Neurocomputing 148 (2015) 17–22

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A genetic algorithm for the maximum edge-disjoint paths problem

Chia-Chun Hsu [a,b,*], Hsun-Jung Cho [a]

[a] Department of Transportation and Logistics Management, National Chiao Tung University, Hsinchu 300, Taiwan, ROC
[b] Department of Industrial and Systems Engineering, North Carolina State University, Raleigh, NC, USA

ARTICLE INFO

Article history:
Received 7 April 2012
Received in revised form
24 September 2012
Accepted 8 October 2012
Available online 7 August 2014

Keywords:
Edge-disjoint paths
MEDP
Genetic algorithm
Ant colony optimization
Disjoint paths

ABSTRACT

Optimization problems concerning edge-disjoint paths have attracted considerable attention for decades. These problems have a lot of applications in the areas of call admission control, real-time communication, VLSI (Very-large-scale integration) layout and reconfiguration, packing, etc. The maximum edge-disjoint paths problem (MEDP) seems to lie in the heart of these problems. Given an undirected graph G and a set of I connection requests, each request consists of a pair of nodes, MEDP is an NP-hard problem which determines the maximum number of accepted requests that can be routed by mutually edge-disjoint (s_i,t_i) paths. We propose a genetic algorithm (GA) to solve the problem. In comparison to the multi-start simple greedy algorithm (MSGA) and the ant colony optimization method (ACO), the proposed GA method performs better in most of the instances in terms of solution quality and time.

1. Introduction

Assigning paths to connection requests is one of the basic operations in the modern communication networks. Each connection request is a pair of physically separated nodes that require a path for information transmission. Given such a set of connection requests, due to the capacity constraints, one wants to assign paths to requests in a way that no two paths share an edge in common. These paths are called the edge disjoint paths (EDPs). The maximum edge-disjoint paths problem (MEDP) maximizes the number of requests that are simultaneously realizable as EDPs. The problem turns out to be one of the classical combinatorial problems in the NP-complete category.

MEDP can be formally stated as follows. Let $G=(V,E)$ be an undirected and connected graph, where $|V|=n$ and $|E|=m$. A sequence of edges $\pi=\{e_1,e_2,\ldots,e_l\}$, where $e_i\in E$, $i=1,\ldots,l$, is called a path of length $l=|\pi|$. Two paths are edge-disjoint if they do not have any edge in common, otherwise we say they interfere. Let $T=\{(s_i,t_i)|i=1,\ldots,\mathrm{I}$ and $s_i\neq t_i\in V\}$ be a list of connection requests. Each request (s_i,t_i) in G is a pair of vertices that asks for a path connecting s_i and t_i. A feasible solution of MEDP is given by a subset $R\subseteq T$, such that each request in R is assigned a path. The assigned paths are pairwise edge-disjoint and denoted by S. The requests in R are called realizable (or accepted) requests. The goal of the maximum edge-disjoint paths problem is to maximize the cardinality of R.

Early works on the edge-disjoint paths problem have focused on the decision problem, which is one of the classical *NP*-complete problems [1]. The investigation of MEDP started in the 1990s and is still ongoing [2–5]. Since that MEDP is an *NP*-hard problem on general graphs, many studies devoted to obtaining good approximation algorithms and exploring more tractable classes of graphs [6]. In the real world, the MEDP has a multitude of applications in the areas of call admission control [7], real-time communication, VLSI layout [2], packing [5], etc. In addition, the routing and wavelength assignment (RWA) problem [8,9] and unsplittable flow problem (UFP) [3,5,6,10] are direct extensions of MEDP. Therefore, the importance of MEDP is significant.

For specific classes of graphs, MEDP can be optimally solved in polynomial time. Such class includes chains, undirected trees, bipartited stars, and undirected (or bidirected) rings, etc. We refer to [6] for more details. For arbitrary graphs, approximation algorithms including simple greedy algorithm, bounded greedy algorithm and shortest-path-first greedy algorithm, were proposed and their approximation ratios were provided [3,5]. However, to the best of our knowledge, there is a lack of efficient algorithms for tackling the MEDP problem on arbitrary graphs. Greedy algorithms and the ant colony optimization (ACO) approach are the only existing methods. In this paper, we present a novel genetic algorithm (GA) for solving the MEDP problem on arbitrary graphs. The GA is a powerful stochastic search method which has been successfully applied to tackle optimization

* Corresponding author. Tel.: +886 921462912.
E-mail addresses: chsu4@ncsu.edu (C.-C. Hsu), hjcho@cc.nctu.edu.tw (H.-J. Cho).

http://dx.doi.org/10.1016/j.neucom.2012.10.046
0925-2312/© 2014 Elsevier B.V. All rights reserved.

problems in engineering and science. It has been broadly used on network optimization problems [11] as well.

The paper is organized as follows. In Section 2, we review one conventional greedy algorithm and an ACO approach for solving MEDP problems. In Section 3 we present our algorithm. The benchmark instances used to test the performance of the GA approach are introduced in Section 4. The computational results and some observations are also provided. In Section 5, we make a conclusion and point out possible directions for future research.

2. Greedy approach and ACO approach for MEDP

A greedy algorithm is a straightforward constructive algorithm that starts from an empty solution and establishes the solution step by step with a greedy strategy. It can often provide a solution in reasonable computational time. The simple greedy algorithm (SGA) for MEDP shown in Algorithm 1 was proposed in [3]. It starts with an empty set S and R, then iteratively assigns a shortest path to the connection request according to the given order. Each time a path is established, all the edges along that path are removed from the graph. The algorithm halts after I iterations.

```
Algorithm 1 Simple Greedy Algorithm (SGA)

Input: $G=(V,E)$ and $T=\{(s_i,t_i)|i=1,\ldots,I\}$
$S \leftarrow \varnothing$, $R \leftarrow \varnothing$;
for $i=1$ to $I$
        if $\exists$ path from $s_i$ to $t_i$ in $G$ then
            $\pi_i \leftarrow$ a shortest path from $s_i$ to $t_i$ in $G$;
            $S \leftarrow S \cup \pi_i$;
            $E \leftarrow E \backslash \{e|e \in \pi_i\}$;
            $R \leftarrow R \cup \{(s_i,t_i)\}$;
   end if
end for
Output: Realizable requests $R$ and edge-disjoint paths $S$
```

The main drawback of SGA is that the solution quality highly depends on the order of the given connection requests. An intuitive way to improve SGA is applying the multi-start simple greedy (MSGA) algorithm [12]. MSGA runs SGA for several times, each time the order of connection requests is randomly permuted. The algorithm then outputs the best solution among all runs.

Other two improved greedy algorithms including the bounded-length greedy algorithm and the shortest-path-first greedy algorithm, were proposed by Kleinberg [3] and Kolliopoulos [5], respectively. The bounded-length greedy algorithm takes an extra parameter D to denote the threshold of route length. A request is accepted only if it can be routed on a path of length at most D. The shortest-path-first greedy algorithm is another modification of SGA. In each iteration, the shortest path of each of the remaining requests is obtained. Then the request that has the path with the shortest length among all the paths is accepted and removed from the request list. This process repeats until no path can be found for any of the remaining requests. Both greedy algorithms have proven to have better performance than SGA [3,5].

However, due to the deterministic decisions that greedy algorithms take during the solution construction procedure, it is sometimes impossible to find the optimal solution. Fig. 1 shows an example (illustrated in [12]) which consists of a network with the connection request $T=\{(v_1,v_7),(v_8,v_{14}),(v_{15},v_{21})\}$. We can observe that the optimal solution should contain all three requests and the paths are shown in boldface. However, since all the greedy algorithms are based on the shortest path algorithm, none of them

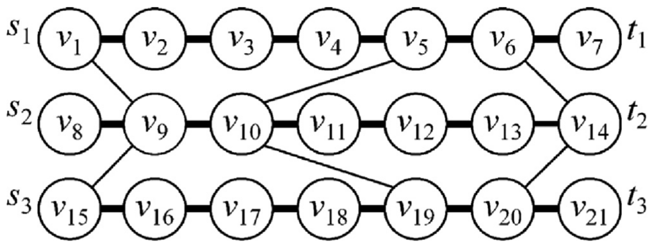


Fig. 1. Instance of the MEDP with $T=\{(v_1,v_7),(v_8,v_{14}),(v_{15},v_{21})\}$. None of the greedy algorithms can find the optimal solution shown in bold font.

can obtain the optimal solution. No matter which connection request we start with, its shortest path always includes one edge that makes it impossible to establish the optimal paths for the other requests. We can observe that the greedy algorithm cannot find a solution of size greater than two.

The application of the ant colony optimization (ACO) to solving the MEDP problem was proposed in [12], which is the only known metaheuristic method so far. The ACO decomposes MEDP into I subproblems $P_i=(G,T_i)$, with $i \in \{1,\ldots,I\}$. Each subproblem itself is trying to find a path for the request $T_i=(s_i,t_i) \in T$ by an ant. During the process of path construction, an ant iteratively moves from one node to another along an edge, the choice of destination can be made either deterministically or stochastically. A constructed ant solution contains I paths which are not necessarily edge-disjoint. An edge-disjoint solution S can be obtained by iteratively remove the path that has the most edges in common with other paths, until the remaining paths are mutually edge-disjoint. A pheromone model τ^i is applied for each subproblem P_i. Each pheromone model τ^i consists of a pheromone value τ_e^i for each edge $e \in E$. We refer readers to [12] for the details of the path construction and pheromone updating procedures.

3. Genetic algorithm for MEDP

A genetic algorithm (GA) mimics the natural evolution processes to gradually improve the solution. Several components are required for solving MEDP by GA: (1) a genetic representation of the solution domain, (2) a way to create an initial population, (3) genetic operators to create new offspring at each generation, and (4) a fitness function for evaluating a solution. The details of these basic components are introduced in this section.

3.1. Decoding and encoding procedure

Since the MEDP problem considers the routes between I terminal pairs, we let an individual (or a solution) contain information of I paths. Each path is encoded by an n-bit string of real values in the interval [0,1]. Each of these values denotes a node's "priority" of being selected into a path during the decoding process. Thereby, an nI-bit string is required to encode an individual. Let $\boldsymbol{u}$ be a $1 \times n$ vector of priority values denoting a path from s to t. A $1 \times n$ binary vector l is the label of the nodes. Note l is set to $\vec{0}$ at the beginning. Decoding of $\boldsymbol{u}$ is a path construction procedure. The procedure starts at the source node s, which is also set as the current node w. The candidate nodes for next move, denoted by C, are the unlabeled nodes adjacent to w. If C is not empty, the current node w will move to the node which has the greatest priority value. The label of w is then set to 1 and the path $\boldsymbol{\pi}$ grows. Otherwise, the procedure backtracks. This construction procedure stops when $w=t$.

```
Algorithm 2 Decoding procedure
Input: $\boldsymbol{u}$ is a priority vector with $n$ elements,
          source $s$,
Algorithm 2 Decoding procedure
        sink $t$,
        $\boldsymbol{A}_w$ is the set of nodes adjacent to node $w$.
  $w \leftarrow s$, $len \leftarrow 1$, $\boldsymbol{\pi} \leftarrow w$, $\boldsymbol{l} \leftarrow \vec{0}$;
  while $w \neq t$
        $C \leftarrow \{j | j \in \boldsymbol{A}_w, l_j = 0\}$;
        if $C \neq \varnothing$ then
          $len \leftarrow len+1$;
          $w \leftarrow \text{argmax}_j\{u_j, j \in C\}$;
          $l_w \leftarrow 1$;
          $\boldsymbol{\pi} \leftarrow [\boldsymbol{\pi} w]$;
    else
          $\pi_{len} \leftarrow 0$;
          $len \leftarrow len-1$;
          $w \leftarrow \pi_{len}$;
      end if
  end while
Output: a path $\boldsymbol{\pi}$
```

After invoking the decoding process I times, I paths are extracted from an individual. Then we apply a simple heuristic called *GMIN* to obtain the EDPs. *GMIN* starts with an empty realizable set, then accepts the request that has the minimum number of interfering requests. The accepted request is then removed from T and its interfering requests are rejected. The procedure iterates until all the requests are either accepted or rejected. Finally, the realizable set R can be obtained.

The encoding procedure generates a code corresponding to the input path. Given a path $\boldsymbol{\pi}$ (denoted by a node sequence), where $|\boldsymbol{\pi}| = len$, the encoding procedure (Algorithm 3) generates a corresponding n-digits encoding $\boldsymbol{u}$. The π_w^{th} element of $\boldsymbol{u}$ is assigned with the priority value $(n-w+1)/n$. For the nodes that are not in $\boldsymbol{\pi}$, their priority values are randomly generated within the range $[0, (n-len+1)/n)$.

```
Algorithm 3 Encoding Procedure
Input: a node sequence $\boldsymbol{\pi}$,
  $len$ is the number of nodes in $\boldsymbol{\pi}$,   $n$ is the number of nodes in
  the graph,
  $\boldsymbol{u} \leftarrow \vec{0}$.
  for $w=1$ to $len$
    if $u_{\pi_w} = 0$ then
      $u_{\pi_w} \leftarrow \frac{n-w+1}{n}$;
    end if
  end for
  $J \leftarrow \{j | u_j = 0\}$;
  for $j \in J$
    $u_j \leftarrow$ random real value in $[0, \frac{n-len}{n})$;
  end for
Output: encoded path $\boldsymbol{u}$
```

3.2. Initial solution

The priority values of all the initial individuals are generated randomly from [0,1]. Some of the initial solutions are further modified by a heuristic method. The method first obtains the shortest path distance d_i of each request (s_i,t_i) (assume the cost of each edge is one). Then the permutation of the connection request is sorted in an ascending order of the shortest path distance. Denote the permuted connection request by T', a realizable request set R' and its corresponding EDPs can be acquired by applying SGA with input (G,T'). We then encode the EDPs to the first initial individual. More individuals can also be modified by exchanging the order of any two requests of T' and applying the same process again.

3.3. Genetic operators

Three genetic operators: crossover, mutation, and self-adaptive operators, are introduced to generate offspring. The first operator is binary which takes two individuals to generate an offspring, in contrast, the other two operators are unary. Let $\boldsymbol{u}^j = (\boldsymbol{u}_1^j, \boldsymbol{u}_2^j, \ldots, \boldsymbol{u}_I^j)$ denote the priority values of the jth solution, where $\boldsymbol{u}_i^j$ is a $1 \times n$ vector of priority values representing the path for the ith request for $i=1,2,\ldots,I$. The crossover operator yields an offspring $\boldsymbol{u}^{new}$ by the weighted linear combination of two individuals p_1 and p_2, i.e., $\boldsymbol{u}_i^{new} = \alpha \boldsymbol{u}_i^{p_1} + (1-\alpha)\boldsymbol{u}_i^{p_2}$ for $i=1,2,\ldots,I$. The crossover operator inherits some characteristics of the two parents, i.e., $\boldsymbol{u}_i^{new}$ is higher (or lower) if their parents, $\boldsymbol{u}_i^{p_1}$ and $\boldsymbol{u}_i^{p_2}$, have higher (lower) priority values. The mutation operator reverses the priority values of a randomly selected request i of the jth individual by letting $\boldsymbol{u}_i^{mu} = \boldsymbol{e} - \boldsymbol{u}_i^j$, where $\boldsymbol{e}$ is an $1 \times n$ vector full of 1s. The operation usually produces a different jth path and brings small change to the selected individual. The self-adaption operator randomly selects an individual $\boldsymbol{u}^{SA}$ and reroutes a rejected request l, i.e., $(s_l,t_l) \in T \backslash R$, by assigning new priority values to $\boldsymbol{u}_l^{SA}$. Two factors are taken into consideration in order to construct a possibly better solution: First, a longer path is less preferred since it may intersect with other paths easily. The all-pairs shortest distance matrix $\mathbb{D}$ obtained at the initialization stage provides useful information for this priority adjustment. Second, the rerouted path better not include an occupied edge (which is already used by other paths). In other words, we want this new path to be composed by the edges that are seldom used. However, edge preferences are hard to manipulate since the encoding scheme is based on node priorities. We use an alternative way which assigns higher priority values to the nodes adjacent to more available edges.

To apply the self-adaption operator, the all-pairs shortest path matrix $\mathbb{D}_{nxn}$, node-arc matrix $\mathbb{A}_{nxm}$ and the $m \times 1$ indicator vector $\boldsymbol{v}^{\boldsymbol{E}}$ obtained in the evaluation step are required. In which $\boldsymbol{v}^{\boldsymbol{E}}$ is defined as follows:

$$v_j^E = \begin{cases} 1 & \textit{if edge } j \textit{ is available} \\ 0 & \textit{otherwise} \end{cases}, j = 1, \ldots, m. \tag{1}$$

The new priority vector $\boldsymbol{u}_l^{SA} \in \mathrm{R}^n$ in block l is determined by a weighted average of distance and usage factors

$$\boldsymbol{v} = (\mathbb{A}\boldsymbol{v}^{\boldsymbol{E}})^{\mathrm{T}} \tag{2}$$

$$\hat{\mathbb{D}}_{t_l} = \| \mathbb{D}_{t_l} \| \cdot \mathrm{e}^{\mathrm{T}} - \mathbb{D}_{t_l} \tag{3}$$

$$\hat{\boldsymbol{u}}_l^{SA} = \frac{\hat{\mathbb{D}}_{t_l}}{\| \hat{\mathbb{D}}_{t_l} \|} + \frac{\boldsymbol{v}}{\| \boldsymbol{v} \|} \tag{4}$$

$$\boldsymbol{u}_l^{SA} = \frac{\hat{\boldsymbol{u}}_l^{SA}}{\| \hat{\boldsymbol{u}}_l^{SA} \|} \tag{5}$$

We use the maximum norm $\boldsymbol{x} = \max \{|x_1|, |x_2|, \ldots, |x_n|\}$ to control the priority value in the interval [0,1]. In (2), $v \in R^n$ represents the number of available adjacent edges of each node. In (3), $\mathbb{D}_{t_l}$ denotes the t_l^{th} row of $\mathbb{D}$. We subtract the one-to-all (node t_l to all nodes) shortest distance value from the maximum value of $\mathbb{D}_{t_l}$. Hence we can obtain the $1 \times n$ vector $\hat{\mathbb{D}}_{t_l}$, in which its ith element has greater value if the distance between node i and node t_l is shorter. In (4), the weighted average of two factors is assigned to $\hat{\boldsymbol{u}}_l^{SA}$. Eq. (5) normalizes the priority values to [0,1].

3.4. Fitness function

Let S^g denote the path set of I paths extracted from an individual. The fitness function is similar to the bicriterion scheme proposed in [12]. The first objective $f(S^g)=|S|$ is the number of EDPs and the second criterion $C(S^g)$ measures the usage of edges that are covered by more than one path. That means $C(S^a)$ is zero if all paths are mutually edge-disjoint.

$$C(S^g)=\sum_{e\in E}\left(\max\left\{0,\left(\sum_{\pi_j\in S^g}\delta^j(S^g,e)\right)-1\right\}\right),$$
$$\text{where }\delta^j(S^g,e)=\begin{cases}1 & if\ \ e\in\pi_j\in S^g\\ 0 & otherwise.\end{cases} \tag{6}$$

For two individuals S_1^g and S_2^g, we say that $f^g(S_1^g)>f^g(S_2^g)$ if and only if

$$f(S_1^g)>f(S_2^g),$$
$$\text{Or }\ (f(S_1^g)=f(S_2^g)\ \text{ and }\ C(S_1^g)<C(S_2^g)). \tag{7}$$

3.5. Improvement heuristic

During the evaluation process, an individual is converted into I paths and *GMIN* is performed to acquire the EDPs. A residual graph $\overline{G}$ can be obtained by removing all edges involved with the EDPs. The proposed improvement heuristics is performed on $\overline{G}$ tries to find a new path to the rejected requests. The function *PathConstruction*, which constructs a path between the endpoints of the unrealizable request (s_{U_j},t_{U_j}), is similar to Algorithm 2. Instead of moving to the node with the highest priority, it selects the node with the smallest index as the succeeding node.

```
Algorithm 4 Improvement Heuristics
Input: $\overline{G}$ is the residual graph,
   $U$ is the unrealizable set,
   $T$ is the connection requests
   if $|U|>0$
      for $j=1$ to $|U|$
         $\pi_{new} \leftarrow PathConstruction\ (s_{U_j},t_{U_j})$;
         if $\pi_{new}$ exists
            $\pi_{U_j} \leftarrow \pi_{new}$;
            Encode $\pi_{U_j}$ to the offspring;   Remove edges in $\pi_{U_j}$
            from $\overline{G}$;
         end if
      end for
   end if
Output: an improved offspring
```

4. Benchmark and experiment results

The set of benchmark instances for MEDP was given in [12]. Seven graphs are considered: the first two graphs, *graph3* and *graph4*, are from the communication network of the Deutsche Telekom AG in Germany. The other three graphs: *AS-BA.R-Wax.v100e190*, *AS-BA.R-Wax.v100e217* and *bl-wr2-wht2.10-50.rand1* were generated with the network generator BRITE. Two mesh graphs: *mesh10* $\times$ *10* and *mesh15* $\times$ *15* are also considered. The main features and quantitative measures are shown in Table 1.

4.1. Instances and platform

We refer to the combination of a graph and a set of connection requests as an instance. For each of the seven graphs, we generate different instances with 0.10|V|, 0.25|V| and 0.40|V| requests

Table 1
Main quantitative measures of the instances.

Graph	|V|	|E|	Min.	Avg.	Max.	Diameter
graph3	164	370	1	4.51	13	16
graph4	434	981	1	4.52	20	22
AS-BA.R-Wax.v100e190	100	190	2	3.80	7	11
AS-BA.R-Wax.v100e217	100	217	2	4.34	8	13
bl-wr2-wht2.10-50.rand1	500	1020	2	4.08	13	23
mesh 10x10	100	180	2	3.60	4	18
mesh 15x15	225	420	2	3.73	4	28

(Min., Avg. and Max. denote the minimum, average and maximum degree, respectively).

separately. Therefore we have 3 instances for each graph and 21 instances in total. We applied the MSGA, ACO and GA on every instance for 30 runs to obtain the best, worst, mean values and standard deviation of the objective values. The average computational time are also recorded. All the algorithms in this paper were implemented in MATLAB and the experiments have been conducted on a PC with Intel® Core i7 CPU @1.6 GHz and 4 Gb of memory running the Windows 7 operating system.

4.2. Computational results

The computational results are shown in Table 2. Comparing the performance of MSGA vs. GA, GA usually obtains better solution quality in less computational time. More specifically, in all 21 instances, GA obtains average values either equal or better than MSGA. Moreover, in 16 out of 21 instances, GA spent less computational time than MSGA did.

For the comparison of ACO and GA, several observations can be made: First, the solution quality of GA obtained in the experiment is comparable with, or in most cases surpasses, that of ACO. More in detail, GA achieves the best solution in 18 out of 21 instances, in which GA beats ACO in 14 instances. In particular, all instances on *graph4* and *mesh15* $\times$ *15* strongly favor the proposed GA over ACO in both computational time and solution quality.

Second, although we notice that ACO performs better than GA in all three instances on graph *bl-wr2-wht2.10-50.rand1*, the differences gradually vanish while the number of request grows (3.2% in the case of 50 pairs, 2.2% in the case of 125 pairs and 0.7% in the case of 200 pairs). However, for *graph4* with 43 pairs, 108 pairs and 173 pairs, GA obtains 11.2%, 13.8% and 7% better values than ACO does, respectively. For mesh15 $\times$ 15 with 23, 57 and 90 pairs, GA performs 14%, 6.3% and 9.0% better than ACO, respectively.

In addition, the 95% confident interval of the performance produced by the three methods for each instance is shown in Table 3. We can observe that, in 14 out of 21 instances, the solution quality of GA is significantly better than that of ACO (their confidence intervals do not overlap). ACO beats GA on two instances (bl-wr2-wht2.10-50.rand1 with 50 and 125 pairs). Moreover, GA also has significant advantage over the MSGA in 14 out of 21 instances. Overall, the proposed GA performs better than the other two methods on 11 out of 21 instances.

4.3. Discussion

Conventional greedy algorithms usually apply the shortest path algorithm to construct the path. Due to its ease and speed in execution, the greedy algorithm is usually implemented in real practice and applied in the on-line case. However, since they establish the paths one by one corresponding to a given order of the connection requests, the quality of the solutions depends heavily on the order of the connection request. We have also seen an example that shows greedy algorithms may not find the

Table 2
Comparison of the results obtained by the MSGA, the ACO and the GA.

Graph	|T|	MSGA					ACO					GA				
		Max.	Min.	$\bar{q}_M$	Std.	$\bar{t}$	Max.	Min.	$\bar{q}_A$	Std.	$\bar{t}$	Max.	Min.	$\bar{q}_G$	Std.	$\bar{t}$
AS-BA.R-Wax.v100e190	10	8	8	**8.0**	0.0	20	8	8	**8.0**	0.0	19	8	8	**8.0**	0.0	10
AS-BA.R-Wax.v100e190	25	14	13	13.4	0.5	49	14	12	13.5	0.6	51	14	13	**13.9**	0.3	43
AS-BA.R-Wax.v100e190	40	20	18	19.1	0.6	77	21	19	20.0	0.4	70	21	20	**20.3**	0.5	98
AS-BA.R-Wax.v100e217	10	7	6	6.8	0.4	20	7	6	6.7	0.4	30	7	7	**7.0**	0.0	15
AS-BA.R-Wax.v100e217	25	13	10	11.5	0.6	48	13	11	11.2	0.5	50	13	12	**12.6**	0.5	54
AS-BA.R-Wax.v100e217	40	22	19	19.9	0.8	77	22	20	21.2	0.5	74	22	21	**21.7**	0.5	93
bl-wr2-wht2.10-50.rand1	50	25	22	23.7	0.7	1081	26	25	**25.8**	0.4	938	26	24	25.0	0.8	788
bl-wr2-wht2.10-50.rand1	125	40	36	38.1	0.9	2746	43	42	**42.5**	0.5	1802	43	39	41.5	0.8	1661
bl-wr2-wht2.10-50.rand1	200	57	55	55.1	0.4	4182	61	59	**60.4**	0.7	2753	61	58	60.0	0.8	3592
graph3	16	15	15	**15.0**	0.0	82	15	15	**15.0**	0.0	24	15	15	**15.0**	0.0	32
graph3	41	33	32	32.1	0.3	174	33	28	30.1	1.0	119	33	32	**32.2**	0.4	93
graph3	65	34	29	32.3	1.1	270	38	33	35.2	1.1	253	39	35	**36.4**	1.0	153
graph4	43	42	42	42.0	0.0	1210	40	36	37.8	1.2	678	42	42	**42.0**	0.0	291
graph4	108	68	62	64.6	1.2	3421	63	58	61.3	1.7	2464	71	68	**69.7**	0.9	1093
graph4	173	75	73	73.1	0.4	5146	82	76	78.8	1.8	4494	85	83	**84.3**	0.7	1665
mesh10X10	10	10	10	**10.0**	0.0	18	10	9	9.9	0.3	20	10	10	**10.0**	0.0	11
mesh10X10	25	17	15	16.3	0.5	50	19	15	16.5	1.0	58	19	17	**17.5**	0.6	45
mesh10X10	40	22	18	19.7	0.8	88	24	20	21.8	1.0	93	24	21	**22.6**	0.8	94
mesh15X15	23	22	19	**20.4**	0.7	194	20	16	17.9	0.9	240	21	20	**20.4**	0.5	90
mesh15X15	57	28	26	27.1	0.6	510	31	27	28.9	1.1	639	32	29	**30.7**	0.7	392
mesh15X15	90	35	32	32.6	0.8	725	38	34	36.2	1.2	966	41	39	**39.4**	0.6	768

(For each approach the first three columns show the maximum, minimum and average value of the number of EDPs in 30 runs. The 4th and 5th column provide the standard deviation and the average computational time. The average value is in boldface and underlined while the result is the best among the three).

Table 3
Confident intervals obtained by the MSGA, the ACO and the GA.

Graph	|T|	MSGA		ACO		GA	
		LB	***UB***	***LB***	***UB***	***LB***	***UB***
AS-BA.R-Wax.v100e190	10	8.0	8.0	8.0	8.0	8.0	8.0
AS-BA.R-Wax.v100e190	25	13.3	13.6	13.3	13.7	**13.8**	**14.0**
AS-BA.R-Wax.v100e190	40	18.9	19.3	19.9	20.1	**20.1**	**20.5**
AS-BA.R-Wax.v100e217	10	6.7	7.0	6.6	6.9	7.0	7.0
AS-BA.R-Wax.v100e217	25	11.2	11.7	11.0	11.4	**12.4**	**12.8**
AS-BA.R-Wax.v100e217	40	19.6	20.1	21.0	21.4	**21.5**	**21.8**
bl-wr2-wht2.10-50.rand1	50	23.4	23.9	**25.7**	**25.9**	24.7	25.3
bl-wr2-wht2.10-50.rand1	125	37.7	38.4	**42.3**	**42.6**	41.2	41.8
bl-wr2-wht2.10-50.rand1	200	55.0	55.2	60.2	60.6	59.7	60.3
graph3	16	15.0	15.0	15.0	15.0	15.0	15.0
graph3	41	32.0	32.2	29.7	30.5	32.0	32.3
graph3	65	31.9	32.7	34.8	35.6	**36.1**	**36.8**
graph4	43	42.0	42.0	37.4	38.2	42.0	42.0
graph4	108	64.2	65.0	60.7	61.9	**69.4**	**70.1**
graph4	173	73.0	73.3	78.1	79.4	**84.0**	**84.5**
mesh10X10	10	10.0	10.0	9.8	10.0	10.0	10.0
mesh10X10	25	16.1	16.5	16.1	16.9	**17.3**	**17.7**
mesh10X10	40	19.5	20.0	21.4	22.1	**22.3**	**22.9**
mesh15X15	23	20.1	20.7	17.6	18.2	20.2	20.6
mesh15X15	57	26.9	27.4	28.5	29.3	**30.4**	**31.0**
mesh15X15	90	32.3	32.9	35.7	36.6	**39.2**	**39.6**

(For each method, the two columns ***LB*** and ***UB*** show the lower bound and upper bound of 95% confident interval for 30 runs on each instance, respectively. The value is in boldface and underlined if the interval surpasses and does not overlap with that of the other two methods).

optimal solution due to its greedy mechanism. Moreover, the repeating calculation of shortest path information causes significantly increasing of computational time while the problem size grows. We can see the phenomenon on Table 2, i.e., instances on bl-wr2-wht2.10-50.rand1 and graph4. The fact that GA only calculates the all-to-all shortest path matrix at the initial stage, is an advantage over the greedy algorithms.

Concerning the comparison of ACO and GA, the pheromone model of ACO consists of a pheromone value for each edge, therefore a I-bit string is required to represent a solution. In contrast, GA uses a I-bit string to characterize a solution, yet GA performs well on most of the instances. In particular, the proposed method to generate initial solution and the improvement heuristics provide huge aid to the performance.

5. Conclusions and future work

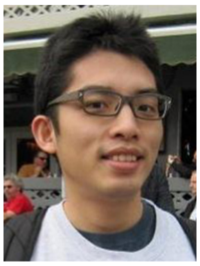
We have proposed a novel genetic algorithm to tackle the MEDP problem. Our approach considers I paths associated with the I connection requests as an individual and applies GMIN to obtain the maximum number of edge-disjoint paths from the path set. In addition to crossover and mutation, a self-adaption operator is proposed to generate offspring. Moreover, an improvement heuristic is used to enhance the solution. A method to generate initial solution provides good quality of the starting population. We have compared our algorithm with multi-start greedy algorithm (MSGA) and ant colony optimization method (ACO). The results showed that the proposed GA method performs better in most of the instances in terms of solution quality and time. In the future, we intend to study different approaches to deepen the performance of our GA method. In addition, we will extend our approach to tackle the generalizations of MEDP, e.g., RWA problem. The RWA problem adds some real-life features into the MEDP problem (e.g., edge capacity, demand, profit, etc.). Some changes of encoding method and genetic operators must be studied.

Acknowledgment

The authors would like to thank Prof. Shu-Cherng Fang of the Industrial and Systems Engineering of North Carolina State University for giving advice and assistance to this article. Also thanks to Dr. Maria J. Blesa Aguilera from ALBCOM research group of the Universitat Politècnica de Catalunya for providing the numerical instances.

References

[1] M.R. Garey, D.S. Johnson, Computers and Intractability: A guide to Theory of NP-completeness, W.H. Freeman and Company, New York-San Francisco, 1979.
[2] A. Frank, Packing paths, circuits, and cuts—a survey, in: B. Korte, L. Lovász, H.J. Prömel, A. Schrijver (Eds.), Paths, Flows, and VLSI-Layout, Springer-Verlag, Berlin, 1990, pp. 47–100.
[3] J. Kleinberg, Approximation Algorithms for Disjoint Paths Problems (PhD thesis), Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1996.
[4] J. Vygen, Disjoint paths (February), Technical report 94816, Research Institute for Discrete Mathematics, University of Bonn, 1994.
[5] S.G. Kolliopoulos, Edge-disjoint paths and unsplittable flow, in: T.F. Gonzalez (Ed.), Handbook of Approximation Algorithms and Metaheuristics, Chapman & Hall/CRC, 2007.
[6] T. Erlebach, Approximation algorithms for edge-disjoint paths and unsplittable flow, Lect. Notes Comput.r Sci. 3484 (2006) 97–137.
[7] U. Adamy, T. Erlebach, D. Mitsche, I. Schurr, B. Speckmann, and E. Welzl, Offline admission control for advance reservations in star networks, in: 2nd Workshop on Approximation and Online Algorithms, LNCS, 3351, 2004, pp. 211–224.
[8] A.E. Ozdaglar, D.P. Bertsekas, Routing and wavelength assignment in optical networks, IEEE/ACM Trans Netw. 11 (2) (2003) 259–272.
[9] H. Choo, V.V. Shakhov, Routing and wavelength assignment in optical WDM networks with maximum quantity of edge disjoint paths, Photon. Netw. Commun. 12 (2006) 145–152.
[10] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, M. Yannakakis, Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems, J. Comput. Syst. Sci. 67 (3) (2003) 473–496.
[11] M. Gen, R. Cheng, L. Lin, Network Models and Optimization, Springer, 2008.
[12] M.J. Blesa, C. Blum, Findinge edge-disjoint paths in networks: An Ant Colony Optimization Algorithm, J. Math. Model. Algorithms 6 (2007) 361–391.



Chia-Chun Hsu received his B.S. degree in Transportation Technology and Management from the National Chiao-Tung University in 2005 and an M.S. degree in Industrial and Systems Engineering from the North Carolina State University in 2011. During his master's degree, he participated in many projects related to development of the roadside traffic detector. Currently he is a dual-Ph. D. candidate of the Department of Industrial and Systems Engineering from North Carolina State University and Transportation Technology and Management from the National Chiao-Tung University. His research interests include software development of roadside detector, bus scheduling, network assignment problem, application of metaheuristic to solve network problems.

Hsun-Jung Cho received his B.S. degree in statistics and his M.S. degree in urban planning from the National Chung Hsing University, Taipei, Taiwan, ROC., in 1978 and 1980, respectively, and his Ph.D. degree in transportation from the University of Pennsylvania, Philadelphia, in 1989. In 1990, he joined the Department of Transportation Technology and Management (TTM), National Chiao Tung University, Hsinchu, Taiwan, ROC. He was Chairman of the TTM Department in 2000. He also was Director of the Transportation Research Center, National Chiao Tung University, from 1998 to 2001. He also is an Associate Editor of the Journal of Net works and Spatial Economics. His research interests include sensitivity analysis of network equilibrium, network design, traffic simulation, intelligent transportation systems, and urban planning.