Global Analysis and Discrete Mathematics
Volume 1, Issue 2, pp. 83–92
ISSN: 2476-5341

A hybrid algorithm for the path center problem

Maryam Rahbari · Jafar Fathali[*] · Reza Mortazavi

Received: 21 October 2015 / Accepted: 27 August 2016

Abstract Let a graph $G = (V, E)$ be given. In the path center problem we want to find a path P in G such that the maximum weighted distance of P to every vertex in V is minimized. In this paper a genetic algorithm and a hybrid of genetic and ant colony algorithms are presented for the path center problem. Some test problems are examined to compare the algorithms. The results show that for almost all examples the hybrid method results better solutions than genetic algorithm.

Keywords Genetic algorithm · Ant colony · Location theory · Path center · Hybrid algorithm

Mathematics Subject Classification (2010) 90B90 · 90B06

1 Introduction

Let $G = (V, E)$ be a graph with vertex set V and edge set E. The vertices of the graph model clients. We attach a positive weight w_i, the *demand of client* i, to every vertex v_i. Every edge $e = [v_i, v_j]$ has a (positive) length $c(e)$. The length of a shortest path from vertex v_i to vertex v_j is denoted as $d(v_i, v_j)$. In

Maryam Rahbari
Department of Mathematics, Shahrood University of Technology, University Blvd., Shahrood, Iran, E-mail: : mr_rahbari@yahoo.co.uk

[*]Corresponding author
Jafar Fathali
Department of Mathematics, Shahrood University of Technology, University Blvd., Shahrood, Iran, E-mail: : fathali@shahroodut.ac.ir

Reza Mortazavi
Department of Mathematics, Damghan University, Damghan, Iran, email:r_mortazavi@du.ac.ir

the path center problem we want to find a path P in G such that the maximum weighted distance of every vertex in G to P is minimized.

Hakimi et al. [12] showed that the path center problem on general network is NP-hard. In the case that the network is a tree Hedetniemi et al. [13] and Slater [17] presented linear time algorithms to solve the problem. For this case when the path has a specified size Minieka [15] proposed a linear time algorithm. For a complete survey on path-location problems the reader is referred to Labbe et al. [14].

In this paper we develop a genetic algorithm and a combination of genetic and ant colony algorithm to approach the optimal solution of the path center problem on a graph.

Genetic algorithms (GAs) are search heuristic methods for solving combinatorial optimization problems. They begin with a feasible solution and seek to improve upon it. GAs were first invented by John Holland in the 196Os, but they have become popular in the operations research literature more recently. Specially many genetic algorithms are applied to solve some location problems such as median problem and hub location problem(e.g. see [1,4,9, 18]). For more details about genetic algorithms the reader is referred to the books by Goldberg [11] and Reeves [16].

Ant colony optimization(ACO) was proposed by Dorigo et al. [7] to solve some difficult combinatorial optimization problems such as the traveling salesman problem and the quadratic assignment problem. The ACO algorithm also was successfully used to tackle some other difficult problems such as the vehicle routing, graph coloring, sequential ordering, job-shop scheduling and location problems (see e.g. Bonabeau et al. [3], Dorigo and Di Caro [5], Dorigo and Stützle [8], Fathali et al. [10] and Zaferanieh and Fathali [19]).

In what follows we define the path center problem in Section 2. A short description of the genetic algorithm and discuss our special implementation of this algorithm for the path center problem are given in Section 2. In Section 3 a hybrid genetic and ant colony algorithm is presented to solve path center problem. Computational results are presented in Section 4. Section 5 contains a summary and conclusions.

2 The problem definition

Let $G = (V, E)$ be a graph with V, $|V| = n$, the set of vertices and E the set of edges. Every edge with the end vertices u and v is presented by e_{uv} and every vertex $v_i \in V$ has a real weight $w(v_i)$ that for simplicity we use w_i. $d(u, v)$ is the length of the shortest path between vertices u and v and $d(P, v)$ is the length of the shortest path between path P and vertex v i.e.

$$d(P, v) = \min_{u \in P} d(u, v). \tag{1}$$

The path center problem asks to find a path P such that the following function is minimized

$$F(P) = \max_{i=1,\dots,n} w_i d(P, v_i). \tag{2}$$

Note that a Hamiltonian path in a graph G is a path which visits each vertex of G exactly once. Thus if there is a Hamiltonian path on the given graph, our objective function is equal to zero since all vertex $i \in V$ is in the path, and consequently the maximum distance of each vertex $i \in V$ to the path itself is minimized and it is equals zero. Therefore in this case the Hamiltonian path is also a path center.

3 The genetic algorithm

In this section we present a genetic algorithm for the path center problem on a network.

In the GAs each chromosome corresponds to a solution for the problem. The measure of quality of a solution is called fitness. A genetic operator called crossover is used to produce new chromosomes from a pair of selected chromosomes. Mutations are used to promote genetic diversity.

3.1 Encoding

In our encoding each chromosome has $k = |P|$ genes where each gene corresponds to the index of a vertex in the selected path P. For example, $[4, 5, 2, 11, 15]$ is a chromosome corresponds to a feasible solution for a problem which is a path from vertex v_4 to v_{15} that passes the vertices v_5, v_2 and v_{11}. Note that in our algorithm the number of genes may not be equal for two different chromosomes.

3.2 Fitness evaluation

The fitness of a chromosome is given by the objective function value of the corresponding solution for the path center problem.

3.3 Population

The population size and the initial population are two effective factors on the convergence of the algorithm. Large populations slow down the GA while small population may not have sufficient genetic diversity to search over the feasible region. In our algorithm we set n as the population size. Selecting a suitable initial population results fast convergence for the algorithm. To achieve a near optimal solution the initial population should be distributed on the whole feasible region and every gene must be presented in the initial population.

We construct n initial paths. To select members of initial population, we start with each of n vertices and move to the adjacent vertices in random. This rule follows until we can not move to any new vertex. Note that however selection vertices with high degrees instead of randomly choosing vertices can be a good idea but it caused reduction in the diversity of the population. Procedure *Initial* can be used to generate the initial population.

```
Procedure [Initial]
Set $S := \emptyset$.
For each vertex $v \in V$ do the following
 1. $P(v) := v$, $EndP := v$.
 2. While $Adj := \{u \in V | u \notin P(v)$, u is adjacent to $EndP\}$ is not empty do
    the following
    (a) Select randomly a vertex $u \in Adj$.
    (b) Add $u$ to $P(v)$.
    (c) $EndP := u$.
    End While.
 3. $S := S \cup \{P(v)\}$
End For.
```

3.4 Selection

To generate new members, we select the member of population with best fitness as the first parent and the second one is chosen randomly from the population. Other methods such as selecting members with better fitness as the parents, selecting both parents randomly or selecting parents with maximum number of genes, can be used. However, our computational experiments showed that this method did not yield better results.

3.5 Generating new members

In a GA the chromosomes of two parents are merged to generate new members (children). Usually by using a crossover the chromosomes of the parents are split into two parts and then combined to generate two new members. Instead of this traditional crossover operation we use the following method to generate a new member. We select the genes that are present in both parents, as a part of genes of the new member. However this part my not be a connected path, so to construct a connected path we should add some vertices to this part. Let $\overline{P}$ be the set of selected vertices, and u and v be the two members of $\overline{P}$ such that $e_{uv} \notin E$ and v lies immediately after u in $\overline{P}$. We should construct a path from u to v and add it to $\overline{P}$. If by adding the shortest path between u and v to $\overline{P}$, $\overline{P}$ does not contain any cycle we add this shortest path to it. Otherwise the path which is joining u and v in the parent that have better fitness is added to $\overline{P}$. In this case if we had a cycle in $\overline{P}$ we would delete the vertex v and all other vertices after it in $\overline{P}$ that they constitute a subpath from v. We repeat this rule until the vertices in $\overline{P}$ make a path.

Then we extend the path from the last vertex by adding one vertex at a time until we can not add a new vertex to the path. In each iteration the vertex t which is adjacent to the last vertex u is selected from the parents probabilistically, with respect to the degree of v and amount of its improvement in the fitness. Let P be the current path, P_1 and P_2 the parents of P, F_P the fitness of P, $f(v)$ amount of improvement in the fitness by adding v to the path, $deg(v)$ the degree of v, $dmax$ the degree of the vertex with maximum degree in the network and

$$g(v) = \alpha \frac{deg(v)}{dmax} + (1-\alpha)\frac{f(v)}{F(P)} \tag{3}$$

where $0 \leq \alpha \leq 1$ is a parameter. Let

$$V_1 = \left\{ v \middle| \begin{array}{l} v \in P_1 \cup \ P_2 \ , \ v \ is \ adjacent \ to \ u \ and \ by \ adding \ it \ to \ P, \\ P \ does \ not \ contain \ any \ cycle \end{array} \right\}$$

then the vertex t is selected to add the path P by using the following probability function

$$p(t) := \frac{g(t)}{\sum_{x \in V_1} g(x)}. \tag{4}$$

3.6 Mutation

The mutation operator helps the GA to escape from local optimum. To perform mutation, after generating a new member, we extend the new path from two sides of the path by adding a vertex at a time to each side until can not be added a new vertex. To add a new vertex in each iteration we select the new vertex v which is adjacent to the last vertex u. The rule of selecting this vertex is the same as the method of extending path in the section of generating new members.

3.7 Replacement

After generating a member and implementing the mutation operator, we should admit this member to the population. If the new member is not identical to an existing member and its fitness value is better than the worst fitness value in the population, then the worst member is replaced by the new one.

3.8 Stopping condition

The stopping rule is usually either the number of iterations, maximum number of iterations after last improvement, or a maximum CPU time. Our algorithm terminates when it repeats n iterations after last improvement and the best solution has not changed.

3.9 The algorithm

The ideas of the previous sections lead to the following algorithm.

```
Algorithm [GAPCP].
Initialization:
Generate an initial population $R$ of size $n$ by running procedure Initial (see Section 3.3).
Find the best member and its fitness value, $f_{best}$ and the worst member and its fitness value, $f_{worst}$ in the $R$.
Iteration counter $r := 0$, $r_{best} := 0$.
Iteration step:
While $r - r_{best} \leq n$ do the following:
 1. $r := r + 1$,
 2. Let $P_1$ be the member of $R$ with best fitness and select $P_2$ randomly from $R$.
 3. Generate a new member:
   (a) Set the common edges in $P_1$ an $P_2$ to $P_{new}$.
   (b) For any two vertices $u$ and $v$ that cause a disconnection in $P_{new}$ do the following:
        i. Find the shortest path $P_{uv}$ from $u$ to $v$.
       ii. If by adding $P_{uv}$ to $P_{new}$, $P_{new}$ does not contain any cycle insert it to $P_{new}$.
           else If by adding the path which is joining $u$ and $v$ in $P_1$ to $P_{new}$, $P_{new}$ does not contain any cycle insert this path to $P_{new}$.
           else delete $v$ and all vertices after it in $P_{new}$ that they constitute a subpath from $v$.
   (c) Let $x$ be the end vertex of $P_{new}$.
   (d) Until we can add a vertex to $P_{new}$ such that $P_{new}$ remains a path do the following:
        i. For any vertex $y \in P_1 \cup P_2 \setminus P_{new}$ adjacent to $x$ calculate $g(y)$ according to the equation 3 and select vertex $t$ by using probability function 4.
       ii. Add $t$ to the end of $P_{new}$.
      iii. Set $x = t$.
 4. Mutation:
   (a) Let $x_1$ and $x_2$ be the first and the last vertex of $P_{new}$, respectively.
```

```
    (b) Until we can add a vertex to $P_{new}$ such that $P_{new}$ remains a path **do**
        the following for $x = x_1, x_2$:
          i. For any vertex $y \in G \setminus P_{new}$ adjacent to $x$ calculate $g(y)$ according
             to the equation 3 and select vertex $t$ by using probability function
             4.
         ii. Add $t$ to $P_{new}$.
        iii. Set $x = t$.
 5. **If** the generated member is not in the population and $F(P_{new}) < f_{worst}$
    then **do** the following:
    (a) Replace the worst member by the generated member.
    (b) Update the worst member of the population and its fitness value, $f_{worst}$.
    (c) **If** $F(P_{new}) < f_{best}$, set $f_{best} = F(P_{new})$ and $r_{best} = r$.
**endwhile**
```

4 The hybrid algorithm

In this section we present a hybrid algorithm for path center problem which is a combination of genetic and ant colony algorithms.

Ant colony algorithms are inspired by the behavior of ants in the real world. They are improvement methods which try to approach optimal solution using information obtained by previous solutions. While ants walk from food sources to the nest and vice versa deposit a substance called pheromone on the ground and form in this way a pheromone trail. Ants can smell pheromone and choose the path with the strongest pheromone trail in probability among many paths toward sources or nest.

Our hybrid algorithm is based on genetic algorithm. In the generating new members and mutation steps we use a memory and pheromone to select and add a new vertex to the path. In this algorithm after generating initial population we assign an amount of pheromone τ to each edge of any initial path according to the fitness of the path. Let $P_1, ..., P_n$ be the members of initial population. For each edge e_{uv} we set

$$\tau(e_{uv}) = \sum_{\{P_i | e_{uv} \in P_i\}} \frac{1}{F(P_i)}. \tag{5}$$

Then in each iteration the pheromone of every edge $e_{uv} \in P_{new}$ is updated as follows:

$$\tau(e_{uv}) = \rho \frac{1}{F(P_{new})} + (1 - \rho)\tau(e_{uv}) \tag{6}$$

where $\rho \in [0, 1]$ is a parameter governing pheromone decay called evaporation coefficient.

In the steps generating new members and mutation of hybrid algorithm to add a new vertex to any side of P_{new} instead of selecting vertex by genetic method we select the edge e_{uv} using the following probability function

$$p(e_{uv}) := \frac{\tau(e_{uv})}{\sum_{e_{xy}\in A}\tau(e_{xy})} \tag{7}$$

where A is the set of vertices that can be added to P_{new}.

Now the hybrid algorithm can be written as follows:

Algorithm [**GACPCP**].
Initialization:
Generate an initial population R of size n by running procedure Initial.
Find the best member and its fitness value, f_{best} and the worst member and its fitness value, f_{worst} in the R.
Let $R = \{P_1, ..., P_n\}$, for any edge e_{uv} set $\tau(e_{uv}) = \sum_{\{P_i|e_{uv}\in P_i\}} \frac{1}{F(P_i)}$.
Iteration counter $r := 0$, $r_{best} := 0$.
Iteration step:
While $r - r_{best} \leq n$ **do** the following:

```
1. $r := r + 1$,
2. Let $P_1$ be the member of $R$ with best fitness and select $P_2$ randomly from $R$.
3. Generate a new member:
   (a) Set the common edges in $P_1$ an $P_2$ to $P_{new}$.
   (b) For any two vertices $u$ and $v$ that cause a disconnection in $P_{new}$ do the following:
       i. Find the shortest path $P_{uv}$ from $u$ to $v$.
       ii. If by adding $P_{uv}$ to $P_{new}$, $P_{new}$ does not contain any cycle insert it to $P_{new}$.
           else If by adding the path which is joining $u$ and $v$ in $P_1$ to $P_{new}$, $P_{new}$ does not contain any cycle insert it to $P_{new}$.
           else delete $v$ and all vertices after it in $P_{new}$ that make a connected subpath.
   (c) Let $x$ be the end vertex of $P_{new}$.
   (d) Until can be added a vertex to $P_{new}$ such that $P_{new}$ remains a path do the following:
       i. Select a vertex $y \in P_1 \cup P_2 \setminus P_{new}$ adjacent to $x$ by using the probability function 7 and add it to the end of $P_{new}$.
4. Mutation:
   (a) Let $x_1$ and $x_2$ be the first and the last vertex of $P_{new}$, respectively.
   (b) Until can be added a vertex to $P_{new}$ such that $P_{new}$ remains a path do the following for $x = x_1, x_2$:
       i. Select a vertex $y \in G \setminus P_{new}$ adjacent to $x$ using the probability function 7 and add it to the $P_{new}$.
5. If the generated member is not in the population and $F(P_{new}) < f_{worst}$ then do the following:
   (a) Replace the worst member by the generated member.
   (b) Update the worst member of the population and its fitness value, $f_{worst}$.
```

(c) Update pheromone of each edge $e_{uv} \in P_{new}$ by using equation 6.
(d) **If** $F(P_{new}) < f_{best}$, set $f_{best} = F(P_{new})$ and $r_{best} = r$.

endwhile

5 Computational results

We test the GAPCP and GACPCP for fifteen test problems (p-median instances) from the ORLIB library, see Beasley [2].

Both methods are coded in C++ and run on a Pentium IV with 3600 MHz CPU and 512 megabytes of RAM. We ran 5 times each method for all problems.

In the Table 1 we present the results of GAPCP and GACPCP algorithms. In this table the column *last improvement iteration* indicates the number of iteration after which no improvement in the solution is attained. The last columns of the table show the CPU time of the algorithms per iteration.

We ran the GAPCP for all the problems with $\alpha = 0, 0.5, 1$ (see equation 3). The best solutions were found for $\alpha = 0.5$. The GACPCP algorithm was performed for all the problems with $\rho = 0.2, 0.5, 0.6, 0.75, 0.9$ (evaporation coefficient) and the best solutions were obtained by $\rho = 0.5$.

Table 1 The results for the test problems

			Objective function		Last improv. iter.		CPU/Iter (sec)	
Test #	n	Edge NO.	GACPCP	GAPCP	GACPCP	GAPCP	GACPCP	GAPCP
pmed1	100	200	59	60	251	138	0.136	0.104
pmed2	100	200	33	68	281	179	0.142	0.123
pmed3	100	200	42	72	487	176	0.140	0.125
pmed4	100	200	55	91	425	101	0.133	0.109
pmed5	100	200	54	77	330	101	0.123	0.100
pmed6	200	800	21	24	597	201	0.404	0.388
pmed7	200	800	32	38	437	209	0.364	0.259
pmed8	200	800	27	38	327	473	0.432	0.281
pmed9	200	800	20	38	246	278	0.427	0.396
pmed10	200	800	23	34	510	317	0.410	0.376
pmed11	300	1800	20	29	467	301	0.891	0.712
pmed12	300	1800	20	30	594	441	0.879	0.832
pmed13	300	1800	30	30	567	351	0.931	1.000
pmed14	300	1800	20	33	694	301	0.880	1.019
pmed15	300	1800	16	16	512	321	0.893	1.095

For five test problems, (pmed4, pmed5, pmed6, pmed11 and pmed14) the GAPCP does not improve the best solution, whereas the GACPCP improves the solution of all problems. The results show that for all problems the GACPCP obtained better solutions than GAPCP.

6 Summary and conclusion

In this paper we proposed a genetic algorithm and a hybrid of genetic and ant colony algorithm to solve the path center problem on a network. The results show for almost all examples the hybrid method results better solutions than genetic algorithm.

References

1. O. Alp, E. Erkut and Z. Drezner, An efficient genetic algorithm for the p-Median Problem, Annals of Operations Research, 122, 21-42 (2003).
2. J.E. Beasley, OR-Library: distributing test problems by electronic mail, Journal of the Operational Research Society, 41, 1069-1072 (1990).
3. E. Bonabeau, M. Dorigo and G. Theraulaz, From natural to artifical swarm inteligence, New York: Oxford University Press (1999).
4. Y.C. Chiou and L.W. Lan, Genetic clustring algorithms, European Journal of Operational Research, 135, 413-427 (2001).
5. M. Dorigo and G. Di Caro, The ant colony optimization meta-heuristic, In New ideas in optimization (D. Corne, M. Dorigo and F. Glover, Eds.), Maidenhead, UK: McGraw-Hill, (1999).
6. M. Dorigo, V. Maniezzo and A. Colorni, Positive feedback as a search strategy, Technical Report 91-016, Milan, Italy: Politecnico di Milano (1991).
7. M. Dorigo, V. Maniezzo and A. Colorni, The Ant System: Optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and CyberneticsPart B, 26, 1-13 (1996).
8. M. Dorigo and T. Stützle, Ant Colony Optimization, MIT Press, Cambridge (MA), (2004).
9. J. Fathali, A genetic algorithm for the p-median problem with pos/neg weights, Applied Mathematics and Computation, 183, 1071-1083 (2007).
10. J. Fathali, H.T. Kakhki and R.E. Burkard, An ant colony algorithm for the pos/neg weighted p-median problem, Central European Journal of Operations Research, 14, 229-246 (2006).
11. D.E. Goldberg, Genetic Algorithms In Search, Optimization And Machine Learning. Menlo Park, CA: Addison-Weseley (1989).
12. S.L. Hakimi, E.F. Schmeichel and M. Labbe, On locating path- or tree-shaped facilities on networks, Networks, 23, 543-555 (1993).
13. S.M. Hedetniemi, E.J. Cockaine and S.T. Hedetniemi, Linear algorithms for finding the jordan center and path center of a tree, Transportation Science, 15, 98-114 (1981).
14. M. Labbe, G. Laporte and I. Rodriguez Martin, Path, tree cycle location, In Fleet Management and Logistics (T.G. Crainic and G. Laporte, Eds.), Kluwer (1998).
15. E. Minieka, The optimal location of a path or tree in a tree network, Networks, 15, 309-321 (1985).
16. C.R. Reeves, Genetic Algorithms, In Modern Heuristic Techniques for Combinatorial Problems, (C.R Reeves, Eds.), Chapter 4 (1993).
17. P.J. Slater, Locating central paths in a graph, Transportation Science, 16: 1-18 (1982).
18. Z. Stanimirovic, J. Kratica and D. Dugosija, Genetic algorithms for solving the discrete ordered median problem, European Journal of Operational Research, 182, 983-1001 (2007).
19. M. Zaferanieh and J. Fathali, Ant colony and simulated annealing algorithms for finding the core of a graph, World Applied Sciences Journal, 7, 1335-1341 (2009).