The modification of genetic algorithms for solving the balanced location problem

Vladimir Filipović
Faculty of Mathematics,
University of Belgrade
Studentski trg 16/IV
11 000 Belgrade, Serbia
vladaf@matf.bg.ac.rs

Jozef Kratica
Mathematical Institute,
Serbian Academy of Sciences
and Arts, Kneza Mihaila 36/III
11 000 Belgrade, Serbia
jkratica@mi.sanu.ac.rs

Aleksandar Savić
Faculty of Mathematics,
University of Belgrade
Studentski trg 16/IV
11 000 Belgrade, Serbia
aleks3rd@gmail.com

Djordje Dugošija
Faculty of Mathematics,
University of Belgrade
Studentski trg 16/IV
11 000 Belgrade, Serbia
dugosija@matf.bg.ac.rs

ABSTRACT

In this paper is described the modification of the existing evolutionary approach for Discrete Ordered Median Problem (DOMP), in order to solve the Balanced Location Problem (LOBA). Described approach, named HGA1, includes a hybrid of Genetic Algorithm (GA) and a well-known Fast Interchange Heuristic (FIH). HGA1 uses binary encoding schema. Also, new genetic operators that keep the feasibility of individuals are proposed. In proposed method, caching GA technique was integrated with the GFI heuristic to improve computational performance. The algorithm is tested on standard instances from the literature and on large-scale instances, up to 1000 potential facilities and clients, which is generated by generator described in [**?**]. The obtained results are also compared with the existing heuristic from the literature.

Categories and Subject Descriptors

G.2.3 [**Mathematics of Computing**]: Discrete Mathematics—*applications*; I.2.8 [**Computing Methodologies**]: Artificial Intelligence—*problem solving, control methods, and search*

Keywords

genetic algorithms, evolutionary computations, discrete location, balanced allocation of customers.

1. INTRODUCTION

The balanced location problem (LOBA) was introduced in [**?**]. in order to consider a discrete facility location problem where the difference between the maximum and minimum number of clients allocated to every potential facility has to be balanced.

Some applications of introduced problem can be found in the field of Territory Design [**?**], where small geographic areas must be grouped into larger geographic clusters according to some planning criteria (for instance, political districting, solid waste collection, school districting). Another application is location of antennas for mobile phones. In [**?**] is studied one similar problem that addressed the minimization of the Gini index when a facility is located.

Two different integer programming formulations for LOBA were were introduced in [**?**], and several families of valid inequalities for these formulations are developed. Then, the exact Branch-and-Cut algorithm was developed which incorporated a good preprocessing techniques which allow to reduce the size of the largest formulation. To get good upper bound for exact algorithm, the interchange heuristic has been implemented. Since the number of available valid inequalities for this formulation is exponential, the most violated inequalities are separated at every node of the branching tree. Both formulations, ware tested in a computational framework in order to discriminate the most promising solution methods. The experimental results are reported on hard generated instances with up to 50 potential plants and 100 customers, since standard ORLIB p-median instances seems to be too easy for this problem.

2. PROBLEM DESCRIPTION

Let we have n clients and m potential facilities, and c_{ij} is distance between client i and potential facility j. We need to establish establish subset X of exactly p facilities, then every client is allocated to the closest facility from X, in order that difference between facility with most and facility with least number of allocated clients can be as small as possible.

Problem can be formulated as integer linear programming

(ILP) model introduced in [?]. Decision variables x are defined as follows: $x_{ij} = 1$ if client i is allocated to facility j and $x_{ij} = 0$ otherwise. Similarly, $y_j = 1$ if facility j is established and $y_j = 0$ otherwise.

3. PROPOSED HYBRID GA METHOD

In this section are described modifications of genetic algorithms for solving DOMP into the methods for solving LOBA. The main emphasis is new objective function and modification of the local search procedure, while all other GA aspects for solving DOMP from paper [?], are only summarized in Section 3.3.

3.1 Representation and objective function

Results from paper [?] indicate that hybrid genetic algorithm based on binary representation (noted as HGA1) outperforms integer encoded genetic algorithm (HGA2). Therefore we decided to adopt only the first evolutionary approach HGA1 in order to solve LOBA.

On the same way as in HGA1 algorithm (described in [?]), feasible solution for LOBA problem is represented by a m-dimensional binary vector with exactly p entries equal to 1. Each gene with value 1 in the genetic code denotes that a certain facility is established, while 0 denotes it is not. According to problem formulation, each client is assigned to its closest established facility. In order to obtain majority of correct individuals in the initial population, the probability of generating ones in the genetic code is set to p/m. The individuals which have a number of ones in their genetic code that is different from p (denoted as k, $k \neq p$) are corrected by adding $p-k$ ones at the end of the genetic code (if $p > k$), or by deleting $k - p$ ones from the end of the genetic code (if $p < k$). Previously described adding/deleting could also be performed at the beginning of the genetic code or in a random way.

After the application of the correction procedure all individuals are feasible, because the number of established facilities is fixed to p ($|X| = p$). The described correction is performed only during initialization phase of the algorithm, since the proposed genetic operators are designed to preserve the feasibility of individuals.

The choice of data structure is very important for the fast implementation of the objective function. Besides the transportation cost matrix, this implementation, as well as implementation described in [?], uses indexed arrays of facility sites. These arrays are sorted in non-decreasing order according to transportation cost for each client. The following notation is used:

> $y_j = 1$, if a facility is established at node j ($j \in X$), 0 if not ($j \notin X$),
>
> x_{ij}=1, if demands at client i are satisfied from a located facility j, 0 if not.

Method for computing objective function is similar to method used in HGA1. It consists of the two strategies. The choice of particular strategy depends on the number of established facilities p. The threshold is $e_0 = c \cdot \sqrt{m}$, where $c = 0.95$ is a constant.

If p is large ($p > e_0$), the algorithm is searching the array of indices (y), looking for the first facility j with $y_j = 1$. That facility i has minimal transportation cost for a given client i (x_{ij}=1). In such case, procedure is fast because the array (y) is dense. The number of steps for every customer is approximately m/p, and the overall run-time complexity is O($n * m/p$).

In case of small value of p, the previously described strategy gives poor results. It is slow because the array (y) is sparse. In that case, another strategy is performed: instead of using the array of indices, this strategy uses a sequence of ordinals (o). The sequence (o) contains only established facilities ($o_k = j$, $j \in X$, where y_j is k-th occurrence of 1 in the array y). By searching the sequence (o), the most suitable established facility for each customer is found. If o_k is minimum, then $x_{ij} = 1$ for $j = o_k$. The array (y) is sparse, so the length of sequence (o) is small, which guarantees fast computation. The construction phase for (o) has O(m) time requirement. Nevertheless, the algorithm creates the sequence (o) only once, and uses it n times (once for every customer). This procedure requires p steps for each customer, so the overall running-time complexity is O($m+n*p$) in this case.

Table 1: Results on instances from [?]

Inst	*Opt*	*Best*	*AMHs*	*AMHt*	*GAs*	*GAt*
20-20-3-10	15	15	*opt*	0	*opt*	0.183
20-20-3-50	3	3	*opt*	1	*opt*	0.19
20-20-6-10	12	12	*opt*	0	*opt*	0.194
20-20-6-50	6	6	*opt*	0	*opt*	0.192
30-30-3-10	25	25	*opt*	3	*opt*	0.2
30-30-3-30	22	22	*opt*	3	*opt*	0.21
30-30-3-100	9	9	10	3	*opt*	0.214
30-30-4-10	24	24	*opt*	3	*opt*	0.217
30-30-4-30	17	17	*opt*	3	*opt*	0.221
30-30-4-100	9	9	10	3	*opt*	0.217
30-50-3-10	45	45	*opt*	4	*opt*	0.205
30-50-3-50	34	34	*opt*	6	*opt*	0.215
30-50-3-100	27	27	*opt*	6	*opt*	0.214
30-50-3-200	13	13	*opt*	5	*opt*	0.22
30-50-6-10	45	45	46	4	*opt*	0.317
30-50-6-50	34	34	*opt*	5	*opt*	0.253
30-50-6-100	19	19	*opt*	4	*opt*	0.261
30-50-6-200	10	10	*opt*	5	*opt*	0.27
30-50-10-10	45	45	46	3	*opt*	0.494
30-50-10-50	27	27	*opt*	4	*opt*	0.262
30-50-10-100	18	18	20	3	*opt*	0.265
30-50-10-200	5	5	*opt*	3	*opt*	0.302
30-100-3-10	94	94	*opt*	9	*opt*	0.216
30-100-3-50	85	85	*opt*	9	*opt*	0.214
30-100-3-100	75	75	*opt*	14	*opt*	0.224
30-100-3-200	63	63	*opt*	9	*opt*	0.23
30-100-6-10	92	92	*opt*	10	*opt*	0.276
30-100-6-50	84	84	*opt*	8	*opt*	0.268
30-100-6-100	64	64	*opt*	8	*opt*	0.276
30-100-6-200	48	48	*opt*	8	*opt*	0.285
30-100-10-10	96	96	*opt*	5	*opt*	0.76
30-100-10-50	77	77	*opt*	5	*opt*	0.421
30-100-10-100	63	63	*opt*	8	*opt*	0.307
30-100-10-200	41	41	*opt*	8	*opt*	0.295
50-50-3-10	46	46	*opt*	34	*opt*	0.259
50-50-3-50	39	39	*opt*	45	*opt*	0.263
50-50-3-100	32	32	*opt*	35	*opt*	0.264
50-50-3-400	11	11	*opt*	43	*opt*	0.269
50-50-6-10	43	43	*opt*	38	*opt*	0.301
50-50-6-50	33	33	*opt*	37	*opt*	0.324
50-50-6-100	24	24	*opt*	36	*opt*	0.328
50-50-6-400	3	3	*opt*	37	*opt*	0.367
50-50-10-10	43	43	45	25	*opt*	0.86
50-50-10-50	30	30	*opt*	30	*opt*	0.6
50-50-10-100	20	20	23	33	*opt*	0.384
50-50-10-400	2	2	*opt*	28	*opt*	0.4
50-100-3-50	87	87	*opt*	61	*opt*	0.268
50-100-3-100	79	79	*opt*	72	*opt*	0.263
50-100-3-400	48	48	*opt*	89	*opt*	0.271
50-100-6-10	95	95	*opt*	95	*opt*	0.994
50-100-6-50	83	83	*opt*	51	*opt*	0.37
50-100-6-100	70	70	76	73	*opt*	0.352
50-100-6-400	35	35	*opt*	80	*opt*	0.382
50-100-10-10	96	96	*opt*	51	*opt*	1.505
50-100-10-50	83	83	*opt*	51	*opt*	0.884
50-100-10-100	69	69	72	53	*opt*	0.455
50-100-10-400	18	18	*opt*	71	*opt*	0.522
100-100-3-1		1			*opt*	1.283
100-100-3-10		96			*best*	0.418
100-100-3-50		91			*best*	0.42
100-100-3-100		86			*best*	0.41
100-100-3-1000		41			*best*	0.416
100-100-3-2000		11			*best*	0.443
100-100-3-2500		3			*best*	0.469
100-100-3-3000		1			*best*	0.524
100-100-3-3500		1			*best*	0.475
100-100-3-5000		1			*best*	0.576

3.2 Local search procedure

The fast interchange heuristic showed to be effective for hybridization with other methods for solving the p-median and p-center problem (see [**?**]). This heuristic can not be directly applied to LOBA without modification. The major difficulty is to compute the difference between the objective values each time when an interchange is performed. In paper [**?**], another generalization of the interchange heuristic (named GFI) is developed.

In the description of the GFI heuristic, the following notation is used:

j: the index of established facility to be deleted from the current solution X

k : the index of non-established facility to be inserted in the current solution X

i: client

$imin(i)$: the index of opened facility that is closest to client i, i.e. $imin(i) = arg\ min_{j \in X} C_{i,j}$

obj: the objective value of the interchanged solution $X \bigcup \{k\} \backslash \{j\}$

$Num(l)$: the number of clients that are assigned to location l

For given (j,k), our GFI heuristic is outlined as follows:

for i:=1 **to** n **do**
 if $(C_{i,j} \geq C_{i,imin(i)})$ **and** $(C_{i,imin(i)} \geq C_{i,k})$
 then
 $imin(i) := k$;
 else
 if(j=$imin(i)$) **and** $(C_{i,j} < C_{i,k})$
 then
 $C_{i,r} := min_{\ s \in X \cup \{k\} \backslash \{j\}}\ C_{i,s}$;
 $imin(i) := r$;
 endif
 endif
endfor
QSort($imin$);
$fmax := max_{\ l \in X}\ Num(l)$;
$fmin := min_{\ l \in X}\ Num(l)$;
obj := $fmax$ - $fmin$;

Function QSort arranges the array $imin$ in non-decreasing order of transportation costs $C_{i,imin(i)}$. Computing min in second case is implemented on the same way as computing the objective value.

Moreover, for already cached individuals the GFI heuristic does not follow a scheme given above, and reads obj value directly from the hash-queue table, as described in [**?**].

3.3 Other GA aspects

Proposed GA implementation uses fine-grained tournament selection - FGTS (described in [**?**]), with parameter the average tournament size set on 5.4.

The standard crossover usually randomly chooses crossover points and simply exchanges the segments of the parents' genetic codes. However, this approach cannot be applied in HGA1, since it may produce incorrect offspring for LOBA.

The number of ones in an offspring may become different from p, although its parents had exactly p ones in their genetic codes. To overcome this problem, the basic crossover is modified in HGA1 and same modification states here: The operator is simultaneously tracing the genetic codes of the parents from right to left searching the position i on which the first parent has 1 and second 0. Similar process is performed starting from the left side of the genetic codes. The operator is searching the position j where the first parent has 0 and the other 1. Genes are exchanged on the i-th and j-th position and the number of located hubs in each individual remains unchanged. Finding positions i and j and exchanging genes is repeated until $j \leq i$. Probability of the crossover is 0.85.

The mutation operator (with frozen bits) is performed by changing a randomly selected gene in the genetic code (0 to 1, 1 to 0). In order to keep the mutated individual feasible, it is necessary to count and compare the numbers of mutated ones and zeros for each individual. If these numbers are not equal, we have to mutate additional genes in order to equalize them. In this way, the mutation operator is preserving p ones in the genetic code. Applied mutation rates are constant through all generations ($0.2/m$, for the basic mutation rate and $0.5/m$, for the mutation rate on frozen genes).

The running time of the GA is improved by caching (see [**?**]). Evaluated objective functions are stored in a hash-queue data structure, together with the corresponding genetic codes. When the same genetic code is obtained again during the GA, the objective value is taken from the hash-queue table, instead of computing the objective function. The cache size is limited to 5000 in this implementation.

The population contains of 150 individuals. A steady-state generation replacement with elitist strategy is used. In this replacement scheme only $N_{nonel} = 50$ individuals are replaced in every generation, while the best $N_{elite} = 100$ individuals are directly passing in the next generation. Duplicated individuals are removed from each generation. Their fitness values are set to zero, so that selection operator prevents them from entering the next generation. Individuals with the same objective function but different genetic codes in some cases may dominate in the population, so it is useful to limit their appearance to some constant. In the method we propose, number of such similar individuals is limited up to 40.

4. EXPERIMENTAL RESULTS

A computational experiment was carried out in order to check the performance of our solution methods. All tests were carried out on an Intel 2.5 GHz single processor with 1GB memory, under Windows operating system. The algorithm was coded in C programming language.

The finishing criterion for proposed method is:

maximal number of generations $N_{gen} = 5000$ or best individual (or best objective) value remains unchanged through $N_{rep} = 2000$ successive generations. For every problem instance, algorithm is executed 20 times. Firstly, we tested our method with a problem data set that creates and uses Marin in [**?**] and obtained results are summarized in Table 1.

In Table 1, column $Inst$ indicates problem instance. Problem instances are denoted with mark $m - n - p - pl$, as described in [**?**], where m is number of facilities, n is number of clients, p is number of facilities to be established and pl is perturbation level, defined in [**?**]. Column Opt represents the optimal solution, column $Best$ is the best known solution so far, $AMHs$ is solution obtained by heuristic described in [**?**], $AMHt$ is time for heuristic described in [**?**] (in seconds),

Table 2: Results on medium-size instances

Inst	*GAs*	*GAt*	*Inst*	*GAs*	*GAt*
200-200-3-100	188	0.924	200-1000-10-100	976	61.083
200-200-3-1000	156	0.794	200-1000-10-1000	885	5.656
200-200-3-2000	122	0.773	200-1000-10-5000	566	5.631
200-200-3-5000	61	0.782	200-1000-10-20000	115	7.943
200-200-6-100	184	7.019	200-1000-20-100	964	146.847
200-200-6-1000	129	1.146	200-1000-20-1000	806	58.312
200-200-6-2000	74	1.195	200-1000-20-5000	358	15.542
200-200-6-5000	20	1.529	200-1000-20-20000	8	13.344
200-200-10-100	177	10.805	300-300-3-100	290	1.201
200-200-10-1000	100	1.716	300-300-3-1000	264	1.186
200-200-10-2000	43	1.569	300-300-3-2000	234	1.209
200-200-10-5000	3	1.877	300-300-3-5000	189	1.19
200-200-20-100	167	33.345	300-300-6-100	285	1.62
200-200-20-1000	56	4.167	300-300-6-1000	242	2.146
200-200-20-2000	8	4.436	300-300-6-2000	200	1.901
200-200-20-5000	2	3.428	300-300-6-5000	122	1.988
200-500-3-100	488	1.365	300-300-10-100	279	24.137
200-500-3-1000	454	1.052	300-300-10-1000	214	2.714
200-500-3-5000	335	1.028	300-300-10-2000	160	2.559
200-500-3-10000	237	1.134	300-300-10-5000	65	3.168
200-500-6-100	484	19.948	300-300-20-100	270	76.374
200-500-6-1000	424	2.054	300-300-20-1000	165	8.864
200-500-6-5000	240	2.137	300-300-20-2000	89	7.732
200-500-6-10000	120	2.581	300-300-20-5000	8	7.657
200-500-10-100	476	28.755	300-1000-3-100	990	3.406
200-500-10-1000	389	2.645	300-1000-3-1000	964	3.126
200-500-10-5000	157	3.13	300-1000-3-5000	879	3.439
200-500-10-10000	41	3.612	300-1000-3-20000	642	3.646
200-500-20-100	465	81.106	300-1000-6-100	985	5.089
200-500-20-1000	319	8.93	300-1000-6-1000	938	4.973
200-500-20-5000	56	7.191	300-1000-6-5000	786	5.676
200-500-20-10000	4	6.885	300-1000-6-20000	423	7.526
200-1000-3-100	988	2.567	300-1000-10-100	979	66.114
200-1000-3-1000	954	1.996	300-1000-10-1000	907	8.202
200-1000-3-5000	817	1.877	300-1000-10-5000	679	8.216
200-1000-3-20000	514	2.221	300-1000-10-20000	232	10.619
200-1000-6-100	984	43.941	300-1000-20-100	970	258.137
200-1000-6-1000	922	3.717	300-1000-20-1000	845	212.015
200-1000-6-5000	697	3.911	300-1000-20-5000	487	25.57
200-1000-6-20000	270	4.968	300-1000-20-20000	42	24.425

Table 3: Results on large-scale instances

Inst	*GAs*	*GAt*	*Inst*	*GAs*	*GAt*
500-500-3-100	492	3.358	500-1000-10-100	984	50.515
500-500-3-1000	472	3.816	500-1000-10-1000	927	162.797
500-500-3-5000	414	2.579	500-1000-10-5000	773	12.502
500-500-3-20000	266	2.773	500-1000-10-20000	396	15.804
500-500-6-100	487	9.892	500-1000-20-100	974	471.569
500-500-6-1000	452	3.756	500-1000-20-1000	873	365.11
500-500-6-5000	357	4.06	500-1000-20-5000	617	18.607
500-500-6-20000	152	5.489	500-1000-20-20000	152	23.552
500-500-10-100	484	23.461	1000-1000-3-100	993	11.384
500-500-10-1000	428	16.035	1000-1000-3-1000	978	9.002
500-500-10-5000	292	5.48	1000-1000-3-10000	909	8.724
500-500-10-20000	62	7.43	1000-1000-3-100000	494	11.433
500-500-20-100	474	223.522	1000-1000-6-100	991	12.568
500-500-20-1000	378	190.226	1000-1000-6-1000	964	19.345
500-500-20-5000	178	11.23	1000-1000-6-10000	839	15.036
500-500-20-20000	4	12.606	1000-1000-6-100000	241	19.938
500-1000-3-100	992	7.131	1000-1000-10-100	986	589.789
500-1000-3-1000	972	5.816	1000-1000-10-1000	945	52.932
500-1000-3-5000	912	5.082	1000-1000-10-10000	760	21.136
500-1000-3-20000	741	5.333	1000-1000-10-100000	69	25.08
500-1000-6-100	987	20.957	1000-1000-20-100	976	1431.506
500-1000-6-1000	951	8.408	1000-1000-20-1000	907	575.818
500-1000-6-5000	849	8.562	1000-1000-20-10000	590	34.782
500-1000-6-20000	562	9.152	1000-1000-20-100000	3	45.09

GAs is solution obtained by proposed method and *GAt* is execution time for proposed method. Empty cell in the table indicates that result is not obtained, or not known.

The data from Table 1 show that the proposed method reached optimal solution for all instances where optimal solution is known in advance. In all other cases, encountered in Table 1, the proposed method reached best known solution so far. Concerning execution time, proposed method clearly outperforms heuristic from [**?**].

Paper [**?**] also describes the structure of problem data set generator for LOBA. It will be useful to analyze behavior of the proposed algorithm on problem data set with larger instances, so we used Marin's generator to create new problem data set with larger instances and we tested our method on that data set. Again, algorithm is executed 20 times for each LOBA problem instance. Tables 2 and 3 contain results that are obtained during second computational experiment. Columns in those tables have same meaning as columns in Table 1.

For LOBA problem instances in Table 2 and Table 3, exact algorithms execution lasted for unacceptable long period of time, so exact solutions are not known. Presented data show GA results, and time(in seconds) needed to obtain such solutions. On all large scale LOBA instances, problems are solved in reasonable running time.

5. CONCLUSIONS

In this paper, we present the modification of one existing evolutionary approach for Discrete Ordered Median Problem (DOMP), in order to solve the Balanced Location Problem (LOBA). We are using binary representation, crossover and mutation operators constructed to keep the individuals feasible. In order to increase the diversity of genetic material, proposed method use mutation with frozen genes. Solution quality is further improved by using efficiently implemented GFI heuristic. Caching GA, as well as its combination with the GFI, is additionally improving performances of the algorithms. Comprehensive computational experiments on Marin's instances and new instances created with Marin's generator demonstrate the robustness of the proposed algorithms with respect to the solution quality and running times. Computational results show that the proposed algorithm outperform existing metaheuristic. The proposed method provides solutions for some LOBA problems not yet investigated in the literature. Further research can be directed to parallelization of the proposed method and testing it on powerful multiprocessor system.

6. ACKNOWLEDGMENTS

This research was partially supported by the Serbian Ministry of Education and Science under project 174010. Authors would like to thank Alfredo Marin for his help in obtaining problem data set and generator for the the balanced location problem.