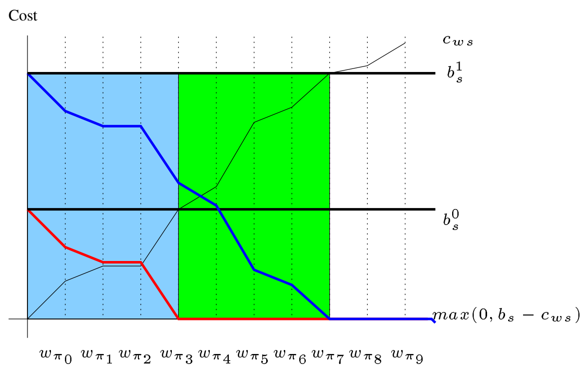

Figure 4: Closing a Warehouse.

priority queues in $O(m \ \log \ n)$. We now show that updating the entries in Equations 4, 5, and 6 takes time linear in the number of updated values. This type of analysis was suggested in [13], since it captures the essence of many incremental algorithms maintaining an output under changes. It is particularly appropriate to analyze local search algorithms, since a move does not change the state much in general.

Updating g^- does not induce any asymptotic cost. Indeed, the update rule for each triplet takes $\Theta(1)$ time and each triplet $\langle a_s, b_s, d_s \rangle$ satisfies

$$a_s^1 \neq a_s^0 \vee b_s^0 \neq b_s^1 \vee d_s^0 \neq d_s^1,$$

which means that the update rule only adds a constant factor to the maintenance of the priority queues.

We now analyze the update of g^+. Define $\Delta(e)$ as the number of values e_{ws} which needs to be updated because of a flip. We show that the algorithm is linear in $\Delta(e)$, while the update of g^+ requires $\Omega(\Delta(e))$, yielding an optimal algorithm. Indeed, the value e_{ws} changes each time a warehouse w is considered by the above update rules for a store s. Hence the amount of work in the update rules is linear in the number of updated e_{ws}. In addition, updating g_w^+ when e_{ws} changes takes constant time. Note also that the update of g^+ is optimal: it is necessary to consider all the updated e_{ws}, since they cannot cancel out. This is due to the fact that either all the b_s increase (closing a warehouses) or all the b_s decrease (opening a warehouse).

Once Equations 1-6 are updated, Equations 7 and 8 can easily be computed in $O(m \ \log \ m)$ time and $O(m)$ time respectively. Overall, the maintenance of Equations 1-8 takes $O(m \ \log \ n + \Delta(e))$ time, which is optimal. Note that, in practice, this cost is dominated by updating of the priority queues, as a profile of our implementation has shown.

Bench	Size	Opt	BEST	AVG	WST	$\mu(TS)$	$\mu(TE)$	$\sigma(S)$	$\sigma(TS)$	$\sigma(TE)$
cap71	16x50	20	932615.75	932615.75	932615.75	0.00	0.11	0.000%	0.00	0.01
cap72	16x50	20	977799.40	977799.40	977799.40	0.01	0.12	0.000%	0.01	0.01
cap73	16x50	20	1010641.45	1010641.45	1010641.45	0.01	0.15	0.000%	0.01	0.01
cap74	16x50	20	1034976.97	1034976.97	1034976.97	0.01	0.15	0.000%	0.00	0.01
cap101	25x50	17	796648.44	796777.48	797508.72	0.03	0.16	0.039%	0.04	0.04
cap102	25x50	20	854704.20	854704.20	854704.20	0.01	0.14	0.000%	0.01	0.01
cap103	25x50	20	893782.11	893782.11	893782.11	0.05	0.20	0.000%	0.04	0.04
cap104	25x50	20	928941.75	928941.75	928941.75	0.01	0.19	0.000%	0.00	0.01
cap131	50x50	16	793439.56	793611.62	794299.85	0.05	0.22	0.043%	0.03	0.04
cap132	50x50	20	851495.32	851495.32	851495.32	0.02	0.21	0.000%	0.01	0.01
cap133	50x50	20	893076.71	893076.71	893076.71	0.06	0.27	0.000%	0.04	0.05
cap134	50x50	20	928941.75	928941.75	928941.75	0.02	0.31	0.000%	0.01	0.01
capa	100x1000	20	17156454.4	17156454.4	17156454.4	0.47	5.77	0.000%	0.30	0.31
capb	100x1000	14	12979071.5	13012432.1	13214718.1	0.74	4.37	0.458%	0.40	0.41
capc	100x1000	13	11505594.3	11512451.7	11551802.8	1.64	4.90	0.127%	1.14	1.15
MO1	100x100	20	1305.95	1305.95	1305.95	0.09	1.02	0.000%	0.03	0.03
MO2	100x100	20	1432.36	1432.36	1432.36	0.16	1.08	0.000%	0.13	0.13
MO3	100x100	9	1516.77	1519.36	1521.47	0.16	1.07	0.154%	0.16	0.16
MO4	100x100	20	1442.24	1442.24	1442.24	0.09	1.02	0.000%	0.04	0.04
MO5	100x100	20	1408.77	1408.77	1408.77	0.13	1.09	0.000%	0.07	0.07
MP1	200x200	20	2686.48	2686.48	2686.48	0.35	2.41	0.000%	0.06	0.06
MP2	200x200	20	2904.86	2904.86	2904.86	0.36	2.45	0.000%	0.13	0.13
MP3	200x200	20	2623.71	2623.71	2623.71	0.32	2.39	0.000%	0.06	0.06
MP4	200x200	19	2938.75	2938.94	2942.63	0.83	2.92	0.029%	0.69	0.69
MP5	200x200	20	2932.33	2932.33	2932.33	0.61	2.73	0.000%	0.42	0.42
MQ1	300x300	[20]	4091.01	4091.01	4091.01	0.67	4.28	0.000%	0.07	0.08
MQ2	300x300	[20]	4028.33	4028.33	4028.33	0.70	4.26	0.000%	0.08	0.15
MQ3	300x300	[20]	4275.43	4275.43	4275.43	0.71	4.31	0.000%	0.13	0.13
MQ4	300x300	[20]	4235.15	4235.15	4235.15	0.76	4.68	0.000%	0.16	0.77
MQ5	300x300	[20]	4080.74	4080.74	4080.74	1.08	4.54	0.000%	0.42	0.42
MR1	500x500	[20]	2608.15	2608.15	2608.15	2.76	9.40	0.000%	1.19	1.29
MR2	500x500	[20]	2654.73	2654.73	2654.73	2.03	8.98	0.000%	0.38	0.44
MR3	500x500	[19]	2788.25	2788.48	2792.83	2.92	9.83	0.036%	2.10	2.07
MR4	500x500	[20]	2756.04	2756.04	2756.04	1.98	8.44	0.000%	0.32	0.31
MR5	500x500	[20]	2505.05	2505.05	2505.05	1.83	8.32	0.000%	0.23	0.28
MS1	1000x1000	[20]	5283.76	5283.76	5283.76	6.93	23.50	0.000%	0.43	0.82
MT1	2000x2000	[19]	10069.80	10070.79	10089.46	33.22	76.75	0.043%	8.61	8.98

Table 1: Experimental Results for the Local Search Algorithm.

Bench	Runs	Opt	$\mu(T)$	$\frac{\mu(T)*133}{850}$	DUALOC	$\frac{DUALOC*133}{850}$
cap41-74	260	260	0.86	.13	<0.01	<0.01
cap81-104	240	240	1.26	.19	<0.01	<0.01
cap111-134	240	198	2.97	.45	<0.01	<0.01
A-C	60	42	83.1	12.99	25.17	3.94
O	100	93	5.49	.85	32.54	5.09
P	100	100	16.60	2.59	369.7	57.85
Q	100	100	34.97	5.47	2913.7	455.90
R	100	99	93.69	14.66	*75964*	*11886.13*
S	100	100	379.6	59.38	*12279*	*1921.30*
T	100	100	1812.3	283.57	*NA*	*NA*

Table 2: Experimental Results of the Genetic Algorithm in [12].

4 Experimental Results

Table 1 depicts the experimental results on the standard OR Library benchmarks for uncapacitated warehouse location, as well as the `M*` instances generated according to the scheme specified in [12]. Recall that the `M*` instances, which capture classes of real UWLPs [12], are very challenging for mathematical programming approaches because they have a large number of suboptimal solutions. Each benchmark was run 20 times on a 850Mhz Intel Pentium III running Linux. The table reports the number of times the optimal solution was found (#Opt), the best (BEST), average (AVG), and worst (WST) solutions found across the runs, and the average time in seconds to the best solution ($\mu(TS)$) and to complete the search ($\mu(TE)$). The last three columns report the standard deviation for the solution (expressed as a percentage for $\sigma(S)$), the time to the best solution, and the time to the completion of the execution. The `cap` benchmarks are from the OR Library. The other problems are the `M*` instances. On the larger `M*` instances, we have not been able to obtain the optimal solution using mathematical programming tools. (Similar problems were experienced in [12] on these instances). The results in column #Opt are in between brackets on these benchmarks, since we are not guaranteed that our best solutions are indeed optimal. It is useful to note that the algorithm has no prior knowledge of the optimal solution, i.e., it cannot terminate early when the optimum solution is found.As can be seen from Table 1, the algorithm is very robust. It finds optimal solutions with very high frequencies on all benchmarks. As predicted by the theoretical results, the algorithm outperforms the tabu search in [1] by many orders of magnitude.

It is particularly interesting to compare our algorithm to the state-of-the-art genetic algorithm [12]. For reference purposes, the results reported in [12] are shown in Table 2, together with scaled running times to take into account the difference in processor speed. The table also lists the running times of DUALOC [6], a dual-based procedure reported

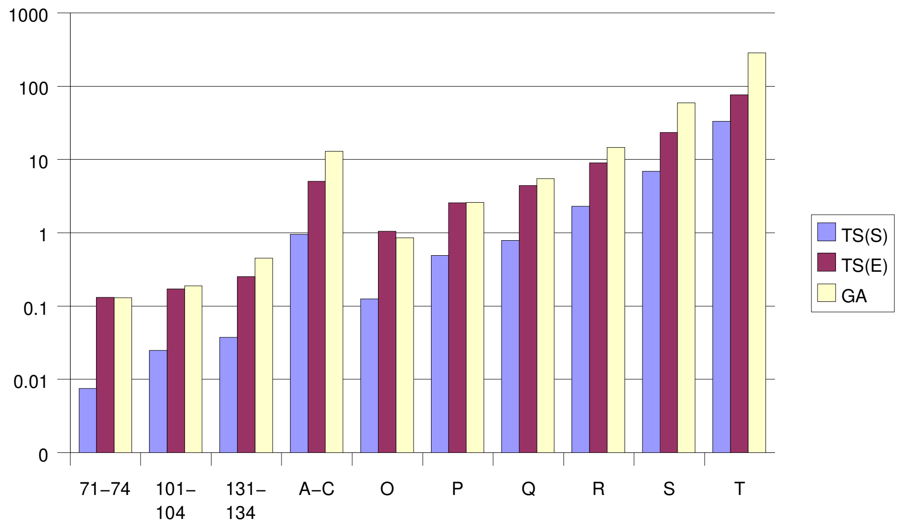

Figure 5: Comparison of the Running Times.

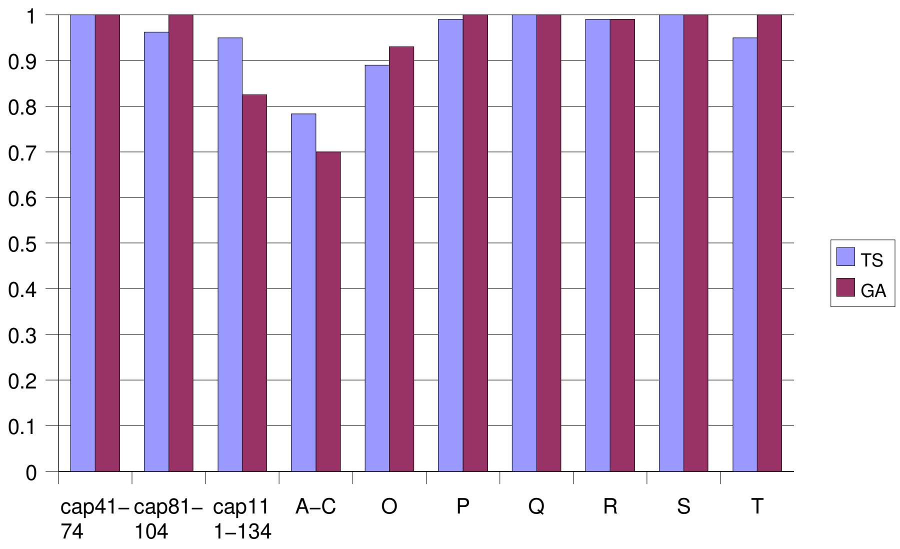

Figure 6: Comparison of the Robustness.

in [12] as the state-of-the-art for mathematical programming. (Our experience with state-of-the-art MIP packages confirms this). Entries in italics correspond to instances where `Dualoc` did not find the optimal solution in the allocated time. The last solution produced by DUALOC is reported to be about 10% above the best solution found by the genetic algorithm. Note also that it is not really clear what [12] reports as running times. In [11], the authors state

> Maximal number of iterations is 20000. If the optimal solution is obtained before 20000 iterations, execution is stopped and results are immediately printed.

There is no such remark in [12] but there is no definition of running times either. It is thus impossible to determine the exact nature of the running times reported in this last paper.

Figure 5 plots a histogram comparings the two algorithms. The y axis gives the running times in seconds on a logarithmic scale, the $TS(S)$ column gives the tabu search time to the best solution, and the $TS(E)$ column gives the running time to the completion. Figure 6 plots a comparison of the robustness of the two algorithms. It reports the frequency of finding the optimal solutions for both algorithms.

Our tabu-search algorithm almost always outperforms the genetic algorithm, while exhibiting essentially the same robustness. For instance, on the largest benchmarks from the OR Library, i.e., `capa`, `capb`, and `capc`, our tabu-search algorithm is at least 2.6 times faster when machines are scaled (and up to 13.7 times faster if we compare the time to find the best solution) and 8.3% more robust. The results seem similar on the `M*` instances, although a direct comparison is difficult, since their actual instances are not available. It should also be mentioned that there is considerable room for improvement in our implementation.

5 Conclusion

The uncapacitated warehouse location problem (UWLP) has been studied heavily in mathematical programming but has received limited attention from the local search community. This paper presented a simple, yet robust and efficient, local search algorithm for the UWLP. The algorithm was evaluated on the standard OR Library benchmarks and on the `M*` instances proposed in [12]. The `M*` instances have a large number of suboptimal solutions and are very challenging for mathematical programming approaches, which exhibit exponential growth in the problem size on these instances. Despite its simplicity, our algorithm outperforms state-of-the-art genetic algorithms on both classes of benchmarks and hence should be of great practical interest.

Acknowledgments

Thanks to Russell Bent for interesting discussions on this paper. Pascal Van Hentenryck is partially supported by NSF ITR Award DMI-0121495.

References

[1] K.S. Al-Sultan and M.A. Al-Fawzan. A tabu search approach to the uncapacitated facility location problem. *Annals of Operations Research*, 86:91–103, 1999.

[2] M.L. Alves and M.T. Almeida. Simulated annealing algorithm for simple plant location problems. *Rev. Invest.*, 12, 1992.

[3] J.E. Beasley. Obtaining Test Problems via Internet. *Journal of Global Optimization*, 8:429–433, 1996.

[4] C. Codognet and D. Diaz. Yet Another Local Search Method for Constraint Solving. In *AAAI Fall Symposium on Using Uncertainty within Computation*, Cape Cod, MA., 2001.

[5] G. Cornuéjols, G.H. Nemhauser, and L. Wolsey. *Discrete Location Theory*, chapter The Uncapacitated Facility Location Problem, pages 119–171. Lecture Note in Artificial Intelligence (LNAI 1865). Wiley, 1990.

[6] D. Erlenkotter. A Dual-Based Procedure for Uncapacitated Facility Location: General Solution Procedures and Computational Experience. *Operations Research*, 26:992–1009, 1978.

[7] L.L. Gao and E.P. Robinson. Uncapacitated Facility Location: General Solution Procedures and Computational Experience. *European Journal of Operations Research*, 76:410–427, 1994.

[8] P. Hansen and N. Mladenovic. An introduction to variable neighborhood search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-heuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer Academic Publishers, 1998.

[9] M. Koerkel. On the Exact Solution of Large-Scale Simple Plant LOcation Problems. *European Journal of Operations Research*, 39:157–173, 1989.

[10] J. Kratica, V. Filipovic, V. Sesum, and D. Tosic. Solving the uncapacitated warehouse location problem using a simple genetic algorithm. In *Proceedings of the XIV*

International Conference on Material handling and warehousing, pages 3.33–3.37, 1996.

[11] J. Kratica, D. Tosic, and V. Filipovic. Solving the uncapacitated warehouse location problem by sga with add-heuristic. In *XV ECPD International Conference on Material Handling and Warehousing*, 1998.

[12] J. Kratica, D. Tosic, V. Filipovic, and I. Ljubic. Solving the Simple Plant LOcation Problemsby Genetic Algorithm. *RAIRO Operations Research*, 35:127–142, 2001.

[13] G. Ramalingam. *Bounded Incremental Computation*. PhD thesis, University of Wisconsin-Madison, 1993.