ABSTRACT

HSU, CHIA-CHUN. A Genetic Algorithm for Maximum Edge-disjoint Paths Problem and Its Extension to Routing and Wavelength Assignment Problem. (Under the direction of Dr. Shu-Cherng Fang.)

Optimization problems concerning edge-disjoint paths in a given graph have attracted considerable attention for decades. Lots of applications can be found in the areas of call admission control, real-time communication, VLSI (Very-large-scale integration) layout and reconfiguration, packing, etc. The optimization problem that seems to lie in the heart of these problems is the maximum edge-disjoint paths problem (MEDP), which is NP-hard. In this dissertation, we developed a novel genetic algorithm (GA) for handling the problem. The proposed method is compared with the purely random search method, the simple greedy algorithm, the multi-start greedy algorithm, and the ant colony optimization method. The computational results indicate that the proposed GA method performs better in most of the instances in terms of solution quality and time.

Moreover, a real-world application of the routing and wavelength assignment problem (RWA), which generalizes MEDP in some aspects, has been performed; and the computational results further confirm the effectiveness of our work. Compared with the bin-packing based algorithms and particle swarm optimization, the proposed method can achieve the best solution on all testing instances. Although it is more time-consuming than the bin-packing based methods, the differences of computational time become small on large instances.

A Genetic Algorithm for Maximum Edge-disjoint Paths Problem and Its Extension to Routing and Wavelength Assignment Problem

by
Chia-Chun Hsu

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Industrial Engineering

Raleigh, North Carolina

2013

APPROVED BY:

Dr. Shu-Cherng Fang Chair of Advisory Committee	Dr. James R. Wilson
Dr. Salah E. Elmaghraby	Dr. Yuan-Shin Lee
Dr. Hsun-Jung Cho	

DEDICATION

This dissertation is dedicated to:

God,

my parents Hung-Chi Hsu and Mei-Chiao Peng,

my little sister Pei-Wen Hsu,

my girlfriend Kuan-Lin Chen

and her parents Ho-Ping Chen and Pi-Hsia Chan,

for their endless love and support.

BIOGRAPHY

Chia-Chun Hsu was born in a lovely family on October 25, 1982. He had a happy childhood and grew up in Taipei, Taiwan. In 2001, he attended National Chiao Tung University and in 2005 received his bachelor degree in Transportation Technology and Management. Then he started his graduate study and began to pursue the Ph.D. degree in 2006 under the supervision of Prof. Hsun-Jung Cho. In 2009, he participated in the dual-Ph.D. program between the College of Management of National Chiao Tung University and the Industrial and Systems Engineering Department of North Carolina State University. In fall 2009, he went to NCSU and started his new life in USA under the supervision of Prof. Shu-Cherng Fang. On March 25th 2013, he passed the oral exam in NCSU and he will receive his Ph.D. degree before the end of 2013 spring semester.

ACKNOWLEDGMENTS

I would like to express my deepest and sincerely gratitude to my advisor Dr. Shu-Cherng Fang, for his guidance over the past four years. His knowledge, wisdom and character would benefit me for life. I also thank my advisor of NCTU Dr. Hsun-Jung Cho for his enormous support and helps during my graduate study. My appreciation also goes to my committee members: Dr. Salah E. Elmaghraby, Dr. James R. Wilson and Dr. Yuan-Shin Lee for their valuable suggestions and comments. It is so lucky to work with my colleagues in FANGroup: Pingke Li, Kun Huang, Lan Li, Tao Huang, Pu Wang, Qingwei Jin, Lu Yu, Yuan Tian, Ye Tian, Zhibin Deng, Ziteng Wang, Jian Luo, Chien-Chia Huang, Tiantian Nie and many others whose names are not listed here. Their encouragement and companion is invaluable to me. Finally, I thank my parents, my sister, and my girlfriend. It is because of them that my years spent on this work have been such a delight.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 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 restrictions, one may want to assign paths to requests in such a way that no two paths share an edge in common. These paths are called edge disjoint paths (EDPs). A natural question to ask is: What is the maximum number of requests that are simultaneously realizable as edge disjoint paths? This is called the maximum edge-disjoint paths (MEDP) problem, which turns out to be one of the classical combinatorial problems in the NP-complete category. It has been extensively studied for decades and can be extended to many real-world applications, e.g., the routing and wavelength assignment (RWA) problem, the call admission problem, the unsplittable flow problem, and the very large-scale integration (VLSI) problem, etc. In this dissertation, we propose a novel genetic-based algorithm to solve the MEDP problem. Moreover, the proposed algorithm is extended for solving the RWA problem. Computational results show that in either case, the proposed method exhibits good performance compared with other existing solution methods.

This chapter intends to introduce some background information on the problems we are going to tackle. The first four sections provide the descriptions, formulation, importance and difficulties of the MEDP problem, respectively. Then an overview and background information on the RWA problem is given in Section 1.5. Following that is an outline of the dissertation in Section 1.6.

1.1 Problem Description

The physical architecture of the network is given in the form of an undirected and connected graph $G = (V, E)$, which consists of a finite set V of vertices and a finite set E of edges, where $|V| = n$ and $|E| = m$. Each edge $e = \{u, v\} \in E$ is said to be incident to u and to v, and $u, v \in V$ are called endpoints of e. A sequence of edges $p = \{e_1, e_2, \dots, e_l\}$ such that $e_i = \{v_i, v_{i+1}\}$ for some $v_i, v_{i+1} \in V$, is called a path of length $l = |p|$, with endpoints v_1 and v_{l+1}. We say that two paths are edge-disjoint (or edge-independent) if they do not have any edge in common. A set of paths is said to consist of edge-disjoint paths (EDPs) if any two paths in the set are edge-disjoint.

Let $T = \{(s_i, t_i) \in V \times V | i = 1, \dots, \mathrm{I} \text{ and } \mathrm{s}_i \neq \mathrm{t}_i\}$ be a set of I connection requests. Each request (s_i, t_i) in a graph G is a pair of vertices that asks for a path that establishes the connection between s_i and t_i. We often use "request (s_i, t_i)" and "request i" interchangeably.

An instance of the maximum edge-disjoint paths (MEDP) problem consists of an undirected graph $G = (V, E)$ and a request set $T \subseteq V \times V$. 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. More precisely, a path p_i between s_i and t_i is assigned to each $(s_i, t_i) \in R$ such that no two paths p_i, $p_j \in S$ $(i, j \in R \text{ and } i \neq j)$ have an edge of the graph in common. The goal of the maximum edge-disjoint paths problem is to maximize the cardinality of R. The requests in R are called realizable (or accepted) requests, those in $T \backslash R$ are the rejected requests. MEDP can be stated in a compact way as follows:

Problem: maximum edge-disjoint paths (MEDP) problem.

Input: undirected graph $G = (V, E)$, connection requests $T = \{(s_i, t_i) \in V \times V | i = 1, \dots, \mathrm{I} \text{ and } \mathrm{s}_i \neq \mathrm{t}_i\}$.

Feasible Solution: a realizable subset $R \subseteq T$ such that there is an assignment of edge-disjoint paths to the requests in R.

Goal: maximize $|R|$.

1.2 The Integer Linear Programming Model for Maximum Edge-disjoint Paths Problem

MEDP has a natural IP formulation based on multicommodity flows. We use an exponentially sized path formulation for convenience. The notations of the MEDP model are defined as follows:

P_i: the set of all simple (cycle-free) paths in G from s_i to t_i, for $i = 1, \dots, \mathrm{I}$.

Q_e: the set of all simple (cycle-free) paths in G that pass along edge e.

y_p: a binary variable indicating whether path p is chosen in the solution, for each $p \in \bigcup_i P_i$.

x_i: a binary variable indicating whether the request i is realizable, for $i = 1, \dots, \mathrm{I}$.

The formulation of MEDP is the following linear integer program:

$$\text{Max} \quad \sum_{i=1}^{\mathrm{I}} x_i \tag{1.1}$$

$$\sum_{p \in P_i} y_p = x_i \qquad i = 1,2, \dots, \mathrm{I} \tag{1.2}$$

$$\sum_{p \in Q_e} y_p \leq 1 \qquad \forall e \in E,\ Q_e = \{p | e \in p\} \qquad (1.3)$$

$$x_i \in \{0,1\} \qquad i = 1,2, \ldots, \mathrm{I} \qquad (1.4)$$

$$y_p \in \{0,1\} \qquad \forall p \in \bigcup_{i=1}^{\mathrm{I}} P_i \qquad (1.5)$$

The objective function (1.1) maximizes the number of realizable connection requests. Constraint (1.2) ensures that each realizable request is assigned a path. Constraint (1.3) ensures that each edge can only be used by at most one path. Constraints (1.4) and (1.5) ensure that all variables are binary.

Enumerating all possible simple paths for each of the connection requests makes solving the model extremely time consuming. Considering the case on a complete graph (in which every pair of distinct vertices is connected by an unique edge), the number of all-possible simple paths that connects a pair of node is

$$P(n-2,0) + P(n-2,1) + P(n-2,2) + \cdots + P(n-2,n-1) + P(n-2,n-2)$$

$= \sum_{j=0}^{n-2} P(n-2,j) = O(n^{n-2})$, where $P(u,v) = u(u-1)(u-2) \ldots (u-v+1)$.

For instance, in a complete graph which has 10 nodes, enumerating all possible simple paths for a pair of nodes has order of 10^8 time complexity. Thus solving this integer linear programming model is not an efficient way for tackling the MEDP problem.

1.3 Importance and applications

Research on the maximum edge-disjoint paths (MEDP) problem has a long history and the corresponding literature is extensive [3, 8, 14, 20, 25, 30, 31]. In recent years, the advent of

the modern high-speed communication networks has brought more focus to the MEDP problem [11]. Many modern network architectures establish a virtual path between any two vertices. In order to achieve guaranteed service quality, the network must reserve sufficient resources (capacity or bandwidth) on the edges along that path. Some requests are rejected if the path does not have sufficient capacity. We want to know how many requests are realizable in a round using edge-disjoint paths, and how many rounds of communication are required to satisfy all requests. The MEDP problem is the essence of these types of problems.

In the real world, the maximum edge-disjoint paths problem has a multitude of applications in the areas of call admission control [37, 38], real-time communication, VLSI (very-large-scale integration) layout [3] and reconfiguration [42], packing [1, 30, 31], etc. In addition, the routing and wavelength assignment (RWA) problem [2, 11, 27], unsplittable flow problem (UFP) [14, 30, 33, 39], and the call admission problem [37, 38] are direct extensions of MEDP. These real-world applications of the maximum edge-disjoint paths problem generalize the original MEDP in one or more aspects. In fact, MEDP is essentially in the heart of several network optimization problems and therefore, its importance is significant. The three classical applications of MEDP are further introduced below.

The routing and wavelength assignment (RWA) problem

Optical networks that apply the wavelength division multiplexing (WDM) technology have attracted enormous attention due to its capability of satisfying the increasing capacity requirements in telecommunication networks [2, 11, 27]. WDM networks allow the simultaneous transmission of different channels along the same optical fiber, by assigning each of them a different wavelength. An optical connection between two nodes is called a

lightpath, which can be characterized by its route and the assigned wavelength.

Given an optical network and a set of lightpath requests, the routing and wavelength assignment (RWA) problem attempts to route and assign a wavelength to each lightpath request subject to the following constraints: (a) wavelength continuity constraint: the same wavelength must be assigned to the entire route if there are no available wavelength converters; and (b) wavelength clash constraint: two lightpaths sharing the same edge have to use different wavelengths.

The objectives of RWA include the minimization of the required number of wavelengths to satisfy all lightpath requests, or maximization of the number of realizable lightpath requests subject to a given number of wavelengths. In later chapters, more details including problem background and related works will be further introduced.

The unsplittable flow problem (UFP)

The unsplittable flow problem is one of the most extensively studied optimization problems in the field of networking [14, 30, 33, 39]. It is essentially a generalization of MEDP in several aspects. For a given undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, each edge e now has a capacity $u(e)$. With respect to the set of connection requests T, each request i has a demand d_i and a profit r_i, assuming that the edge capacities, demands and profits are positive real numbers. A feasible solution is given by selecting a subset of requests and assigning a path from s_i to t_i for each realizable request i, subject to the following constraints: (i) for an edge e, the sum of demands of all the accepted requests that pass through e cannot exceed the capacity $u(e)$; (ii) for an accepted request i, it must send d_i units of demand through a single route. One can gain the profit r_i if request i is accepted. The goal is to maximize the total profit.

It is easy to see that MEDP is a special case of UFP in which $u(e) = 1$ for every $e \in E$, and $d_i = r_i = 1$ for every request i.

Call admission control problem

The call admission control problem is a vital optimization problem encountered in the operations of communication networks [37, 38]. Given an undirected graph and a set of connection requests, each request has a certain bandwidth requirement and time specification of its starting time and duration. If a request is accepted, then a path has to be routed between the pair of nodes and the required amount of bandwidth is reserved on all links along that path during the time period.

In addition, each call is associated with some profits, which the network provider will gain if the desired connection is established. The goal is to maximize the total profits obtained from the accepted requests without violating the edge capacity constraints at any time.

1.4 Difficulties of the maximum edge disjoint paths problem

Most of the early works on the edge-disjoint paths problem have focused on the version of a decision problem, which determines either all the connection requests can be realizable by edge-disjoint paths or certifies that such a routing does not exist. This decision problem is one of the classical *NP*-complete problems [1, 24]. Substantial efforts have been made to the identification of polynomial solvable cases for the decision problem, we refer to the surveys by Frank [3] and Vygen [16] for more details.

The investigation of MEDP started in the 1990s and is still ongoing [3, 14, 16, 30, 31].

Some classes of graphs are able to be checked whether all requests are realizable in polynomial time, but if the answer is no, it is *NP*-hard to compute the maximum number of realizable requests. Reference [33] provides some examples. Since MEDP is an *NP*-hard problem on general graphs [39], many studies were devoted to obtaining good approximation algorithms and exploring more tractable classes. For instance, MEDP on chains can be solved in polynomial time since the routing for each request in a chain is uniquely determined by its endpoint (this fact also holds for arbitrary trees). Hence, the connection requests can be treated as a set of intervals on the real line and the problem of finding a maximum number of disjoint intervals is known to be solvable in linear time. We refer to the survey in [33] for more details and other tractable graphs (e.g., bidirected chains, undirected trees, bidirected stars).

1.5 Routing and wavelength assignment problem

In recent decades, the number of bandwidth-intensive applications in telecommunications such as HD video, video conferencing, HD digital broadcasting and streaming over the internet, have grown rapidly. The technology of fiber-optics can be an attractive candidate for meeting the above-mentioned needs because of its huge transmission bandwidth (~50 Tbps), low signal attenuation, low signal distortion, low power requirement, small space requirement, and low cost. This section starts with the background of optical fibers and WDM networks, and then gives a precise description of the routing and wavelength problem.

1.5.1 Background

Corning Glass Works developed commercial optical fibers successfully in 1970, with

attenuation low enough for communication purposes. In the meanwhile, GaAs semiconductor lasers were developed, which were suitable for transmitting light through optical cables for long distances. Starting from 1975, the first commercial fiber-optics communications system was developed, and it operated at a bit rate of 45 Mbps with repeater spacing up to 10 km. The second generation of fiber-optics communication was developed for commercial use in the early 1980s. By 1987, these systems were operating at bit rates up to 1.7 Gbps with repeater spacing up to 50 km.

Later, scientists developed dispersion-shifted fibers which allowed the third-generation fiber-optics systems operating commercially at a bit rate of 2.5 Gbps with repeater spacing in excess of 100 km. Finally, the fourth generation of fiber-optics communication systems used optical amplification to reduce the need for repeaters and wavelength-division multiplexing to increase data capacity. These two technologies improved the system capacity dramatically since 1992. By 2001, such systems operated at a bit rate of 10 Tbps. Finally, a bit-rate of 14 Tbps was reached over a single 160 km line using optical amplifiers in 2006.

In telecommunications or computer networks, multiplexing is a method to combine multiple analog message or digital data streams into one signal over one shared medium. The use of such a technique can further increase the capacity of optical fibers. Four main types of multiplexing are available: (a) space-division multiplexing (SDM); (b) time-division multiplexing (TDM); (c) code-division multiplexing (CDM); and (d) frequency-division (or wavelength-division) multiplexing (FDM).

SDM simply implies different point-to-point wires for different channels, for instance, stereo audio cable with one pair of wires for the left channel and another for the right channel.

For TDM, two or more bit streams or signals are transferred as sub-channels in one communication channel, but are physically taking turns on the channel. The time domain is divided into several recurrent time slots of fixed length, one for each sub-channel. The optical TDM bit rate is the aggregate rate over all channels in the system. A disadvantage of TDM is that it requires that each node has to be perfectly synchronized to the same time clock and be capable of handling the aggregate bit rate of all channels. On the other hand, CDM assigns a code to each transmission and also requires the source and destination nodes to synchronize to the same time base.

FDM combines several digital signals into one medium by sending signals in several distinct frequencies over that medium. One of the most common applications is cable television. Only one cable reaches a customer's home but the service provider can send multiple television channels or signals simultaneously over that cable to all subscribers. Receivers must tune to the appropriate frequency (channel) to access the desired signal. Wavelength-division multiplexing (WDM) is a variant technology used in optical communications. Since wavelength and frequency are tied together through a simple directly inverse relationship, the two terms actually describe the same concept. WDM operates by dividing the optical transmission spectrum into many non-overlapping wavelengths and each wavelength supports one communication channel. It allows multiple channels to coexist on a single fiber and does not require nodes to synchronize to the same time clock. Hence WDM has become the favorite multiplexing technique for optical networks.

1.5.2 WDM networks

Wavelength-division multiplexing (WDM) is a technology which multiplexes a number of

optical carrier signals onto a single optical fiber by using different wavelengths (i.e. colours) of laser light. The number of wavelengths that each fiber can carry simultaneously is limited by the physical characteristics of fibers and the optical technology of combining the wavelengths and separating them off. In early WDM systems, each fiber could only provide two channels. Modern systems can handle up to 370 signals and can thus expand a basic 273 Gbps system over a single fiber pair to a bit rate over 101Tbps.

Figure 1 [17] is a block diagram of a basic WDM transmission system. The transmitter comprises a laser and a modulator. The laser is the light source, which generates an optical carrier signal at either a fixed or a tunable wavelength. The carried signal is modulated by an electronic signal and is sent to the multiplexer (MUX). The multiplexer combines several optical signals on different wavelengths (denoted by l_1, l_2, l_3 and l_4 in Figure 1) into a single optical signal, which is transmitted to a common output port or optical fiber. The network medium can be a simple fiber link, a passive star coupler, or any type of optical network. The demultiplexer (DMUX) uses optical filters to separate the received optical signal into multiple optical signals on different wavelengths, which are then sent to the receivers. The receiver has a detector that can convert an optical signal to an electronic signal. Optical amplifiers are used at appropriate locations in the transmission system to maintain the power strength of an optical signal.

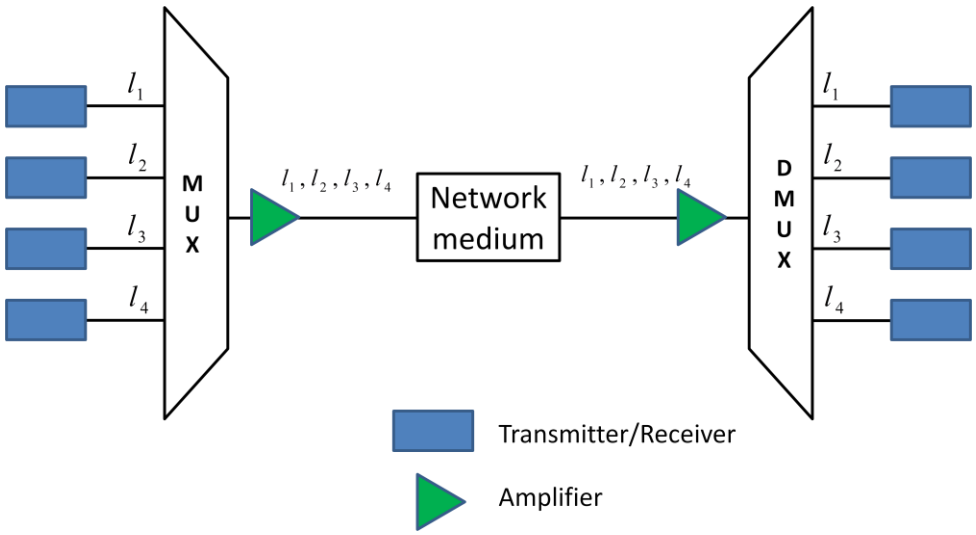


Figure 1 A WDM transmission system [17]

A wavelength-routed optical WDM network typically consists of routing nodes interconnected by WDM fiber links in an arbitrary physical topology. Each routing node employs several transmitters and receivers for transmitting signals to and receiving signals from fiber links, respectively. Each link operates in WDM and supports a certain number of optical channels (or wavelengths). A routing node can be connected to an access node, which is an interface between the optical network and the electronic client networks. An access node performs traffic aggregation and E/O conversion functions on the source side. On the destination side, traffic deaggregation and O/E conversion are performed. The architecture of a wavelength-routed WDM network is shown in Figure 2 [17]. In the remainder of this work, we assume that each routing node is connected to an access node, and we refer to this integrated unit as a node.

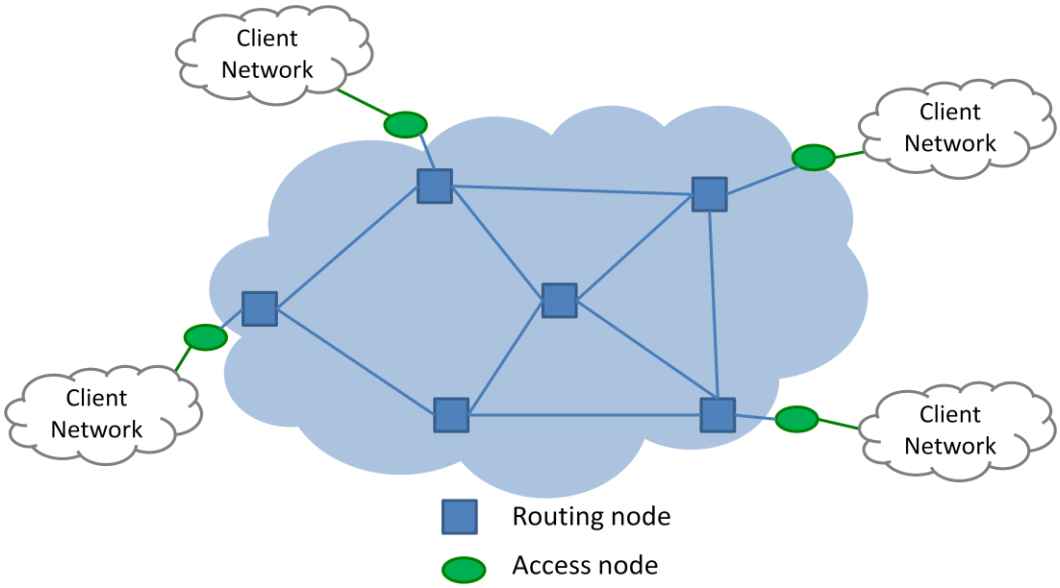


Figure 2 A wavelength-routed WDM network [17]

1.5.3 Problem description

In a wavelength-routed WDM network, end users communicate with each other via all-optical WDM-channels, which are referred to as lightpaths. A lightpath is used to establish a connection between two nodes, and it can be characterized by its route and the occupied wavelength. In the absence of wavelength converters, a lightpath must use the same wavelength on all fiber links which it traverses, which is known as the *wavelength continuity constraint*. In addition, lightpaths that share a common physical link cannot use the same wavelength, which is known as the *wavelength clash constraint*. Figure 3 illustrates a wavelength-routed network in which three lightpaths have been set up on two different wavelengths.

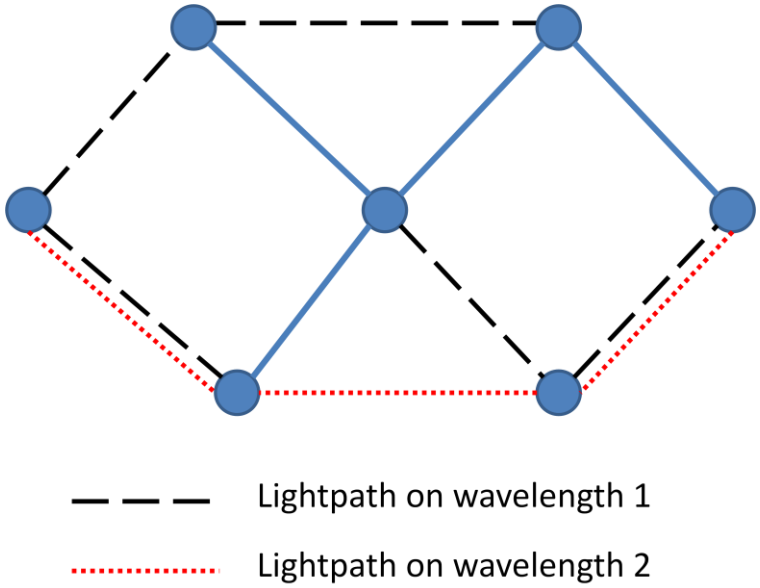


Figure 3 A wavelength-routed network with three lightpaths

Given a set of connection requests, the problem of setting up lightpaths by routing and allocating a wavelength to each connection is called the routing and wavelength assignment (RWA) problem. In general, connection requests are of three types: static, incremental and dynamic. We only consider the static case, which means the entire set of connection requests is known in advance. The routing and wavelength assignment operations are performed off-line.

The RWA problem is to establish routes and assign wavelengths for the connections while minimizing network resources such as the number of wavelengths or the number of fibers in the network. Alternatively, one may attempt to connect as many requests as possible for a given number of wavelengths. In this work, we consider the former case assuming that the available number of wavelengths is unlimited. The objective is to minimize the number of wavelengths used to establish connections for all requests.

To be precise, given an undirected graph $G = (V, E)$ in which each edge $e \in \mathrm{E}$ is an optical fiber link in the physical network, and a request set $T = \{(s_i, t_i) \in V \times V | i = 1, \ldots, \mathrm{I} \text{ and } s_i \neq t_i\}$, the routing and wavelength assignment (RWA) problem searches for a set of lightpaths $\pi = \{\pi_1, \pi_2, \ldots, \pi_\mathrm{I}\}$ in G, each corresponds to one request $(s_i, t_i) \in T$, and assigns a set of wavelengths $w = \{w_1, w_2, \ldots, w_\mathrm{I}\}$ to these paths. Path π_i and π_j, $i \neq j$, cannot be assigned the same wavelength if they share a common edge. The objective is to minimize the number of wavelengths required to satisfy all requests in T.

A feasible solution to the RWA problem consists of a path set π and the assigned wavelength set w. Each path $\pi_i \in \pi$ connects the request (s_i, t_i) and is assigned the wavelength $w_i \in w$ such that the wavelength clash constraints hold. The RWA problem can be stated in a compact way as follows:

Problem: routing and wavelength assignment (RWA) problem.

Input: undirected graph $G = (V, E)$ and a set of connection requests $T = \{(\mathrm{s}_i, \mathrm{t}_i) \in V \times V | i = 1, \ldots, \mathrm{I} \text{ and } s_i \neq t_i\}$.

Feasible Solution: a path set π to connect all requests and a corresponding wavelength set w such that the wavelength clash constraint holds.

Goal: minimize the number of required wavelengths.

1.5.4 Mathematical model of RWA

In reference [2], the RWA problem is formulated as an integer linear programming problem with a general multicommodity flow formulation. The authors assume that the number of

available wavelengths is limited and the goal of their model is to maximize the number of accepted requests. In our work, we assume there are I units of available wavelengths (I is the number of requests). Thus in the worst case, where each wavelength is assigned to exactly one request, all connection requests can still be satisfied. The goal of our model is to minimize the number of utilized wavelengths to satisfy all requests. Some notations are defined below.

L: set of indices of available wavelengths (on each edge), $= \{1, 2, \dots, \mathrm{I}\}$.

$x_{l,\pi}$: a binary variable, $= 1$ if wavelength $l \in L$ is assigned to path π; and $= 0$ otherwise.

P_i: the set of all simple (cycle-free) paths in G from source s_i to terminal t_i, for $i = 1, \dots, \mathrm{I}$.

Q_e: the set of all simple paths in G that pass along edge e.

y_l: a binary variable, $= 1$ if wavelength $l \in L$ is utilized; and $= 0$ otherwise.

The problem formulation is given by

$$\text{Minimize} \quad \sum_{l=1}^{\mathrm{I}} y_l \tag{1.6}$$

Subject to

$$\sum_{\pi \in Q_e} x_{l,\pi} \leq 1 \qquad \forall e \in E,\ l \in L \tag{1.7}$$

$$\sum_{\pi \in P_i} \sum_{l=1}^{\mathrm{I}} x_{l,\pi} = 1 \qquad i = 1,2, \dots, \mathrm{I} \tag{1.8}$$

$$x_{l,\pi} \leq y_l \qquad \pi \in \bigcup_{i=1}^{\mathrm{I}} P_i,\ l \in L \tag{1.9}$$

The objective (1.6) minimizes the total number of utilized wavelengths. Constraint (1.7)

is the wavelength clash constraint, that is, for the paths in Q_e, the wavelength l is assigned to at most one of them. In other words, paths using the same edge must employ different wavelengths. Constraint (1.8) represents the demand constraint, which ensures each request is assigned exactly a path and a wavelength. Constraint (1.9) ensures y_l will equal 1 if wavelength l is used by one or more paths.

As mentioned in Section 1.2, enumerating all-possible paths is extremely time-consuming and only applicable in very small-sized networks. The number of all-possible paths that connect a pair of nodes in a complete graph is $O(n^{n-2})$. Thus it is unlikely to tackle the RWA problem by solving the above integer linear programming due to its rapidly increasing number of variables and constraints.

1.6 Outline of the dissertation

The dissertation is organized as following: Chapter 2 includes two parts. The first part is the literature review of the MEDP problem, where some known approximation ratios, existing methods and genetic algorithms for path-related problems are reviewed. The second part is a survey of the RWA problem, where the background, related works, two existing methods and lower bounds of the problem are provided. In Chapter 3, we propose a novel genetic algorithm for solving the MEDP problem, including the encoding/decoding scheme, a method to produce the initial population, a fitness function, three reproduction operators, an improvement heuristic, and the population management method. The testing instances and comparisons of computational results obtained by using existing methods and the proposed genetic algorithm are provided in Chapter 4. In Chapter 5, we develop a heuristic method

which employs the proposed GA-based method to solve the RWA problem. The computational results show that the proposed methods outperform the bin-packing based methods and the particle swarm optimization (PSO). Concluding remarks and future research directions are given in Chapter 6.

Chapter 2 Literature Review

In this chapter, we provide a review of the maximum edge-disjoint paths problem (MEDP) and one of its extended real-world applications – routing and wavelength assignment problem (RWA). In Section 2.1, a special case of MEDP known as edge-disjoint Menger problem, in which all connection requests are composed by repetitions of the same pair (s, t), is discussed. Section 2.2 summarizes the approximation ratios of most well-known approximation algorithms for MEDP on a general graph. Detailed descriptions of these approximation algorithms are given in Section 2.3. In Section 2.4, some encoding schemes and genetic operators for solving path-related problems are introduced. Related works on the RWA problem are reviewed in Section 2.5, particle swarm optimization (PSO) and the state-of-art bin-packing based methods are given in Sections 2.6 and 2.7, respectively. Finally, lower bounds of solving the RWA problem are provided in Section 2.8.

2.1 A special case: Menger's Theorem

One extreme case of MEDP is that all of the I pairs of connection requests are the same, i.e., all requests are between two vertices $s, t \in V$. In this case, the number of edge-disjoint paths can be viewed as a measurement of how well a given pair of vertices is connected. A different way of measuring the connectivity is to determine the smallest number of edges whose deletion from the graph disconnects every path between the pair. In 1927, Karl Menger [19] proved an elegant theorem, which states that the maximum number of edge-disjoint paths between a given pair of connection requests in a graph equals the

minimum number of edges whose deletion disconnects the pair.

To be more precise, given $s, t \in V$ in graph G, let set S be a collection of edges. We say S is an "$s-t$ edge-separating set" if every $s-t$ path contains an edge of S. We denote the minimum cardinality of an $s-t$ edge-separating set by $\lambda(s,t)$ and the maximum number of edge-disjoint $s-t$ paths in G by $\nu(s,t)$. Since each edge-disjoint $s-t$ path must contain at least one edge in the $s-t$ edge-separating set, we have $\lambda(s,t) \geq \nu(s,t)$. Menger further proved that $\lambda(s,t) \leq \nu(s,t)$ in his theorem.

Theorem 1 (Karl Menger, 1927) *In an undirected graph G, if vertices s and t are not adjacent, $\lambda(s,t) = \nu(s,t)$.*

2.2 Known approximation ratios for MEDP

Since the maximum edge-disjoint paths problem with connection requests on a general graph is proven to be NP-hard, many works have proposed approximation algorithms for solving the problem [1, 8, 14, 20, 23, 30, 31, 32, 33]. A good approximation algorithm runs in polynomial time to reach a solution guaranteed to be close enough to the optimal solution. The sense of "closeness'' can be described by the "approximation ratio" ρ. A ρ-approximation algorithm for MEDP runs in polynomial time to output a feasible solution R satisfying $|R| \geq \mathrm{OPT}/\rho$, where OPT is the optimal objective value and $\rho > 1$ is the approximation ratio.

For a general graph, known approximation algorithms for MEDP include the simple greedy algorithm, bounded greedy algorithm and shortest-path first greedy algorithm can be found in Kleinberg [14]. The bounds of approximation ratios are summarized below, while

the detailed descriptions of each algorithm will be given in Section 2.3.

Theorem 2 (**Erlebach, 2006** [33]) *The simple greedy algorithm has an approximation ratio of* $n-1$ *for MEDP in a directed or undirected graph with* n *vertices, and the bound is tight.*

Theorem 3 **(Kleinberg, 1996** [14]**)** *The bounded greedy algorithm with a parameter* $D=\lceil\sqrt{m}\rceil-1$ *has an approximation ratio of* $O(\sqrt{m})$ *for MEDP in a directed or undirected graph with* m *edges.*

Chekuri and Khanna [8] showed that for MEDP, the shortest-path-first greedy algorithm gives an $O(n^{2/3})$ approximation for undirected graphs and an $O(n^{4/5})$ approximation for directed graphs. In the same article, an $O(\sqrt{n\log n})$ approximation algorithm was also shown for acylic graphs. In Varadarajan and Venkataraman's work [20], the approximation ratio for directed graphs was improved to $O((n\log n)^{2/3})$. The next theorem provides the best known approximation ratio for MEDP in terms of the number of vertices.

Theorem 4 **(Chekuri and Khanna, 2003** [8]**; Varadarajan and Venkataraman, 2004** [20]**)** *The shortest-path-first greedy algorithm for MEDP achieves an approximation ratio of* $O(min\{\sqrt{m},n^{2/3}\})$ *for undirected graphs and* $O(min\{\sqrt{m},(n\log n)^{2/3}\})$ *for directed graphs.*

Lastly, an essential inapproximabilty result for directed graphs has been obtained by Guruswami et al. [39].

Theorem 5 **(Guruswami et al., 1999** [39]**)** *For MEDP in a directed graph with* m *edges, there cannot be an* $m^{0.5-\varepsilon}$*-approximation algorithm for any* $\varepsilon>0$ *unless* $P=NP$.

Very few metaheuristics algorithms have been proposed for solving MEDP. The ant colony optimization (ACO) approach presented in [23] is the only known metaheuristic for MEDP. The details of ACO approach will be given in Section 2.3.3.

2.3 Existing solution methods for MEDP

Known solution methods for the MEDP include the "LP relaxation and rounding" method, the "greedy algorithms", and the "Ant Colony Optimization" approaches, which are presented in this section.

2.3.1 LP relaxation and rounding method

The formulation of MEDP shown in Section 1.2 is an integer linear program whose complexity grows exponentially in terms of the problem size. Relaxing (1.4) and (1.5) by $x_i \in [0,1]$ and $y_\pi \in [0,1]$ respectively, leads to an LP relaxation such that an optimal fractional solution can be acquired in polynomial time. Then the rounding techniques are applied to covert the fractional solution into an integral solution. However, the gap between the fractional optimum and integral optimum can be large. A brick-wall graph shown in Figure 4 is a simple example demonstrating this phenomenon.

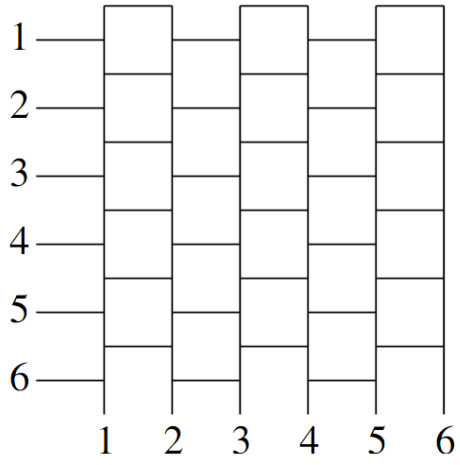


Figure 4 The brick-wall graph

Assume there are six connection requests {(1,1), (2,2), (3,3), (4,4), (5,5), (6,6)} in Figure 4. When the capacity of each edge is 1, any two paths interfere with each other. We can easily see that only 1 request is realizable. However, solving the LP relaxation will obtain an objective value of 3, since it routes 0.5 of each request without violating any constraint. This shows that the fractional solution obtained by using the LP relaxation cannot guarantee much about the original problem. The rounding approach may result in an integral solution far away from desired. However, the situation becomes better as the edge capacity increases. We refer to reference [33] for more details.

2.3.2 Greedy algorithms

A greedy algorithm starts with an empty solution set and constructs a feasible solution step by step utilizing a greedy strategy. Due to its ease and speed in execution, a greedy algorithm is usually implemented for on-line real practice. In this case, the requests are presented one by one and the algorithm has to accept or reject the request sequentially without knowing future requests.

The pseudocode of the simple greedy algorithm (SGA) for MEDP is given in **Algorithm 1**. It starts with empty sets S and R, then iteratively assigns a shortest path, if there is one, to a connection request following the order that the request set is given. Each time a path is assigned, all the edges along that path are removed from the graph. The algorithm halts after I iterations. Unfortunately, SGA does not achieve a good approximation ratio in general. The work in [3] shows that SGA has the approximation ratio $n-1$ for MEDP in general graphs.

Algorithm 1 Simple Greedy Algorithm (SGA)

Input: $G = (V, E)$ and $T = \{(s_i, t_i) | i = 1, \dots, \mathrm{I}\}$
Begin:
1. $S \leftarrow \emptyset, R \leftarrow \emptyset$;
2. **for** $i = 1$ to I
3. **if** $\exists$ path from s_i to t_i in G **then**
4. $p_i \leftarrow$ a shortest path from s_i to t_i in G ;
5. $S \leftarrow S \cup p_i$;
6. $E \leftarrow E \setminus \{e | e \in p_i\}$;
7. $R \leftarrow R \cup \{(s_i, t_i)\}$;
8. **end if**
9. **end for**

End
Output: Realizable requests *R* and edge-disjoint paths *S*

It is easy to see that the solution quality of SGA highly depends on the order of connection requests. In the worst case, SGA may route the first request on a very long path that interferes with all other requests. This is the main drawback of SGA. An intuitive way to solve this problem is applying the multi-start simple greedy (MSGA) algorithm [23], shown in **Algorithm 2**. MSGA runs SGA for N times, in each iteration the order of connection requests is randomly regenerated. The algorithm then outputs the best solution of the N possible solutions.

Algorithm 2 Multi-start simple greedy algorithm (MSGA)

Input: G, T and N, where N is the number of restarts
Begin:
1. $S_{\text{best}} \leftarrow \emptyset, R_{\text{best}} \leftarrow \emptyset$;
2. $T_1 \leftarrow T$;
3. **for** $j = 1$ to N
4. $(R_j, S_j) \leftarrow$ Simple Greedy Algorithm SGA(G, T_j);
5. **if** $|R_j| > |R_{\text{best}}|$ **then**
6. $S_{best} \leftarrow S_j$;
7. $R_{best} \leftarrow R_j$;
8. **end if**
9. $\rho \leftarrow$random permutation of $(1, 2, \dots, \text{I})$;
10. $T_{j+1} = \{(s_{\rho(i)}, t_{\rho(i)}) | i = 1, \dots, \text{I}\}$;
11. **end for**

End
Output: Realizable requests R_{best} and edge-disjoint paths S_{best}

Another improved greedy algorithm is the bounded-length greedy algorithm shown in **Algorithm 3**, proposed by Kleinberg [14]. It 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. In other words, requests whose endpoints are at distance larger than D will be rejected. The algorithm has an approximation ratio of ∞ if every request can only be routed with length at least $D + 1$. If this happens, the algorithm will increase D by one and run again. Kleinberg proved that the bounded-length greedy algorithm with parameter $D = \lceil\sqrt{\text{m}}\rceil - 1$ can achieve an approximation ratio of $O(\sqrt{\text{m}})$ for MEDP in a directed or undirected graph with m edges.

Algorithm 3 Bounded-length Greedy Algorithm (BGA)

Input: $G = (V, E)$, $T = \{(s_i, t_i) | i = 1, \ldots, \mathrm{I}\}$ and D
Begin
1. **do**
2. S $\leftarrow \emptyset$, R $\leftarrow \emptyset$;
3. **for** $i = 1$ to I
4. $p_i \leftarrow$ a shortest path from s_i to t_i in G **;**
5. **if** shortest path p_i exists **and** $|p_i| \leq D$ **then**
6. $S \leftarrow S \cup p_i$;
7. $E \leftarrow E \setminus \{e | e \in p_i\}$;
8. $R \leftarrow R \cup \{(s_i, t_i)\}$;
9. **end if**
10. **end for**
11. D $\leftarrow$ D $+ 1$;
12. **while** $|R| < 1$

End
Output: Realizable requests *R* and edge-disjoint paths *S*

A further modification of the greedy algorithm is the shortest-path-first greedy algorithm proposed by Kolliopoulos and Stein [30, 31], shown in **Algorithm 4**. First, the algorithm acquires the shortest path for each connection request. The request that has the path with the shortest length among all paths is accepted and removed from the request set. Then the algorithm repeats the same "greedy" strategy until no path can be found for all remaining requests. Obviously, the algorithm accepts requests in a non-decreasing order of the path length. It has been shown that the worst-case approximation ratio of **Algorithm 4** is at least as good as that of bounded greedy algorithm. Kolliopoulos and Stein [30, 31] proved that the algorithm achieves an approximation ratio of $\lceil \sqrt{m} \rceil$ in a directed or undirected graph with m edges.

Algorithm 4 Shortest-path-first Greedy Algorithm

Input: $G = (V, E)$ and $T = \{(s_i, t_i) | i = 1, \dots, \mathrm{I}\}$
Begin:
1. $S \leftarrow \emptyset, R \leftarrow \emptyset$;
2. **While** R contains a request that can be routed in G
3. $(s_i, t_i) \leftarrow$a request in T such that its shortest path has minimum length among all requests in T;
4. $R \leftarrow R \cup \{(s_i, t_i)\}$;
5. $T \leftarrow T \setminus \{(s_i, t_i)\}$;
6. $p_i \leftarrow a$ shortest path from s_i to t_i in G
7. $E \leftarrow E \setminus \{e | e \in p_i\}$;
8. **end while**

End
Output: Realizable requests *R* and edge-disjoint paths *S*

2.3.3 Ant-colony optimizationn

Ant colony optimization (ACO) was initially proposed by Marco Dorigo in 1992 in his PhD dissertation [21]. The idea of ACO comes from observing the exploitation of food resources by ants. In the beginning, ants wander randomly. If an ant finds food, it leaves pheromone on the trail back to the colony. Other ants are likely to follow the trail instead of keep travelling at random. If one eventually finds food, it also leaves pheromone to reinforce the path. On the other hand, the pheromone on paths evaporates gradually, thus reducing its strength of attraction. The pheromone density becomes higher on the shorter paths than the longer ones, therefore a shortest path between the food source and the ants' nest may be found eventually.

The application of ant colony optimization (ACO) to solving MEDP is the only known metaheuristic method. In [23], MEDP is decomposed into I subproblems $P_i, i \in \{1, \dots, \mathrm{I}\}$. Each subproblem $P_i = (G, T_i)$, where $T_i = (s_i, t_i)$, is trying to find a path for request T_i on

G by an ant. In other words, I ants are assigned for the I connection requests.

A constructed ant solution S^a contains I paths which are not necessarily edge-disjoint. An edge-disjoint solution S is generated by iteratively removing the path that has the most edges in common with other paths, until the remaining paths are mutually edge-disjoint. Let $f(S^a) = |S|$ denote the number of edge-disjoint paths obtained from S^a. Since two solutions S_1^a and S_2^a may have the same number of EDPs, i.e., $f(S_1^a) = f(S_2^a)$, a second criterion is introduced to quantify the non-disjointness of an ant solution. Define $\mathrm{C}(\mathrm{S^a})$ as follows:

$$\mathrm{C}(S^a) = \sum_{e \in E}(\max\{0, \left(\sum_{p_j \in \mathrm{S^a}} \delta^j(S^a, \mathrm{e})\right) - 1\}), \tag{2.1}$$

$$\text{where } \delta^j(S^a, e) = \begin{cases} 1, & \text{if } e \in p_j \in S^a, \\ 0, & \text{otherwise.} \end{cases}$$

For an ACO intermediate solution S^a, $\mathrm{C}(S^a) \geq 0$ measures the usage of edges that are covered by more than one path. That means $\mathrm{C}(S^a)$ is zero if all paths are mutually edge-disjoint. Generally speaking, a decrease of $\mathrm{C}(S^a)$ may imply an increase of the number of EDP. Thus we can define an ordering $g(\cdot)$ as follows: For two ACO intermediate solutions S_1^a and S_2^a, we say that $g(\mathrm{S_1^a}) > g(\mathrm{S_2^a})$ if and only if

$$f(S_1^a) > f(S_2^a), \tag{2.2}$$
$$\text{Or } (f(S_1^a) = f(S_2^a) \text{ and } \mathrm{C}(S_1^a) < \mathrm{C}(S_2^a))$$

The pheromone model is critical for the ant colony optimization approach. Since the problem is decomposed into I subproblems, 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 $e \in \mathrm{E}$. All pheromone values are in the range $[\tau_{\min}, \tau_{\max}]$, where $\tau_{\min}$ and $\tau_{\max}$ are user-defined parameters. We denote the set of I pheromone models by $\tau = \{\tau^1, \dots, \tau^{\mathrm{I}}\}$. **Algorithm 5** carries the pseudocode of a basic ACO algorithm. The procedure $InitializePheromoneValues(\tau)$ sets all the initial pheromone values to be value $\tau_{\min}$. In each iteration, N_{sol} ant solutions are constructed by applying the function $ConstructFullPath(s_{\rho_i}, t_{\rho_i})$ I times (with I ants), where ρ is a permutation of $(1, 2, \dots, \mathrm{I})$. During the process of path construction, an ant iteratively moves from one node to another along an available edge, the choice of destination can be made either deterministically or stochastically. We randomly draw a number d_{rate} between 0 and 1. If $d_{rate} \leq 0.75$, the next step destination is chosen deterministically. Otherwise, the choice is made stochastically.

After I paths are constructed, the value of the variable S_{gbest} will be updated if the solution improves. Finally, the pheromone values are updated depending on the edges included in S_{gbest}. We refer readers to [23] for the details of the path construction and pheromone updating procedures.

Algorithm 5 Basic ACO Algorithm

Input: G, T
Begin:
1. $S_{\text{gbest}} \leftarrow \emptyset$;
2. $InitializePheromneValues(\tau)$;
3. **while** termination condition is false
4. $\rho \leftarrow$(1,2…,I);
5. **for** $j = 1$ to N_{sol}
6. $S_j^a \leftarrow \emptyset$;
7. **for** $i = 1$ to I
8. $p_{\rho_i} \leftarrow \text{ConstructFullPath}(s_{\rho_i}, t_{\rho_i})$;
9. $S_j^a \leftarrow S_j^a \cup p_{\rho_i}$;
10. **end for**
11. $\rho \leftarrow$random permutation of $(1, 2, \dots, \text{I})$;
12. **end for**
13. Choose $S_{best}^a \in \{S_j^a | j = 1, \dots, \text{N}_{\text{sol}}\}$ such that
$$g(S_{best}^a) \geq g(S^a), \forall S^a \in \{S_j^a | j = 1, \dots, \text{N}_{\text{sol}}\}$$
14. **if** $f(S_{best}^a) > f(S_{gbest}^a)$ **then**
15. $S_{gbest}^a \leftarrow S_{best}^a$;
16. $\text{UpdatePheromoneValues}(\tau, S_{gbest}^a)$;
17. **end if**
18. **end while**
End
Output: Realizable requests R_{best} and disjoint paths S_{gbest} generated from $\text{S}_{\text{gbest}}^{\text{a}}$

In [23], the author also proposed an enriched version of ACO for MEDP. The following four features are added to modify the way of exploring the solution space.

Sequential versus parallel solution construction: While constructing an ACO solution, instead of establishing a path for one request after another, the paths are built in parallel. That is, in each step, an ACO procedure either adds exactly one more edge, or takes a

backtracking move. This feature changes the dynamics of the searching process that may lead to different results.

Candidate list strategy: This is a mechanism to restrict the number of available choices for consideration at each construction step. For instance, when applying ACO to the traveling salesman problem, a restriction on checking a few nearby nodes only may significantly improve the solution efficiency and quality. The modified ACO for MEDP considers only "good" choices at each construction step to speed up the process.

Different search phases: The pheromone update scheme is an important component of ACO. In the basic algorithm, all the paths (including the non-disjoint paths) of the ant solution S_{gbest} are used for updating the pheromone values. The author of [23] proposed a two-phase scheme. In the first phase, only the edge-disjoint paths are used for updating the pheromone values. The second phase kicks in when no improvement can be found over a certain period of time by using all paths to update pheromone values. Once the second phase results in any improvement, the algorithm returns to using the first phase.

Partial destruction of solutions: This mechanism helps the algorithm escape from the local solutions by removing and reconstructing some paths of the solution. This procedure is initiated once the algorithm fails to improve over a certain period of time.

In general, ACO approach has advantages over MSGA in terms of solution quality as well as computational time. The details of comparison on several benchmark instances can be found in [23].

2.4 Genetic algorithms for path-related problems

The genetic algorithm (GA) is a stochastic search method for optimization problems. It mimics the natural evolution processes using crossover, mutation and selection mechanisms to gradually improve the solution. Let $P(t)$ denote a population set of individuals in generation t and $C(t)$ the set of offspring generated by genetic operators. A general structure of the genetic algorithm is given below.

Genetic Algorithm

Begin:

1. $t \leftarrow 0$;
2. initialize $P(t)$;
3. evaluate $P(t)$;
4. **while** (terminal condition not met) do
5. recombine $P(t)$ to yield $C(t)$;
6. evaluate $C(t)$;
7. select $P(t+1)$ from $P(t)$ and $C(t)$;
8. $t \leftarrow t+1$;
9. **End**

End

Since MEDP considers the paths between several terminal pairs, we review the application of genetic algorithms for the shortest path problem in this section. The shortest path problem is to find a path between two nodes such that the path length is minimized. It is a fundamental problem involved in many applications on transportation, routing, and communications. For real-world applications, multiple and conflicting objectives are taken into consideration. Gen and Cheng [22] proposed a genetic algorithm to solve the shortest

path problem. The encoding schemes and genetic operators are summarized below.

2.4.1 Encoding methods

A gene in a chromosome is characterized by two factors: “locus” denotes the position of the gene within the structure of the chromosome, and “allele” represents the value of the gene. In [22], three different encoding schemes are investigated:

Variable-Length encoding

The variable-length encoding method is a straightforward method which consists of a sequence of positive numbers that represent the indices of nodes through which a path passes. Given a graph with n nodes, the length of the chromosome is between 1 and n. The advantage of this approach is that the mapping from a chromosome to a solution is a 1-to-1 mapping. The disadvantage is that, in general, the genetic operators shown in the next section may generate an infeasible chromosome, or in other words, a path that does not exist. Thus repairing techniques are usually applied to ensure the feasibility of the chromosome. Figure 5 shows an example of variable-length chromosome and its decoded path.

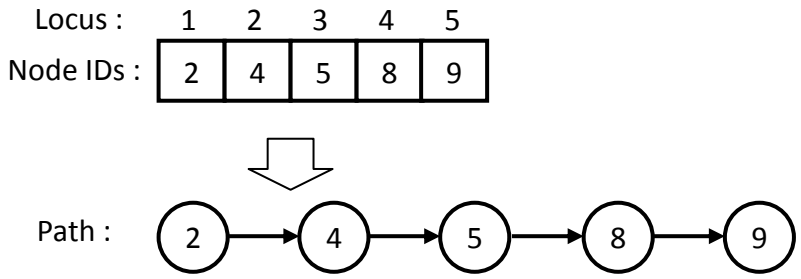


Figure 5 An example of variable-length chromosome and its decoded path

Fixed-Length encoding

This method uses a fixed-length chromosome to represent a path. To encode an arc node i to j, put j in the ith locus of the chromosome. This process is reiterated from the source node and terminated at the sink node. If a node u is not passed by the route, randomly select a node from the set of nodes that connect with u, and put it in the uth locus. The advantages of fixed-length encoding method are: (1) any permutation of the encoding corresponds to a path; (2) any path has a corresponding encoding. The disadvantages are : (1) some different chromosomes may correspond to the same path (n-to-1 mapping); (2) special genetic operators are required to generate a feasible chromosome. Figure 6 shows an example of fixed-length encoding and its decoded path.

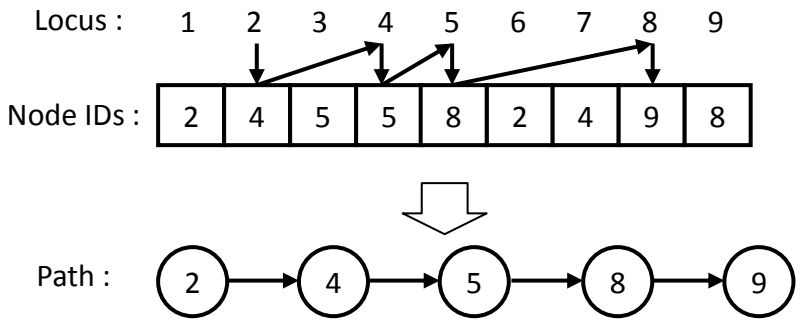


Figure 6 An example of fixed-length chromosome and its decoded path

Priority-based encoding

The priority-based method also uses a fixed length chromosome to represent a path. Given that there are n nodes, a path is encoded by a chromosome with n genes. The "locus" denotes the node ID and the "allele" represents the priority of the node. The priorities of nodes are used for constructing the path. The decoding procedure starts from scanning the source node and labeling the node with the highest priority among all nodes that are adjacent to the source node. The labeled node is put into the path. This scanning procedure restarts at the labeled node and continues until the path reaches the sink node. Illustration of the priority-based encoding method and its decoded path is shown in Figure 7. Let node 1 and node 9 be the source and sink node. At the beginning, node 2 and 4 are candidates for the next node and their priority values are 4 and 1, respectively. Since node 2 has greater priority, it is labeled and put into the path. The nodes adjacent to node 2 are node 1, 3 and 5. Node 1 is removed from the candidate set since it is already in the path. Compared with node 3, node 5 has a higher priority and, hence, it is put into the path. Repeat the process until a complete

path (1-2-5-8-9) is found.

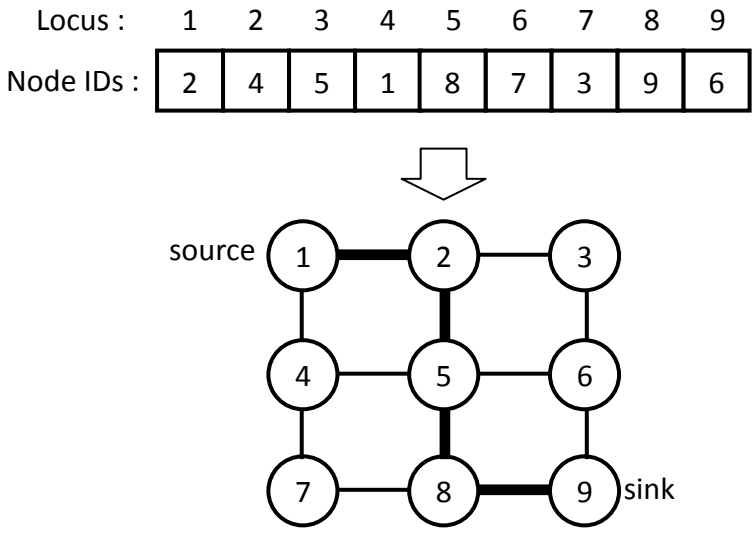


Figure 7 An example of priority-based chromosome and its decoded path

The priority-based encoding has several advantages: (1) any permutation of the encoding corresponds to a path; (2) most of the existing genetic operators can be applied; (3) any path has a corresponding encoding; (4) any point in the solution space is accessible through genetic operations. The disadvantage is also the n-to-1 mapping which lowers the searching efficiency. For instance, [2,4,5,3,8,7,1,9,6] and [3,4,5,2,8,7,1,9,6] both denote the same path (1-2-5-8-9) in Figure 7. The comparison of the three encoding methods is made in [22] and shown in Table 1.

Table 1 Summary of the performance of the three encoding methods

Chromosome Design	Space	Time	Feasibility	Uniqueness	Locality	Heritability
Variable-length	m	$O(m \log m)$	poor	1-to-1	worse	worse
Fixed-Length	n	$O(n \log n)$	worse	n-to-1	worse	worse
Priority-based	n	$O(n \log n)$	good	n-to-1	good	good

2.4.2 Genetic operators

Genetic operators mimic the process of heredity of genes to create new offspring at each generation. Using different operators may cause a huge difference in the performance of the GA procedure, therefore we reviewed below some different operators for the shortest path problem encoded by the priority-based representation.

Order Crossover

Order-crossover can be viewed as an extension of two-point crossover. It avoids the illegality caused by the simple two-point crossover. The procedure is described as follows and is illustrated in Figure 8.

Input: Two parents.

Step1: Select one substring from one parent randomly.

Step2: Generate a proto-child by copying the substring into the corresponding position of it.

Step3: Delete the nodes which are already in the proto-child from the second parent.

Step4: Place the nodes into the unfixed position of the proto-child according to the order of the sequence in the second parent.

Output: offspring.

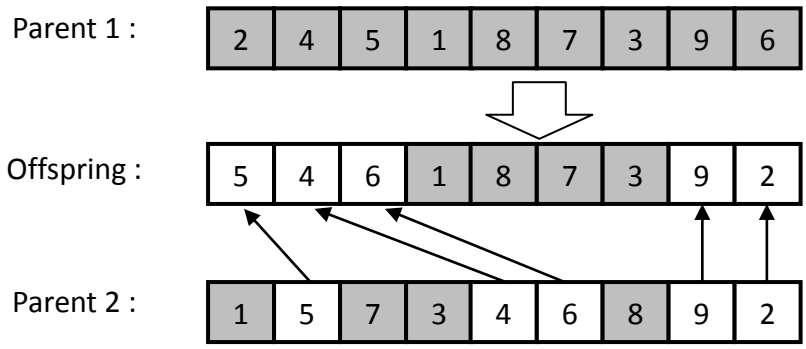


Figure 8 An illustration of Order Crossover

Position-based Crossover

Position-based crossover is essentially a uniform crossover for the permutation representation together with a repairing procedure. It can also be viewed as a variation of the order crossover where the nodes are selected separately. The procedure is illustrated in Figure 9.

Input: Two parents.

Step1: Select a set of positions from one parent randomly.

Step2: Generate a proto-child by copying the nodes on the positions into the corresponding position of it.

Step3: Delete the nodes which are already in the proto-child from the second parent.

Step4: Place the nodes into the unfixed position of the proto-child according to the order of the sequence in the second parent.

Output: offspring.

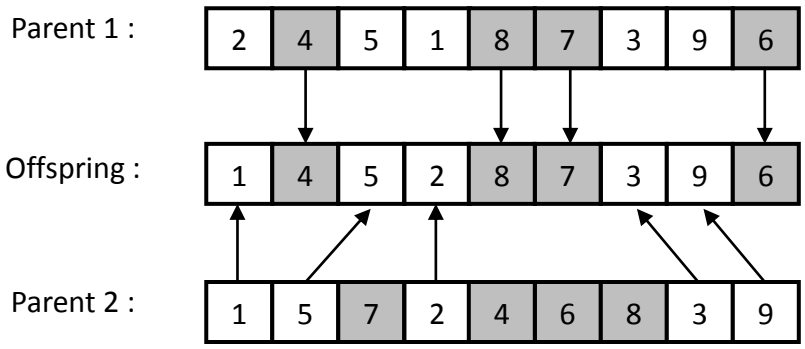


Figure 9 An illustration of Position-based Crossover

Inversion Mutation

This operator randomly selects two positions on an individual and then inverts the substring between these two positions. It is illustrated in Figure 10.

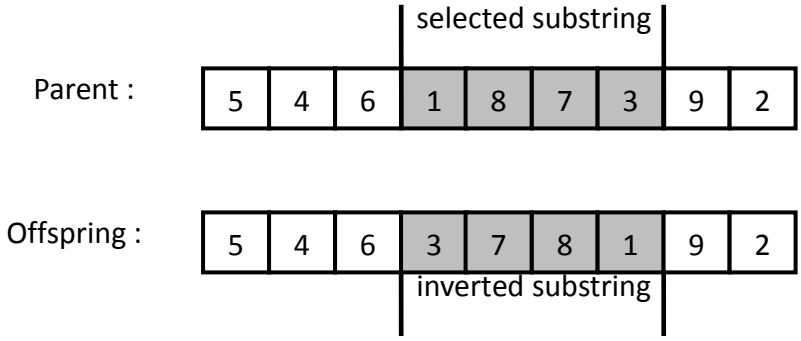


Figure 10 An illustration of Inversion Mutation

Insertion Mutation and Swap Mutation

Insertion mutation selects an element at random and inserts it in a random position as illustrated in Figure 11. Swap mutation randomly selects two elements and swaps the elements on the position as illustrated in Figure 12.

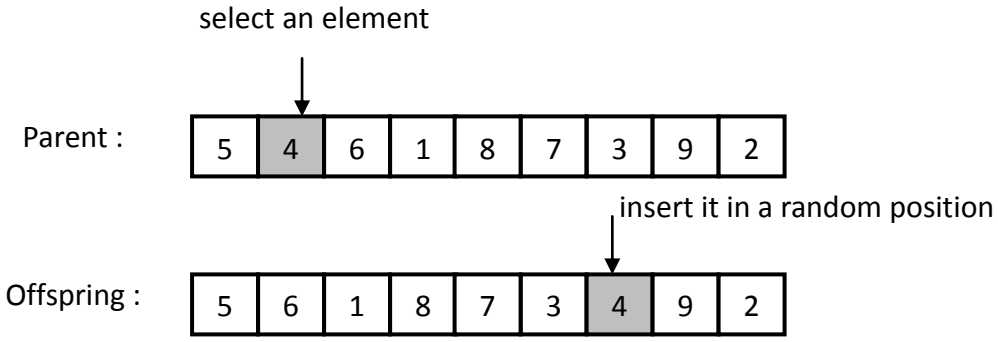


Figure 11 An illustration of Insertion Mutation

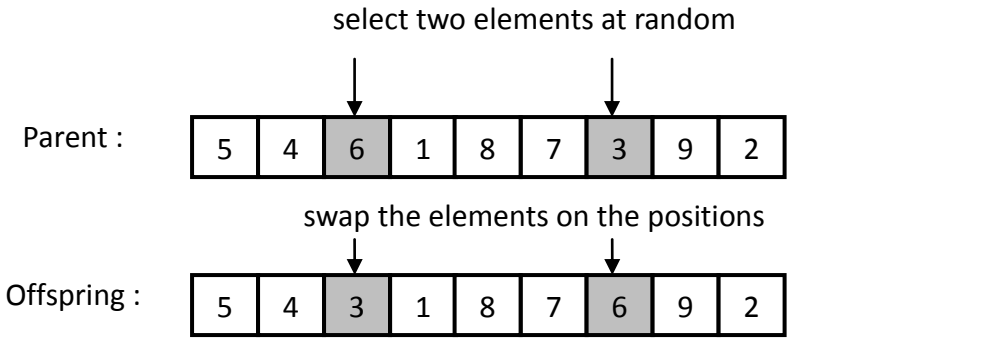


Figure 12 An illustration of Swap Mutation

2.5 Related works on RWA

The RWA problem was proven to be NP-complete [13] in 1992. The first heuristic method was proposed in [13]. Since then, different heuristic methods have been developed. Reference [12] covers different approaches and variants developed in the 1990s for RWA. A functional classification of RWA heuristics can be found in [15]. In the literature, the approaches for solving the RWA problem can be divided into two main categories. One decomposes the problem into two subproblems, the routing subproblem and wavelength assignment problem [4, 9, 10, 35, 41] to be solved separately. The other one solves the two subproblems simultaneously [26, 36, 27, 45].

Bannerjee and Mukherjee [9] employed a multicommodity flow formulation combined with randomized rounding to calculate the route for each request. After that, the wavelength assignment subproblem is solved based on the graph-coloring techniques. In which the graph, called “the conflict graph”, is built with one node corresponding to each request (and its route) and an edge exists between two nodes if their associated routes share one edge. Reference [10] also used the two-phase decomposition strategy to solve the RWA problem. First, one or more candidate routes are determined for each request by the kth-shortest path algorithm. Then the wavelength assignment problem is tackled by solving an instance of the partitioning coloring problem (PCP) defined over a partitioned conflict graph. The authors proved that the decision version of PCP is NP-complete, and proposed six heuristic methods for solving PCP. In [35], the same decomposition scheme was employed, but new algorithms for each phase were proposed. In the routing phase, candidate routes are precomputed by an edge-disjoint-paths based approach. That is, several edge disjoint paths are precomputed as

path candidates for each request. Next, a Tabu-search for solving PCP was proposed to solve the wavelength assignment problem. The initial feasible solution of PCP is provided by one of the six methods provided in [10], then a Tabu-search attempts to improve the solution by removing one color. The computational results indicated that the proposed Tabu search outperforms the best heuristic for PCP.

Generally speaking, the routing problem may be solved by using a shortest-path algorithm, an EDP-based algorithm, or a combinatorial optimization algorithm [15]. The first two types are sequential algorithms, while the last one tales a combinatorial approach.

Consequently, the wavelength assignment problem can be handled by a sequential or combinatorial approach. The sequential approach sorts routes according to different schemes. For example, routes can be sorted in descending order of their lengths. Then a wavelength is assigned to the sorted routes. For the combinatorial approach, a number of heuristic methods based on well-known graph coloring methods have been proposed.

Although dividing RWA into two subproblems allows the use of existing algorithms, good solutions for each subproblem do not guarantee a good solution to the RWA problem. Hence some algorithms treat the RWA problem as an integral problem. The first such heuristic method called Greedy-EDP-RWA was developed in [27]. It employs the solution technique in [14] to solve the maximum edge-disjoint paths problem. Compared with the one in [9], Greedy-EDP-RWA was reported to run much faster to reach an equally good solution.

The state-of-art heuristic for RWA was proposed in [26]. The author adapted some ideas from bin-packing heuristics to the RWA problem by considering each connection request as an item and copies of the original graph as bins. The weight of an item is set to be the number

of links in routing a request. To say that a bin does not have enough capacity for two items is equivalent to saying that two requests cannot be routed on the same copy of the original graph with edge-disjoint paths. Four bin-packing based heuristics were proposed in [26]: (i) first fit heuristic (FF-RWA), (ii) best fit heuristic (BF-RWA), (iii) first fit decreasing heuristic (FFD-RWA) and (iv) best fit decreasing heuristic (BFD-RWA). Computational results showed that FFD-RWA and BFD-RWA both outperform Greed-EDP-RWA [27]. Detailed descriptions of the bin-packing based algorithms will be provided later.

In [34], BFD-RWA is embedded into a biased random-key genetic algorithm. A chromosome is a vector of real numbers in the interval [0, 1]. Each gene is associated with a connection request. The requests are sorted in a non-decreasing order in terms of the sum of their lengths and genes before BFD is applied. Computational results indicate that better solutions can be found than those obtained by a multistart variant of BFD in less time on average. In recent years, other soft computing techniques such as the particle swarm optimization (PSO) [4], artificial bee colony (ABC) [44] and memetic algorithm [36] were applied to solve the RWA problem. For the PSO and ABC algorithms, several route candidates are precomputed for each request using the kth-shortest paths algorithm. A particle or a population of bees represents a set of I route IDs. Each ID represents a route which connects the corresponding connection request. During the search process, the route IDs are recombined according to different evolutionary scheme. Then a bin-packing based method is used to solve the wavelength assignment problem.

Different local search approaches were proposed in [36] and [7]. Both references construct the initial solution by BFD-RWA [26]. In [7], a variable neighborhood descent

(VND) and an iterated local search (ILS) were developed. The experimental results showed that VND-ILS is able to improve the solution quality significantly. In [36], a memetic algorithm which includes the ILS, mutation, and recombination operators was proposed. In addition, a multilevel algorithm was applied to address large size instances. The results showed that this method can be considered as the most sophisticated heuristic algorithm known in the literature.

2.6 Particle swarm optimization for RWA

2.6.1 Introduction of PSO

Particle Swarm Optimization (PSO) is an evolutionary and population based optimization algorithm, which was developed by Kennedy and Eberhart in 1995 [28]. It was inspired by the simulation of social behavior, such as bird flocking and fish schooling to find food sources. Swarm optimization takes advantages of the cooperation between individuals. In PSO, each member of the swarm is called a particle (or an individual), which utilizes two pieces of important information in a decision process. The first is their own experience; that is, the best position and its fitness value they have experienced so far. The second is other individuals' experience; that is, they have knowledge of how their neighbor individuals perform. Namely, they know the best position and its fitness value their neighbors have found so far.

The PSO algorithm initially places a number of particles in the search space randomly. Each particle evaluates its current location, and then determines its movement through the search space by combining its own current and best-fitness locations with those of one or

more members of the swarm, with some random perturbations. Specifically, the velocity of each particle is iteratively adjusted according to the best position visited by itself so far (denoted by $pbest$) and the best position obtained so far by any particle among the neighbors of the particle (denoted by $gbest$). Then the next iteration starts after all particles have been moved. The swarm eventually is likely to move close to an optimum location (food source). The pseudocode of PSO is given as below.

Particle swarm optimization

Begin:
1. Random initialization of the whole swarm;
2. **Repeat**
3. Evaluate each particle;
4. Update the current best ($gbest$ and $pbest$) positions;
5. **For** each particle
6. Update velocities;
7. Move to the new position;
8. **End for**
9. **Until** Stopping criteria

End

Let p_{pb}, p_{gb}, and x_i^k denote the previous best, global best, and current position of particle i, respectively. The velocity v_i^{k+1}, is updated according to the following equations (the superscripts denote the iteration):

$$v_i^{k+1} = w \times v_i^k + c_1 \times r_1 \times \left(p_{pb} - x_i^k\right) + c_2 \times r_2 \times \left(p_{gb} - x_i^k\right) \quad (2.3)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (2.4)$$

where $i = 1, \ldots, N$, and N is the population size. The parameter w is the inertia weight, which controls the impact of the previous velocity. The parameters c_1 and c_2 are two positive constants, where c_1 is the cognitive learning factor that represents the attraction toward the best position it had searched so far; and c_2 is the social learning factor that represents the attraction that a particle has toward the success of its neighbors. Two random numbers r_1 and r_2 are uniformly distributed in the range $[0, 1]$. Equation (2.3) determines the ith particle's new velocity v_i^{k+1}, while (2.4) moves the particle i to the new position x_i^{k+1} by adding the new velocity to the current position x_i^k. Figure 13 shows the description of the velocity and position updates of a particle in a two-dimensional space.

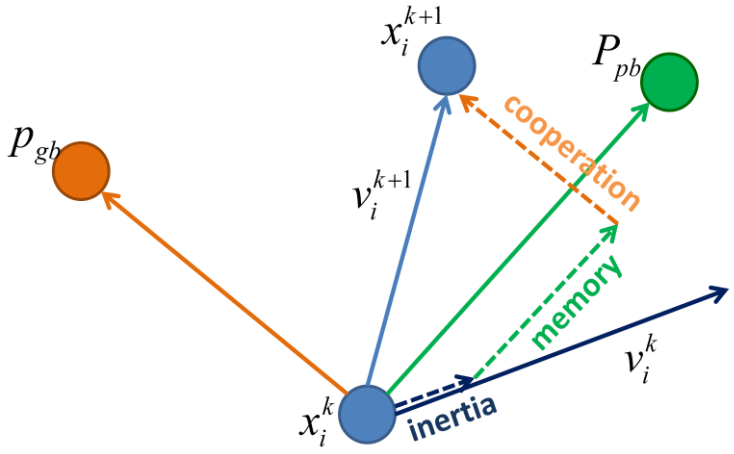


Figure 13 The velocity and position updates of a particle in a two-dimensional space

2.6.2 PSO for RWA

A PSO technique for solving RWA was proposed in [4]. In order to apply PSO for solving the RWA problem, the general PSO equations are modified so that PSO can be mapped for RWA. The velocity of movement is either influenced according to the global best or local best

position, but not both at the same time. The equations are as follow:

$$v_i^{k+1} = \alpha \times C_1 \times \left(P_{gb} - x_i^k\right) + (1-\alpha) \times C_2 \times \left(P_{lb} - x_i^k\right) \quad (2.5)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (2.6)$$

where $i = 1, \ldots, N$, and N is the population size. The parameter α is either 0 or 1; C_1 and C_2 are social learning parameters (here we let C_1=C_2), P_{gb} and P_{lb} represent the global best position and local best position, respectively.

In the PSO for RWA, the position and velocity of a particle is represented as vectors of route ids. Before the PSO starts, a *k*-shortest paths algorithm is used to produce several path candidates for each of the connection requests. Each path is identified by a given unique route-id. The particle and its velocity are represented as 1-by-I vectors of route ids. Each particle is attached with an edge usage table which shows the edge usage in terms of routes traversing over an edge in the network. This table helps determine which edges of the network will be overloaded if the routes of the current particle are chosen.

In the PSO for RWA, velocity is also a vector of route-ids that will be replaced in the current particle according to the global or local best particle. The minus operator is redefined as follows: $(P_{gb} - x_i^k)$ denotes the different routes between the gbest and the current particle. Similarly, $\left(P_{lb} - x_i^k\right)$ represents the routes that are different in the lbest and the current particle. Two social learning parameters C_1 and C_2 determine the number of routes that will be replaced. The parameter α is used to determine whether the new velocity is affected by the global best or local best particle, but not both in a single iteration for a

particle.

The add operator in equation (2.6) is redefined: the application of velocity v_i^{k+1} to the particle x_i^k means the routes in v_i^{k+1} will replace the corresponding routes in x_i^k. The particle will move to the next position x_i^{k+1} which represents a new candidate solution to the problem.

Equation (2.7) is used to quantify the quality of the solution represented by each particle of the swarm in terms of their fitness value.

$$f(x) = \frac{1}{\mathrm{APL} + \mathrm{NWL}} \tag{2.7}$$

where APL is the average path length and NWL is the number of wavelengths required to satisfy all requests. The value of NWL is obtained by calculating the number of edge disjoint path sets among the predetermined paths. The pseudocode of PSO algorithm for RWA is given below.

Particle swarm optimization for RWA

Begin:

1. For each connection request, randomly select a route from the k-shortest paths.
2. **Repeat**
3. Evaluate each particle;
4. Update the current best (gbest and lbest) positions;
5. **For** each particle
6. Find the differences routes between the current best particles;
7. Among the route set, find a given number of routes that traverse the most congested edges;
8. Replace those routes in the current particles;
9. **End for**
10. **Until** Stopping criteria

End

In addition, some strategies are proposed to improve searching ability: First, while applying velocity to a current particle, the routes that traverse the most congested edges of the network should be selected. Second, instead of replacing the route by the route of gbest (lbest), replace it with an alternative route from gbest (lbest). We refer readers to [4] for the detailed descriptions about the strategies.

2.7 BIN-packing based methods for RWA

The bin packing problem is a classical combinatorial optimization problem that has been widely studied in the literature. Given is a list of I items of various sizes, and identical bins with a limited capacity. To solve the problem, it is necessary to pack these items into the minimum number of bins, without violating the capacity constraints. Four classical algorithms for the bin packing problem are the First Fit (FF), Best Fit (BF), First Fit Decreasing (FFD) and Best Fit Decreasing (BFD) algorithms. The FF algorithm packs each item into the bin with the lowest index. On the other hand, the BF algorithm packs each item into the bin which leaves the least room left over after packing the item. The FFD and BFD algorithm first place larger items into bins and then fill up remaining space with smaller items.

To apply bin-packing methods to solve the RWA problem, we must define bins, items, and their corresponding size in terms of optical networks. Skorin-Kapov of [26] considered using lightpath requests to represent items and using duplicates of graph G to represent bins.

Each copy of G, i.e., G_j, $j = 1,2, \ldots,$ corresponds to one wavelength. Let the size of each lightpath π_i be represented by the length of its shortest path SP_i in graph G_j. To solve the RWA problem, we wish to pack as many items (lightpaths) into a minimum number of bins (copies of G), and hence the number of used wavelengths is minimized.

The FF algorithm runs as follows. Fisrt, only one copy of G, bin G_1, is created. Higher indexed bins are created as needed. Lightpath requests are selected and routed on the lowest indexed copy of G if the length of the shortest path on such graph is less than the threshold H, which is set to be $\max(diam(G), \sqrt{|E|})$ as suggested in [27]. If a lightpath is routed in bin G_j, the lightpath is assigned wavelength j and the edges along such path are removed from G_j. A new bin is created if no existing bin can accommodate the request. On the other hand, the FFD sorts the requests in a nonincreasing order in terms of the lengths of their shortest paths in G. The motivation is that, the connection request with the longest shortest path is usually harder to route. Therefore the strategy of considering these requests first then filling up the remaining space with the requests having the shortest routes may lead to fewer wavelengths used. The pseudocode of FF and FFD algorithms are shown as below.

Algorithm 6 FF_RWA (FFD_RWA) algorithm

Input:
$G = (V, E)$;
$T = \{(s_i, t_i) | i = 1, \ldots, \mathrm{I}\}$;
H;
Begin:
1. (ONLY FOR FFD_RWA: sort demands T in a non-increasing order in terms of the lengths of their shortest paths in G)
2. $\pi = \emptyset$;

3. Create $G_1 \coloneqq G$;
4. BINS $\coloneqq \{G_1\}$;
5. **For** $i = 1$ to I
6. $\pi_i = \emptyset$;
7. **For** $j = 1$ to $|\text{BINS}|$
8. Find shortest path SP_i^j for (s_i, t_i) in G_j;
9. **If** $\left|\text{SP}_i^j\right| \leq H$ **then**
10. $\pi_i = \text{SP}_i^j$;
11. $w_i = j$;
12. Remove edges in π_i from G_j;
13. **Break**;
14. **End if**
15. **End for**
16. **If** $\pi_i = \emptyset$ **then**
17. NEW $= |\text{BINS}| + 1$;
18. Create $G_{\text{NEW}} \coloneqq G$;
19. BINS $=$ BINS $\cup \{G_{\text{NEW}}\}$;
20. Find shortest path SP_i^{NEW} for (s_i, t_i) in G_{NEW};
21. $\pi_i = \text{SP}_i^{\text{NEW}}$;
22. $w_i =$ NEW;
23. Remove edges in π_i from G_{NEW};
24. **End if**
25. $\pi = \pi \cup \pi_i$;
26. $T = T \backslash (\text{s}_i, \text{t}_i)$;
27. **End for**

End

Output: π and w

The Best Fit bin packing algorithm routes requests in the bin which they fit "best". The best bin is considered to be the one in which the request can be routed on the shortest path. In other words, assume there are B existing bins, bin J is the best bin for lightpath request i if and only if $J = \text{argmin}_{j=1,\ldots,\text{B}}\left|\text{SP}_i^j\right|$, where SP_i^j denotes the shortest path of request i in G_j. The pseudocode of BF and BFD algorithm are shown as follows.

Algorithm 7 BF_RWA (BFD_RWA) algorithm

Input:

$G = (V, E)$;

$T = \{(s_i, t_i) | i = 1, \dots, \mathrm{I}\}$;

H;

Begin:

1. (ONLY FOR BFD_RWA: sort demands T in a non-increasing order in terms of the lengths of their shortest paths in G)
2. $\pi = \emptyset$;
3. Create $G_1 \coloneqq G$;
4. BINS $\coloneqq \{\mathrm{G}_1\}$;
5. **For** $i = 1$ to I
6. $\pi_i = \emptyset$, $\mathrm{l}(\pi_i) = \infty$;
7. BestBIN $= 0$;
8. **For** j $= 1$ to $|\mathrm{BINS}|$
9. Find shortest path SP_i^j for (s_i, t_i) in G_j;
10. **If** $\left|\mathrm{SP}_i^j\right| \leq \mathrm{H}$ and $\left|\mathrm{SP}_i^j\right| < |\pi_i|$ **then**
11. $\pi_i = \mathrm{SP}_i^j$;
12. $w_i = j$;
13. BestBIN $= j$;
14. **End if**
15. **End for**
16. **If** $\pi_i \neq \emptyset$ **then**
17. Remove edges in π_i from G_{BestBIN};
18. **else**
19. NEW $= |\mathrm{BINS}| + 1$;
20. Create $G_{\mathrm{NEW}} \coloneqq G$;
21. BINS $=$ BINS $\cup \{G_{\mathrm{NEW}}\}$;
22. Find shortest path $\mathrm{SP}_i^{\mathrm{NEW}}$ for (s_i, t_i) in G_{NEW};
23. $\pi_i = \mathrm{SP}_i^{\mathrm{NEW}}$;
24. $w_i = \mathrm{NEW}$;
25. Remove edges in π_i from G_{NEW};
26. **End if**
27. $\pi = \pi \cup \pi_i$;
28. $T = T \backslash (s_i, t_i)$;

29. **End for**
End
Output: π and w

2.8 Known lower bounds

Since the known algorithms for the RWA problem are heuristics, it is useful to have a good lower bound in order to assess the quality of suboptimal solutions. Finding good lower bounds is not trivia. The task may still be time-consuming. Different approaches have been developed to determine lower bounds. We can either estimate a lower bound according to the problem instance's properties, or relax constraints of the problem formulation to solve an easier problem. Usually, estimations are easily available but are often far below the optimal solution. Lower bounds obtained by relaxation can be tighter, but at the cost of computational time.

For simplicity, we only introduce an easy lower bound for the RWA problem using the estimation approach provided in [26] as below.

$$\mathrm{LB} = \max\left\{\max_{u\in V}\left\lceil\frac{\Delta_l(u)}{\Delta_p(u)}\right\rceil, \left\lceil\frac{\sum_{i=1}^{\mathrm{I}}|SP_i|}{|E|}\right\rceil\right\} \tag{2.8}$$

where $\Delta_l(u)$ is the logical degree of node u, i.e., the number of requests in which node u is the source node; $\Delta_p(u)$ represents the physical degree of node u; $|SP_i|$ is the length of the shortest path in G of request (s_i, t_i). The lower bound has two elements. The first one represents the maximum ratio of logical to physical degree of any node in G, rounded up to the first higher integer. If a node u has $\Delta_p(u)$ adjacent edges and is one of the endpoints for $\Delta_l(u)$ requests, at least one physical link will have $\left\lceil\frac{\Delta_l(u)}{\Delta_p(u)}\right\rceil$ requests routed over it.

Therefore a number of $\left\lceil \frac{\Delta_l(u)}{\Delta_p(u)} \right\rceil$ wavelengths are required due to the wavelength clash constraint. The second component of (2.8) is the distance (assume each edge has one unit of cost) of each request's shortest path divided by the number of edges available in the graph.

Two other lower bounds were proposed in [5, 36]. In [36], the lower bound is obtained by relaxing the wavelength continuity constraint, thus the RWA becomes a multicommodity flow problem, where each request is a unique commodity with one unit of demand. The commodity need to be routed through the problem instance's network G. The other lower bound introduce in [5] also relaxes the wavelength continuity constraint and translates the problem into a maximum cut problem. Both are more sophisticated methods to obtain stronger lower bounds than the one provided in (2.8), but they are not applicable for large instances due to their prohibitive computation time.

Chapter 3 Proposed genetic algorithm for MEDP

In this chapter, we propose a genetic algorithm for solving the maximum edge-disjoint paths problem. A typical genetic algorithm has four basic components: (i) a genetic representation, (ii) a method to find an initial solution, (iii) an evaluation function in terms of the fitness of an individual and (iv) genetic operators that produce offspring. A good genetic representation is a key issue while using the genetic algorithm. Here we adopt the priority-based encoding method to represent a path by an $1 \times n$ vector, in which each element is a real value in [0,1]. Each individual includes I paths with such representation method. Throughout the rest of the chapter, we will use the terms "individual," "solution," and "chromosome" interchangeably.

In Section 3.1, we discuss how to transform MEDP with pre-determined paths into a maximum independent set (MIS) problem. A greedy algorithm for solving MIS can be applied to extract the edge-disjoint paths from a set of given paths. In Section 3.2, the encoding/decoding procedures are given. A simple heuristic is proposed to generate the initial population in Section 3.3. The genetic operators are described and some small examples are provided in Section 3.4. In Section 3.5, we present a simple heuristic to improve the solution after evaluating the offspring. Finally, the evaluation and selection mechanisms are given in Sections 3.6 and 3.7, respectively.

3.1 MEDP with pre-determined paths

The objective of MEDP is to maximize the realizable connection requests through edge-disjoint paths. Two questions arise naturally:

1. How to construct the path between two terminals of a connection request?

2. If the paths are known, how to decide whether a request should be accepted or rejected?

Most of the existing methods are greedy algorithms which usually apply the shortest path algorithm to construct the path, then remove all edges along that path from the graph. Therefore the second question is not relevant. Since greedy algorithms build the paths one by one corresponding to a given order of the connection requests, the quality of the solution depends heavily on the given order. In our work, instead of removing edges from the graph after a path is built, we begin with relaxing the edge-disjoint constraint and assigning a path to every request, then obtain the maximum number of EDPs among all of these paths.

Here we describe a key idea of the proposed approach. Given an undirected graph G and a set of connection requests $T = \{(s_i, t_i) \in V \times V | i = 1, \dots, \mathrm{I} \text{ and } s_i \neq t_i\}$. A set of paths $S_P = \{p_i |\, i = 1, \dots \mathrm{I}\}$ is also given (these paths are not necessarily edge-disjoint), where each path p_i connects the terminal pair (s_i, t_i). How do we find the maximum number of edge-disjoint paths from S_P? A conflict graph $\tilde{G} = (\tilde{V}, \tilde{E})$ is built with each node $i \in \tilde{V}$ corresponding to a connection request (s_i, t_i) in the original MEDP. Hence $|\tilde{V}| = \mathrm{I}$. And there is an edge between two nodes $u, v \in \tilde{\mathrm{V}}$ if the two paths p_u and p_v have some edges in common in G. In this way, solving MEDP with pre-determined paths is equivalent to finding a so-called "maximum independent set (MIS)" on $\tilde{G}$.

In graph theory, we call a set of nodes an independent set in a graph, if there are no two of which are adjacent. A maximum independent set (MIS) is the largest independent set for a given graph. Finding an MIS in a graph is a well-known NP-complete problem. Two greedy algorithms, *GMIN* and *GMAX*, have been investigated in [32]. *GMIN* selects a vertex of the minimum degree and removes it and its neighbors from the graph. This process is iterated on the remaining graph until no vertex remains. The set of selected vertices then form an independent set. In contrast, *GMAX* deletes a vertex of the maximum degree until no edges remain. In this case, the set of remaining vertices is an independent set. In our proposed method, *GMIN* is applied to find the MIS on $\widetilde{\mathrm{G}}$ since it can achieve a better lower bound than *GMAX* [32].

3.2 Encoding/Decoding procedures

Representing paths in a graph is critical for developing a genetic algorithm for MEDP. Different methods for encoding a path on a graph were reviewed in Section 2.4.1. The priority-based encodings method uses a fixed-length code to represent a path. Although several encodings may correspond to the same path (n-to-1 mapping), priority-based encoding has some good characteristics compared to other methods (see Table 1). Thus we adopt this scheme to represent a path. The priority values are assigned in the interval $[0,1]$.

A solution of MEDP involves several paths, which means that an individual needs to carry the information of these paths. Each individual contains I vectors, with each vector i has n elements (or priority values) representing the specific path corresponding to the connection request (s_i, t_i). Let u_i^j denote the ith vector of individual j,

Figure 14 shows the structure of the chromosomes. Each individual j, denoted by $\boldsymbol{u}^j$, is a collection of vectors $\{u_1^j, u_2^j, \ldots, u_i^j, \ldots, u_I^j\}$, where u_i^j is a $1 \times n$ vector of priority values representing the path that connects (s_i, t_i).

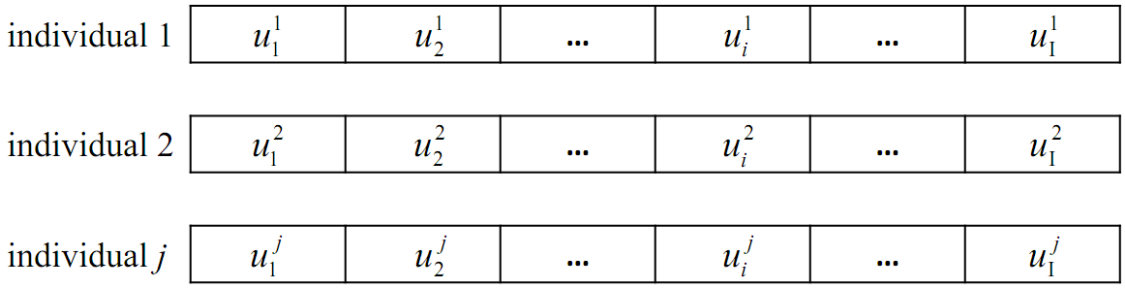


Figure 14 The structure of a chromosome

Algorithm 8 describes the details of the decoding procedures. Basically, decoding is a procedure of path construction. At the beginning, the path p only contains the source node s. The current node, denoted by cur, is set to be s. All the unlabeled and neighbor nodes of cur, which are the candidates for the next move, are denoted by C. If C is not empty, the node with the greatest priority value in C is added into p and becomes the current node. The label of the current node is set to be 1. If C is empty, the path backtracks by setting the second last node in p to be the current node and removing the last node of p. The path construction procedure stops when the path reaches the destination node ($cur = t$). If two or more nodes have the same priority, choose the one with the smallest node index to break the tie. For example, if nodes 1, 3, 7 have the same priority values, then node 1 is chosen to be the next node

Algorithm 8 Decoding Procedure

Input: a $1 \times n$ vector u denoting a path from s to t,
source s,
sink t,
A_{cur} is a set of nodes adjacent to node cur.

Begin:

1. $cur \leftarrow s,\ len \leftarrow 1,\ p \leftarrow \{s\},\ l \leftarrow \vec{0}$;
2. **while** $cur \neq t$
3. $C \leftarrow \{\mathrm{j} | \mathrm{j} \in A_{cur}, l_{\mathrm{j}} = 0\}$;
4. **if** $C \neq \emptyset$ **then**
5. $len \leftarrow len + 1$;
6. $cur \leftarrow \mathrm{argmax_j}\{u_{\mathrm{j}},\ \mathrm{j} \in C\}$;
7. $l_{cur} \leftarrow 1$;
8. $p \leftarrow [p\ cur]$;
9. **else**
10. $\pi_{len} \leftarrow 0$;
11. $len \leftarrow len - 1$;
12. $cur \leftarrow \pi_{len}$;
13. **end if**
14. **end while**

End

Output: a path p between s and t

Algorithm 9 shows the pseudocode of the encoding procedure. Given a node sequence p, the encoding procedure generates a priority vector $u = (u_1, \ldots, u_n)$. Starting from $w = 1$, the element u_{p_w} is assigned the priority value $\frac{n-w+1}{n}$. Thus the starting node will have the highest priority, and the second node in p will be assigned the second highest priority, and so on. For the nodes that are not in the path, their priority values are randomly generated within the range [0, $\frac{n-|p|}{n}$). In this way any of the nodes in p has higher priority than those

that are not. For instance, the encoding procedure of the path $(1, 2, 5, 8, 9)$ on a graph with 9 nodes is as follows: For nodes 1, 2, 5, 8, 9, their priority values are $u_1 = \frac{9-1+1}{9}$, $u_2 = \frac{9-2+1}{9}$, $u_5 = \frac{9-3+1}{9}$, $u_8 = \frac{9-4+1}{9}$, $u_9 = \frac{9-5+1}{9}$, respectively. For the nodes not in the path (nodes 3, 4, 6 and 7), their priority values are randomly drawn from $[0, \frac{9-5}{9}]$. A code representing the path can be $u = (1, {}^{8}/_{9}, {}^{4}/_{9}, {}^{3}/_{9}, {}^{7}/_{9}, {}^{2}/_{9}, {}^{1}/_{9}, {}^{6}/_{9}, {}^{5}/_{9})$.

Algorithm 9 Encoding Procedure

Input: a node sequence p denoting a path,
n is the number of nodes in the graph.
Begin
1. $u \leftarrow \vec{0}$.
2. **for** w=1 to $|p|$
3. **if** $u_{p_w} = 0$ then
4. $u_{p_w} \leftarrow \frac{n-w+1}{n}$;
5. **end if**
6. **end for**
7. J $\leftarrow \{j|u_j = 0\}$;
8. **for** j $\in$ J
9. $u_j \leftarrow$ random real value in $[0, \frac{n-|p|}{n})$;
10. **end for**
End
Output: encoded path u

As described in the previous section, the I paths in an individual are not necessarily edge-disjoint. Once the paths are determined, an $I \times I$ path relation matrix $\mathbb{P}$ can be generated, where $p_{uv} = 1$ if path u and v share the same edge; otherwise $p_{uv} = 0$. A simple heuristic *GMIN* is applied for obtaining the paths that are actually edge-disjoint. At

each iteration, the request that has the least number of interfering requests is accepted (two requests interfere if their paths have an edge in common). Then the request and all the interfering requests are removed from the request list. Repeat the iteration until the request list is empty. The detail of *GMIN* is described in **Algorithm 10**, where M is a set of request indices, and N_u is the set of indices of the requests interfering with request u.

Algorithm 10 *GMIN* for MEDP

Input: $\mathbb{P}$ is a IxI matrix.
Begin:
1. $R \leftarrow \emptyset$, $\mathrm{M} \leftarrow \{1,2,\dots,\mathrm{I}\}$,$e \leftarrow (1,1,\dots,1)^{\mathrm{T}}$;
2. **while** $\mathrm{M} \neq \emptyset$
3. $x = \mathbb{P}e$;
4. $u = \mathrm{argmin}_{i\in\mathrm{M}}\{x_i\}$;
5. $\mathrm{N}_u \leftarrow \{v|\mathrm{p}_{uv} = 1\}$;
6. $R \leftarrow R \cup \{u\}$;
7. $\mathrm{M} \leftarrow \mathrm{M}\backslash(\mathrm{N}_u \cup \{u\})$;
8. set the q^{th} column and row of $\mathbb{P}$ to zeros, $\forall q \in (\mathrm{N}_u \cup \{u\})$;
9. **end while**

End
Output: a realizable set R

3.3 Initial population

The initial priority values of all the individuals are generated randomly by drawing values from $[0,1]$. A total of J individuals are further modified by a heuristic method. The heuristic method works as follows. First, the shortest path distance d_i is calculated (letting the distance of each edge be one) for each request i. The connection requests is sorted in an ascending order of distance and the sorted permutation is denoted by ρ_s. The corresponding

connection request is denoted by T_s. Considering the permuted request set T_s, a realizable request set and its corresponding edge-disjoint paths can be obtained by applying the Simple Greedy Algorithm (SGA) as described **Algorithm 1** in Section 2.3.2. We then encoded these edge disjoint paths as the first individual.

More individuals can be initialized by exchanging the order of two requests in T_s and then run another SGA to obtain a new realizable set of requests and the corresponding edge-disjoint path. To avoid swapping the request that has the "longest" shortest path with the one has the "shortest" shortest path, which is very likely to generate a worse solution, we cut T_s in half and forbid the swap operation taking place between different halves. Figure 15 illustrates the swap operation. The pseudocode of initialization heuristic is given in **Algorithm 11**. We denote individual j as $\boldsymbol{u}^j = \{u_1^j, u_2^j, \dots, u_i^j, \dots u_{\mathrm{I}}^j\}$, where u_i^j is a $1 \times n$ priority vector denoting a path for the request i.

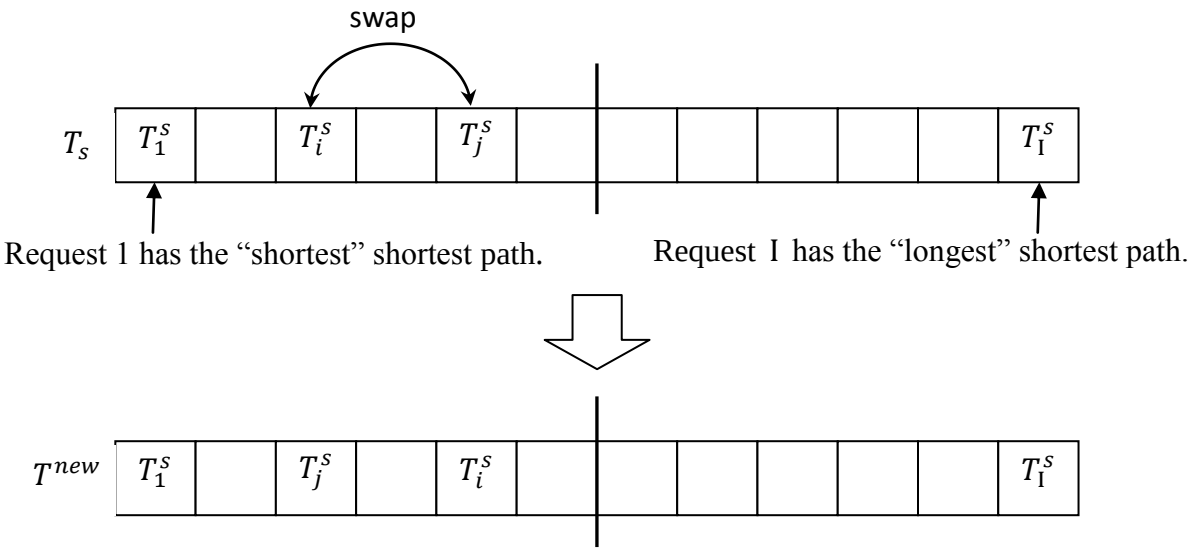


Figure 15 Swap operation generates a new initial individual

Algorithm 11 Initialization Heuristic

Input: G, T;

J is the number of individuals generated by this heuristic;

Begin:

1. $j \leftarrow 1$;
2. $\rho \leftarrow$(1,2…,I);
3. **for** $i = 1$ to I
4. $d_i \leftarrow$ shortest path distance between (s_i, t_i)**;**
5. **end for**
6. $\rho_s \leftarrow$*SortRequest*(ρ, d)**;**
7. $T_i^s = \{(s_{\rho_s(i)}, t_{\rho_s(i)}) | i = 1, \dots, \mathrm{I}\}$**;**
8. $(R, S) \leftarrow SimpleGreedyAlgorithm(G, T_s)$**;**
9. $\boldsymbol{u}^j \leftarrow encoding(S)$**;**
10. **while** $j < J$
11. $\rho^{new} \leftarrow \mathrm{SwapRequest}(\rho_s)$;
12. $T_i^{new} = \{(s_{\rho^{new}(i)}, t_{\rho^{new}(i)}) | i = 1, \dots, \mathrm{I}\}$;
13. $(R, S) \leftarrow \mathrm{SimpleGreedyAlgorithm}(G, T^{new})$;
14. $\boldsymbol{u}^{j+1} \leftarrow \mathrm{encoding}(S)$;
15. $j \leftarrow j + 1$;
16. **end while**

End

Output: J initial individuals $\boldsymbol{u}^1, \boldsymbol{u}^2, \dots, \boldsymbol{u}^J$

3.4 Genetic operators

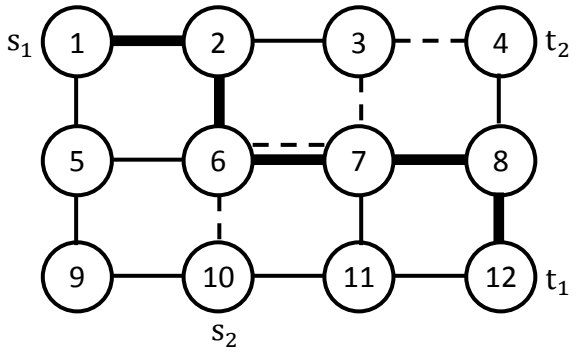
Genetic operators mimic the process of heredity of genes to create the offspring. Using different operators may cause a great impact on GA performance. In Section 2.4.2, we examined several operators for priority-based representation. In general, permutation representation may yield illegal offspring by using the two-point or multi-point crossover because some priority values may be missed or duplicated. Therefore, a repairing procedure

is required when these approaches are applied. Here we use the real-valued priorities. Since most of the individuals are randomly generated, any two priority values are unlikely to be the same. Consequently, the repairing process is unnecessary. Three genetic operators are introduced as follows.

3.4.1 Crossover Operator

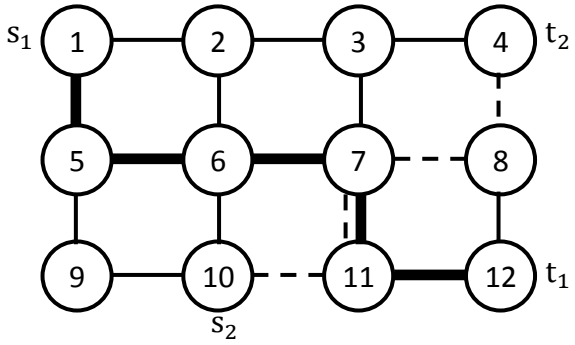
The crossover operator generates one offspring by the weighted linear combination of parents. The parents are chosen by roulette-wheel and the weight is randomly generated. The essence of this operator is blind random search, hence there is no guarantee that the offspring generated by this method is better than its parent.

We use a simple example of a 3x4 mesh graph with the connection requests $T = \{(1,12),(10,4)\}$ to illustrate the process. The path set represented by the first individual $\boldsymbol{u}^1 = \{u_1^1, u_2^1\}$ is shown in Figure 16, in which u_1^1, u_2^1 denote the first individual's priority vectors for the two requests. The bold line and the dashed line denote the paths decoded from u_1^1 and u_2^1, which correspond to the first and second requests, respectively. Figure 17 shows the path set of the second individual $\boldsymbol{u}^2 = \{u_1^2, u_2^2\}$. We can see that the number of EDPs of individuals 1 and 2 are both one. Figure 18 is the path set represented by the offspring generated by letting $u_1^{crs} = \alpha u_1^1 + (1-\alpha)u_1^2 \; and \; u_2^{crs} = \alpha u_2^1 + (1-\alpha)u_2^2$ with $\alpha = 0.5$. Luckily, the number of EDPs is increased by one.



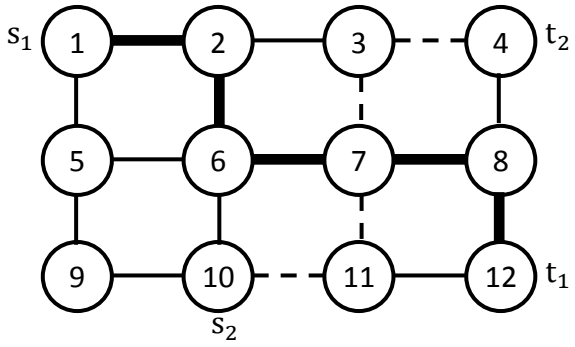
	1	2	3	4	5	6	7	8	9	10	11	12
u_1^1	**0.31**	**0.92**	0.30	0.22	0.13	**0.80**	**0.74**	**0.63**	0.15	0.21	0.43	**0.50**
u_2^1	0.51	0.48	**0.73**	**0.68**	0.41	**0.90**	**0.82**	0.43	0.30	**0.33**	0.21	0.10

Figure 16 Chromosome 1 and its representative path set



	1	2	3	4	5	6	7	8	9	10	11	12
u_1^2	**0.13**	0.42	0.33	0.20	**0.94**	**0.9**	**0.81**	0.59	0.57	0.40	**0.73**	**0.62**
u_2^2	0.32	0.40	0.52	**0.68**	0.44	0.10	**0.87**	**0.71**	0.23	**0.11**	**0.94**	0.80

Figure 17 Chromosome 2 and its representative path set



	1	2	3	4	5	6	7	8	9	10	11	12
u_1^{crs}	**0.22**	**0.67**	0.315	0.21	0.535	**0.85**	**0.775**	**0.61**	0.36	0.305	0.58	**0.56**
u_2^{crs}	0.415	0.44	**0.625**	**0.68**	0.425	0.50	**0.845**	0.57	0.265	**0.22**	**0.575**	0.45

Figure 18 The offspring and its representative path set

3.4.2 Mutation Operator

The mutation operator generates one offspring from an individual. In some sense, the priority value of a node represents the "preference" of a node while we are building the path. The offspring generated by the mutation operator (denoted by $\boldsymbol{u}^{mu}$) has the same priority vectors as its parent, except for a randomly picked vector u_j^{mu} is mutated by letting $u_j^{mu} = e - u_j^p$, where e is an $1 \times n$ vector of 1s, $\boldsymbol{u}^p$ is the parent individual and j is an integer randomly draw from $[1, \mathrm{I}]$. The mutation operator can be stated as follow:

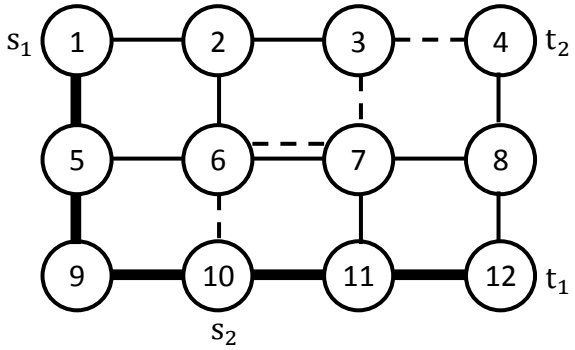
$$u_i^{mu} = \begin{cases} e - u_i^p, & \text{if } i = j \\ u_i^p, & \text{otherwise} \end{cases}, \text{ for } i = 1, \dots, \mathrm{I}. \qquad (3.1)$$

For instance, mutating u_1^1 in Figure 16 gives a new priority vector u_1^{mu} as follows.

$$u_1^{mu} = e - u_1^1$$
$$= [1.00\ 1.00\ 1.00\ 1.00\ 1.00\ 1.00\ 1.00\ 1.00\ 1.00\ 1.00\ 1.00\ 1.00]$$
$$- [0.31\ 0.92\ 0.30\ 0.22\ 0.13\ 0.80\ 0.74\ 0.63\ 0.15\ 0.21\ 0.43\ 0.50]$$
$$= [0.69\ 0.08\ 0.70\ 0.78\ 0.87\ 0.20\ 0.26\ 0.37\ 0.85\ 0.79\ 0.57\ 0.50] \text{ and}$$

$u_2^{mu} = u_2^1$.

The new offspring is shown in Figure 19, the path decoded from u_1^{mu} is in bold line. Obviously, two edge-disjoint paths can be found now.



	1	2	3	4	5	6	7	8	9	10	11	12
u_1^{mu}	**0.69**	0.08	0.70	0.78	**0.87**	0.20	0.26	0.37	**0.85**	**0.79**	**0.57**	**0.50**
u_2^{mu}	0.51	0.48	**0.73**	**0.68**	0.41	**0.90**	**0.82**	0.43	0.30	**0.33**	0.21	0.10

Figure 19 The offspring generated by mutation operator

3.4.3 Self-Adaption Operator

The self-adaption operator generates one offspring $\boldsymbol{u}^{SA}$ from a randomly selected individual. Similar to the mutation operator, it randomly selects and reroutes a rejected request l, where $(s_l, t_l) \in$ T\R, by assigning new priority values to u_l^{SA}. The priority vectors for other

requests stay the same as its ancestor. Two factors are taken into consideration in order to construct a better solution: First, a longer path (assume that the distance of each edge is 1) is less preferred because its "intersection" with other paths carries a higher probablility. The all-pairs shortest distance matrix $\mathbb{D}$ obtained at the initialization stage provides useful information for this priority adjustment.

Second, if we want to reroute a path, the new path better not include an occupied edge (which is already taken 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 chromosome is encoded as node priorities. An alternative way to serve the purpose is to assign a higher priority value to a node, which is adjacent to more available edges.

To apply the self-adaption operator, the all-pairs shortest path matrix $\mathbb{D}_{nxn}$, the incidence matrix $\mathbb{A}_{nxm}$ and the $m \times 1$ indicator vector $\boldsymbol{v^E}$ obtained in the evaluation process are required. We define $\boldsymbol{v^E}$ as follows:

$$v_j^E = \begin{cases} 1, & \text{if edge } j \text{ is available} \\ 0, & \text{otherwise} \end{cases}, \text{ for } j = 1, \dots, m. \tag{3.2}$$

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:

$$\boldsymbol{v} = (\mathbb{A}\boldsymbol{v^E})^{\mathrm{T}} \tag{3.3}$$

$$\widehat{\mathbb{D}}_{t_l} = \left\|\mathbb{D}_{t_l}\right\| \cdot \mathrm{e}^{\mathrm{T}} - \mathbb{D}_{t_l} \tag{3.4}$$

$$\hat{u}_l^{SA} = \frac{\widehat{\mathbb{D}}_{t_l}}{\left\|\widehat{\mathbb{D}}_{t_l}\right\|} + \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} \tag{3.5}$$

$$u_l^{SA} = \frac{\hat{u}_l^{SA}}{\left\|\hat{u}_l^{SA}\right\|} \tag{3.6}$$

Here we use the maximum norm $\|\boldsymbol{x}\| = \max\{|x_1|, |x_2|, \dots, |x_n|\}$ to control the priority value in the interval $[0,1]$. In (3.3), $\boldsymbol{v}$ is a $1 \times n$ vector representing the number of available adjacent edges of each node. In (3.4), $\mathbb{D}_{t_l}$ 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 $\mathbb{D}_{t_l}$. In (3.5), the weighted average of two factors is assigned to $\hat{u}_l^{SA}$. Equation (3.6) normalizes the $\hat{u}_l^{SA}$ to [0,1] and assigns it to u_l^{SA}.

Take the individual in Figure 16 as example. If we apply the operator to the second request, that is, $l = 2, (s_2, t_2) = (10,4)$, we have

$$\boldsymbol{v} = [1\ 1\ 3\ 2\ 3\ 2\ 2\ 1\ 2\ 3\ 3\ 1]$$

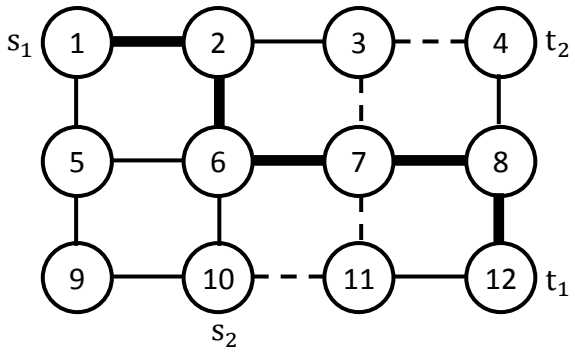
$$\mathbb{D}_4 = [3\ 2\ 1\ 0\ 4\ 3\ 2\ 1\ 5\ 4\ 3\ 2]$$

$$\widehat{\mathbb{D}}_4 = [2\ 3\ 4\ 5\ 1\ 2\ 3\ 4\ 0\ 1\ 2\ 3]$$

$$\hat{u}_2^{SA} = [\frac{11}{15}\ \frac{14}{15}\ \frac{9}{5}\ \frac{5}{3}\ \frac{6}{5}\ \frac{16}{15}\ \frac{19}{15}\ \frac{17}{15}\ \frac{2}{3}\ \frac{6}{5}\ \frac{7}{5}\ \frac{14}{15}]$$

$$u_2^{SA} = [\frac{11}{27}\ \frac{14}{27}\ 1\ \frac{25}{27}\ \frac{2}{3}\ \frac{16}{27}\ \frac{19}{27}\ \frac{17}{27}\ \frac{10}{27}\ \frac{2}{3}\ \frac{7}{9}\ \frac{14}{27}]$$

Figure 20 illustrates the paths represented by the new individual. We can see that the self-adaption operator reroutes the second request to a better path (in dash line).



	1	2	3	4	5	6	7	8	9	10	11	12
u_1^1	**0.31**	**0.92**	0.30	0.22	0.13	**0.80**	**0.74**	**0.63**	0.15	0.21	0.43	**0.50**
u_2^{SA}	0.41	0.52	**1**	**0.93**	0.67	0.59	**0.70**	0.63	0.37	**0.67**	**0.78**	0.52

Figure 20 The offspring generated by self-adaption operator

3.5 Improvement heuristics

Researchers have shown that local improvement heuristics add a great deal of benefit to GA. In our algorithm, the proposed heuristics are performed after an offspring is evaluated. In the evaluation process, an individual is decoded into I paths. Then *GMIN* is performed to acquire the maximum number of EDPs and a realizable set R. A residual graph, denoted by $\bar{G}$, which is the graph after removing all edges involved with the EDPs from the original graph G, is obtained. Note that in the residual graph $\bar{G}$, there may exist some paths which can connect some of the rejected requests. The purpose of the improvement procedure is to find some new paths for the unrealizable connection requests in $\bar{G}$. If such paths can be found, they must be edge-disjoint to all other paths in S since the routes are build in $\bar{G}$. The

new paths are then encoded to the offspring and the solution is improved. The pseudocode of the improvement heuristic is shown in **Algorithm 12**.

Algorithm 12 Improvement Heuristics

Input: $\bar{G}$ is the residual graph,
U is the unrealizable set,
T is the connection requests
Begin:
1. **if** $|U|>0$
2. **for** $j = 1$ to $|U|$
3. $p_{new} \leftarrow$*PathConstruction*(s_{U_j}, t_{U_j});
4. **if** p_{new} exists
5. $p_{U_j} \leftarrow p_{new}$;
6. Encode p_{U_j} to the offspring;
7. Remove edges in p_{U_j} from $\bar{G}$;
8. **end if**
9. **end for**
10. **end if**
End
Output: an improved offspring

The function *PathConstruction*, which constructs a path between the endpoints of the unrealizable request (s_{U_j}, t_{U_j}), is similar to **Algorithm 8**. The only difference is that, instead of moving to the node with the highest priority, it selects the node with the smallest index as the succeeding node.

An example of employing this heuristic is shown in Figure 21, Figure 22 and Figure 23. An instance of a 3x4 mesh graph with given connection requests $\mathrm{T} = \{(2,12), (10,4), (9,3)\}$ and some pre-determined paths are illustrated in Figure 21. In Figure 22, two EDPs are obtained by applying *GMIN*. The unrealizable request $(9,3)$ can be reconnected by using the

subroutine *PathConstruction*. A new path (9, 5, 1, 2, 3) is found as shown in the dotted lines in Figure 23. The new path is then encoded and the improved offspring has three EDPs now.

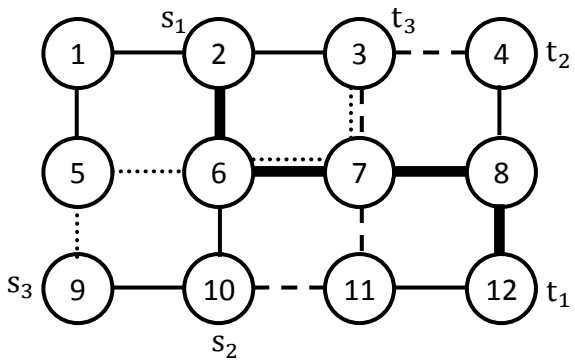


Figure 21 Three paths of corresponding requests

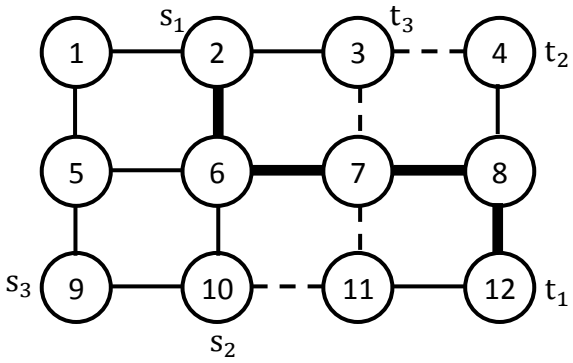


Figure 22 Two EDPs found by *GMIN*

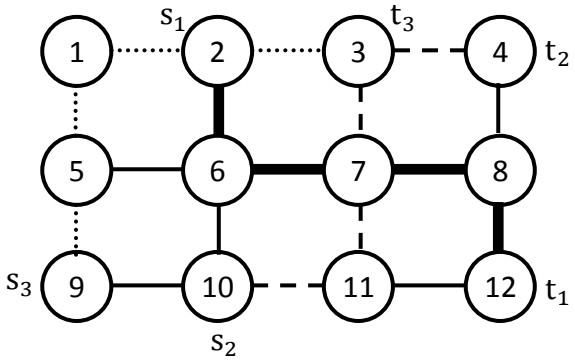


Figure 23 A new EDP found by the improvement heuristics

3.6 Fitness function and evaluation

Let S denote a set of edge disjoint paths extracted from a path set $S_P = \{p_i | \, i = 1, \ldots \mathrm{I}\}$ by *GMIN*. The path set S_P represents the paths decoded from an individual $\boldsymbol{u} = \{u_i \in [0,1]^n | \, i = 1, \ldots \mathrm{I}\}$. The fitness function is similar to the bicriteria scheme in Section 2.3.3, where the first objective $f(S_P) = |\mathrm{S}|$ is the number of EDPs and the second criterion $\mathrm{C}(S_P)$ measures the usage of edges that are traversed by more than one path.

$$\mathrm{C}(S_P) = \sum_{e \in E} (\max\{0, \left(\sum_{i=1}^{\mathrm{I}} \delta^i(S_P, \mathrm{e})\right) - 1\}),$$

$$\text{where } \delta^i(S_P, e) = \begin{cases} 1, & \text{if } e \in p_i \in S_P, \\ 0, & \text{otherwise.} \end{cases}$$

The second criterion measures the degree of "non-disjointness" of an individual. If all paths in S_P are edge-disjoint, $\mathrm{C}(S_P)$ is zero. In general, $\mathrm{C}(S_P)$ increases while paths in S_P have more edges in common. A comparison operator $f^{\mathrm{g}}(\cdot)$ is defined as follows. For the two path sets S_P^1 and S_P^2, which are decoded from two individuals $\boldsymbol{u}^1$ and $\boldsymbol{u}^2$, respectively, we say that $f^{\mathrm{g}}(\boldsymbol{u}^1) > f^{\mathrm{g}}(\boldsymbol{u}^2)$ if and only if

$$f(S_P^1) > f(S_P^2),$$
$$\text{or } (f(S_P^1) = f(S_P^2) \text{ and } \mathrm{C}(S_P^1) < \mathrm{C}(S_P^2)).$$

3.7 Population management and selection method

Two issues are worth mentioning regarding the population management. First, duplicate individuals are forbidden. Two individuals are considered identical if the decoded solutions are identical (i.e., the decoded I paths are the same, but not the priority values). Each time a new individual is generated, it is compared to the individuals in the population; if there already exists the same individual, the new individual is discarded and another individual is generated. The duplicate-checking is performed after the improvement heuristic (**Algorithm 12**) is executed.

The second issue iWith N individuals generated in each generation, let Mut and Crs represent the number of individuals generated by the mutation and crossover operators, respectively. To avoid being trapped in local solutions, we let Mut and Crs vary as follows:

$$Mut = round(minMut + ite \times (\frac{maxMut - minMut}{maxIte})),$$
$$Crs = N - Mut,$$

where $minMut$, $maxMut$ are parameters that indicate the minimum and maximum number of the individuals generated by the mutation operator in each generation. The quantity ite denotes the number of consecutive iterations that the algorithm has failed to improve the best-known solution. The quantity $maxIte$ is the maximum tolerable iteration, the algorithm terminates when $ite = maxIte$.

For the selection method, we apply the $(\mu + \lambda)$ selection method which picks the best

μ individuals from the parents and λ individuals from the offspring. Many researchers prefer to use this method to deal with combinatorial optimization problems. Note that the duplicate-checking is also adopted to prevent the selection of identical individuals.

3.8 Summary

Putting together all procedures designed in Sections 3.2 to Section 3.7, we now have a proposed GA based algorithm for solving the maximum edge-disjoint paths problem. Each connection request is assigned a path and each path is encoded into an n-element vector by the priority-based encoding scheme. An individual is composed of I such vectors, in which the ith vector represents the path connecting (s_i, t_i). A heuristic called *GMIN* is employed to solve a MIS problem to obtain the edge-disjoint paths among the I paths. For the reproduction procedure, three genetic operators were proposed to produce offspring by manipulating the priority values of one or two individuals. The self-adaption operator reroutes the path according to two factors: distance and edge usage rate. The main idea is that, a node which is closer to the terminal point and adjacent to more unused edges is more likely to have higher priority. Moreover, a heuristic method is proposed to further improve the quality of solution.

Chapter 4 Computational results

In this chapter, we intend to investigate the performance of the proposed genetic algorithm. In order to achieve this goal, we apply the proposed algorithm to various instances with different network structure and connection requests. The design of experiment is presented in Section 4.1. Features of the testing instances and the way to conduct experiments are outlined in Section 4.2. We also compare the proposed algorithm with the random search method, greedy algorithms and ant colony optimization in Section 4.3. Concluding remarks are made in Section 4.4.

4.1 Design of experiment

An instance of MEDP consists of a graph G and a set of connection requests T. To compare the proposed GA approach with the existing algorithms, we considered seven graphs representing different networks, in which two of them are parts of real telecommunication networks and others are randomly generated. The characteristics of these graphs will be given in the next section. For each graph, we generate different instances with $0.10|V|$, $0.25|V|$ and $0.40|V|$ requests, separately. Consequently, we have 3 instances for each graph and 21 instances in total. We applied each algorithm on every instance for 30 runs to obtain the best, worst, mean and standard deviation of the objective values. The average computational times are also recorded. With this information we can compute confidence intervals for the objective values obtained by the three algorithms for every instance. For a given instance, we say that the performance of two algorithms are significantly different if

their confidence intervals do not overlap.

GA is a stochastic optimizer since it involves some random factors during the search process. Hence the first thing we want to know is that whether our GA procedure performs better than the pure random search. The result is presented in Section 4.3.1. Secondly, we would like compare the performance of GA with state-of-the-art optimization algorithms for MEDP. Here we choose the multi-start simple greedy algorithm and ACO as described in Sections 2.3.2 and 2.3.3, respectively.

4.2 Problem generation and computational experiments

A set of benchmark instances for MEDP was given in [23]. Seven graphs are considered in our experiment. The first two graphs, **graph3** and **graph4,** were created by researchers of the Computational Optimization & Graph Algorithm group at the Technische Universität Berlin. The structures of these two graphs 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**, are generated with the network generator BRITE. In addition, 2 mesh graphs composed of **mesh10x10**, **mesh15x15** are also included. The main features and quantitative measures are shown in Table 2. We refer to [23] for the parameters used for the generation of the network topologies using BRITE.

Table 2 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
mesh10x10	100	180	2	3.60	4	18
mesh15x15	225	420	2	3.73	4	28

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

All the algorithms in our experiment were implemented in MATLAB. The experiments have been conducted on a PC with Intel® Core i7 CPU @1.6GHz and 4 Gb of memory running the Windows 7 operating system. All the algorithms were implemented on the same data structures. Information about the shortest paths in the respective graphs is provided to all of them as input. Notice that the greedy algorithms need to partially recompute this information iteratively in the solution construction process, but this work is not necessary for GA and ACO approaches.

4.3 Experimental results

In this section, we report and analyze the computational results. In the first section, the performance of the pure random search vs. GA is given in Section 4.3.1. Section 4.3.2 provides the computational results of SGA, MSGA and the proposed GA. We can observe the clear advantages of the proposed GA over the other two greedy algorithms. Section 4.3.3 shows the experimental results of ACO. The confidence intervals of the solution quality

obtained by MSGA, ACO and the proposed GA are plotted for comparisons.

4.3.1 Random search vs. GA

The comparison between the proposed GA method and a purely random search method is given in this section. Note that the initial population of GA is randomly generated in order to make a fair comparison. The steps of the random search method are described as follows:

Step 0. Set the current best objective value to 0.
Step 1. Randomly generate 10 solutions.
Step 2. Evaluate the solutions.
Step 3. Update the current best solution if there is any improvement.
Step 4. Stop if the termination conditions are met. Otherwise repeat Steps 1 to 3.

As described in Chapter 3, the proposed GA method generates 10 offspring in each iteration and terminates when no improvement can be found for several iterations. We observed that in general, GA terminates in less than 200 iterations. In other words, less than 2000 solutions were investigated in each run of GA. Therefore, we let the random search algorithm halt after generating 2000 solutions (or 200 iterations). Both algorithms were executed for 30 times and the mean values and confidence intervals of their solution quality are stored. The choice of the executing times is because that the number 30 is the boundary between small and large samples.

Figure 24 and Figure 25 show the evolution of the current best solution generated by the proposed GA and the random search method on the instances of AS-BA.R-Wax.v100e190 with 40 connection requests and mesh10X10 with 40 requests. The solid line and dash line are the mean values of the current best solution obtained by GA and the random search

during the search process, respectively. The dotted lines above and under the mean values show the upper and lower bounds of the 95% confidence intervals. The confidence intervals are calculated $\bar{x} \pm 1.96\left(\frac{\sigma}{\sqrt{30}}\right)$, where $\bar{x}$, σ are the mean and standard deviation of objective values, respectively.

Some observations can be made from these two figures. At the beginning, two confidence intervals overlap (since both methods have their initial solutions randomly generated). The best solution obtained by GA is enhanced rapidly in the next few iterations, and the progress slows down after 10 iterations but the improvement is still ongoing. From the two figures, we can see that the proposed GA method has a clear advantage over the random search algorithm.

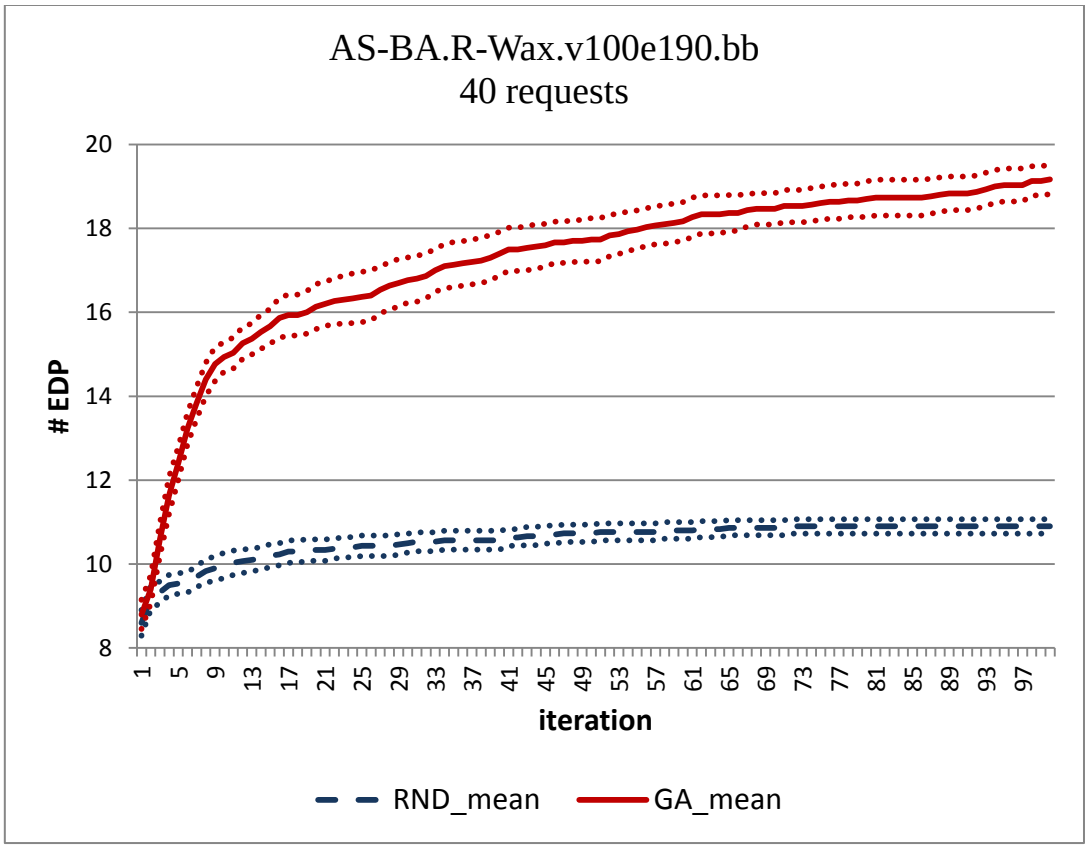


Figure 24 Evolution of the solution quality obtained by GA and random search on AS-BA.R-Wax.v100e190 with 40 connection requests (upper and lower dot lines denote the boundaries of 95% confidence intervals)

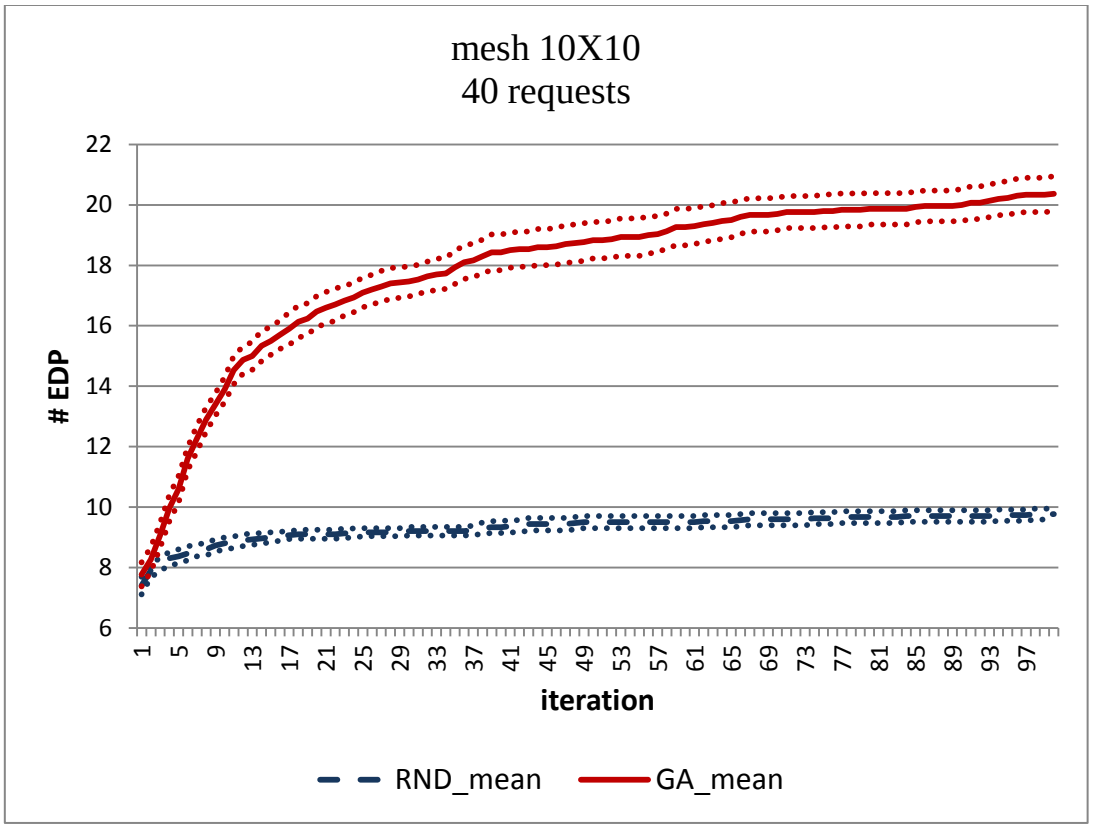


Figure 25 Evolution of the solution quality obtained by GA and random search on mesh10X10 with 40 connection requests (upper and lower dot lines denote the boundaries of 95% confidence intervals)

4.3.2 Greedy algorithms vs. GA

The comparison of the greedy algorithms and the proposed Genetic Algorithm is shown in Table 3. The first column gives the name of the graph tested and the second column shows the number of connection requests, which are the 10, 25, and 40% of the number of nodes of the graphs. For the simple greedy algorithm (SGA), the first column shows the number of the EDPs obtained from the instance and the second column provides the computational time. For the multi-start simple greedy algorithm (MSGA) and the genetic algorithm with 3 initial

solutions, 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 for each instance. The average value is underlined and in boldface when the result is the best among the three. The last column shows the values of $(\bar{q}_G - \bar{q}_M)/\bar{q}_M$, where $\bar{q}_G$ and $\bar{q}_M$ stand for the average value obtained by the proposed GA and MSGA, respectively.

We observe that MSGA has a clear advantage over SGA. This shows that the order of the connection requests is crucial in achieving good performance. However, since there is no obvious way to predetermine a good order, we apply MSGA which permutes the order of the connection requests in random and run SGA with the new request list. The price we have to pay for running MSGA is the significantly increased computational time.

Comparing the performance of MSGA vs. the proposed GA, we observe that in generally, GA obtains better solution quality in less computational time. More specifically, in all 21 instances, GA obtains average values either equal to or better than MSGA. Moreover, in 16 out of 21 instances, GA spent less computational time than MSGA did. In the last column of Table 3, we can observe that for the same graph, the advantage of the proposed GA is more distinct when the number of connection requests grows. This phenomenon occurs in AS-BA.R-Wax.v100e190, graph3, graph4, mesh10X10 and mesh15X15.

Table 3 Comparison of the results obtained by SGA, MSGA and proposed GA with 3 initial populations

Graph	Number of requests	SGA		MSGA					GA with 3 initial					$\frac{(\bar{q}_G - \bar{q}_M)}{\bar{q}_M}$
		q	***t***	***max***	***min***	$\bar{q}_M$	***std***	***t***	***max***	***min***	$\bar{q}_G$	***std***	***t***	
AS-BA.R-Wax.v100e190	10	**8.0**	0.8	8	8	**8.0**	0.0	19.5	8	8	**8.0**	0.0	10.0	0.0%
AS-BA.R-Wax.v100e190	25	12.0	2	14	13	13.4	0.5	48.6	14	13	**13.9**	0.3	43.0	3.7%
AS-BA.R-Wax.v100e190	40	16.0	3	20	18	19.1	0.6	76.9	21	20	**20.3**	0.5	98.1	6.1%
AS-BA.R-Wax.v100e217	10	6.0	0.7	7	6	6.8	0.4	19.5	7	7	**7.0**	0.0	15.1	2.3%
AS-BA.R-Wax.v100e217	25	9.0	1.8	13	10	11.5	0.6	48.1	13	12	**12.6**	0.5	54.3	10.0%
AS-BA.R-Wax.v100e217	40	19.0	2.8	22	19	19.9	0.8	77.0	22	21	**21.7**	0.5	93.0	9.1%
bl-wr2-wht2.10-50.rand1	50	21.0	57.2	25	22	23.7	0.7	1081.0	26	24	**25.0**	0.8	788.3	5.7%
bl-wr2-wht2.10-50.rand1	125	34.0	121.2	40	36	38.1	0.9	2746.1	43	39	**41.5**	0.8	1661.4	9.1%
bl-wr2-wht2.10-50.rand1	200	55.0	194.6	57	55	55.1	0.4	4182.0	61	58	**60.0**	0.8	3592.2	8.9%
graph3	16	**15.0**	2.3	15	15	**15.0**	0.0	81.7	15	15	**15.0**	0.0	32.4	0.0%
graph3	41	32.0	5.7	33	32	32.1	0.3	173.8	33	32	**32.2**	0.4	92.6	0.3%
graph3	65	29.0	9.2	34	29	32.3	1.1	270.4	39	35	**36.4**	1.0	152.8	12.9%
graph4	43	**42.0**	44.8	42	42	**42.0**	0.0	1210.4	42	42	**42.0**	0.0	291.0	0.0%
graph4	108	60.0	104.2	68	62	64.6	1.2	3420.5	71	68	**69.7**	0.9	1093.1	7.9%
graph4	173	73.0	175	75	73	73.1	0.4	5146.1	85	83	**84.3**	0.7	1664.6	15.2%
mesh10X10	10	**10.0**	0.6	10	10	**10.0**	0.0	18.2	10	10	**10.0**	0.0	11.0	0.0%

Table 3 Continued

Graph	Number of requests	SGA		MSGA					GA with 3 initial					$\frac{(\bar{q}_G - \bar{q}_M)}{\bar{q}_M}$
		q	***t***	***max***	***min***	$\bar{q}_M$	***std***	***t***	***max***	***min***	$\bar{q}_G$	***std***	***t***	
mesh10X10	25	14.0	1.4	17	15	16.3	0.5	50.2	19	17	**17.5**	0.6	45.0	7.4%
mesh10X10	40	17.0	2.2	22	18	19.7	0.8	88.3	24	21	**22.6**	0.8	94.1	14.5%
mesh15X15	23	19.0	6.2	22	19	**20.4**	0.7	194.1	21	20	**20.4**	0.5	90.2	0.0%
mesh15X15	57	23.0	14.9	28	26	27.1	0.6	510.2	32	29	**30.7**	0.7	392.0	13.2%
mesh15X15	90	32.0	22.9	35	32	32.6	0.8	725.4	41	39	**39.4**	0.6	768.0	20.9%

4.3.3 MSGA/ACO vs. GA

The results obtained by MSGA, Ant Colony Optimization (ACO) and the proposed GA are shown in Table 4. The five columns under each method show the maximum, minimum, average value, standard deviation and the average computational time of 30 runs, respectively. The average values are underlined and in boldface when the result is the best among the three.

Some observations can be made from the results shown in Table 4. First, the solution quality of the proposed GA obtained in the experiment is comparable with, or in most cases surpasses, that of the other two algorithms. More in detail, GA achieves the best solution in 18 out of 21 instances, in which GA beats MSGA and ACO in 15 instances. In particular, all instances on graph4 and mesh15X15 strongly favor the proposed GA over ACO in both computational time and solution quality. 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 mesh15X15 with 23, 57 and 90 pairs, GA performs 14%, 6.3% and 9.0% better than ACO, respectively. On the other hand, although ACO performs better than GA in the three instances on graph bl-wr2-wht2.10-50.rand1, the performance differences are small (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).

Table 4 Comparison of the results obtained by MSGA, ACO and the proposed GA with 3 initial populations

Graph	Number of requests	MSGA					ACO					GA with 3 initial				
		max	***min***	$\bar{q}_M$	***std***	***t***	***max***	***min***	$\bar{q}_A$	***std***	***t***	***max***	***min***	$\bar{q}_G$	***std***	***t***
AS-BA.R-Wax.v100e190	10	8	8	**8.0**	0.0	19.5	8	8	**8.0**	0.0	19.1	8	8	**8.0**	0.0	10.0
AS-BA.R-Wax.v100e190	25	14	13	13.4	0.5	48.6	14	12	13.5	0.6	50.6	14	13	**13.9**	0.3	43.0
AS-BA.R-Wax.v100e190	40	20	18	19.1	0.6	76.9	21	19	20.0	0.4	69.6	21	20	**20.3**	0.5	98.1
AS-BA.R-Wax.v100e217	10	7	6	6.8	0.4	19.5	7	6	6.7	0.4	30.2	7	7	**7.0**	0.0	15.1
AS-BA.R-Wax.v100e217	25	13	10	11.5	0.6	48.1	13	11	11.2	0.5	49.6	13	12	**12.6**	0.5	54.3
AS-BA.R-Wax.v100e217	40	22	19	19.9	0.8	77.0	22	20	21.2	0.5	73.9	22	21	**21.7**	0.5	93.0
bl-wr2-wht2.10-50.rand1	50	25	22	23.7	0.7	1081.0	26	25	**25.8**	0.4	938.0	26	24	25.0	0.8	788.3
bl-wr2-wht2.10-50.rand1	125	40	36	38.1	0.9	2746.1	43	42	**42.5**	0.5	1802.3	43	39	41.5	0.8	1661.4
bl-wr2-wht2.10-50.rand1	200	57	55	55.1	0.4	4182.0	61	59	**60.4**	0.7	2753.2	61	58	60.0	0.8	3592.2
graph3	16	15	15	**15.0**	0.0	81.7	15	15	**15.0**	0.0	23.8	15	15	**15.0**	0.0	32.4
graph3	41	33	32	32.1	0.3	173.8	33	28	30.1	1.0	118.7	33	32	**32.2**	0.4	92.6
graph3	65	34	29	32.3	1.1	270.4	38	33	35.2	1.1	253.4	39	35	**36.4**	1.0	152.8
graph4	43	42	42	42.0	0.0	1210.4	40	36	37.8	1.2	678.4	42	42	**42.0**	0.0	291.0
graph4	108	68	62	64.6	1.2	3420.5	63	58	61.3	1.7	2464.0	71	68	**69.7**	0.9	1093.1
graph4	173	75	73	73.1	0.4	5146.1	82	76	78.8	1.8	4494.2	85	83	**84.3**	0.7	1664.6
mesh10X10	10	10	10	**10.0**	0.0	18.2	10	9	9.9	0.3	20.3	10	10	**10.0**	0.0	11.0

Table 4 Continued

Graph	Number of requests	MSGA					ACO					GA with 3 initial				
		max	***min***	$\bar{q}_M$	***std***	***t***	***max***	***min***	$\bar{q}_A$	***std***	***t***	***max***	***min***	$\bar{q}_G$	***std***	***t***
mesh10X10	25	17	15	16.3	0.5	50.2	19	15	16.5	1.0	58.3	19	17	**17.5**	0.6	45.0
mesh10X10	40	22	18	19.7	0.8	88.3	24	20	21.8	1.0	92.9	24	21	**22.6**	0.8	94.1
mesh15X15	23	22	19	**20.4**	0.7	194.1	20	16	17.9	0.9	239.5	21	20	**20.4**	0.5	90.2
mesh15X15	57	28	26	27.1	0.6	510.2	31	27	28.9	1.1	638.7	32	29	**30.7**	0.7	392.0
mesh15X15	90	35	32	32.6	0.8	725.4	38	34	36.2	1.2	966.1	41	39	**39.4**	0.6	768.0

In addition to comparing the maximum, minimum and average values, we plot the confidence intervals in Figure 26—46 to show a clearer picture. Each figure has three segments indicating the 95% confident intervals of the performance of the three algorithms on the same instance. The middle of each segment denotes the average value. 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). For the two instances AS-BA.R-Wax.v100e190.bb with 40 requests and AS-BA.R-Wax.v100e217.bb with 40 requests, we further performed a paired t-test to determine if the results obtained by the proposed GA and ACO are significantly different. For the first instance, the mean difference is -0.17, SD=0.14, N=30, t(29)=1.15, two-tail p=0.26. A 95% C.I. of the mean difference is (-0.46, 0.13). For the second instance, the mean difference is -0.13, SD =0.11, N= 30, t(29)=-1.16, two-tail p=0.25. A 95% C.I. of the mean difference is (-0.37, 0.1). Therefore there are no significant differences between the performances obtained by GA and ACO on both instances. On the other hand, ACO outperforms GA significantly on two instances: bl-wr2-wht2.10-50.rand1 with 50 and 125 pairs. At last, the proposed GA also has significant advantage over the MSGA in 14 out of 21 instances.

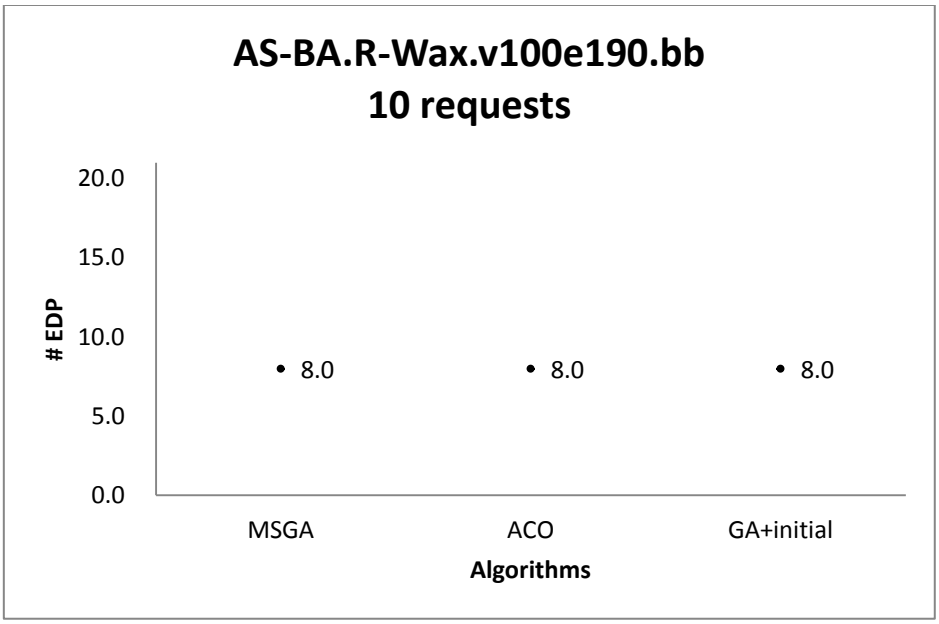


Figure 26 Confidence intervals of the solution quality obtained by three algorithms on AS-BA.R-Wax.v100e190.bb with 10 requests

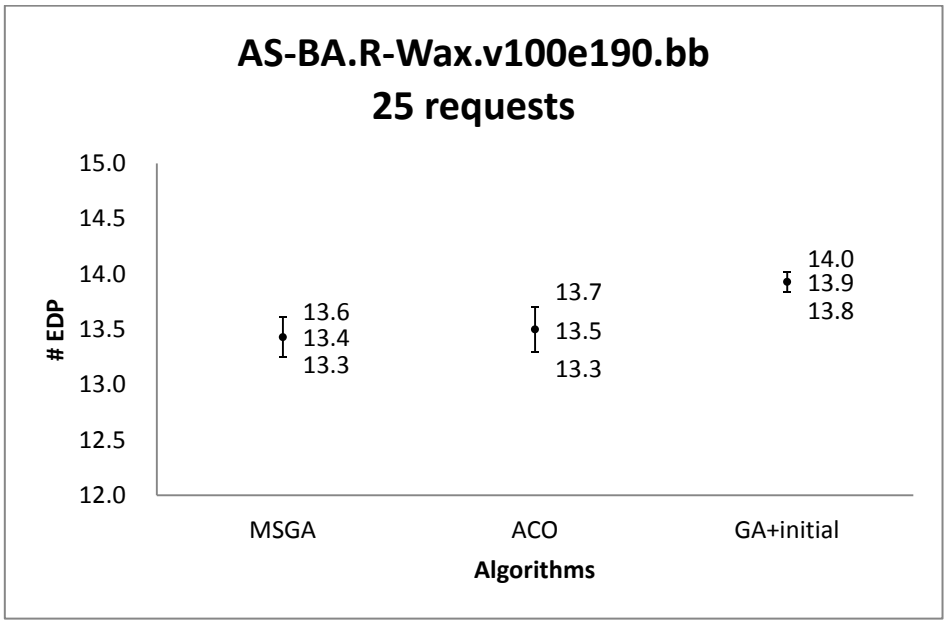


Figure 27 Confidence intervals of the solution quality obtained by three algorithms on AS-BA.R-Wax.v100e190.bb with 25 requests

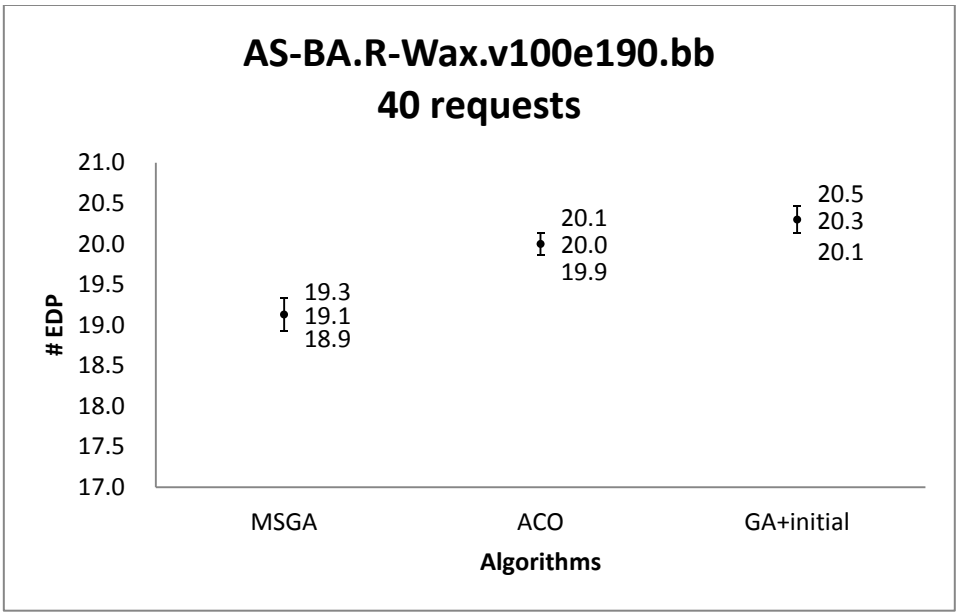


Figure 28 Confidence intervals of the solution quality obtained by three algorithms on AS-BA.R-Wax.v100e190.bb with 40 requests

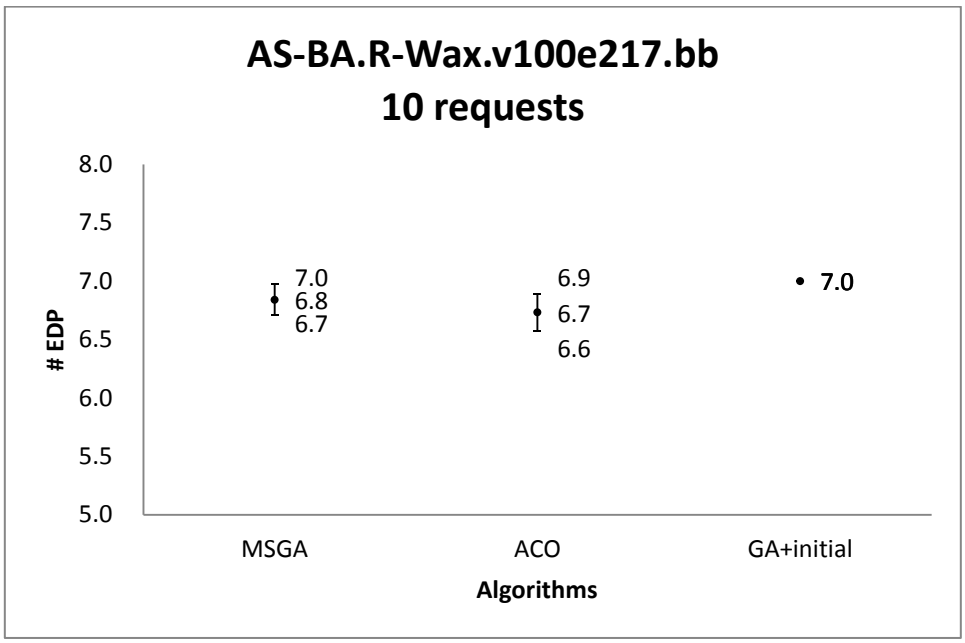


Figure 29 Confidence intervals of the solution quality obtained by three algorithms on AS-BA.R-Wax.v100e217.bb with 10 requests

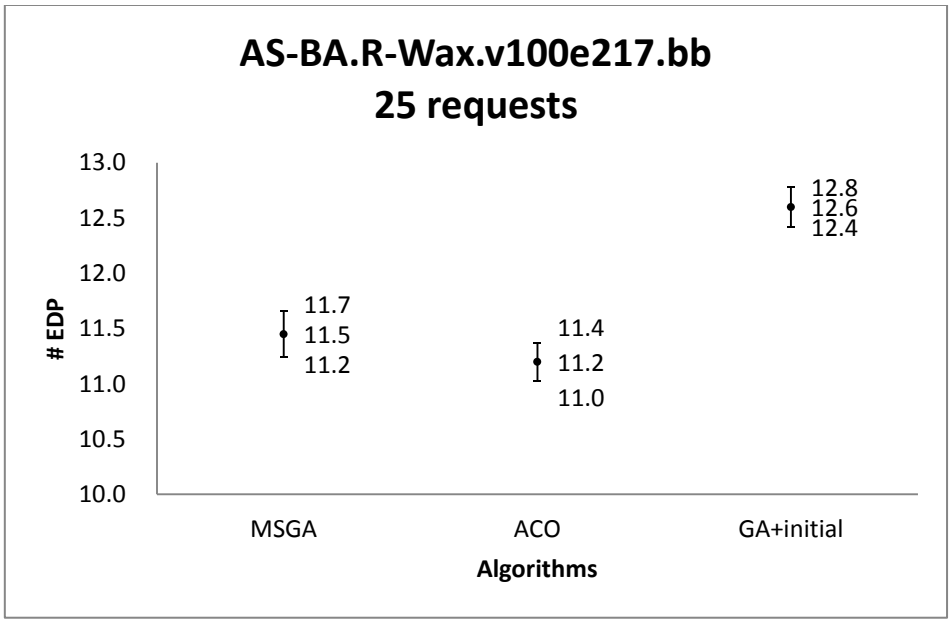


Figure 30 Confidence intervals of the solution quality obtained by three algorithms on AS-BA.R-Wax.v100e217.bb with 25 requests

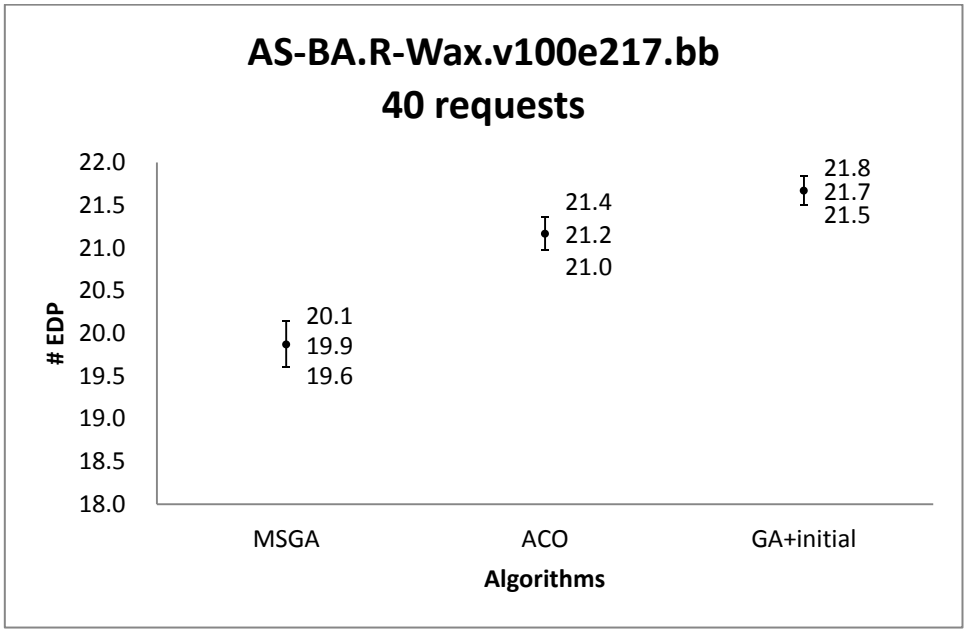


Figure 31 Confidence intervals of the solution quality obtained by three algorithms on AS-BA.R-Wax.v100e217.bb with 40 requests

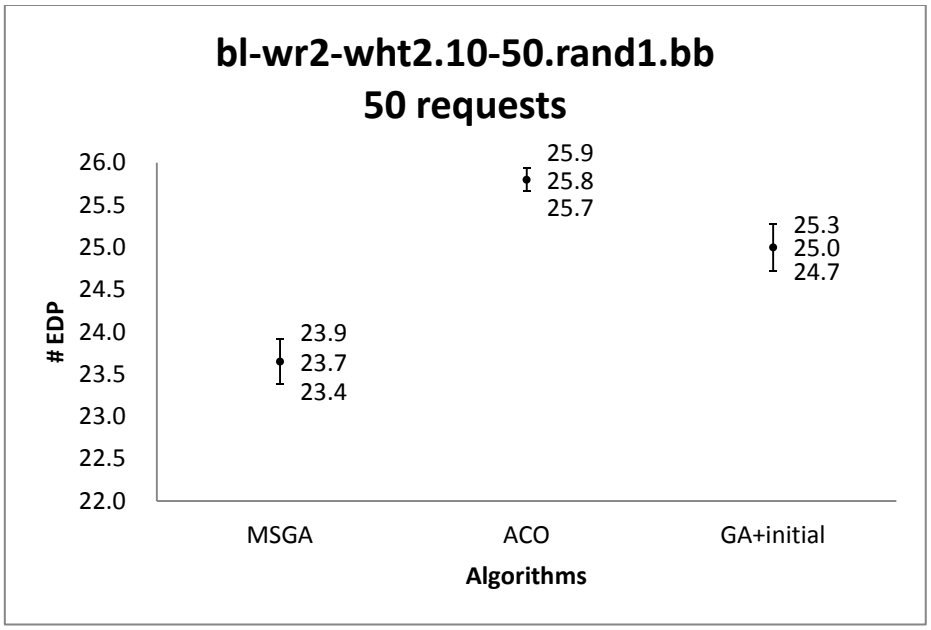


Figure 32 Confidence intervals of the solution quality obtained by three algorithms on bl-wr2-wht2.10-50.rand1.bb with 50 requests

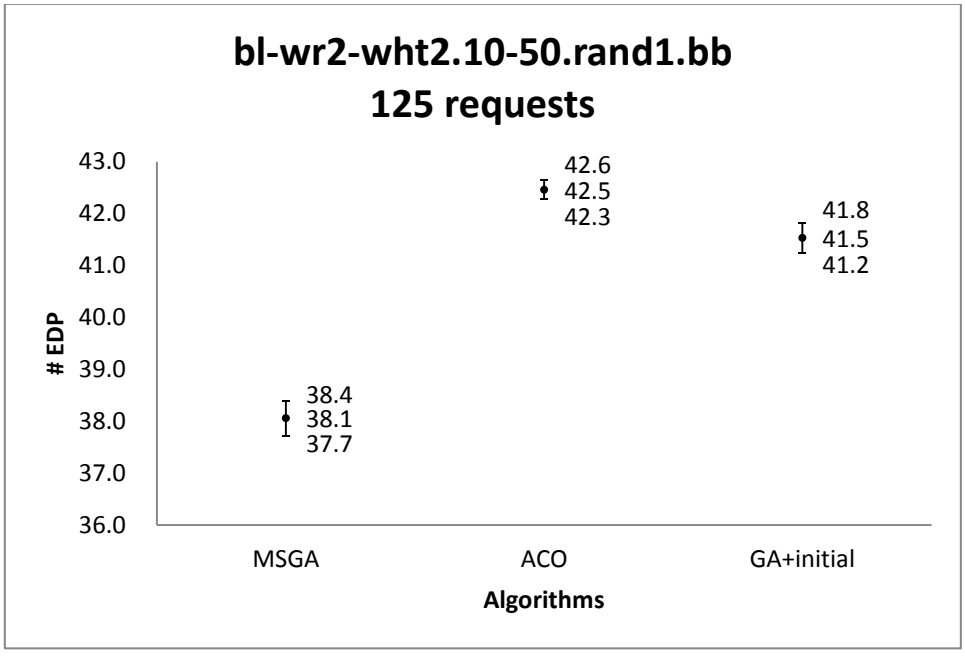


Figure 33 Confidence intervals of the solution quality obtained by three algorithms on bl-wr2-wht2.10-50.rand1.bb with 125 requests

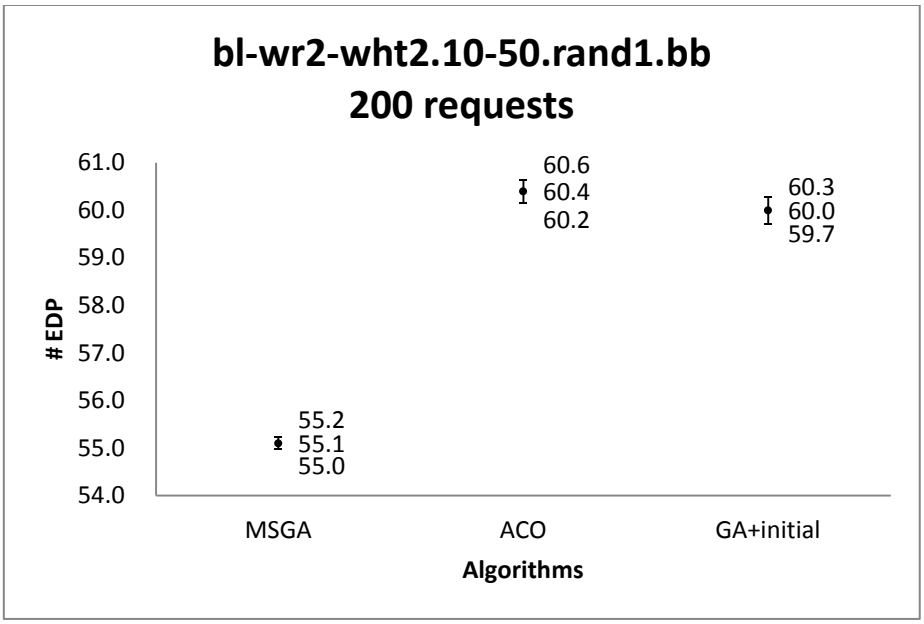


Figure 34 Confidence intervals of the solution quality obtained by three algorithms on bl-wr2-wht2.10-50.rand1.bb with 200 requests

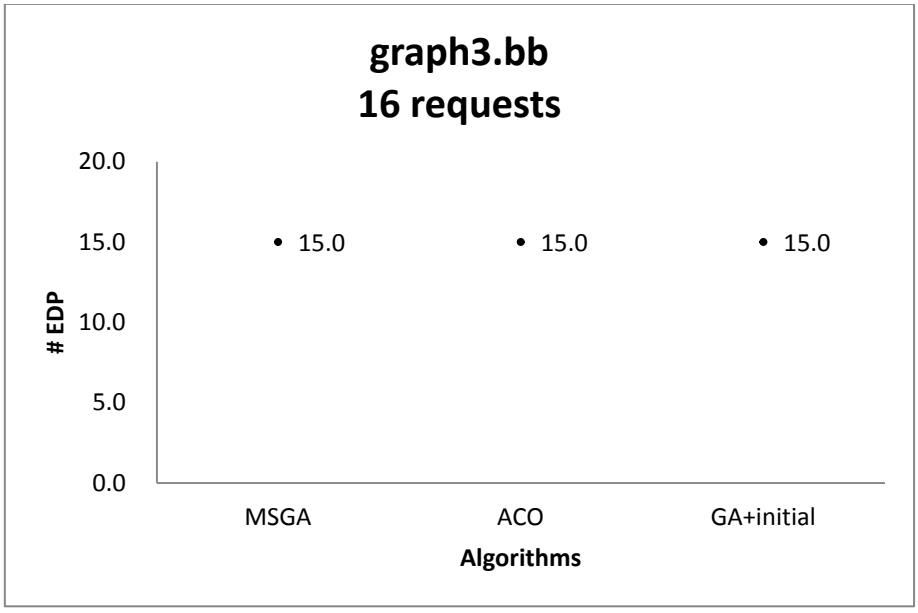


Figure 35 Confidence intervals of the solution quality obtained by three algorithms on graph3.bb with 16 requests

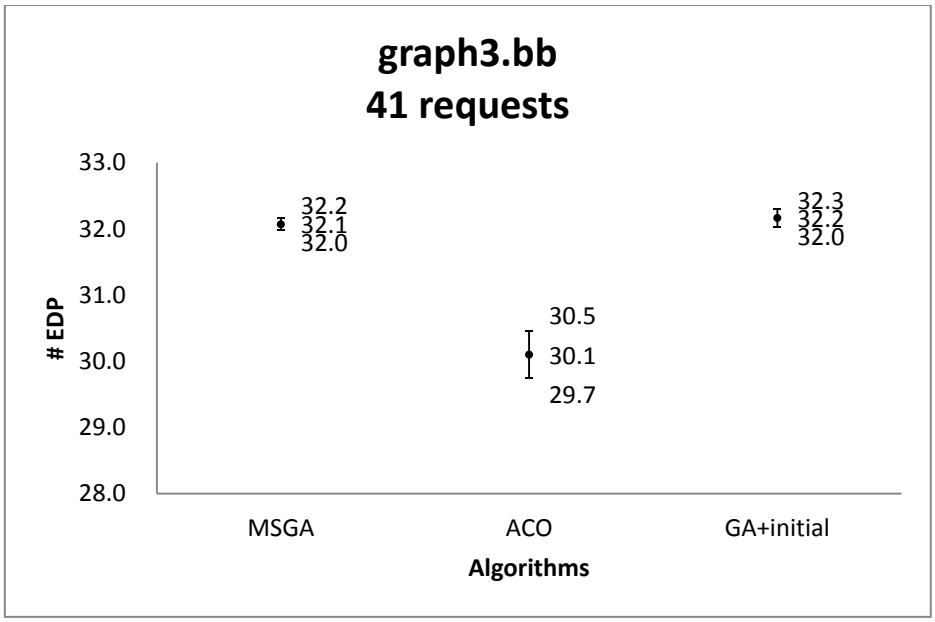


Figure 36 Confidence intervals of the solution quality obtained by three algorithms on graph3.bb with 41 requests

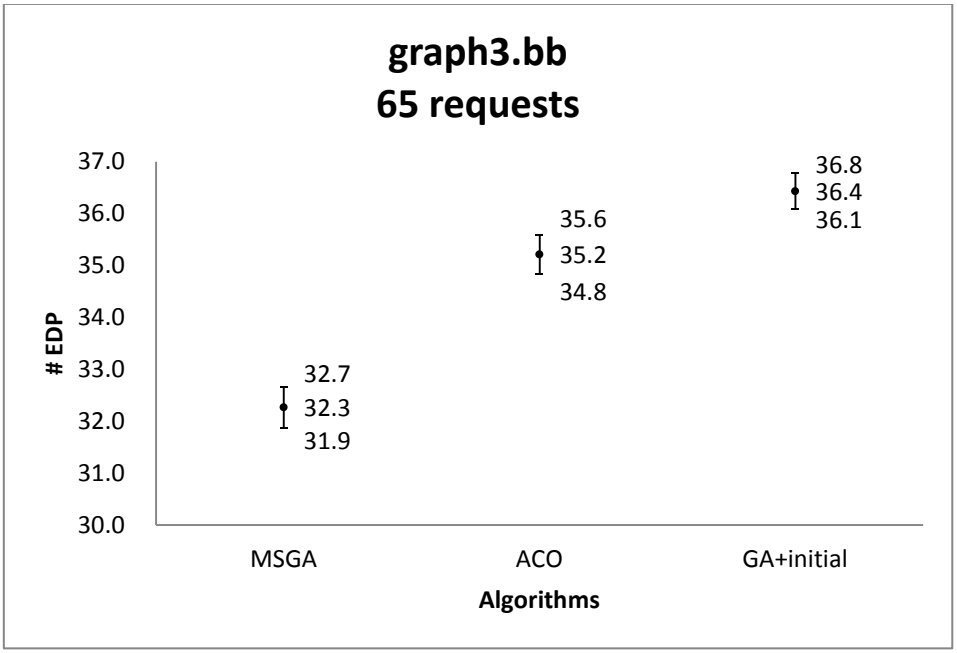


Figure 37 Confidence intervals of the solution quality obtained by three algorithms on graph3.bb with 65 requests

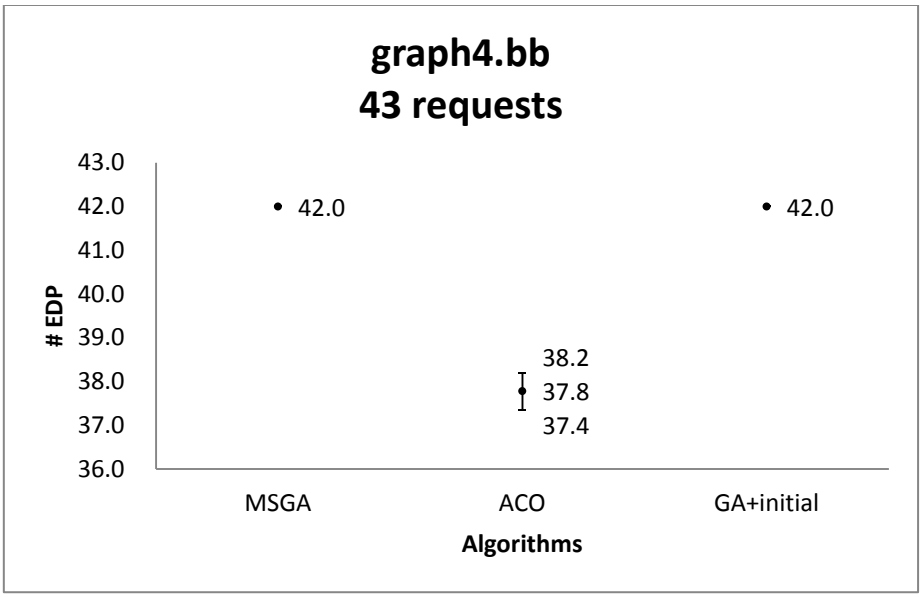


Figure 38 Confidence intervals of the solution quality obtained by three algorithms on graph4.bb with 43 requests

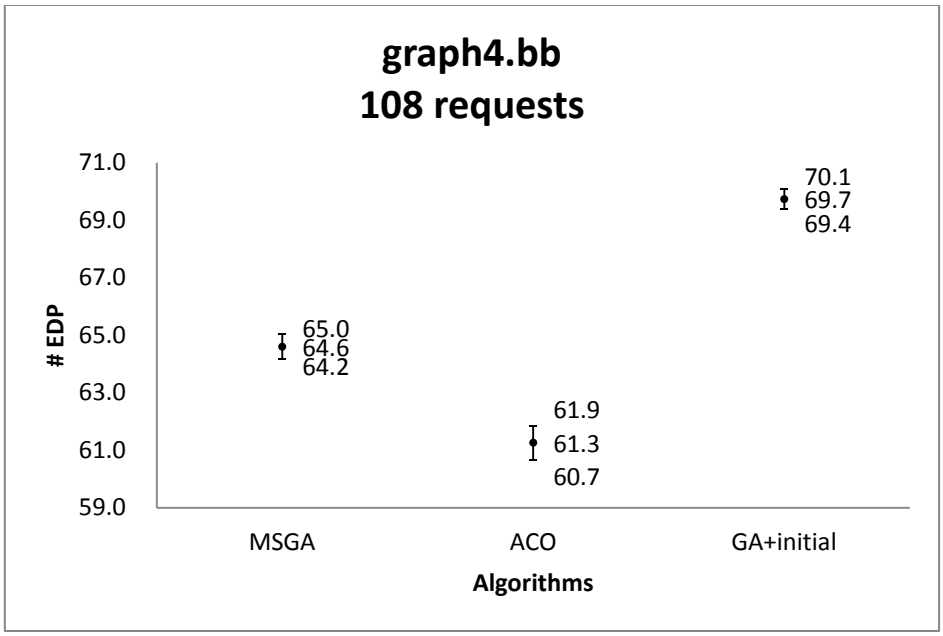


Figure 39 Confidence intervals of the solution quality obtained by three algorithms on graph4.bb with 108 requests

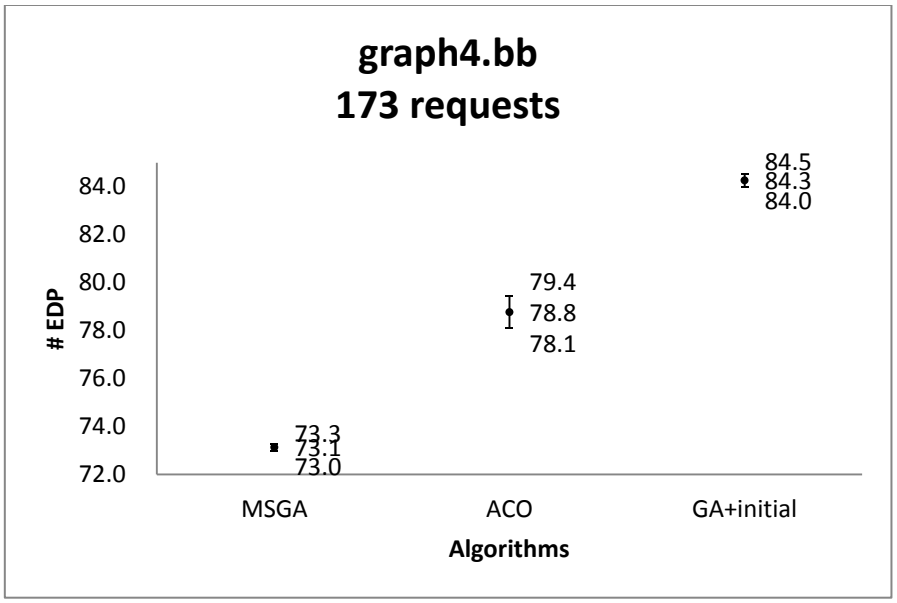


Figure 40 Confidence intervals of the solution quality obtained by three algorithms on graph4.bb with 173 requests

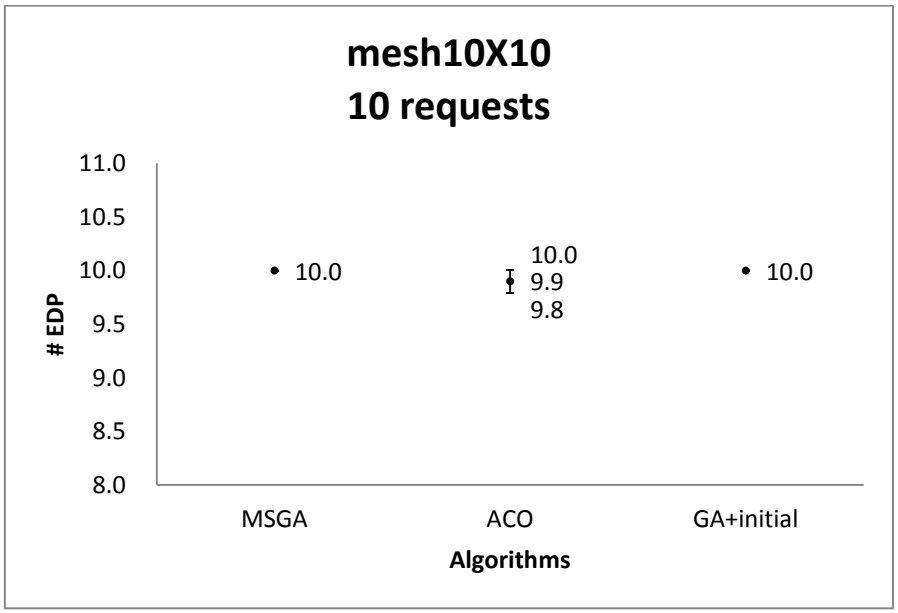


Figure 41 Confidence intervals of the solution quality obtained by three algorithms on mesh10X10 with 10 requests

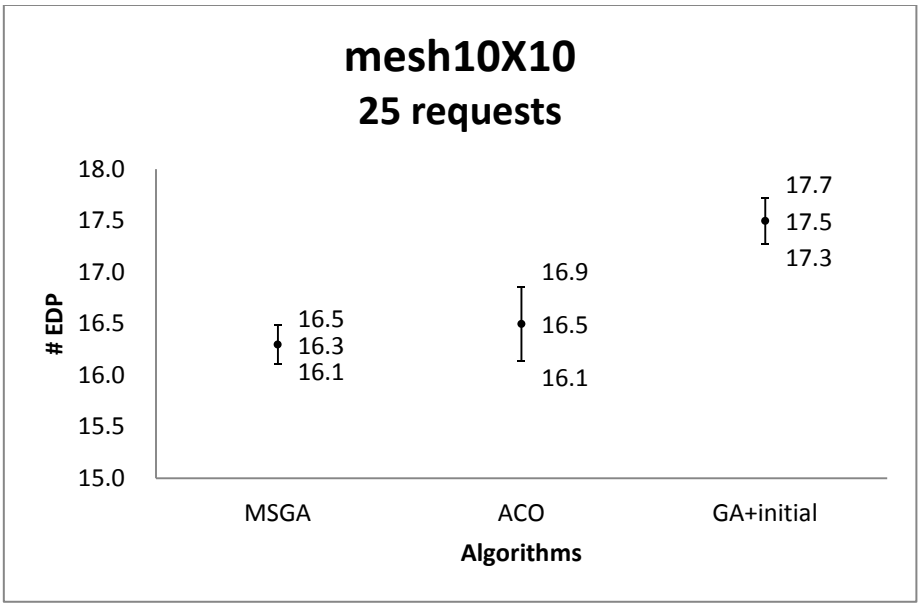


Figure 42 Confidence intervals of the solution quality obtained by three algorithms on mesh10X10 with 25 requests

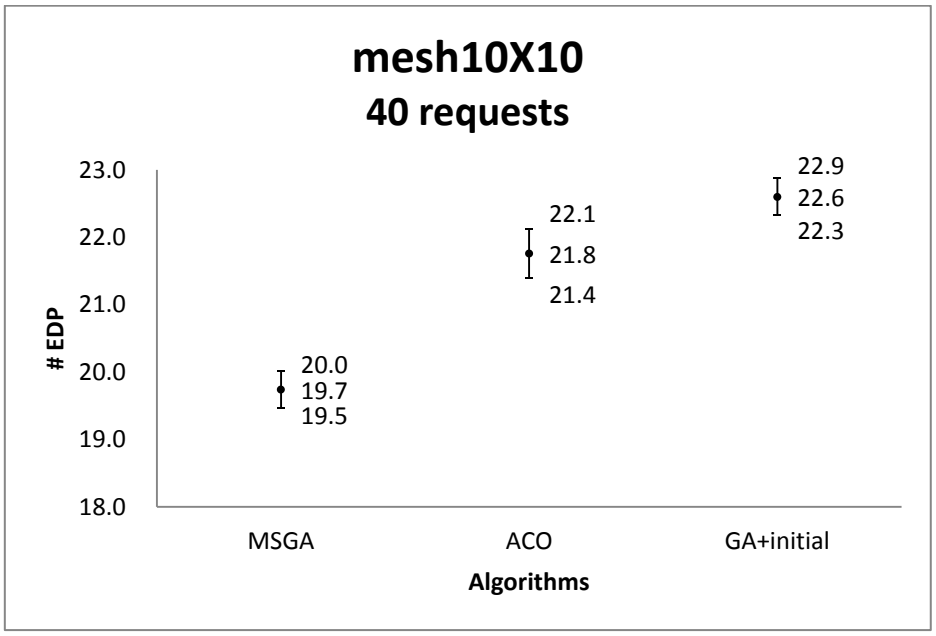


Figure 43 Confidence intervals of the solution quality obtained by three algorithms on mesh10X10 with 40 requests

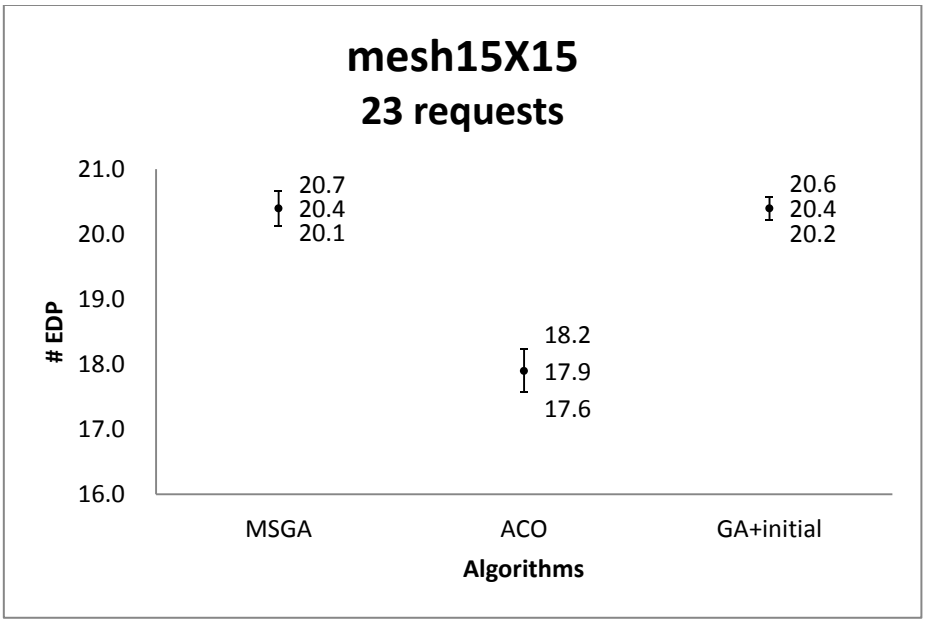


Figure 44 Confidence intervals of the solution quality obtained by three algorithms on mesh15X15 with 23 requests

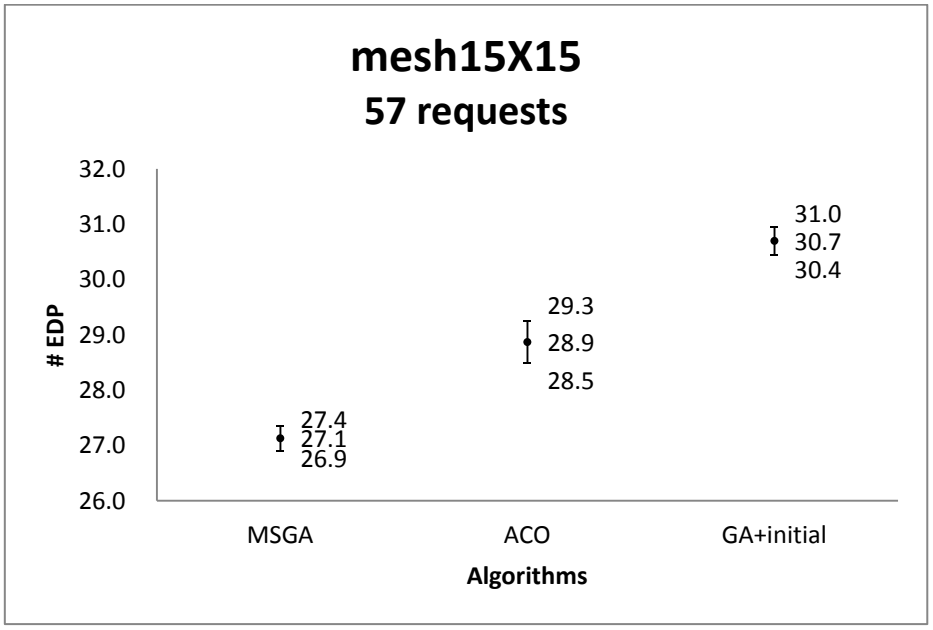


Figure 45 Confidence intervals of the solution quality obtained by three algorithms on mesh15X15 with 57 requests

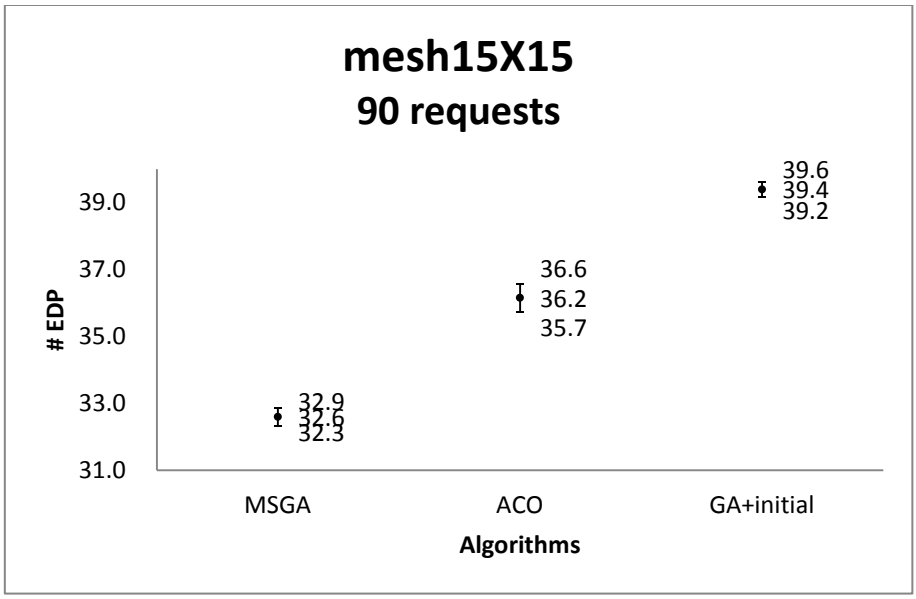


Figure 46 Confidence intervals of the solution quality obtained by three algorithms on mesh15X15 with 90 requests

4.4 Summary

We have compared the proposed genetic algorithm with a purely random search method to confirm the effectiveness of *GA_MEDP*. In addition, compared with the simple greedy algorithm, multi-start greedy algorithm and ant colony optimization method, the proposed GA method performs better or much better in most of the cases in terms of the solution quality and computation time.

Chapter 5 Solving the RWA problem

In this chapter, we develop a heuristic method based on the GA approach proposed in Chapter 3 to solve the routing and wavelength assignment problem. To evaluate its performance, we compare the proposed method with the state-of-art bin-packing based methods [26] and the particle swarm optimization approach [4].

In Section 5.1, we describe the details of the proposed heuristic method for solving the RWA problem. An illustration on a small instance is given in Section 5.2. Features of the testing instances and parameter-tuning are outlined in Section 5.3. The comparison of the performance of the proposed algorithm with the bin-packing based methods and particle swarm optimization are given in Section 5.4. Concluding remarks are made in Section 5.5.

5.1 Proposed method

Finding edge-disjoint paths can be useful for solving the RWA problem since a set of EDPs can be assigned to the same wavelength. In addition, more requests assigned to one wavelength may lead to fewer wavelengths required to satisfy all requests. Therefore an MEDP solution algorithm can be useful for solving the RWA problem. In Chapter 3, we have developed a GA-based method to solve the MEDP problem, the subroutine is called GA_MEDP, which has two inputs (G,T) and two outputs (R,S), where R denotes the set of realizable (or accepted) requests and S is the edge-disjoint paths set.

A set of edge-disjoint paths can be assigned to the same wavelength since no two paths share any edge. Thus an intuitive idea of solving the RWA problem can be developed as

follows. Solve the MEDP problem with (G, T) and assign one wavelength to the accepted requests and remove these requests from T. Then solve another MEDP in G with the rest of T and assign another wavelength to these accepted requests. The procedure is repeated until T is empty. The pseudocode of this basic algorithm is given in **Algorithm 13**.

Algorithm 13 basic MEDP-RWA algorithm

Input: $G = (V, E)$ and $T = \{(s_i, t_i) | i = 1, \dots, \mathrm{I}\}$
Begin:
1. $w = \emptyset, \pi = \emptyset$;
2. $l = 1$;
3. **While** $T \neq \emptyset$
4. $(R, S) = GA\_MEDP(G, T)$
5. $w_i = l, \forall i \in R$;
6. $\pi_i = p_i, \forall i \in R$;
7. $T = T \backslash R$;
8. $l = l + 1$;
9. **end while**

End
Output: π and w

This basic algorithm was used for solving the RWA problem in [27]. Since it takes all requests in T into consideration at each step, the efficiency is low, especially when $|T|$ is large. In our proposed method, instead of dealing with the whole request set, we divide T into several batches and solve the MEDP problem with only one batch at a time. The batch size B is a user-defined value. The tuning process of B is given in Section 5.3.2.

Similar to the preprocessing on the order of T in bin-packing algorithm (FFD and BFD), the shortest path of each request in G is precomputed, then T is sorted in a non-decreasing order of the shortest path distances. Although these paths are unlikely to be the final routes,

they still provide good information about the minimum units of resources (edges) they occupy in G. Thus a better solution could be secured if we first consider the request with longer shortest path distance and then fill up the remaining space with the request with shorter shortest path length.

After the adjustment of T is made, the first B requests in T are selected. The current wavelength is denoted by the variable wl initialized to be 1. Then *GA_MEDP* is executed to find the maximum number of edge disjoint paths among the selected requests in G. The current wavelength wl is assigned to the accepted requests and each of the obtained edge-disjoint paths is assigned to the corresponding lightpaths. The residual graph, where all edges used by the paths are removed, is stored in the variable G_{wl}. The rejected requests at this stage remain in T and will be included in the next batch.

Before starting *GA_MEDP* with the next batch, the algorithm scans all the remaining requests in T in a backward manner. Starting from the last request, which has the shortest distance of shortest path in G, the algorithm tries to find a shortest path to route the request in G_{wl}. If such path exists, the request is assigned wavelength wl and removed from T. After the backward-scanning process is done, wl is increased by 1. Another batch of requests is selected and *GA_MEDP* is executed again. The algorithm halts when T becomes empty. We call this proposed method the *GA_MEDP_RWA* algorithm, whose pseudocode is shown in **Algorithm 14**.

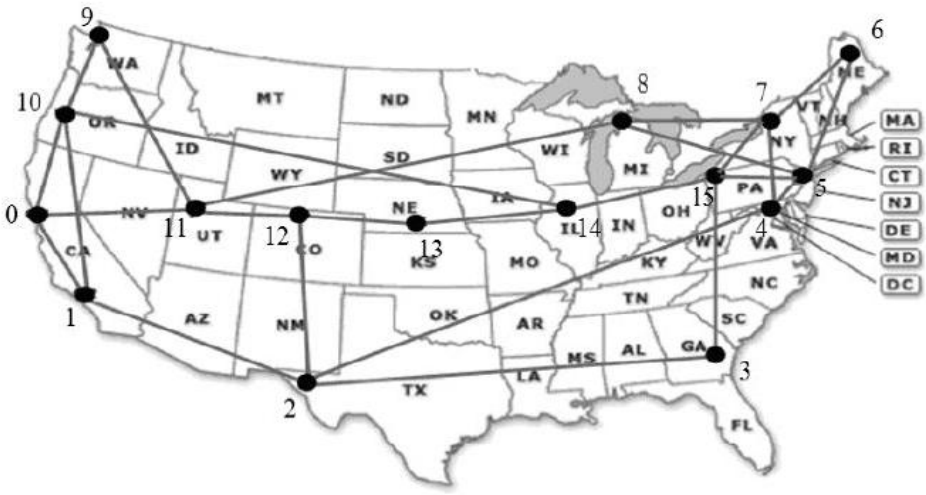
Algorithm 14 *GA_MEDP_RWA* algorithm

Input: $G = (V, E)$, $T = \{(s_i, t_i) | i = 1, \ldots, \mathrm{I}\}$ and batch size B
Begin:
1. $w = \emptyset, \pi = \emptyset, \sigma = \{\sigma_i = i | i = 1, \ldots, \mathrm{I}\}$;
2. $l = 1$;
3. Sort T in non-increasing order of their shortest paths distance in G
4. **While** $T \neq \emptyset$
5. $T' = Get\_Req(T, B)$;
6. $(R, S, G_{wl}) = GA\_MEDP(G, T')$;
7. $w_i = l, \forall i \in R$;
8. $\pi_i = p_i, \forall i \in R$;
9. $T = T \backslash (s_i, t_i), \forall i \in R$;
10. $\sigma = \sigma \backslash i, \forall \mathrm{i} \in R$;
11. **for** $j = |T|$ to 1
12. Find shortest path SP_{σ_j} for $(s_{\sigma_j}, t_{\sigma_j})$ on G_{wl};
13. **If** $\left|\mathrm{SP}_{\sigma_j}\right| < \infty$ then
14. $w_{\sigma_j} = l$;
15. $\pi_{\sigma_j} = \mathrm{SP}_{\sigma_j}$;
16. $T = T \backslash \left(s_{\sigma_j}, t_{\sigma_j}\right)$;
17. $\sigma = \sigma \backslash \sigma_j$;
18. **end if**
19. **end for**
20. $l = l + 1$;
21. **end while**
End
Output: π and w

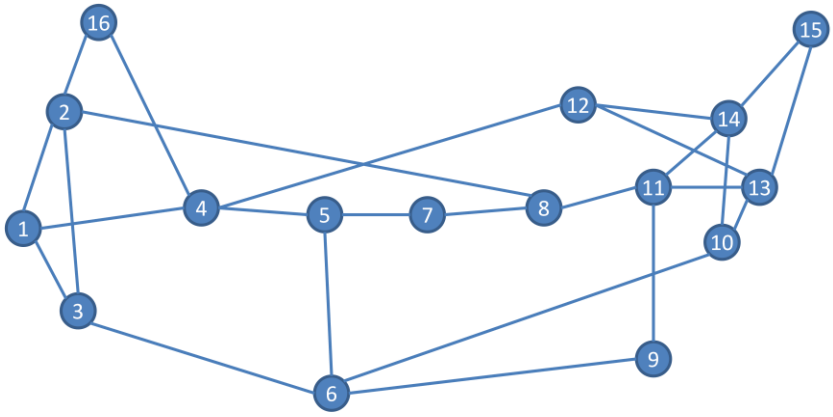
5.2 An illustration

We use a small instance to demonstrate how *GA_MEDP_RWA* works. The National Science Foundation (NSF) Network with 16 nodes and 25 edges has been taken as the benchmark graph in many papers. It is shown in Figure 47. There are 17 randomly-generated requests

listed in Table 5. The third column of Table 5 shows the shortest-path length of each request (assuming each edge has 1 unit cost). The permutation has been adjusted accordingly.



(a)



(b)

Figure 47 Illustration of NSF network with 16 nodes and 25 edges

Table 5 The randomly generated connection request set

Index	Request	length	Index	Request	length
1	(10, 16)	4	10	(11, 12)	2
2	(7, 12)	3	11	(8, 16)	2
3	(13, 16)	3	12	(8, 13)	2
4	(9, 12)	3	13	(7, 8)	1
5	(1, 9)	3	14	(1, 4)	1
6	(3, 16)	2	15	(2, 8)	1
7	(5, 9)	2	16	(2, 16)	1
8	(4, 14)	2	17	(1, 3)	1
9	(5, 16)	2			

Initially, the current wavelength is set to be one. Given the batch size of 5, the algorithm employs GA_MEDP to solve an MEDP problem on G with the first five requests, which are the requests 1, 2, 3, 4 and 5. As a result, requests 1, 3, 4, 5 are accepted. The output edge-disjoint paths are the lightpath routes and the current wavelength is assigned to them. The residual graph is kept in memory and the accepted requests are removed from T. The updated table is shown below.

Table 6 The updated request set after the first run of *GA_MEDP*

Index	Request	wavelength	Index	Request	wavelength
1	~~(10, 16)~~	1	10	(11, 12)	
2	(7, 12)		11	(8, 16)	
3	~~(13, 16)~~	1	12	(8, 13)	
4	~~(9, 12)~~	1	13	(7, 8)	
5	~~(1, 9)~~	1	14	(1, 4)	
6	(3, 16)		15	(2, 8)	
7	(5, 9)		16	(2, 16)	
8	(4, 14)		17	(1, 3)	
9	(5, 16)				

Next, starting from the last request in T, which is request 17, the algorithm is trying to find a shortest path in the residual graph for each request. In this case, shortest paths for request 17, 14, and 12 have been found successfully. They are also assigned to the current wavelength, which is 1, and removed from T. The updated table is shown in Table 7.

Table 7 The updated request set after the backward-scanning process

Index	Request	wavelength	Index	Request	wavelength
1	~~(10, 16)~~	1	10	(11, 12)	
2	(7, 12)		11	(8, 16)	
3	~~(13, 16)~~	1	12	~~(8, 13)~~	1
4	~~(9, 12)~~	1	13	(7, 8)	
5	~~(1, 9)~~	1	14	~~(1, 4)~~	1
6	(3, 16)		15	(2, 8)	
7	(5, 9)		16	(2, 16)	
8	(4, 14)		17	~~(1, 3)~~	1
9	(5, 16)				

So far, we have decided the routes for the requests using the first wavelength. Since the request set is not empty yet, the second run of GA_MEDP is initiated. The current wavelength is set to two and a new batch of 5 requests, which includes requests 2, 6, 7, 8 and 9, are selected. Fortunately, all of them can be accepted and routed in edge-disjoint paths. Followed by the backward-scanning process, it turns out only request 15 can fit into the current residual graph, hence requests 2, 6, 7, 8, 9 and 15 are assigned to wavelength 2 as shown in Table 8.

Table 8 The updated request set after the second run of GA_MEDP and the backward-scanning

Index	Request	wavelength	Index	Request	wavelength
1	~~(10, 16)~~	1	10	(11, 12)	
2	~~(7, 12)~~	2	11	(8, 16)	
3	~~(13, 16)~~	1	12	~~(8, 13)~~	1
4	~~(9, 12)~~	1	13	(7, 8)	
5	~~(1, 9)~~	1	14	~~(1, 4)~~	1
6	~~(3, 16)~~	2	15	~~(2, 8)~~	2
7	~~(5, 9)~~	2	16	(2, 16)	
8	~~(4, 14)~~	2	17	~~(1, 3)~~	1
9	~~(5, 16)~~	2			

Finally, the third batch including all the remaining requests 10, 11, 13 and 16 is selected. As a result, all of them can be accepted and assigned to wavelength 3. Then the algorithm halts. For this small example, the lower bound shown in (2.8) is 3, which means the proposed methods found an optimal solution.

Table 9 The result of applying *GA_MEDP_RWA* on the small example

Index	Request	wavelen	Index	Request	wavelen
1	~~(10, 16)~~	1	10	~~(11, 12)~~	3
2	~~(7, 12)~~	2	11	~~(8, 16)~~	3
3	~~(13, 16)~~	1	12	~~(8, 13)~~	1
4	~~(9, 12)~~	1	13	~~(7, 8)~~	3
5	~~(1, 9)~~	1	14	~~(1, 4)~~	1
6	~~(3, 16)~~	2	15	~~(2, 8)~~	2
7	~~(5, 9)~~	2	16	~~(2, 16)~~	3
8	~~(4, 14)~~	2	17	~~(1, 3)~~	1
9	~~(5, 16)~~	2			

5.3 Testing instances and parameter tuning

A testing instance (G, T) of the RWA problem contains an undirected graph G and a request set T. In Section 5.3.1 we outline some basic characteristics of the testing graphs and the procedure to generate the requests. There are 67 testing instances in total. In Section 5.3.2, the fine tuning process of the parameter B is provided.

5.3.1 Testing instances

In order to evaluate the performance of the proposed method for solving RWA, numerical testing is conducted in comparison with the bin-packing method and the PSO method reported in the literature [4, 26]. An instance of RWA consists of an undirected network G and a set of connection requests T. For the network topology, we use 15 benchmark networks provided in [36, 44, 46] which assume the patterns and sizes of some real-life

telecommunication networks. The topology of NSFNET and EON are the most studied realistic networks in the literature. Network CHNNET and ARPANET are provided by [46], USAnet is given by [44], and other instances are taken from [36]. The main quantitative characteristics of these networks are shown in Table 10.

Table 10 Main quantitative characteristics of the instances

Graph	$\|V\|$	$\|E\|$	Min.	Avg.	Max.	Diameter
CHNNET	15	27	3	3.6	5	5
NSFNET	16	25	2	3.1	4	4
NewYork	16	49	2	6.1	11	3
ARPANET	20	32	3	3.2	4	6
EON	20	39	2	3.9	7	5
France	25	45	2	3.6	10	5
Norway	27	51	2	3.8	6	7
cost266	37	57	2	3.1	5	8
janos-us-ca	39	61	2	3.1	5	10
giul	39	86	3	4.4	8	6
piro40	40	89	4	4.5	5	7
USAnet	46	75	2	3.3	5	11
Germany50	50	88	2	3.5	5	9
zib54	54	80	1	3.0	10	8
ta2	65	108	1	3.3	10	8

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

Regarding the connection requests, for each network, different numbers of connection requests are randomly generated according to a given probability p, i.e., the probability that there is a request between a pair of nodes is p. The mechanism to generate the requests is provided in Algorithm 15. Four instances are generated for each network with p equals to 0.2, 0.4, 0.6 and 0.8, respectively. For the networks of smaller size, namely, CHNNET, NSFNET, NewYork, ARPANET, EON, France and Norway, one more instance with $p = 1$

is generated. That means every pair of different nodes of the network requests a connection between them. There are 67 testing instances in total, which are listed in Table 11. The last two numbers of the instance's name indicate the value p that is used to generate the instance. For example, instance CHNNET_02 is generated on network CHNNET with $p = 0.2$ and CHNNET_1 is generated with $p = 1$.

Algorithm 15 Request generator

Input: $G = (V, E),\ p$
Begin:
1. $T = \emptyset$;
2. **for** $i = 1$ to $|V|$
3. **for** $j = 1\ to\ |V|$
4. **if** $j > i$ **and** rand(0,1) $\leq p$
5. $T = T \cup \{(i, j)\}$;
6. **end if**
7. **end for**
8. **end for**
9. $\rho \leftarrow$random permutation of $(1, 2, \dots, \mathrm{I})$;
10. $T = \{(s_{\rho(i)}, t_{\rho(i)}) | i = 1, \dots, \mathrm{I}\}$;

End
Output: T

Table 11 Testing instances

name	graph	$\|V\|$	$\|E\|$	$\|T\|$
CHNNET_02	CHNNET	15	27	22
CHNNET_04				36
CHNNET_06				71
CHNNET_08				97
CHNNET_10				105
NSF_02	NSFNET	16	25	28
NSF_04				54
NSF_06				63
NSF_08				101
NSF_10				120
NewYork_02	NewYork	16	49	20
NewYork _04				47
NewYork _06				66
NewYork _08				96
NewYork _10				120
ARPANET_02	ARPANET	20	32	46
ARPANET_04				75
ARPANET_06				112
ARPANET_08				169
ARPANET_10				190
EON_02	EON	20	39	33
EON_04				85
EON_06				106
EON_08				147
EON_10				190
France_02	France	25	45	62
France _04				109
France _06				177
France _08				237
France _10				300

Table 11 Continued

name	graph	$\|V\|$	$\|E\|$	$\|T\|$
Norway_02	Norway	27	51	76
Norway_04				136
Norway_06				199
Norway_08				283
Norway_10				351
cost266_02	cost266	37	57	121
cost266_04				259
cost266_06				391
cost266_08				528
janos-us-ca_02	janos-us-ca	39	61	144
janos-us-ca_04				299
janos-us-ca_06				463
janos-us-ca_08				604
giul_02	giul	39	86	147
giul_04				312
giul_06				427
giul_08				585
piro40_02	piro40	40	89	169
piro40_04				302
piro40_06				459
piro40_08				607
USAnet_02	USAnet	46	75	208
USAnet_04				427
USAnet_06				619
USAnet_08				825
Germany50_02	Germany50	50	88	237
Germany50_04				519
Germany50_06				774
Germany50_08				954

Table 11 Continued

name	graph	$\|V\|$	$\|E\|$	$\|T\|$
zib54_02	zib54	54	80	269
zib54_04				577
zib54_06				851
zib54_08				1127
ta2_02	ta2	65	108	432
ta2_04				836
ta2_06				1218
ta2_08				1656

5.3.2 Tuning the batch size

In **Algorithm 14**, the batch size B is the only parameter to be tuned and is indeed an important factor affecting the performance. A bigger value of B means that the algorithm considers more requests at the same time and tries to route them in the same graph by edge-disjoint paths, thus may yield a better solution at the cost of longer computational time.

Two instances USAnet_08 and janos-us-ca_08 are tested to decide the best batch size from six possible settings of B: $\{5, 10, 15, 20, 25, 30\}$. Figure 48 and Figure 50 show the 95% confidence intervals of the number of wavelengths obtained on janos-us-ca_08 and USAnet_08 with different settings of B, respectively, while Figure 49, Figure 51 indicate the 95% confident intervals of computational time. It is obvious that the solution quality gets better and the computational time increases with bigger batch size. We finally decided that $B = 20$ is a good balance between the solution quality and computation time. All the results shown in Section 5.4 are outcomes by setting $B = 20$.

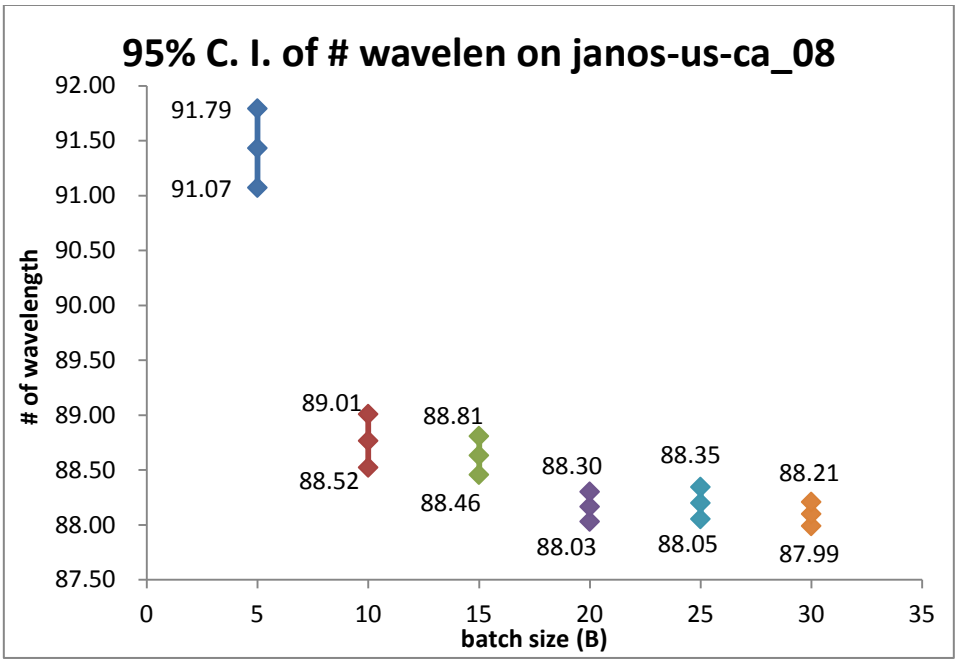


Figure 48 95% C. I. of the objective value obtained with different B on janos-us-ca_08

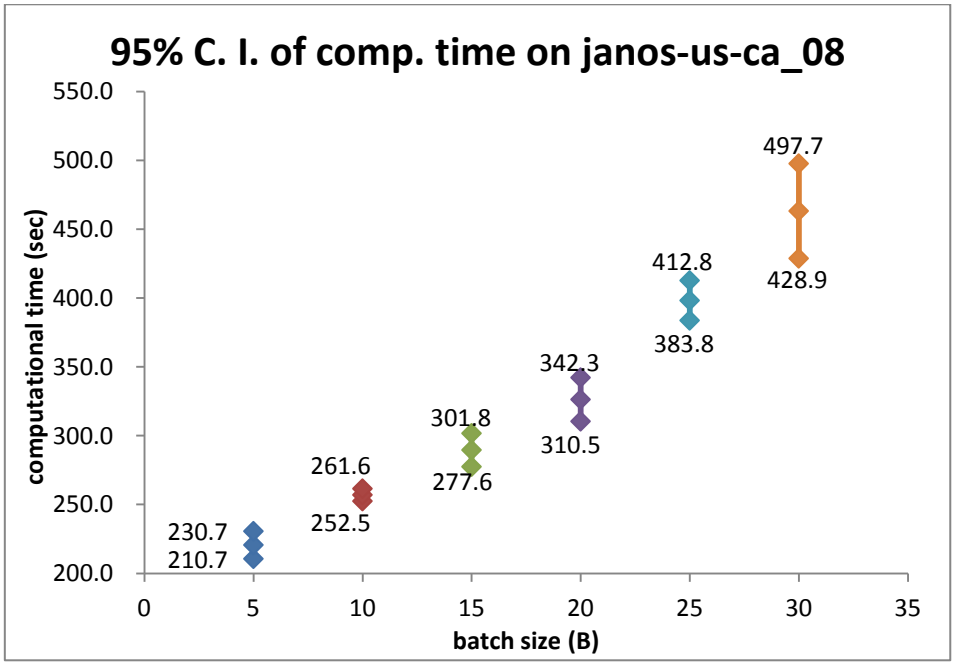


Figure 49 95% C. I. of the computational time with different B on janos-us-ca_08

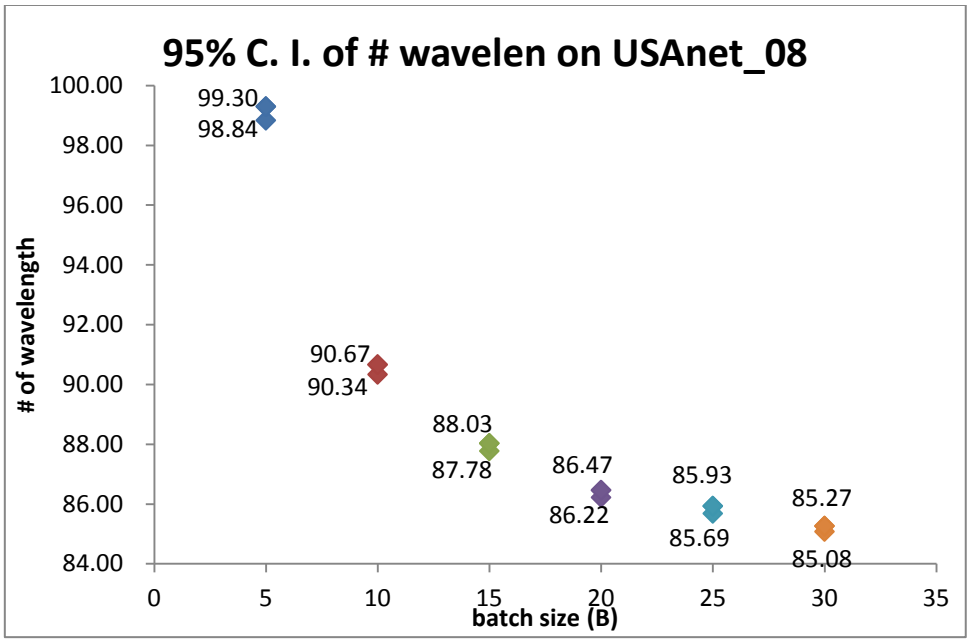


Figure 50 95% C. I. of the objective value obtained with different B on USAnet_08

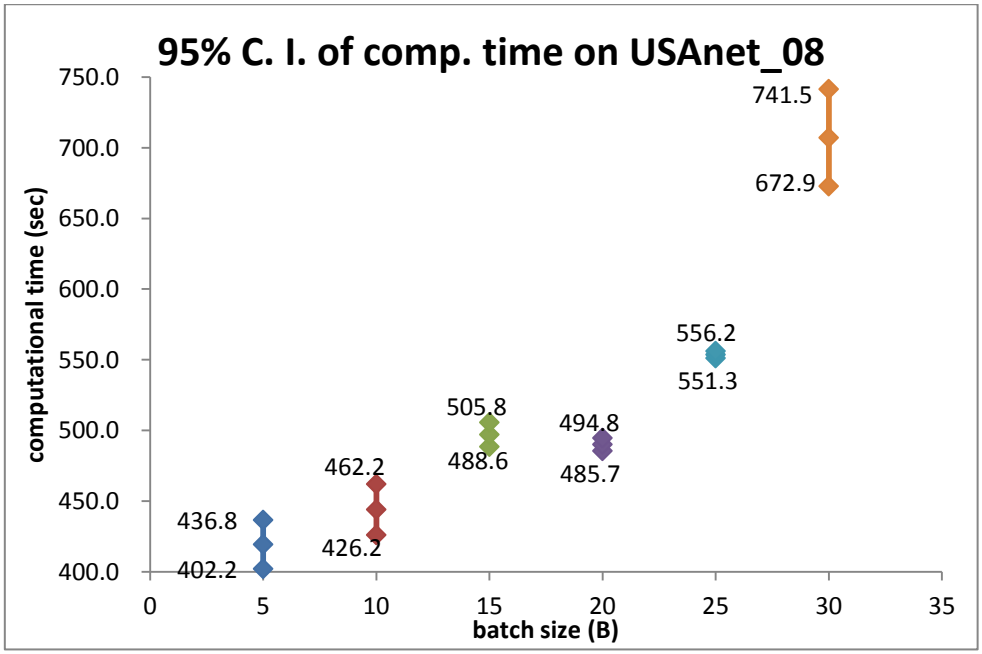


Figure 51 95% C. I. of the computational time with different B on USAnet_08

5.4 Computational experiments

The computational results and comparisons with the bin-packing based heuristic and PSO are reported in Sections 5.4.1 and 5.4.2, respectively. The three algorithms were implemented in MATLAB and the experiments were conducted on a PC with Intel® Core i7 CPU @1.6GHz and 4 Gb of memory running the Windows 7 operating system. The experiments were conducted by applying PSO and the proposed method on each instance for 30 runs to obtain the best, worst, mean and standard deviation of the objective values. The average computational time are also recorded. Regarding the four bin-packing methods, each method only needs to be executed once, and the results and computational time are recorded.

5.4.1 GA_MEDP_RWA vs. bin-packing based methods

The comparison of the proposed method and the four bin-packing based methods is shown in Table 12. The first column gives the name of the tested instance. For the FF, FFD, BF and BFD methods, the objective value, which is denoted by # wl, and the computation time t_{FF}, t_{FFD}, t_{BF} and t_{BFD} were recorded, respectively. The first three columns under *GA_MEDP_RWA* show the maximum, average, and minimum objective values that were obtained in 30 runs. The fourth column is the mean computation time. The objective value is underlined and in boldface when it is the best among the five. Here we say that a solution method achieves the best objective value on an instance if the value obtained is less than or equal to that obtained by other methods. We also say that a solution method achieves the worst objective value if the value obtained is greater than that obtained by other methods.

Table 12 Results of *GA_MEDP_RWA* and bin-packing based methods (time unit: sec)

time unit (sec)	*GA_MEDP_RWA*				FF		FFD		BF		BFD	
instance	*max*	*avg*	*min*	$\bar{t}$	# wl	t_{FF}	# wl	t_{FFD}	# wl	t_{BF}	# wl	t_{BFD}
CHNNET_02	4	4.0	**4**	2.07	**4**	0.24	**4**	0.14	**4**	0.11	5	0.14
CHNNET_04	5	4.4	**4**	3.02	5	0.18	**4**	0.15	5	0.23	**4**	0.26
CHNNET_06	11	11.0	**11**	10.66	**11**	0.70	**11**	0.62	**11**	0.81	**11**	0.96
CHNNET_08	14	14.0	**14**	13.50	15	1.02	**14**	1.17	**14**	1.21	**14**	1.51
CHNNET_10	16	15.0	**15**	16.31	**15**	1.48	**15**	1.42	16	1.75	**15**	2.20
NSF_02	5	5.0	**5**	2.41	6	0.34	**5**	0.17	6	0.19	6	0.18
NSF_04	8	7.3	**7**	5.80	9	0.37	9	0.35	8	0.52	8	0.50
NSF_06	9	8.8	**8**	7.97	10	0.46	10	0.49	9	0.59	9	0.64
NSF_08	14	13.2	**13**	15.11	17	1.06	15	1.10	14	1.50	14	1.47
NSF_10	17	16.6	**16**	22.89	19	1.70	17	1.45	17	1.96	17	2.08
NewYork_02	2	2.0	**2**	1.37	**2**	0.14	**2**	0.08	**2**	0.07	**2**	0.08
NewYork_04	3	3.0	**3**	2.65	**3**	0.23	**3**	0.20	**3**	0.23	**3**	0.23
NewYork_06	5	4.1	**4**	3.93	**4**	0.33	**4**	0.34	**4**	0.34	**4**	0.33
NewYork_08	7	6.0	**6**	6.75	7	0.64	**6**	0.58	7	0.62	**6**	0.81
NewYork_10	8	8.0	**8**	7.03	**8**	0.73	**8**	0.80	**8**	1.02	**8**	2.30
ARPANET_02	10	9.1	**9**	10.58	**9**	0.55	**9**	0.53	**9**	0.64	**9**	0.86
ARPANET_04	13	12.1	**12**	16.96	**12**	1.05	**12**	1.29	**12**	1.31	**12**	1.76
ARPANET_06	21	21.0	**21**	25.98	**21**	2.70	**21**	2.68	**21**	3.39	**21**	4.58
ARPANET_08	29	29.0	**29**	32.31	30	5.46	**29**	6.14	30	7.82	**29**	10.94
ARPANET_10	33	33.0	**33**	62.85	34	6.85	**33**	7.77	34	10.19	**33**	12.68
EON_02	4	3.1	**3**	3.19	**3**	0.22	4	0.17	4	0.34	4	0.36
EON_04	9	8.3	**8**	14.74	10	1.10	9	1.14	**8**	1.53	**8**	1.54
EON_06	11	11.0	**11**	17.52	13	1.21	**11**	1.18	**11**	1.94	**11**	1.90
EON_08	14	13.7	**13**	26.63	16	2.20	**13**	2.99	**13**	3.61	**13**	3.72
EON_10	19	18.1	**18**	38.17	22	3.64	**18**	3.58	**18**	5.77	**18**	5.73
France_02	8	8.0	**8**	8.87	**8**	1.30	**8**	1.27	**8**	1.57	**8**	1.72
France_04	13	12.8	**12**	16.05	14	4.29	13	4.90	13	5.14	13	6.03
France_06	22	22.0	**22**	39.07	**22**	8.73	**22**	10.55	**22**	11.69	**22**	15.27
France_08	27	26.3	**26**	47.09	28	13.25	27	14.00	27	18.53	**26**	24.20
France_10	34	34.0	**34**	60.09	**34**	22.54	**34**	23.99	**34**	29.93	**34**	35.84

Table 12 Continued

	GA_MEDP_RWA				FF		FFD		BF		BFD	
instance	*max*	*avg*	*min*	$\bar{t}$ (s)	# wl	t_{FF}	# wl	t_{FFD}	# wl	t_{BF}	# wl	t_{BFD}
Norway_02	9	8.6	**8**	9.26	10	1.67	**8**	1.78	9	1.90	**8**	2.17
Norway_04	15	14.6	**14**	18.47	15	3.75	15	4.12	15	4.56	15	5.43
Norway_06	22	21.4	**21**	32.84	22	7.87	22	8.20	22	9.73	22	12.32
Norway_08	30	29.5	**29**	46.41	31	15.05	30	16.59	31	19.79	30	24.59
Norway_10	37	36.6	**36**	63.77	37	21.99	**36**	23.75	38	28.58	**36**	36.13
cost266_02	19	18.2	**18**	46.68	19	7.34	**18**	7.44	19	10.38	19	12.40
cost266_04	35	34.1	**33**	159.69	35	27.28	**33**	28.58	36	36.35	35	47.28
cost266_06	54	53.1	**53**	237.55	54	67.18	**53**	70.90	55	98.00	**53**	117.30
cost266_08	68	67.2	**67**	275.29	**67**	120.73	68	144.22	69	190.55	68	273.40
janos-us-ca_02	26	26.0	**26**	54.75	27	16.61	**26**	16.53	27	27.40	27	27.58
janos-us-ca_04	40	39.6	**39**	97.42	42	49.52	**39**	59.49	43	70.75	40	88.45
janos-us-ca_06	68	67.8	**67**	210.36	71	110.99	69	124.10	72	157.50	68	200.32
janos-us-ca_08	89	88.2	**88**	326.39	92	199.09	92	212.16	93	257.39	91	345.67
giul_02	11	10.3	**10**	26.93	**10**	7.40	**10**	8.11	**10**	8.89	**10**	11.24
giul_04	21	20.2	**19**	82.66	**19**	29.10	**19**	31.70	**19**	37.42	**19**	46.16
giul_06	25	24.6	**24**	150.73	25	48.91	25	55.96	25	59.10	**24**	73.12
giul_08	36	34.9	**34**	226.85	35	113.85	**34**	125.20	36	134.25	**34**	181.70
piro40_02	17	17	**17**	55.51	**17**	10.67	**17**	12.58	**17**	15.20	**17**	18.50
piro40_04	28	28	**28**	118.50	**28**	34.98	**28**	38.00	**28**	43.05	**28**	54.44
piro40_06	46	46	**46**	208.81	**46**	61.98	**46**	68.00	**46**	84.33	**46**	109.15
piro40_08	60	60	**60**	338.61	**60**	105.97	**60**	119.35	**60**	140.20	**60**	181.52
USAnet_02	25	24.0	**23**	86.13	25	35.62	26	37.95	26	45.15	25	56.09
USAnet_04	47	45.9	**45**	355.64	47	130.43	**45**	147.76	48	176.43	**45**	228.50
USAnet_06	67	66.1	**65**	422.63	68	244.00	**65**	287.58	68	346.66	66	432.89
USAnet_08	88	86.5	**85**	490.20	88	475.69	**85**	544.98	89	587.11	86	791.38
Germany50_02	21	20.7	**20**	109.28	21	35.86	21	36.10	21	46.10	22	58.58
Germany50_04	45	44.2	**43**	350.24	45	152.90	44	163.00	48	217.60	48	276.50
Germany50_06	61	60.5	**60**	584.92	63	347.70	61	410.61	63	430.89	65	544.74
Germany50_08	76	74.7	**73**	921.59	77	552.50	75	725.90	79	843.70	76	1098.40

Table 12 Continued

	GA_MEDP_RWA				FF		FFD		BF		BFD	
instance	*max*	*avg*	*min*	$\bar{t}$	# wl	t_{FF}	# wl	t_{FFD}	# wl	t_{BF}	# wl	t_{BFD}
zib54_02	32	31.1	**30**	149.44	33	52.28	31	55.01	35	77.70	33	91.67
zib54_04	67	65.7	**65**	406.90	67	209.80	**65**	220.50	73	304.64	71	379.55
zib54_06	92	90.0	**88**	762.56	91	442.95	89	522.85	99	696.78	98	889.98
zib54_08	122	119.2	**117**	1446.41	**117**	994.60	**117**	939.60	130	1229.60	127	1457.22
ta2_02	35	34.6	**34**	411.59	35	148.93	**34**	163.00	35	188.75	35	250.99
ta2_04	72	70.1	**69**	945.79	72	507.30	**69**	542.50	73	635.57	70	813.04
ta2_06	99	97.4	**96**	1865.20	98	1311.70	98	1484.00	100	1881.30	97	2047.40
ta2_08	131	129.7	**128**	2835.57	130	1935.30	129	2577.90	135	2644.43	**128**	3713.46

Some observations can be made based on Table 12. First, the numbers of times that each method achieved the best objective values among the 67 instances are summarized in Figure 52. *GA_MEDP_RWA* achieved the best objective value on all instances, while the FFD method reaches the best value on 53 instances. Both performed better than the other three bin-packing based methods. Concerning the worst case shown in Figure 53, *GA_MEDP_RWA* found the solution with the worst objective value for 5 instances. This shows the effectiveness and robustness of the proposed method.

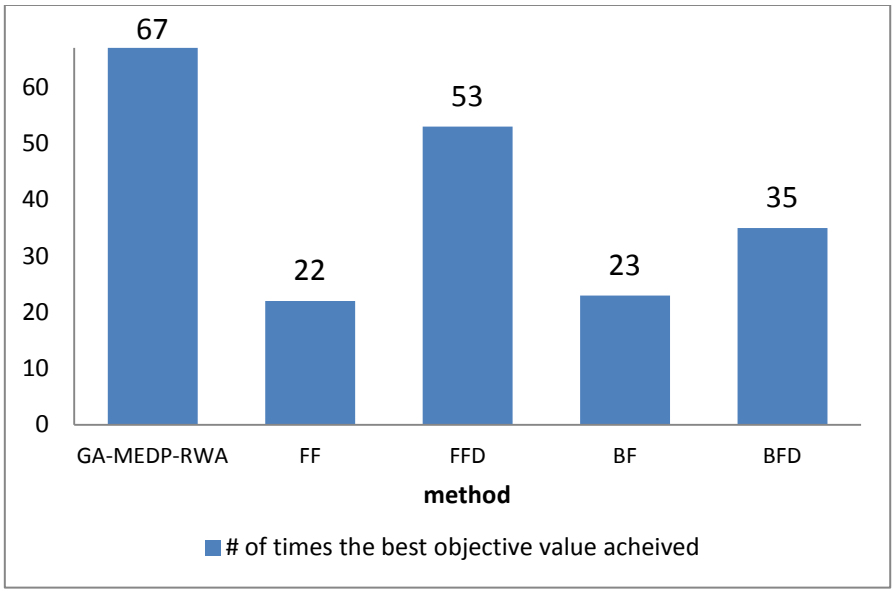


Figure 52 Number of times that the best value is achieved by different methods among 67 instances

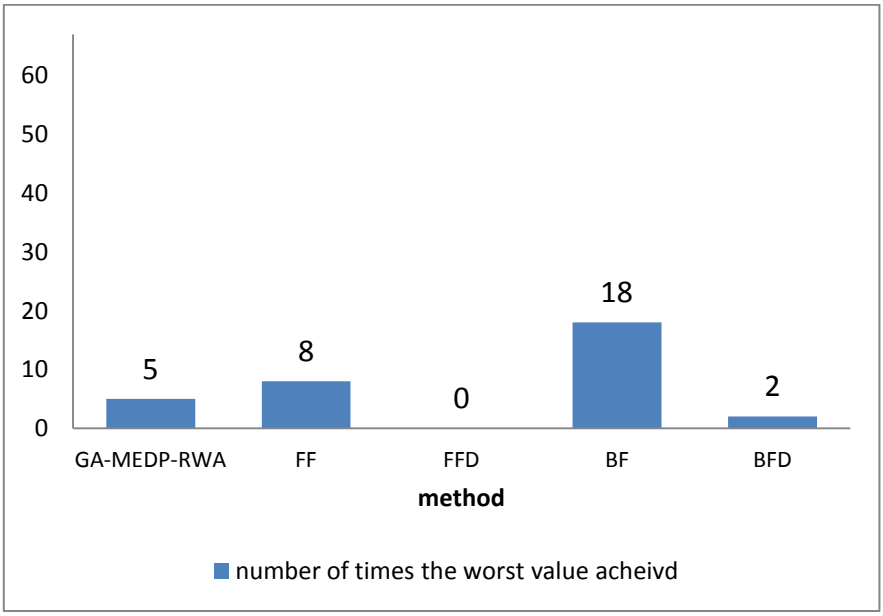


Figure 53 Number of times that the worst value is achieved by different methods among 67 instances

The second observation is that, bin-packing based methods have clearly advantages in terms of the computation time, especially, on those relatively small instances. For example, the average computational time of *GA_MEDP_RWA* on network CHNNET and NSF can be ten to twenty times that of bin-packing based methods. However, the difference of computation time becomes smaller as the problem size grows. We use the instances on the four networks: Norway, giul, Germany50, and ta2, whose sized are in an ascending order, to demonstrate how the computation time changes with the problem size. Define the relative differences to be $(\bar{t} - t_{method})/\bar{t}$, where $\bar{t}$ is the average computation time of *GA_MEDP_RWA* and t_{method} is the computation time of one of the four bin-packing based methods ($method$ can be FF, FFD, BF or BFD, etc). The relative differences for all the instances on the four networks are shown in Figure 54 -57, respectively.

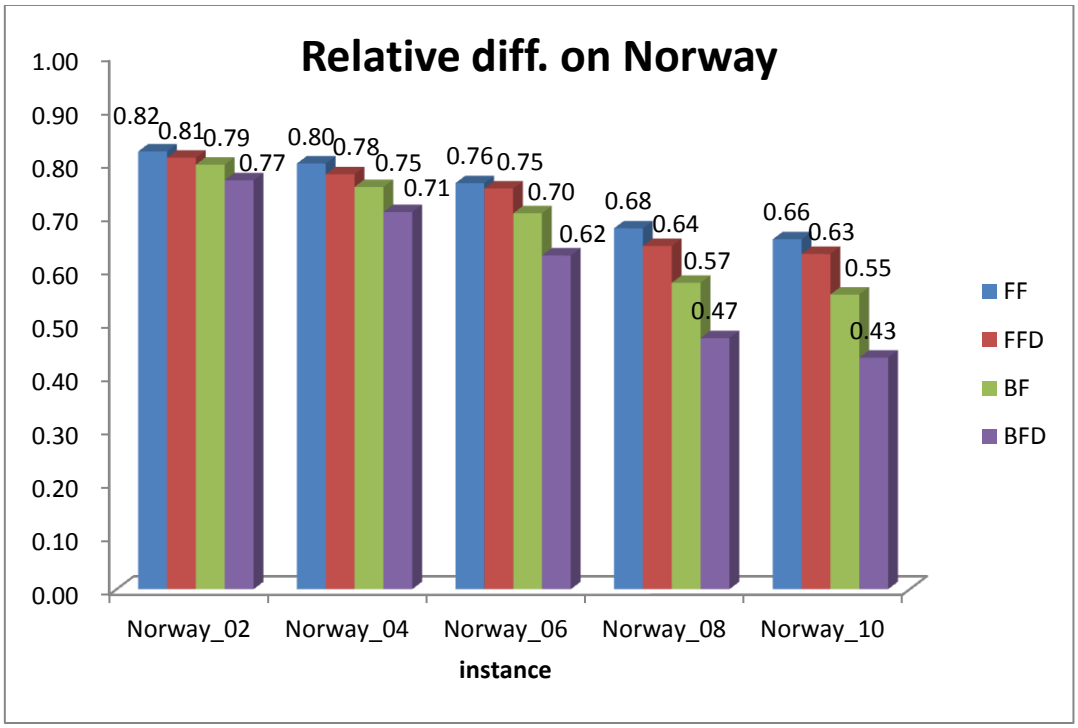


Figure 54 Relative difference of computational time on graph "Norway"

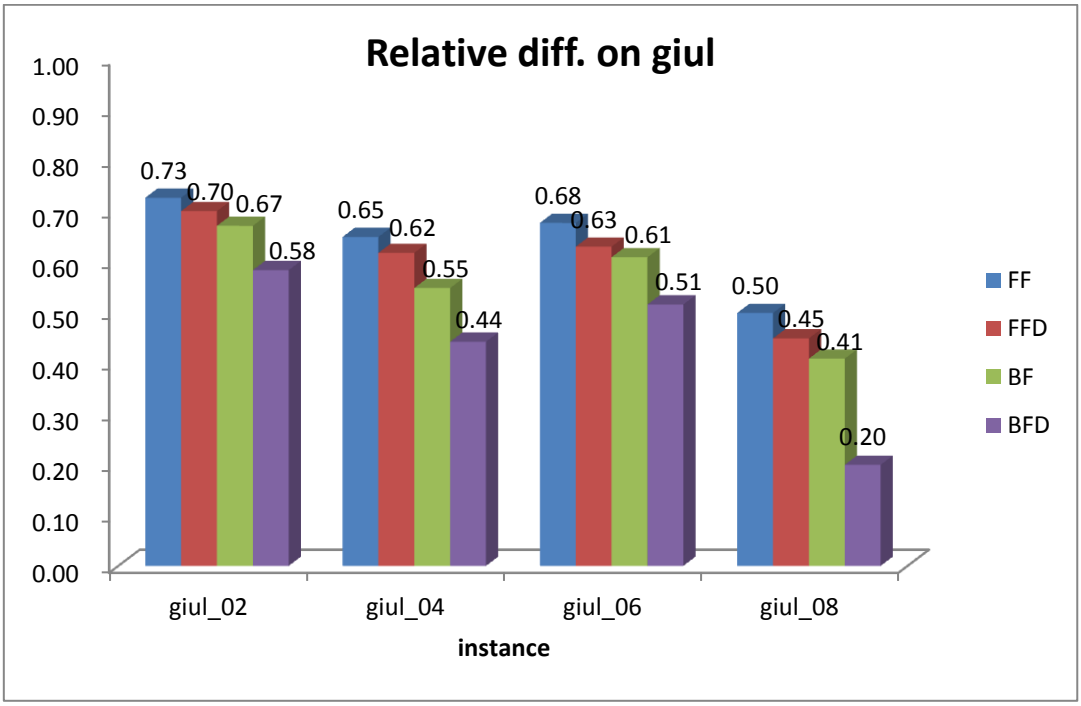


Figure 55 Relative difference of computational time on graph "giul"

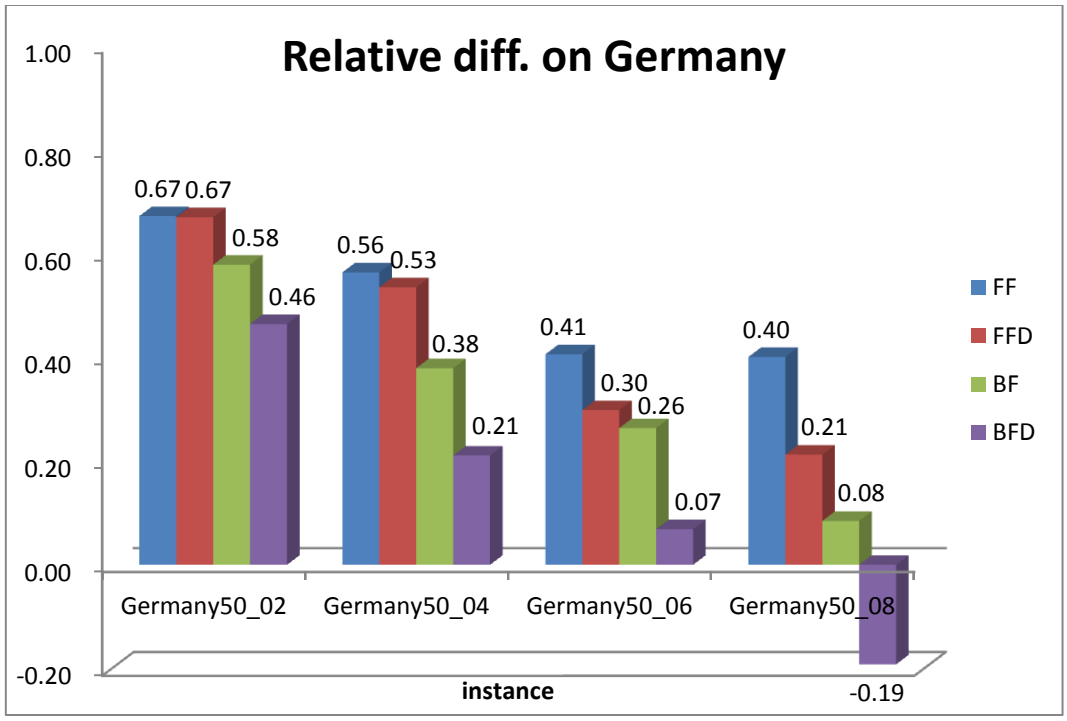


Figure 56 Relative difference of computational time on graph "Germany"

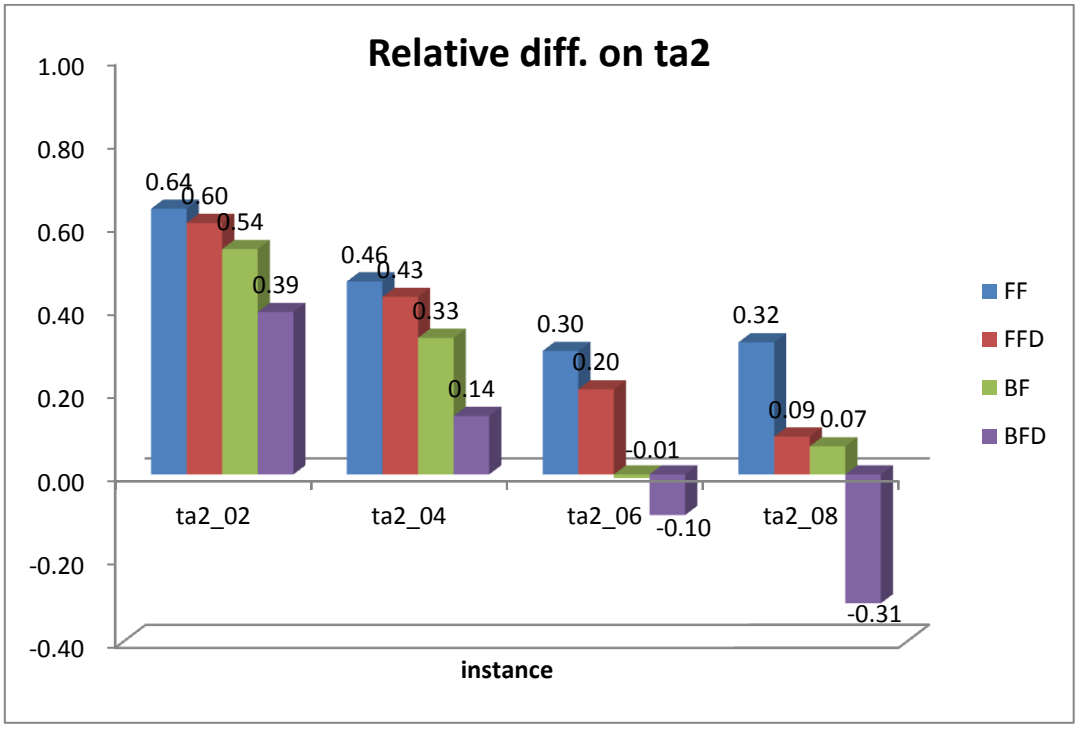


Figure 57 Relative difference of computational time on graph "ta2"

In Figure 54, we can observe that the relative difference tends to go down as the number of connection requests increases. The trend also exists in Figure 55, Figure 56 and Figure 57 and most of other instances. In addition, the relative difference becomes smaller as the network size grows. In Figure 56 we can see that the proposed method spent less time on Germany50_08 than BFD did. In Figure 57, the same situation happened on ta2_06 and ta2_08. Since the four networks of Norway, giul, Germany50 and ta2 are in an ascending order of the network size, by observing Figure 54-57, we can see the phenomenon that the relative differences drop as the network size grows.

5.4.2 GA_MEDP_RWA vs. PSO

The comparison of the proposed method and PSO is shown in Table 13. The minimum objective value is underlined and in boldface if it is the better one. Notice that in this comparison the computation time can be neglected, since we set a weak termination condition to let PSO explore more possible solutions. Therefore, PSO always spends more computation time. We can observe that PSO achieved the best values on 3 relatively small instances while GA_MEDP_RWA achieved the best objective values on all instances. On bigger instances, the performances of PSO are not comparable at all to that of *GA_MEDP_RWA*. The obtained objective values of PSO are twice more than that obtained by *GA_MEDP_RWA*.

The difference in the path construction methods of PSO, *GA_MEDP_RWA* and bin-packing based methods is crucial for distinct performance. Briefly speaking, in *GA_MEDP*, a path can be adjusted by manipulating the priority values. These priority values can be changed during the reproduction and local improvement procedures according to different network conditions and request topologies. In bin-packing based methods, the shortest paths are established iteratively to fit into the residual graphs. Therefore, the number of network resources (edges) to route each request is minimized. On the contrary, in PSO, recombining route ids from predetermined path candidates is lacking of ability to adapt different situations. The deficiency becomes more significant on instance of bigger networks or with more requests.

Table 13 Results obtained by *GA_MEDP_RWA* and PSO (time unit: sec)

	GA_MEDP_RWA					PSO				
instance	max	avg	std	min	avg time	max	avg	std	min	avg time
CHNNET_02	4	4.0	0.00	**4**	2.07	7	5.9	0.55	5	12.97
CHNNET_04	5	4.4	0.50	**4**	3.02	9	7.5	0.63	7	18.60
CHNNET_06	11	11.0	0.00	**11**	10.66	18	15.5	1.36	13	30.39
CHNNET_08	14	14.0	0.00	**14**	13.50	24	20.4	1.28	18	48.95
CHNNET_10	16	15.0	0.18	**15**	16.31	25	21.5	1.63	18	78.96
NSF_02	5	5.0	0.00	**5**	2.41	7	6.0	0.10	**5**	6.75
NSF_04	8	7.3	0.48	**7**	5.80	10	9.0	0.16	8	13.75
NSF_06	9	8.8	0.38	**8**	7.97	11	10.0	0.20	9	16.19
NSF_08	14	13.2	0.38	**13**	15.11	17	15.5	1.48	14	29.02
NSF_10	17	16.6	0.49	**16**	22.89	20	18.9	0.69	18	36.03
NewYork_02	2	2.0	0.00	**2**	1.37	3	2.9	0.31	**2**	7.37
NewYork_04	3	3.0	0.00	**3**	2.65	6	5.2	0.46	4	18.78
NewYork_06	5	4.1	0.35	**4**	3.93	8	6.8	0.50	6	25.51
NewYork_08	7	6.0	0.18	**6**	6.75	10	9.1	0.69	8	38.35
NewYork_10	8	8.0	0.00	**8**	7.03	12	10.7	0.60	10	50.17
ARPANET_02	10	9.1	0.31	**9**	10.58	10	9.1	0.74	**9**	10.58
ARPANET_04	13	12.1	0.31	**12**	16.96	19	16.3	1.27	14	62.15
ARPANET_06	21	21.0	0.00	**21**	25.98	28	24.9	1.26	22	93.86
ARPANET_08	29	29.0	0.00	**29**	32.31	40	36.5	1.48	33	133.70
ARPANET_10	33	33.0	0.00	**33**	62.85	46	41.2	2.47	36	172.60
EON_02	4	3.1	0.31	**3**	3.19	5	4.3	0.09	4	10.40
EON_04	9	8.3	0.48	**8**	14.74	12	10.6	0.49	10	35.34
EON_06	11	11.0	0.00	**11**	17.52	15	13.0	0.62	12	46.61
EON_08	14	13.7	0.47	**13**	26.63	20	18.2	3.75	17	48.92
EON_10	19	18.1	0.25	**18**	38.17	25	22.9	3.74	22	73.32
France_02	8	8.0	0.00	**8**	8.87	14	11.5	0.86	10	50.60
France_04	13	12.8	0.38	**12**	16.05	22	19.3	1.26	17	86.48
France_06	22	22.0	0.00	**22**	39.07	34	30.4	1.67	27	129.16
France_08	27	26.3	0.48	**26**	47.09	45	40.3	1.82	38	290.72
France_10	34	34.0	0.00	**34**	60.09	54	48.9	2.36	45	236.64

Table 13 Continued

	GA-MEDP-RWA					PSO				
instance	max	avg	std	min	avg time	max	avg	std	min	avg time
Norway_02	9	8.6	0.50	**8**	9.26	15	13.1	0.92	11	61.56
Norway_04	15	14.6	0.50	**14**	18.47	25	22.5	1.31	20	101.40
Norway_06	22	21.4	0.49	**21**	32.84	37	32.6	2.06	29	202.11
Norway_08	30	29.5	0.51	**29**	46.41	48	43.6	1.70	40	254.75
Norway_10	37	36.6	0.50	**36**	63.77	59	54.1	2.60	49	323.64
cost266_02	19	18.2	0.41	**18**	46.68	26	23.8	1.09	21	105.22
cost266_04	35	34.1	0.45	**33**	159.69	54	50.9	1.81	47	239.95
cost266_06	54	53.1	0.31	**53**	237.55	79	73.2	2.07	70	436.38
cost266_08	68	67.2	0.43	**67**	275.29	99	93.4	2.81	88	703.53
janos-us-ca_02	26	26.0	0.00	**26**	54.75	40	36.1	2.23	31	288.18
janos-us-ca_04	40	39.6	0.49	**39**	97.42	66	60.5	2.79	56	529.74
janos-us-ca_06	68	67.8	0.41	**67**	210.36	111	102.7	4.02	94	959.93
janos-us-ca_08	89	88.2	0.38	**88**	326.39	144	134.1	5.88	122	1239.46
giul_02	11	10.3	0.48	**10**	26.93	16	14.5	0.86	13	294.44
giul_04	21	20.2	0.50	**19**	82.66	32	28.8	1.56	26	451.61
giul_06	25	24.6	0.49	**24**	150.73	42	38.3	2.02	34	738.65
giul_08	36	34.9	0.51	**34**	226.85	59	53.5	2.57	49	990.63
piro40_02	17	17	0.00	**17**	55.51	22	20.0	0.74	19	159.88
piro40_04	28	28	0.00	**28**	118.50	36	33.7	0.99	32	300.88
piro40_06	46	46	0.00	**46**	208.81	66	58.7	3.27	52	474.73
piro40_08	60	60	0.00	**60**	338.61	86	79.4	2.85	73	684.40
USAnet_02	25	24.0	0.49	**23**	86.13	40	36.4	2.03	32	446.56
USAnet_04	47	45.9	0.50	**45**	355.64	76	69.5	2.91	65	983.93
USAnet_06	67	66.1	0.58	**65**	422.63	106	98.1	3.74	92	1418.73
USAnet_08	88	86.5	0.68	**85**	490.20	139	126.5	4.18	120	1715.20
Germany50_02	21	20.7	0.45	**20**	109.28	37	33.5	1.72	31	379.56
Germany50_04	45	44.2	0.48	**43**	350.24	74	69.4	2.11	66	683.95
Germany50_06	61	60.5	0.51	**60**	584.92	102	95.7	2.70	91	969.87
Germany50_08	76	74.7	0.77	**73**	921.59	126	120.5	3.21	115	1271.90

Table 13 Continued

	GA-MEDP-RWA					PSO				
instance	max	avg	std	min	avg time	max	avg	std	min	avg time
zib54_02	32	31.1	0.51	**30**	149.44	75	64.1	4.23	58	619.63
zib54_04	67	65.7	0.70	**65**	406.90	159	141.6	6.80	133	1462.19
zib54_06	92	90.0	0.96	**88**	762.56	208	195.2	5.29	187	2955.29
zib54_08	122	119.2	1.05	**117**	1446.41	289	266.3	8.90	250	4147.09
ta2_02	35	34.6	0.49	**34**	411.59	75	70.3	2.59	66	1453.68
ta2_04	72	70.1	0.71	**69**	945.79	152	141.8	4.05	133	2852.21
ta2_06	99	97.4	0.79	**96**	1865.20	208	198.7	5.49	181	3867.39
ta2_08	131	129.7	0.69	**128**	2835.57	284	271.0	5.56	258	5953.52

5.5 Summary

We have developed the heuristic method *GA_MEDP_RWA* for solving the RWA problem. It combines the idea of bin-packing method and edge-disjoint paths together. The method considers a batch of requests at a time and solves the corresponding MEDP problem. The remaining requests are then scanned backward such that the one with shorter shortest paths tries to fit into the existing wavelengths first. In such manner, the algorithm constructed a solution consequently.

The computational result confirmed the effectiveness of the proposed method. Compared with the bin-packing based methods and the PSO approach, *GA_MEDP_RWA* can find the best solution on all instances. Although the proposed method takes longer time than the bin-packing methods for small instances, the relative difference of computational time becomes smaller as the problem size grows. We also pointed out that different route-establishing mechanism might be a crucial factor causing the differences of performance between the three methods.

Chapter 6 Conclusion and future research

In this chapter, we summarize our work and point out some possible directions for future research.

6.1 Summary of work done

The maximum edge-disjoint paths (MEDP) problem plays an important role in modern communication networks. Real-world applications of MEDP include VLSI layout, the routing and wavelength assignment problem, call admission control problem, etc. In the first two chapters, the background and complexity of MEDP are provided. Some existing solution methods and their approximation ratios are also reviewed. In Chapter 3, we proposed a novel genetic-based algorithm called *GA_MEDP* for solving the MEDP problem. Each individual in *GA_MEDP* is a collection of paths, in which each path is associated with one connection request and is encoded as a vector of priority values in the range of $[0, 1]$. To generate a feasible solution, a heuristic called *GMIN* is used to obtain a set of edge-disjoint paths from the path set that the individual represents. Then a bicriteria fitness function is used to evaluate the individual. In the reproducing stage, three genetic operators are proposed to create offspring by manipulating the priority values. In addition, an improvement heuristic is provided to further enhance the offspring. The computation results reported in Chapter 4 show that, compared with the multi-start greedy algorithm and ant colony optimization method, the proposed method performs better in most instances in terms of solution quality and time.

We further apply *GA_MEDP* for a real-world application on optical communication

networks – routing and wavelength assignment (RWA) problem. The RWA problem is a graph optimization problem which generalizes MEDP in some aspects and has been extensively studied for decades. In Section 1.5, the background and formulation of RWA are provided. Related works and two solution methods are reviewed in Chapter 2. Firstly, the state-of-the-art bin-packing based method, which has four variants: FF, FFD, BF and BFD. It considers the requests to represent "items" and copies of the graph to represent "bins". Then classic solution methods for the bin-packing problem are used to tackle RWA. Secondly, the particle swarm optimization (PSO), in which each particle is represented by a set of route-ids. For each request, a route-id selected from a set of predetermined route candidates for the request is assigned. The wavelength assignment is taken care of by evaluating the number of sets of edge-disjoint paths among the routes that the particle represents. In Chapter 5, we proposed a method called *GA_MEDP_RWA* for RWA. It combines the idea of the bin-packing method and edge-disjoint paths together. The method considers only a number of requests at a time and solves the corresponding MEDP problem. The remaining requests are then scanned backward such that the one with shorter shortest paths tries to fit into the existing wavelengths first. In such manner, the algorithm solves RWA back and forth until a solution is constructed. The experimental results show that, compared with the other two methods, *GA_MEDP_RWA* can find the best solution among all testing instances. Although the proposed method takes longer computational time than the bin-packing methods for small instances (e.g., ten to twenty times that of bin-packing based methods on network CHNNET and NSF), the relative difference of computational time becomes smaller as the problem size grows.

6.2 Future research

In this dissertation, we have developed a genetic algorithm for solving the MEDP problem; and we have extended the proposed method to tackle the RWA problem. Both methods for solving the MEDP and RWA problems have demonstrated their effectiveness as shown in Chapters 4 and 5. There are some possible directions that may lead to the improvement of these methods. For *GA_MEDP*, although its application to MEDP has shown some promising results, applying additional features to the search process or trying different encoding schemes to enhance the solution quality and efficiency may be worthwhile for investigation. On the other hand, it would be of high interest to explore potential advantages of employing other metaheuristics such as electromagnetism-like mechanism (EM) method, particle swarm optimization (PSO) and artificial bee colony (ABC) for solving MEDP problems.

For the RWA problem, the performance of the proposed *GA_MEDP_RWA* has been verified by several experiments on realistic network topologies. The algorithm is able to solve small and medium size instances in reasonable time. However, for very large instances (for example, $|T| > 1{,}000{,}000$), finding a solution is extremely time-consuming. A divide-and-conquer approach called the multilevel algorithm may be worth studying to enhance the proposed method for solving large-size problems. Finding different ways to identify good permutations of the request set and developing local search methods are also needed to further improve the algorithm.

Further studies on extending the proposed GA to tackle other generalizations of MEDP, for example, the unsplittable flow (UF) problem and the call admission control problem, are also interesting research topics.

References

[1] A. Baveja and A. Srinivasan. *Approximation algorithms for disjoint paths and related routing and packing problems.* Mathematics of Operations Research, 25(2): 255-280, 2000.

[2] A. E. Ozdaglar and D. P. Bertsekas. *Routing and wavelength assignment in optical networks*. IEEE/ACM Transactions on Networking, 11(2): 259-272, 2003.

[3] A. Frank. *Packing paths, circuits, and cuts–a survey*. In B. Korte, L. Lovász,H.J. Prömel and A. Schrijver, editors, Paths, Flows, and VLSI-Layout, Springer-Verlag, Berlin, 47–100, 1990.

[4] Ali Hassan. *Particle swarm optimization for routing and wavelength assignment in next generation WDM networks*. PhD thesis, Department of Electronics Engineering, Queen Mary University of London, 2010.

[5] A. R. Sharafat and O. R. Ma'rouzi. *The most congested cutest: Deriving a tight lower bound for the chromatic number in the RWA problem.* IEEE Communication Letters, 8(7): 473-475, 2004.

[6] A. Schrijver. *Combinatorial Optimization: Polyhedral and Efficiency*. Springer – Verlag, Berlin, 2003.

[7] A. X. Martins, C. Duhamel, P. Mahey, R. R. Saldanha and M. C. de Souza. *Variable neighborhood descent with iterated local search for routing and wavelength assignment*. Computers and Operations Research, 39(9): 2133-2141, 2012.

[8] C. Chekuri and S. Khanna. *Edge disjoint paths revisited*. In Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'03), 628–637, 2003.

[9] D. Banerjee and B. Mukherjee. *A practical approach for routing and wavelength assignment in large wavelength-routed optical networks.* IEEE Journal on Selected Areas in Communications, 14(5): 903-908, 1996.

[10] G. Li and R. Shmha. *The partition coloring problem and its application to wavelength routing and assignment.* In Proceedings of the First Workshop on Optical Networks, 2000.

[11] H. Choo and V. V. Shakhov. *Routing and wavelength assignment in optical WDM networks with maximum quantity of edge disjoint paths*. Photonic Network Communications, 12: 145-152, 2006.

[12] H. Zang, J. P. Jue and B. Mukherjee. *A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks.* Optical Networks Magazine, 1: 47-60, 2000.

[13] I. Chlamtac, A. Ganz and G. Karmi. *Lightpath communications: an approach to high bandwidth optical WAN's*. IEEE Transactions on Communications, 40(7): 1171-1182, 1992.

[14] J. Kleinberg. *Approximation Algorithms for Disjoint Paths Problems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1996.

[15] J. S. Choi, N. Golmie, F. Lapeyrere, F. Mouveaux and D. Su. *A functional classification of routing and wavelength assignment schemes in DWDM networks: Static Case*. In Proceedings of the 7th International Conference on Optical Communications and Networks, 1109-1115, 2000.

[16] J. Vygen. *Disjoint paths*. Technical Report 94816. Research Institute for Discrete Mathematics, University of Bonn, February 1994.

[17] J. Zheng, H. T. Mouftah. *Optical WDM networks: concepts and design principles*, Wiley-IEEE Press, August 2004.

[18] K. E. Parsopoulos and M. N. Vrahatis. *Particle Swarm Optimization and Intelligence: Advances and Applications*. ISBN 1615206671, 9781615206674. IGI Global Snippet, 2009.

[19] K. Menger. *Zur allgemeinen kurventheorie*. Fund. Math, 10: 96–115, 1927.

[20] K. Varadarajan and G. Venkataraman. *Graph decomposition and a greedy algorithm for edge-disjoint paths*. In Proceedings of the 15th Annual ACM–SIAM Symposiumon Discrete Algorithms (SODA'04), 379–380, 2004.

[21] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italie, 1992.

[22] M. Gen, R. Cheng and L. Lin. *Network Models and Optimization*, Springer, 2008.

[23] M. J. Blesa and C. Blum. *Findinge edge-disjoint paths in networks: an ant colony optimization algorithm*. Journal of Mathematical Modeling and Algorithms, 6: 361-391, 2007.

[24] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to Theory of NP-completeness*. W.H. Freeman and Company, New York-San Francisco, 1979.

[25] N. Robertson and P. D. Seymour. *Graph Minors XIII*: *The disjoint paths problem*. Journal of Combinatorial Theory B, 63: 65-110,1995.

[26] N. Skorin-Kapov. *Routing and wavelength assignment in optical networks using bin packing based algorithms.* European Journal of Operational Research, 177: 1167-1179, 2007.

[27] P. Manohar, D. Manjunath and R. K. Shevgaonkar, *Routing and wavelength assignment in optical networks from edge disjoint path algorithms*. IEEE Communications Letters, 6(5): 211-213, 2002.

[28] R. Poli, J. Kennedy and T. Blackwell, *Particle swarm optimization - An overview*. Swarm intelligence, 1(1): 33-57, 2007.

[29] S. Fortune, J. Hopcroft and J. Wyllie. *The directed subgraph homeomorphism problem*. Theoretical Computer Science, 10(2): 111-121, 1980.

[30] S. G. Kolliopoulos. *Edge-disjoint paths and unsplittable flow*. In Handbook of Approximation Algorithms and Metaheuristics, ed. T. F. Gonzalez, Chapman & Hall / CRC, 2007.

[31] S. G. Kolliopoulos and C. Stein. *Approximating disjoint-path problems using greedy algorithms and packing integer programs.* In Proceedings of 6th integer programming and Combinatorial Optimization Conference (IPCO VI), LNCS 1412: 153-168, 1998.

[32] S. Sakai, M. Togasaki and K. Yamazaki. *A note on greedy algorithms for the maximum weighted independent set problem*. Discrete Applied Mathematics, 126: 313-322, 2003.

[33] T. Erlebach. *Approximation algorithms for edge-disjoint paths and unsplittable flow*. Lecture Notes in Computer Science, 3484: 97-137, 2006.

[34] T. F. Noronha, M. G. C. Resende and C. C. Ribeiro. *A biased random-key genetic algorithm for routing and wavelength assignment*. Journal of Global Optimization, 50(3): 503-518, 2011.

[35] T. F. Noraha and C. C. Ribeiro. *Routing and wavelength assignment by partition colouring*. European Journal of Operational Research, 171(3): 797-810, 2006.

[36] T. Fischer, K. Bauer, P. Merz and K. Bauer. *Solving the routing and wavelength assignment problem with a multilevel distributed memetic algorithm*. Memetic Computing, 1(2): 101-123, 2009.

[37] U. Adamy, T. Erlebach, D. Mitsche, I. Schurr, B. Speckmann, and E. Welzl. *Off-line admission control for advance reservations in star networks*. In 2nd Workshop on Approximation and Online Algorithms, LNCS 3351: 211-224, 2004.

[38] U. Adamy, C. Ambuehl, R. S. An, and T. Erlebach. *Call control in rings.* In Proceedings of the 29th International Colloquium on Automata, Languages and Programming ICALP 2002, LNCS 2380: 788-799, 2002.

[39] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. *Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems*. Journal of Computer and System Sciences, 67(3): 473-496, 2003.

[40] W. F. Abd-El-Wahed, A. A. Mousa, and M. A. El-Shorbagy. *Integrating particle swarm optimization with genetic algorithms for solving nolinear optimization problems*. Journal of Computational and Applied Mathematics, 235(5): 1446-1453, 2011.

[41] W. J. Yoon, D. H. Kim, M. Y. Chung, T. J. Lee and H. Choo. *Routing with maximum EDPs and wavelength assignment with path conflict graphs*. In Proceedings of the 2006 international conference on Computational Science and Its Applications, Vol II: 856-865, 2006.

[42] W. T. Chan, F. Y. L. Chin and H. F. Ting. *Escaping a grid by edge-disjoint paths*. Algorithmica, 36 (4): 343-359, 2003.

[43] X. Guan, S. Guo, W. Gong and C. Qiao. *A new method for solving routing and wavelength assignment problems in optical networks*. Journal of Lightwave Technology, 25(8): 1895-1909, 2007.

[44] Y. S. Kavian, A. Rashedi, A. Mahani and Z. Ghassemlooy. *Routing and wavelength assignment in optical networks using artificial bee colony algorithm*. Optik - International Journal for Light and Electron Optics, In Press, Corrected Proof, available online 31 May 2012.

[45] Y. Wang, T. H. Cheng and M. H. Lim. *A Tabu search algorithm for static routing and wavelength assignment problem*. IEEE Communications Letters, 9(9): 841-843, 2005.

[46] P. Leesutthipornchai, C. Charnsripinyo and N. Wattanapongsakorn. *Solving multi-objective routing and wavelength assignment in WDM network using hybrid evolutionary computation approach*. Computer Communications, 33(18): 2246-2259, 2010.