Proceedings of the Federated Conference on Computer Science and Information Systems pp. 165–169
DOI: 10.15439/2019F158
ISSN 2300-5963 ACSIS, Vol. 18

Customized Genetic Algorithm for Facility Allocation using *p*-median

S. D. de S. Silva
Universidade Federal do Amazonas
Manaus, Brasil
Email: srgio.deo@gmail.com

M. G. F. Costa
Universidade Federal do Amazonas
Manaus, Brasil
Email: mcosta@ufam.edu.br

C. F. F. Costa Filho
Universidade Federal do Amazonas
Manaus, Brasil
Email: ccosta@ufam.edu.br

***Abstract*—The p-median problem is classified as a NP-hard problem, which demands a long time for solution. To increase the use of the method in public management, commercial, military and industrial applications, several heuristic methods has been proposed in literature. In this work, we propose a customized Genetic Algorithm for solving the p-median problem, and we present its evaluation using benchmark problems of OR-library. The customized method combines parameters used in previous studies and introduces the evolution of solutions in stationary mode for solving PMP problems. The proposed Genetic Algorithm found the optimum solution in 37 of 40 instances of p-median problem. The mean deviation from the optimal solution was 0.002% and the mean processing time using CPU core i7 was 17.7s.**

I. Introduction

Facility location problems (FLP) are usually employed for solving public, commercial, industrial and military problems. In these problems, service demand points must be attended by a limited number of facilities. The p-median problem (PMP) is a type of FLP problem that aims searching a given location that minimizes the sum of the distances between N demand points and the nearest facility [1].

The computational complexity theory classifies the PMP as a non-polynomial hard problem (NP-hard problem). Meta-heuristic methods are usually used for solving NP-hard problems whose optimal solution method does not exist or is not known: Greedy Interchange (GI) [2], Neighborhood (N) and Exchange [2], Semi-Lagrangean relaxation [3], Simulated Annealing (SA) [4], Tabu Search (TS) [5], Genetic Algorithm (GA) [6-7].

To enable comparative studies of these methods, benchmarking data bases are used. The Operational Research (OR) library [8] and the Traveling Salesman Problem (TSP) [9] are the most used ones [6,7,10,11,12,13].

In the comparisons made in [6] and [11], the GA heuristic stands out as the best one in terms of time and precision of solution. Nevertheless, concerning the precision of the solutions, the GAs presented in these works have a worse result than the customized GAs, presented in [7], as well as when compared to GA combinations with other heuristics, presented in [11].

In [6], the authors used the OR-library [14] and two others more simple databases to evaluate several methods used in PMP solution: ADE (Alp, Drezner and Erkut) GA, Gamma Heuristic (GH), SA, Myopic, Exchange and N. The algorithm known as ADE GA performs a greedy search using the genetic material of two individuals randomly selected, evaluating all the possible combinations of generated offspring. The algorithm found solutions with an average distance from the optimal solution (OPT solution) of 0.41%, in 85% of the OR-library problems, and an average time of 18 seconds.

In [11] the authors performed a comparative study of a GA, an N algorithm and a hybrid GA and N algorithm, using the TSP-library. The GA proposed by the authors is similar to ADE GA, differing only in the use of an algebraic method to select a pair of parents. The GA converged to a solution in less time than the other heuristics. The CPU average time was 126.8min. The GA presented solutions with an average distance from the OPT solution of 0.000016%, and an average time of 391.5min.

In [7], a simple GA was compared to ADE GA, using a subset of OR-library. This GA investigates the use of *p* centroids to find the initial solutions of the algorithm. The GA found OPT solutions in 14 of the 15 subset problems. The average CPU time was 60.1s and 0.2s, for the simple GA and ADE GA, respectively. The average deviation was 0.007% and 0.02% for simple GA and ADE GA, respectively.

Table I shows a summary of the main characteristics of the GA used for solving the *p*-median problem in [6, 7, 11].

This work aims investigating the customization of GA for solving PMP problems. Three steps of the GA are investigated: selection operator, crossover operator and population updating. This investigation has the objective of generating a best performance of GA in finding OPT solutions for PMP problems.

The random selection operator employed in [6] and [11] does not take into account the individual's fitness when they are selected for crossover. The ranking selection operator employed in [7] assigns a selection probability to individuals directly proportional to their position in a ranking of the fitness function. In this work we investigate the use of the roulette wheel selection operator. The difference between the

This work was supported by *Samsung Eletronica da Amazonia*, under the terms of the Brazilian Federal Law number 8.387/91.

TABLE I.
GENETIC ALGORITHM CHARACTERISTICS USED FOR SOLVING THE P-MEDIAN PROBLEM IN [6, 7, 11].

Paper	Data base	GA characteristics			Results	
		Selection	Crossover	Heuristics studied	GA Deviation from OPT	Faster Heuristic
[6]	OR - Library, Alberta, Galvão e Koerkel	Random	Merging	ADE, GH, SA, Myopic, Exchange, BV	Up to 0.41% from OPT at 85% of OR problems. 0% at Alberta problems.	ADE
[11]	TSP – Library	Random	Merging	GA [11], BV, Hybrid between GA [11] and N	Up to 0.008% from OPT at 100% of TSP problems.	GA [11]
[7]	OR – Library (15 problems)	Ranking-based	Partial Match	ADE, GA [7]	GA [7]	ADE GA

ranking operator and the roulette wheel selection operator is that, the last one assign individuals a selection probability directly proportional to their fitness value.

We also propose using the single-point crossover operator. Differently from the merging operators [6,11] and partial match operators [7], the single-point crossover operator generate offspring without evaluating the parents. This implies in less processing time demand.

At last, we propose use a steady-state population updating [15]. In this updating mode, the fitness of children is compared to their parent's fitness. When the fitness value of the offspring is lower than their father's fitness, they are discarded. Offspring with better fitness values than their fathers are preserved with a probability of 75%.

The results obtained in this study are compared with the results obtained with the ADE GA [6] and simple GA [7]. For this comparison, we employed PMPs of OR-library and did a benchmark of the machines used for simulations in these previous works.

II. METHODS

A. Proposed Genetic Algorithm

Genetic Algorithm is a stochastic optimization algorithm, inspired by the theory of evolution of Charles Darwin [16]. Since its proposition, it has been effectively applied in the solution of complex problems, like TSP [9] and PMP [6,7,11].

In GA, initially, a population of chromosomes is randomly generated. In the sequence, the individuals of this population are modified by applying evolution operators, iteratively. A chromosome represents a solution to the problem. The fitness value of each chromosome is evaluated through an objective function of the problem.

The implementation of GA usually consists of three steps: the definition of the genetic codification model, the definition of the objective function and the parameterization of the evolution operators.

In this work, the genetic codification model uses the facility indexes and the objective function is given by the PMP. The structure of the proposed GA is presented in the steps of Algorithm 1.

Algorithm 1 Proposed Genetic Algorithm

Begin
Randomly generate the initial population
Compute fitness of population
Repeat for x generations
  Roulette wheel selection of 2 parents
  One-point crossover, at a 95% probability
  One-gene random mutation, at a 5% probability
  Compute fitness
  Replace the parents with lower fitness than the children, at a 75% probability
  Introduce a random chromosome to the population
Until population has converged
End

Genetic codification

As stated before, the genetic codification uses the facility indexes. The same approach was also used in [6, 7, 11]. Figure 1 shows an encoded chromosome representing a solution in a PMP problem with 8 facilities to be allocated among 100 possible locations.

1	20	31	4	76	91	62	100

Fig. 1 Example of an encoded chromosome used in a PMP problem with 8 facilities

Compute fitness

According to equation 1, the goal of the PMP is minimize f: the sum of the distances between the demand points and the nearest facility.

$$f = \sum_{i=1}^{n} \sum_{j=1}^{p} d_{ij} x_{ij} \tag{1}$$

$$d_{ij} = \text{distance between point i and point j}$$

$$x_{ij} = \begin{cases} 1, \text{if demand i is attended by j} \\ 0, \text{otherwise} \end{cases}$$

$$n = number\ of\ total\ locations$$

$$p = number\ of\ medians$$

Roulette Wheel Selection

The selection operator used is the roulette wheel operator [17,18]. In this study, the selection operator assigns a probability value to each individual that is inversely proportional to its fitness value. The inversely dependence is due to the fact that, in the *p*-median problem, best individuals are those with lower *f* values, given in equation (1). Therefore, fitter individuals are the most likely to have children. This behavior favors the generation of more fit individuals. Table II illustrates the probability values used by different selection operators for four individuals.

TABLE II.
SELECTION OPERATORS CHARACTERISTICS

Chromosome	Fitness Value	Probability of selection operator (%)		
		Roulette Wheel	Random	Ranking
1	200	28.8	25	30
2	900	6.3	25	20
3	100	57.6	25	40
4	800	7.2	25	10

One-point Crossover

The one-point crossover operator is used in this study [18]. This operator randomly generates a reference point to permutate genes between fathers. The crossover probability used is 95%. Figure 2 illustrates the genetic permutation performed by the one-point crossover operator. To avoid repeated indexes in the offspring, we do a scan in the genes of each child, and replace the repeated index with another value randomly selected.

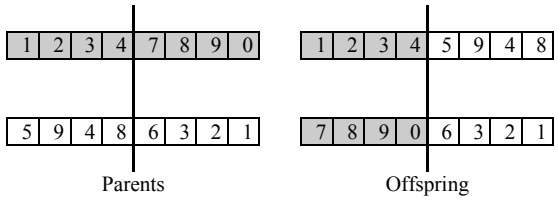

Fig. 2 Illustration of one-point crossover operation

One-gene random mutation

The mutation operator used randomly selects one gene [7], with probability of 5%, and performs a mutation. Figure 3 illustrates the mutation operator. One gene with index value of 5 is selected and replaced with the index value of 7. The replacing value is random selected.

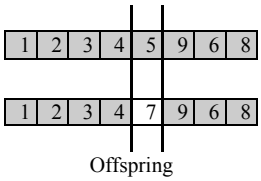

Fig. 3 Illustration of one-gene mutation operation

B. Metrics

In this study, the metrics used for performance evaluation of GA algorithms are the number of optimal solutions found, the percentual deviation of a non-optimal solution from the OPT solution, and the algorithm convergence time.

C. PMP data base for benchmarking

Aiming to compare the results obtained in this study with the results obtained in other two works [6,7], we use the PMP section of OR-library. For each problem, are given: the number of points, *N*; the number of facilities, *p*; the OPT solution and a matrix with the distances between each pair of points.

D. CPU benchmarking

The algorithm convergence time depends on the CPU model and the clock of the machine used for its implementation. Therefore, to compare the results of the GA used in this study with the GAs used in [6,7], we performed the benchmark between CPUs using the Dhrystone (D) method [19].

Using a default algorithm with integer numbers, the D method assigns a numeric value to each CPU. This value represents the number of millions of Dhrystone instructions processed per second (DMIPS) per MHz of clock. The DMIPS value of the machine used in this study (Core I7 7500U) is made equal to 1. The DMIPS values of the machines used in [6,7] are then divided by it and ratio values are obtained. The last column of Table III shows these ratio values. As shown, the machines used in [6] and [7] process 15.1% and 104.1%, respectively, of the DMIPS processed by the machine used in this study.

TABLE III.
BENCHMARK COMPARISON OF THREE DIGITAL COMPUTERS

CPU	Clock (MHz)	DMIPS/ MHz	Product	Ratio
Pentium III [6]	733	3.4	2492.2	0.151
Core I7 4770K [7]	2000	8.57	17140	1.041
Core I7 7500U This study	1800	9.1	16380	1

Product = (Clock*DMIPS/MHz) Ratio= (Product/16380)

III. RESULTS

Table IV shows, for the GA proposed in this study, and for the GAs proposed in [6] and [7], the following results: the number of OPT solutions; the percentual deviations from non-optimal solutions to OPT solutions and the GA algorithm processing time. The processing time of the GAs proposed in [6] and [7] are multiplied by the ratio value shown in Table III. Similarly to [6], the results of this study were produced by a C++ code, implementing 10 runs for each one of the 40 OR-library PMP problems. The best results are in bold.

TABLE IV.
EXPERIMENTAL RESULTS

Problem	*N*	*p*	Optimal Solution	Number of solutions	*p*/N (%)	ADE GA [6]		GA [7]		GA proposed (GAP)		Best deviation from optimal solution		
						Fitness value	Time (s)	Fitness value	Time (s)	Fitness value	Time (s)	ADE GA	GA	GAP
Pmed1	100	5	5819	75287520	5.00	OPT*	0.015	OPT	0.104	OPT	**0.001**	0	0	0
Pmed2	100	10	4093	1.73E+13	10.00	OPT	0.015	OPT	0.94	OPT	**0.008**	0	0	0
Pmed3	100	10	4250	1.73E+13	10.00	OPT	0.03	OPT	0.209	OPT	**0.003**	0	0	0
Pmed4	100	20	3034	5.36E+20	20.00	OPT	0.03	OPT	1.3	OPT	**0.026**	0	0	0
Pmed5	100	33	1355	2.95E+26	33.00	OPT	**0.045**	OPT	3.3	OPT	0.046	0	0	0
Pmed6	200	5	7824	2.54E+09	2.50	OPT	0.06	OPT	2.7	OPT	**0.005**	0	0	0
Pmed7	200	10	5631	2.25E+16	5.00	OPT	0.075	OPT	4.1	OPT	**0.026**	0	0	0
Pmed8	200	20	4445	1.61E+27	10.00	OPT	**0.105**	OPT	14.8	OPT	0.129	0	0	0
Pmed9	200	40	2734	2.05E+42	20.00	OPT	**0.181**	OPT	32.3	OPT	0.519	0	0	0
Pmed10	200	67	1255	1.45E+54	33.50	1256	0.301	OPT	41.4	OPT	1.2	0.080	0	0
Pmed11	300	5	7696	1.96E+10	1.67	OPT	0.256	OPT	28.8	OPT	**0.002**	0	0	0
Pmed12	300	10	6634	1.4E+18	3.33	OPT	0.181	OPT	47.8	OPT	**0.066**	0	0	0
Pmed13	300	30	4374	1.73E+41	10.00	OPT	**0.316**	OPT	78.4	OPT	0.64	0	0	0
Pmed14	300	60	2968	9.04E+63	20.00	OPT	**0.663**	OPT	301.8	OPT	2.9	0	0	0
Pmed15	300	100	1729	4.16E+81	33.33	1733	0.949	1731	343.6	OPT	14.8	0.231	0.116	0
Pmed16	400	5	8162	8.32E+10	1.25	OPT	0.346	-	-	OPT	**0.009**	0	-	0
Pmed17	400	10	6999	2.58E+19	2.50	OPT	0.361	-	-	OPT	**0.096**	0	-	0
Pmed18	400	40	4809	1.97E+55	10.00	OPT	**0.843**	-	-	OPT	0.999	0	-	0
Pmed19	400	80	2845	4.23E+85	20.00	2846	2	-	-	OPT	42.2	0.035	-	0
Pmed20	400	133	1789	1.3E+109	33.25	1792	0.949	-	-	OPT	15.95	0.168	-	0
Pmed21	500	5	9138	2.55E+11	1.00	OPT	0.572	-	-	OPT	**0.016**	0	-	0
Pmed22	500	10	8579	2.46E+20	2.00	OPT	0.678	-	-	OPT	**0.107**	0	-	0
Pmed23	500	50	4619	2.31E+69	10.00	OPT	2.4	-	-	OPT	**2.31**	0	-	0
Pmed24	500	100	2961	2E+107	20.00	2962	3.2	-	-	OPT	15.7	0.034	-	0
Pmed25	500	167	1828	7.9E+136	33.40	1832	4.8	-	-	OPT	105.9	0.219	-	0
Pmed26	600	5	9917	6.37E+11	0.83	OPT	1	-	-	OPT	**0.013**	0	-	0
Pmed27	600	10	8307	1.55E+21	1.67	OPT	1.2	-	-	OPT	**0.16**	0	-	0
Pmed28	600	60	4498	2.77E+83	10.00	4499	3.7	-	-	OPT	23.96	0.022	-	0
Pmed29	600	120	3033	1E+129	20.00	3035	6.6	-	-	OPT	93.422	0.066	-	0
Pmed30	600	200	1989	2.5E+164	33.33	1997	11.9	-	-	OPT	251.54	0.402	-	0
Pmed31	700	5	10086	1.38E+12	0.71	OPT	2.2	-	-	OPT	**0.035**	0	-	0
Pmed32	700	10	9297	7.3E+21	1.43	OPT	2	-	-	OPT	**0.224**	0	-	0
Pmed33	700	70	4700	3.37E+97	10.00	OPT	**6.8**	-	-	OPT	11.73	0	-	0
Pmed34	700	140	3013	5E+150	20.00	3015	9.8	-	-	3014	39.94	0.066	-	0.033
Pmed35	800	5	10400	2.7E+12	0.63	OPT	2.3	-	-	OPT	**0.048**	0	-	0
Pmed36	800	10	9934	2.8E+22	1.25	OPT	2.8	-	-	OPT	**0.232**	0	-	0
Pmed37	800	80	5057	4.1E+111	10.00	5058	11.4	-	-	5058	33.43	0.02	-	0.02
Pmed38	900	5	11060	4,87E+12	0.56	OPT	4.3	-	-	OPT	**0.104**	0	-	0
Pmed39	900	10	9423	9.14E+22	1.11	OPT	4	-	-	OPT	**0.256**	0	-	0
Pmed40	900	90	5128	5.1E+125	10.00	5133	19.9	-	-	5130	112.53	0.098	-	0.039
Average results Pmed1-15						0.2s		60.1s		1.35s		0.0154	0.007	0
Average results Pmed1-40						2.7s		-		17.7s		0.0360	-	0.002
Number of problems solved optimally						28		14		37				

IV. Discussion

A. Proposed GA vs ADE GA [6]

The GA proposed in this study achieved OPT solutions in 37 of the 40 PMPs shown in Table IV. The non-OPT solutions present a mean deviation of 0.002% from OPT solution, corresponding to a mean time of 17.7s. ADE GA [6] presents OPT solutions in 28 of the 40 PMPs. The non-OPT solutions present a mean deviation of 0.036% from OPT solutions, corresponding to a mean time of 2.7s in a CPU Core I7 7500U at 1.8GHz.

Considering the 28 PMPs that both methods achieved OPT solutions, the proposed GA and the ADE GA [6] achieved best results in 21 and 7 of them, respectively. In the 7 PMPs that ADE GA [6] achieved best results, 6 of them occurred between Pmed1 and Pmed20. This range corresponds to less complex problems. To evaluate the performance difference between the two methods, we applied a Qui-Square test in the following 2x2 contingency table: $[21\ 7; 7\ 21]$, and found $\chi^2 = 14$. For 1 degreed freedom, and a significance level of 99%, the critical level is $t_c = 6.63$. As $\chi^2 > t_c$, the difference between the proposed GA algorithm and ADE GA [6] is statistically significant.

From Table IV we also observe that when the ratio p/N increases, ADE GA [6] presents results significantly lower than the results obtained in this study. In the range Pmed21 to Pmed40, ADE GA [6] achieved OPT solutions in 10 of the 20 PMPs, with mean deviation of 0.046% from the OPT solutions, while the GA proposed in this study achieved OPT solutions in 17 of the PMPs, with mean deviation of 0.004% from the OPT solutions. For the instances Pmed5, Pmed10, Pmed15, Pmed20, Pmed25 and Pmed30, in which the ratio p/N is around 33%, the GA proposed in this study found all the OPT solutions, while ADE GA [6] found solutions with mean deviation of 0.18% from the OPT solutions. We believe that, for more complex PMP problems (N>900), the GA algorithm proposed in this study would obtain better values than ADE GA [6].

B. Proposed GA vs GA proposed in [7]

The GA proposed in [7] obtained solutions only for problems in the range Pmed1 to Pmed15. In this range, it obtained OPT solution in 14 PMPs, with a deviation of 0.07% from the OPT solution. The GA proposed in this study obtained OPT solutions in all this range.

Table IV shows that the GA proposed in this study converged in a shorter time than GA proposed in [7]. The last one is 44 times slower. This result suggests that the centroid technique used for population initialization in [7] as well as the continuous population updating have a negative impact in convergency time of the GA algorithm, making it slower.

V. Conclusion

A customization of GA operators for solving the *p*-median problem is proposed in this study. When applied to solve the PMPs of OR-library, the proposed algorithm found OPT solutions in 37 of 40 PMPs, with a mean deviation of 0.002% and with a mean time of 17.7s.

Acknowledgments

This research was financial supported by Samsung *Electronica da Amazonia Ltda*, under the terms of the Brazilian Federal Law number 8.387/91, through an agreement signed with Center for R&D in Electronic and Information Technology- CETELI/UFAM.

References

[1] O. Kariv; S.L. Hakimi. The *p*-median problems. In: An Algorithmic Approach to Network Location Problems. SIAM Journal on Applied Mathematics, 1274, Real World Applications. Philadelphia, 37, 539-560, 1979

[2] R.Whitaker. A fast algorithm for the greedy interchange for large-scale clustering and median location problems. INFOR 21, 95-108, 1983

[3] C. Beltran, C. Tadonki, J. Vial. Solving the *p*-median problem with a semi-lagrangian relaxation, Logilab Report, HEC, University of Geneva, Switzerland, 2004

[4] F. Chiyoshi, R.D. Galvão. A statistical analysis of simulated annealing applied to the *p*-median problem. Annals of Operational Research 96:61–74, 2000. doi: 10.1023/A:1018982914742

[5] S. Salhi. Defining tabu list size and aspiration criterion within tabu search methods. Computers and Operations Research 29, 67–86, 2002. doi: 10.1016/S0305-0548(00)00062-9

[6] O. Alp, E. Erkut, Z. Drezner. An efficient genetic algorithm for the *p*-median problem. Annals Operational Research 122:21–42, 2003. doi: 10.1023/A:1026130003508

[7] S. Satoglu. M. Oksuz. G. Kayakutlu, K. Buyukozkan. A genetic algorithm for the *p*-Median facility location problem. GJCI2016 – Global Joint Conference on industrial engineering, Istanbul, 2016.

[8] J. E. Beasley. OR-library: distributing test problems by electronic mail. Journal of Operations Research Society 41:1069–1072, 1990. doi: 10.2307/2582903

[9] G. Reinelt. TSLIB – a traveling salesman library. ORSA Journal of Computing, 3, pp. 376-384, 1991. doi: 10.1287/ijoc.3.4.376

[10] H. Chen, N.S. Flann, D.W. Watson. Parallel genetic simulated annealing: A massively parallel SIMD approach. IEEE Transactions of Parallel Distributed Computation, 9 (Feb. 1998), pp. 126-136, 1998. doi: 10.1109/71.663870

[11] Z. Drezner, J. Brinberg, N. Mladenovic, S. Salhi. New heuristic algorithms for solving the planar *p*-median problem. Comp. Operations Research, 62, pp. 296-304, 2015. doi: 10.1016/j.cor.2014.05.010

[12] D. F. Albdaiwi, H.h. AboelFotoh. A GPU-based genetic algorithm for the *p*-median problem, Journal of Supercomputing, 73, pp 4221-4244, 2010. doi: 10.1007/s11227-017-2006-x

[13] J. A. Moreno-Perez, J. M. Moreno-Vega, N. Mladenovic, Tabu Search and Simulated Annealing in *p*-median Problems. Talk at the Canadian Operational Research Society Conference, Montreal, 1994.

[14] J. E. Beasley, 'OR-library', 1985. [Online]. Available: http:// http://people.brunel.ac.uk/~mastjjb/jeb/orlib/pmedinfo.html. [Accessed: 04- Jul- 2019]

[15] D. Corus and P. S. Oliveto. Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. IEEE Tran. on Evolut. Comp., 2017. doi: 10.1109/TEVC.2017.2745715

[16] J. Holland. Adaption in natural and artificial systems. The University of Michigan Press, Ann Arbor, 1975.

[17] M. Vavouras, K. Papadimitriou, I. Papaefstathiou,. High-speed FPGA-based implementations of a genetic algorithm, in: International Symposium on Systems, Architectures, Modeling, and Simulation, (IEEE2009), pp. 9–16, 2009.

[18] K. Deliparaschos.; G. Doyamis, S. Tzafestas. A parameterised genetic algorithm IP core: FPGA design, implementation and performance evaluation Int. Journal of Electronics, 95, pp. 1149-1166, 2008.

[19] R. P. Weicker, "Dhrystone: a synthetic systems programming benchmark," Communications of the ACM, vol. 27, no. 10, pp. 1013–1030, Oct 1984. 41.