Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Discrete Mathematics 39 (2012) 21–28

Electronic Notes in DISCRETE MATHEMATICS

www.elsevier.com/locate/endm

Variable Neighborhood Search for Solving the Balanced Location Problem

Jozef Kratica [a,1] Markus Leitner [b,2] Ivana Ljubić [c,3]

[a] *Mathematical Institute, Serbian Academy of Sciences and Arts, Kneza Mihaila 36/III, 11 000 Belgrade, Serbia*

[b] *Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria*

[c] *Department of Statistics and Operations Research, University of Vienna, Austria*

Abstract

In this paper we propose a general variable neighborhood search approach for the balanced location problem. Next to large shaking neighborhoods, the embedded variable neighborhood descent utilizes three neighborhood structures that focus on different solution aspects. By a computational study, we show that this VNS outperforms existing methods with respect to average solution quality and stability.

Keywords: discrete location, balanced allocation of clients, variable neighborhood search

[1] Email: jkratica@mi.sanu.ac.rs This research was partially supported by Serbian Ministry of Science under grant 174010

[2] Email: leitner@ads.tuwien.ac.at

[3] Email: ivana.ljubic@univie.ac.at

1571-0653/$ – see front matter
doi:10.1016/j.endm.2012.10.004

1 Introduction

We consider the *Balanced Location Problem* (LOBA), a variant of the discrete facility location problem in which one aims to open p facilities and balance the number of clients allocated to them. Hereby, each client must be assigned to its closest open facility. More formally, we are given m potential facilities, n clients, a parameter $p \in \mathbb{N}$ and transportation costs (distances) $c_{ij} \geq 0$, between each client i and each potential facility j. In a feasible solution we open exactly p facilities and assign each client to an open facility with the minimum transportation cost. Let u and l be the maximum and minimum number of clients assigned to any open facility, respectively. Then the objective is to determine a set of p open facilities such that the difference between u and l is minimized.

LOBA has applications in the location of antennas for mobile communication, or in the territory design [4], where small geographic areas must be grouped into larger geographic clusters according to some planning criteria like political districting, solid waste collection, or school districting. Note that a similar problem that addresses the minimization of the Gini index has been studied by Drezner et al. [1].

LOBA has been introduced by Marin [5] who proposed two different integer programming formulations and a branch-and-cut algorithm, utilizing preprocessing techniques to reduce the problem size and a so-called *interchange heuristic* (cf. Section 3) to derive good upper bounds. Furthermore, he developed several families of valid inequalities to strengthen these formulations. Computational results have been reported on instances with up to 50 potential facilities and 100 clients. Recently, Filipović et al. [2] suggested to solve LOBA using a hybrid metaheuristic called HGA which combines a genetic algorithm with a fast interchange heuristic. HGA is based on a binary encoding scheme and uses caching techniques to improve its computational performance. In a computational study performed in [2], the authors compared HGA with the heuristic method from [5] using instances from [5] plus a set of significantly larger instances with up to 1000 potential facilities and 1000 clients. The obtained results document a superior performance of HGA in most of the cases.

2 VNS for the Balanced Location Problem

We initialize the VNS by randomly selecting p facilities to create an initial solution. We then use three neighborhood search procedures that are exchanged

in a systematic way within the embedded variable neighborhood descent algorithm. Random moves in larger, so-called shaking neighborhoods are performed in order to escape from local optima and to explore new regions of the search space. In this section we elaborate on the main ingredients of the proposed VNS approach. A detailed description of the variable neighborhood search metaheuristic can be found in, e.g., [3].

Evaluating the Objective Function

As in many discrete location problems, for each location decision in LOBA (i.e., for each subset of open facilities) we can determine the optimal assignments in polynomial time. Therefore, the natural way to store the solutions is to use a set of p indices corresponding to open facilities. It is therefore crucial to choose appropriate data structures in order to be able to efficiently evaluate the objective value of a solution.

Our data structures depend on the size of the parameter p and an auxiliary parameter c (set to $c = 0.95$ in the default implementation (see also [2])). If p is large, i.e., $p > c \cdot \sqrt{m}$, for each client i, we pre-compute a list F_i of all facilities sorted in non-decreasing order according to c_{ij}. For each client i, we simply iterate through F_i until the first open facility is found. Clearly, this facility has minimal transportation costs to i among all open facilities. In the worst case, the number of steps for each client is $m - p \leq m - c \cdot \sqrt{m}$ and thus the overall run-time complexity is $O(n \cdot (m - p)) = O(nm)$. For reasonable solutions, however, we expect that on average only m/p (rather than $m - p$) facilities need to be processed for each client until the first open facility occurs. In this case, the expected total complexity is given by $O(n \cdot m/p) = O(n\sqrt{m})$.

If p is small, i.e., $p \leq c \cdot \sqrt{m}$, this strategy is slow since most facilities will not be active. Thus, it is likely that one would need to consider a significant proportion of list F_i for each client i before finding an open facility. To avoid this situation, for each client i we iterate through the list L of open facilities and identify an open facility $j(i)$ with the minimum transportation cost to i ($j(i) = \arg\min_{j \in L} c_{ij}$). Thus, the overall run-time complexity of this procedure is $O(n \cdot p)$.

Shaking Neighborhoods

Since only the facilities with a maximum and minimum number of assigned clients are considered when computing the objective value of a solution, it may be even more likely than for other problems to get trapped in poor local optima when considering local search neighborhoods which are relatively restricted. Thus, it is crucial to use larger shaking neighborhoods in order to escape from

such local optima.

To perform a move in the k-th shaking neighborhood, $k_{\min} \leq k \leq k_{\max}$, k out of the p currently open facilities are chosen uniformly at random and closed. Afterwards, k among the now $m - p + k$ closed facilities are chosen uniformly at random and opened.

Embedded Variable Neighborhood Descent

We improve candidate solutions using a variable neighborhood descent (VND) based on three neighborhoods, denoted by LS_1, LS_2, LS_3. These are applied in the usual way, i.e., if an improving solution within the LS_i neighborhood could be identified, the search continues within LS_1. Otherwise, if the solution turns out to be locally optimal w.r.t. LS_i, we resort to LS_{i+1}, for $i = 1, 2$.

Overall, if x denotes the current incumbent and $o(x)$ its objective value, and VND terminates with solution x'', three cases may happen: (i) if $o(x'') > o(x)$, a new shaking move (with increased shaking size) is performed on x and the VND is repeated, (ii) if $o(x'') < o(x)$, x'' is the new incumbent x and a shaking move with the previous shaking size is performed, and (iii) if $o(x'') = o(x)$ then x'' is used a new incumbent with probability p_{move} in which case the shaking size remains identical while with probability $1 - p_{\text{move}}$, the incumbent x is kept and the next shaking neighborhood is considered.

Neighborhood Structures

LS_1 and LS_2 focus on the two most influential aspects of a solution, namely on the open facilities with a largest and smallest number of assigned clients. For a given candidate solution x, let $F(x)$ denote its open facilities and $F_{\max}(x) \subseteq F(x)$ and $F_{\min}(x) \subseteq F(x)$, be the sets of facilities with the largest and smallest number of assigned clients, respectively.

The neighborhood $LS_1(x)$ consists of all solutions x' that can be constructed by closing exactly one facility from $F_{\max}(x)$ and opening another facility instead. In case a neighboring solution that decreases the objective value exists, x is replaced by such a solution yielding the maximum decrease.

On the contrary, the neighborhood $LS_2(x)$ consists of all solutions x' that differ from a given solution x by closing one facility from $F_{\min}(x)$ and opening a new, originally closed facility instead. As for LS_1, in case at least one improving solution exists in $LS_2(x)$, a neighboring solution maximizing the improvement of the objective value is chosen as the next one.

Finally, the neighborhood LS_3 consists of those solutions x' that differ from

the current one x in exactly one facility from $F(x) \setminus (F_{\max}(x) \cup F_{\min}(x))$ which is closed, and a new one from $F \setminus F(x)$ is opened instead. In other words we have $LS_3 = \{x' \mid |F(x') \cap F(x)| = |F(x)| - 1 \text{ and } |F(x') \setminus F(x)| = 1\}$.

All three local search procedures are based on swaps between open and closed facilities. More precisely, we apply the *best improvement* strategy in which we search for a pair (j_1, j_2) of facilities $(j_1 \in F(x),\ j_2 \notin F(x))$ that brings the highest gain with respect to the current solution.

Efficient Implementation of Swap Moves

An obvious option for implementing swap moves would be to apply the fast interchange heuristics (cf. [2]) to each candidate pair (j_1, j_2). A significant speed-up can, however, be achieved when implementing an one-to-all rather than many one-to-one calculations. Here, we divide each swap into two parts:

(i) Close one facility and correspondingly update the objective value.

(ii) Open a new facility

In this step, after closing facility j, we need to identify the closest open facility for each client originally assigned to j. In the second step, when opening a new facility j', for each client we only need to compare its distance to j' with the distance to the actual closest facility from $F(x) \setminus \{j\}$. Observe, that when searching any of the three proposed neighborhood structures we only need to consider the first step once for each facility that may be closed. In LS_1, for example, often $|F_{\max}(x)| = 1$, i.e., only one facility may be closed.

Note that the whole VND scheme is collapsed into a single local search procedure if $F_{\max}(x) = F_{\min}(x) = F(x)$ since all three neighborhoods would be identical in this case. To search this neighborhood, we need to consider swaps between every open and every closed facility.

In the general case the size of each neighborhood LS_i, $i = 1, 2, 3$, is given by $|F_{\text{close}}(x)| \cdot |F(x) \setminus F_{\text{close}}(x)| \leq p \cdot (m - p)$. Hereby, $F_{\text{close}}(x)$ denotes the set of facilities that may be closed in neighborhood i, e.g., $F_{\text{close}}(x) = F_{\max}(x)$ for LS_1.

Closing a facility with u assigned clients can be done in $O(u \cdot m)$, since we either need $O(u \cdot (m-p))$ steps if p is large or $O(u \cdot p)$ steps if p is small. In a one-to-all calculation we then need to try whether any of the closed $m-p$ facilities yields an improvement. Since the latter can be performed in $O(n \cdot (m - p))$ steps, the overall complexity of searching any of our neighborhoods is $O(pnm)$. This worst case could, however, only occur if all open facilities may be closed and there are $O(n)$ clients assigned to each open facility. Thus, in particular LS_1 and LS_2 can be searched much faster in practice.

3 Experimental results

Our computational study has been performed on a single core of an Intel quad core processor with 2.5GHz, 1GB RAM, with Windows XP. For the VNS, which has been implemented in C, we set parameters $k_{\min} = 2$, $k_{\max} = \min\{p, 20\}$, $p_{\text{move}} = 0.2$, and used an iteration limit of $N_{\max} = 1000$ as termination criterion.

Tables 1 and 2 compare the performance of the VNS, with the genetic algorithm (HGA) from [2], and the heuristic method (AMH) from [5]. Since, VNS and HGA are not deterministic, each experiment has been repeated 20 times. We used benchmark instances from [2,5] with randomly generated distances between clients and potential facility sites. We report computational results on a subset of 12 small, 16 medium, and 16 large instances from this very large benchmark collection.

In Tables 1 and 2, the first column (Inst) identifies the used instance which is denoted with $m - n - p - pl$, where m, n, and p, denote the numbers of potential facility sites, clients, and facilities that need to be opened, respectively. The last integer pl gives the "perturbation level", a parameter used to create the corresponding instance, cf. [5]. For HGA and VNS, we report best found solutions o_{best}, average objective values o_{avg}, together with corresponding standard deviations of relative errors in parenthesis (in percent), average runtimes for finding the best solution of each run t_{best} in seconds, average total runtimes t_{avg} in seconds, and the number of runs $\#_{\text{best}}$ in which the best solution was found. For small instances, i.e., in Table 1, we also report best found solution values o and needed runtimes t in seconds for the exact branch-and-cut (B&C) approach from [5] as well as for the heuristic method

Table 1
Comparison of B&C, AMH, GA, and VNS on some small instances.

	B&C		*AMH*		*GA*					*VNS*				
Inst	*opt*	*t*	*o*	*t*	o_{best}	o_{avg}	t_{best}	t_{avg}	$\#_{\text{best}}$	o_{best}	o_{avg}	t_{best}	t_{avg}	$\#_{\text{best}}$
50-100-3-10	96	950	96	62	96	96.0 (0.0)	0.0	0.3	20	96	96.0 (0.0)	0.0	0.0	20
50-100-3-50	87	884	87	61	87	87.0 (0.0)	0.0	0.3	20	87	87.0 (0.0)	0.0	0.0	20
50-100-3-100	79	876	79	72	79	79.2 (0.8)	0.0	0.3	19	79	**79.0 (0.0)**	0.0	0.1	20
50-100-3-400	48	1007	48	89	48	48.0 (0.0)	0.0	0.3	20	48	48.0 (0.0)	0.0	0.1	20
50-100-6-10	95	-	95	95	95	95.0 (0.0)	0.0	1.0	20	95	95.0 (0.0)	0.0	0.1	20
50-100-6-50	83	-	83	51	83	83.0 (0.0)	0.0	0.4	20	83	83.0 (0.0)	0.0	0.1	20
50-100-6-100	70	-	76	73	70	70.9 (6.1)	0.0	0.4	19	70	**70.0 (0.0)**	0.0	0.1	20
50-100-6-400	35	-	35	80	35	35.0 (0.0)	0.0	0.4	20	35	35.0 (0.0)	0.0	0.1	20
50-100-10-10	96	1195	96	51	96	96.0 (0.0)	0.0	1.5	20	96	96.0 (0.0)	0.0	0.1	20
50-100-10-50	83	635	83	51	83	83.0 (0.0)	0.0	0.9	20	83	83.0 (0.0)	0.0	0.1	20
50-100-10-100	69	508	72	53	69	**69.0 (0.0)**	0.0	0.5	20	69	69.6 (1.7)	0.0	0.1	16
50-100-10-400	18	-	18	71	18	18.0 (0.0)	0.0	0.5	20	18	18.0 (0.0)	0.0	0.2	20

Table 2
Comparison of GA and VNS on some medium and large size instances.

	GA					*VNS*				
Inst	o_{best}	o_{avg}	t_{best}	t_{avg}	$\#_{\text{best}}$	o_{best}	o_{avg}	t_{best}	t_{avg}	$\#_{\text{best}}$
200-200-3-100	188	188.7 (0.3)	0.1	0.9	7	188	**188.0 (0.0)**	0.0	2.3	20
200-200-3-1000	156	156.0 (0.0)	0.0	0.8	20	156	156.0 (0.0)	0.0	2.6	20
200-200-3-2000	122	122.0 (0.0)	0.0	0.8	20	122	122.0 (0.0)	0.0	3.1	20
200-200-3-5000	61	61.0 (0.0)	0.0	0.8	20	61	61.0 (0.0)	0.0	3.3	20
200-200-6-100	184	185.8 (0.8)	1.3	7.0	8	184	**184.0 (0.0)**	0.1	3.7	20
200-200-6-1000	129	129.0 (0.0)	0.1	1.1	20	129	129.0 (0.0)	0.1	5.3	20
200-200-6-2000	74	74.0 (0.0)	0.1	1.2	20	74	74.0 (0.0)	0.1	5.7	20
200-200-6-5000	20	20.0 (0.0)	0.3	1.5	20	20	20.0 (0.0)	0.7	6.7	20
200-200-10-100	177	180.6 (1.7)	0.0	10.8	8	177	**177.0 (0.0)**	0.3	6.1	20
200-200-10-1000	100	100.7 (1.1)	0.4	1.7	11	100	**100.2 (0.4)**	3.5	9.2	16
200-200-10-2000	43	43.0 (0.0)	0.2	1.6	20	43	43.0 (0.0)	0.5	9.9	20
200-200-10-5000	3	4.3 (26.2)	0.3	1.9	3	3	**3.2 (12.5)**	4.0	10.4	16
200-200-20-100	167	167.4 (0.4)	1.6	33.3	12	167	167.7 (0.4)	3.7	8.6	8
200-200-20-1000	56	**56.7 (4.0)**	1.2	4.2	15	56	62.3 (6.9)	10.7	15.7	3
200-200-20-2000	8	10.7 (16.8)	0.8	4.4	1	8	**9.6 (12.5)**	8.5	17.9	6
200-200-20-5000	2	2.2 (18.3)	1.0	3.4	17	2	**2.0 (0.0)**	3.3	15.0	20
1000-1000-3-100	993	994.0 (0.0)	0.2	11.4	1	993	**993.0 (0.0)**	1.2	117.6	20
1000-1000-3-1000	978	978.0 (0.0)	1.9	9.0	20	978	978.0 (0.0)	0.9	124.3	20
1000-1000-3-10000	909	909.0 (0.0)	1.0	8.7	20	909	909.0 (0.0)	3.2	133.1	20
1000-1000-3-100000	494	498.1 (3.7)	3.4	11.4	18	494	**494.0 (0.0)**	18.8	154.6	20
1000-1000-6-100	991	991.0 (0.0)	0.6	12.6	20	990	**990.0 (0.0)**	29.4	206.7	20
1000-1000-6-1000	964	964.0 (0.0)	2.2	19.3	20	964	964.0 (0.0)	3.0	270.6	20
1000-1000-6-10000	839	840.1 (0.1)	3.8	15.0	9	839	**839.5 (0.1)**	84.1	285.4	15
1000-1000-6-100000	241	242.6 (1.4)	8.0	19.9	15	241	**241.0 (0.0)**	82.9	358.2	20
1000-1000-10-100	986	990.8 (0.2)	46.9	589.8	2	986	**987.5 (0.1)**	52.3	293.3	3
1000-1000-10-1000	945	945.5 (0.2)	19.2	52.9	18	945	**945.0 (0.0)**	75.5	413.8	20
1000-1000-10-10000	760	**763.9 (0.8)**	5.6	21.1	2	760	769.5 (1.1)	200.3	470.9	3
1000-1000-10-100000	69	**78.7 (14.6)**	9.6	25.1	3	69	95.1 (18.0)	380.2	616.8	1
1000-1000-20-100	976	982.6 (0.7)	74.1	1431.5	1	978	**978.0 (0.0)**	23.4	531.6	20
1000-1000-20-1000	907	**913.8 (0.7)**	120.8	575.8	7	907	917.4 (0.3)	173.7	764.8	1
1000-1000-20-10000	590	**608.8 (4.4)**	10.2	34.8	1	604	638.8 (3.1)	733.7	937.3	1
1000-1000-20-100000	3	**5.2 (33.5)**	22.8	45.1	1	4	9.3 (56.5)	909.2	1289.4	2

AMH from the same paper. For the branch-and-cut a dash indicates that the time limit of one hour has been reached and thus the found solution is not necessarily optimal. We note, that the results of the branch-and-cut approach and AMH have been obtained on a similar, yet slightly different hardware, and thus the runtimes are not directly comparable.

From Table 1, we conclude that VNS always found an optimal or best known solution for small instances very fast. Furthermore, on medium and large instances it slightly outperforms HGA with respect to average solution quality and appears to be more stable on average. With respect to the necessary runtime, the picture is more diverse, i.e., whether VNS or GA is faster heavily depends on the considered instance.

4 Conclusions

In this paper we proposed a robust and effective variable neighborhood search metaheuristic for the balanced location problem. This VNS utilizes large shaking neighborhoods in order to escape from local optima as well as an embedded variable neighborhood descent consisting of three neighborhood structures focusing on different solution aspects.

By a computational study, we showed that the VNS is able to reach optimal solutions very fast and consistently for small instances that can be solved to proven optimality with a previously proposed branch-and-cut approach. For larger instances, we showed that this VNS outperforms a genetic algorithm with respect to stability and average solution quality. In future work, we want to study a parallel version of this VNS in order to achieve a significant speed-up. Furthermore, its hybridization with exact methods appears to be a promising direction for future research.

References

[1] Drezner, T., Z. Drezner, and J. Guyse, *Equitable service by a facility: Minimizing the Gini coefficient*, Comp & Oper. Res. **36** (2009), 3240–3246.

[2] Filipović, V., J. Kratica, A. Savić, and Dj. Dugošija, *The modification of genetic algorithms for solving the balanced location problem*, Proceedings of the 5th Balkan Conference in Informatics, in press.

[3] Hansen, P., N. Mladenović, and J.A. Moreno-Pérez, *Variable neighbourhood search: algorithms and applications*, Annals Oper. Res. **175** (2010), 367–407.

[4] Kalcsics, J., S. Nickel, and M. Schröder, *Towards a unified territory design approach - Applications, algorithms and GIS integration*, Top **13** (2005), 1–56.

[5] Marin, A., *The discrete facility location problem with balanced allocation of customers*, Euro. J. Oper. Res. **210** (2011), 27–38.

[6] Stanimirović, Z., J.Kratica, and Dj. Dugosija, *Genetic algorithms for solving the discrete ordered median problem*, Euro. J. Oper. Res. **182** (2007), 983–1001.