A comparative study on evolutionary algorithms for the agent routing problem in multi-point dynamic task

Sai Lu

School of Automation,
Beijing Institute of Technology,
Beijing, China
Email: 2220170505@bit.edu.cn

Bin Xin*

Key Laboratory of Intelligent Control and
Decision of Complex Systems,
Beijing Institute of Technology,
Beijing, China
Email: brucebin@bit.edu.cn
*Corresponding author

Lihua Dou

Beijing Advanced Innovation Center for
Intelligent Robots and Systems,
Beijing Institute of Technology,
Beijing, China
Email: doulihua@bit.edu.cn

Ling Wang

Department of Automation,
Tsinghua University,
Beijing, China
Email: wangling@tsinghua.edu.cn

Abstract: The agent routing problem in multi-point dynamic task (ARP-MPDT) proposed recently is a novel permutation optimisation problem. In ARP-MPDT, a number of task points are located at different places and their states change over time. The agent must go to the task points in turn to execute the tasks, and the execution time of each task depends on the task state. The optimisation objective is to minimise the time for the agent to complete all the tasks. In this paper, five evolutionary algorithms are redesigned and tried to solve this problem, including a permutation genetic algorithm (GA), a variant of the particle swarm optimisation (PSO) and three variants of the estimation of distribution algorithm (EDA). In particular, a dual-model EDA (DM-EDA) employing two probability models was proposed. Finally, comparative tests confirm that the DM-EDA has a stronger adaptability than the other algorithms though GA performs better for the large-scale instances.

Keywords: multi-point dynamic task; estimation of distribution algorithm; EDA; dual-model.

Reference to this paper should be made as follows: Lu, S., Xin, B., Dou, L. and Wang, L. (2020) 'A comparative study on evolutionary algorithms for the agent routing problem in multi-point dynamic task', *Int. J. Automation and Control*, Vol. 14, Nos. 5/6, pp.571–592.

Biographical notes: Sai Lu received his BS from the Harbin Engineering University, Harbin, China in 2017, and he is currently pursuing his MAEng in Beijing Institute of Technology, Beijing, China. His current research interests include manufacturing scheduling and intelligent optimisation methods.

Bin Xin received his BS in Information Engineering and PhD in Control Science and Engineering, both from the Beijing Institute of Technology, Beijing, China in 2004 and 2012, respectively. He was an Academic Visitor at the Decision and Cognitive Sciences Research Centre, University of Manchester from 2011 to 2012. He is currently a Professor with the School of Automation, Beijing Institute of Technology. His current research interests include search and optimisation, evolutionary computation, combinatorial optimisation and multi-agent systems. He is an Associate Editor of the *Journal of Advanced Computational Intelligence and Intelligent Informatics* and the journal *Unmanned Systems*.

Lihua Dou received her BS, MS and PhD in Control Theory and Control Engineering from the Beijing Institute of Technology, Beijing, China in 1979, 1987 and 2001, respectively. She is currently a Professor at the Control Science and Engineering at State Key Laboratory of Intelligent Control and Decision of Complex Systems, School of Automation, Beijing Institute of Technology. Her research interests include multi-objective optimisation and decision, pattern recognition and image processing.

Ling Wang received his BSc and PhD from the Department of Automation, Tsinghua University, China in 1995 and 1999, respectively. Now, he is a Full Professor at the Department of Automation, Tsinghua University. His research interests include theories and algorithms for intelligent optimisation and scheduling. He has authored five academic books and over 200 refereed papers. He was a recipient of the Outstanding Paper Award at the International Conference on Machine Learning and Cybernetics in 2002, Best Paper Award at International Conference on Intelligent Computing in 2011, Top Cited Article Award by Engineering Applications of Artificial Intelligence (Elsevier), National Natural Science Award (second place) in 2014, Science and Technology Award of Beijing City in 2008, and Natural Science Award (first place in 2003 and second place in 2007) nominated by the Ministry of Education (MOE) of China.

This paper is a revised and expanded version of a paper entitled 'A comparative study on evolutionary algorithms for the agent routing problem in multi-point dynamic task' presented at Intelligent Simulation Optimization and Scheduling, Changsha, China, 14–16 September 2018.

1 Introduction

The agent routing problem in the multi-point dynamic task (ARP-MPDT) is a novel permutation combinatorial optimisation problem (Lu et al., 2018). In this problem, the agent visits all task points distributed at different locations only once and completes each task with dynamic state. When the state of each task changes over time, the time cost for the agent to finish the task will change dynamically. The agent spends time on moving between two task points and executing each task. We call the two parts as the travelling time and the execution time, respectively. The objective is to find an agent's route to minimise the total time consisting of the travelling time and the execution time. If the execution time is neglected, we can find that ART-MPDT will degenerate into the travelling salesman problem (TSP) (Chen and Chen, 2011; Tang et al., 2013). Therefore, TSP can be considered as a special case of ARP-MPDT. As a NP-hard problem, ARP-MPDT has been rarely researched. However, many real-world tasks can be abstracted as ARP-MPDT such as fire fighting, disaster relief and large area searching.

Different evolutionary algorithms are tried and designed to solve this problem including estimation of distribution algorithm (EDA), genetic algorithm (GA) and particle swarm optimisation (PSO) algorithm. Evolutionary algorithms have been widely applied in different areas (Panigrahi et al., 2007; Bhaduri and Banerjee, 2011; Abderrezek et al., 2019; Chaudhary et al., 2015). Currently, in order to effectively solve nonlinear equations system (NES), a general framework based on the dynamic repulsion technique and evolutionary algorithms was proposed (Gong et al., 2018; Liao et al., 2018). As a rising evolutionary algorithm, EDA is an effective optimisation method to solve combinatorial optimisation problems (Lozano et al., 2006; Wang and Fang, 2012; Wang et al., 2013). The node histogram matrix (NHM) was introduced and it is suitable to solve the quadratic assignment problem (QAP) and the linear ordering problem (LOP) very well (Ceberio et al., 2012). The edge histogram matrix (EHM) was also introduced, and it matches the TSP and the flow shop scheduling problem (FSSP) better than NHM. In this paper, in view of the coexistence of execution time and travelling time in ARP-MPDT, the EDA employing both NHM and EHM (DM-EDA) is proposed to solve the agent routing problem (Zhong et al., 2010). The selection ratio of the two models is regulated dynamically. In GA, an adaptive crossover operator and a mutation operator are designed to adjust the crossover probability and the mutation probability of each chromosome, respectively. The random key technique in continuous PSO is applied to transform the continuous search space to a permutation-based space (Baioletti et al., 2017).

In this paper, the problem was described in detail firstly. Then, the five algorithms were successively designed to solve this problem, and the data results were comprehensively analysed. Finally, the stronger adaptability of DM-EDA than the other algorithms was confirmed.

2 Problem description

In order to simplify the model of the problem and highlight the research focus, the actual task model will be simplified and the key parameters are extracted. So, before the models are established, the following assumptions are made:

a The agent moves at a constant speed on a two-dimensional plane where all task points are located.

b The agent moves between any two task points along a straight line.

c The agent has a constant capacity and is able to complete all tasks.

d The agent leaves from its initial position at $t = 0$ to execute its first task.

e After reaching a task point, the agent will reduce the state of the current task to a threshold which means the task is completed.

f Before the current task is completed, the agent is not allowed to go to another task point.

2.1 Problem model

The APP-MPDT can be described as follows. There are multiple tasks which have different initial states to be carried out in different locations. The agent must go to each task point to perform the task. The state of each task changes over time, and its dynamic model can be defined by two parameters: the initial state and the state growth index. After the agent starts to execute the task, the state of this task will decrease because the growth index of the state is changed by the agent. When the state of the task is lower than a threshold, the agent completes this task and leaves for another task point. The agent needs to find a task execution sequence to minimise the total time of completing all tasks.

Table 1 Notations of the model parameters

Notation	*Implication*
V	The speed of the agent
β	The capability parameter of the agent
$P_0 = (x_0, y_0)$	The initial position of the agent
n	The number of tasks
$S_i(0)$	The initial state of task point i
α	The state growth index of task i
$P_i = (x_i, y_i)$	The position of task point i
t_i^r	The time when the agent reaches task point i
t_i^l	The time when the agent leaves task point i
Δt_i	The execution time of the agent at task point i
ξ	The state threshold indicating task completion

In this paper, the dynamic model of task i is defined as an exponential function which is determined by S_i, α_i and β. The formula is as follows:

$$S_i(t) = \begin{cases} S_i(0) \cdot e^{\alpha_i \cdot t} & t \le t_i^r \\ S_i\left(t_i^r\right) \cdot e^{(\alpha_i - \beta) \cdot \left(t - t_i^r\right)} & t > t_i^r \end{cases} \tag{1}$$

where i = 1, 2, …, n. According to this dynamic model, the execution time Δt_i can be calculated according to the following formula as long as t_i^r is given.

$$\Delta t_i = \frac{\ln(\xi) - \ln\left(S_i\left(t_i^r\right)\right)}{\alpha_i - \beta} \tag{2}$$

The distance between P_j and P_k is $D_{j,k}$, and it can be calculated according to the formula:

$$D_{j,k} = \begin{cases} \sqrt{\left(x_j - x_k\right)^2 + \left(y_j - y_k\right)^2} & j \neq k \\ \infty & j = k \end{cases} \tag{3}$$

where j, k = 0, 1, 2, …, n. The solution of this problem can be represented as a permutation. The search space can be denoted as follows:

$$\mathbf{z} = \left\{z(1), z(2), \ldots, z(n) \mid z(i) \in \{1, 2, \ldots, n\}, z(i) \neq z(j), \forall i \neq j\right\}. \tag{4}$$

The schedule of the agent to perform all tasks can be expressed as the following form:

$$T = \left\{\left(t_{z(1)}^r, t_{z(1)}^l\right), \left(t_{z(2)}^r, t_{z(2)}^l\right), \ldots, \left(t_{z(n)}^r, t_{z(n)}^l\right)\right\}. \tag{5}$$

Given the task access sequence, the schedule of the agent can be calculated one by one from the first variable in T according to the following formulas:

$$t_{z(1)}^r = \frac{D_{0,z(1)}}{V} \tag{6}$$

$$t_{z(i)}^l = t_{z(i)}^r + \Delta t_{z(i)} \quad i = 1, 2, \ldots, n \tag{7}$$

$$t_{z(i+1)}^r = t_{z(i)}^l + \frac{D_{z(i),z(i+1)}}{V} \quad i = 1, 2, \ldots, n-1. \tag{8}$$

The objective of this problem is to minimise the total time of executing all tasks. In the other words, the time to leave the last task point needs to be the shortest. The optimisation model of ARP-MPDT is given as follows:

$$\min f(\mathbf{Z}) = t_{z(n)}^l \tag{9}$$

subject to:

$$\alpha_i \leq \beta \quad \forall i = 1, 2, \ldots, n \tag{10}$$

$$S_i\left(t_i^l\right) \leq \xi \quad \forall i = 1, 2, \ldots, n \tag{11}$$

$$t_{z(i)}^r \leq t_{z(i)}^l \leq t_{z(i+1)}^r \leq t_{z(i+1)}^l \quad i = 1, 2, \ldots, n-1 \tag{12}$$

where equation (9) defines the objective function and $t_{z(n)}^l$ can be calculated by equations (6), (7) and (8). Equation (10) guarantees that each task can be completed by the agent, and the formula constrains the existence of the solution. Equation (11) means that the agent is not allowed to leave for another point until the current task is finished and the states of task points can be calculated successively by equations (1) and (6).

Equation (12) is the basic constraint of the schedule which ensures the validity of the solution.

3 EDAs for ARP-MPDT

3.1 Overview of EDA

EDA is an effective optimisation method to solve permutation optimisation problems (Zhou and Sun, 2007). Its basic procedure is as follows:

Step 1 Initialise a population and the probability model randomly.

Step 2 Calculate the fitness value of each individual.

Step 3 Select individuals (samples) into dominant group according to the selection strategy.

Step 4 Update the probability model using the samples.

Step 5 Generate new individuals according to the sampling strategy.

Step 6 Judge the termination condition. If the termination condition is satisfied, stop the algorithm and output results. Otherwise, go to Step 2.

EDA searches in the solution space by sampling the probability model which expresses the statistical information of the solutions which have better fitness values. So, it is vital to select a suitable probability model of EDA to reflect the problem feature. In EDAs for the permutation optimisation problem, NHM or EHM is frequently-used to collect different statistical information of dominant solutions.

In ARP-MPDT, all of the task points and the available paths between any two task points constitute a graph. Each task access sequence corresponds to the group which consists all the task points and the directed paths from the first task point to the last task point in graph. NHM reflects the absolute position information of task points in the graph, and EHM reflects the statistics of the directed paths in the graph.

In ARP-MPDT, the total time consists the execution time of each task and the travelling time. The execution time depends on the state information of the task and t_i^r, and the travelling time relies on the length of each path. Therefore, it is unreasonable to choose one from NHM and EHM without prior knowledge. For example, when the execution time tasks up most of the total time in an ARP-MPDT, the agent's travelling time is not a primary element. It is better to select NHM as the probability model of EDA to collect the information of individuals in dominant group. Conversely, EHM is suitable for the instance when the travelling time tasks up most of the total time.

In this paper, three variants of EDA with different probability models are designed to solve ARP-MPDT. In EDA1 and EDA2, NHM and EHM are used respectively to describe the feature of this problem and generate new solutions by sampling. Both NHM and EHM are employed in EDA3. All of these algorithms select samples by the truncation selection and initialise the populations by sampling initial probability models. Because of the difference of the probability models in the three EDAs, the methods of updating and sampling probability models are different.

The methods of updating and sampling probability models in EDAs are shown in the following sections.

The notations of EDAs are explained as follows:

G_{NHM} node histogram matrix

G_{EHM} edge histogram matrix

λ the selection ratio coefficient of NHM to EHM

Z population containing individuals generated by using NHM and EHM

N the population size

N_Z the number of individuals generated by the probability models

η proportion of samples to population

Z_b the group of samples for updating NHM and EHM

N_b the number of individuals in Z_b

Z_e the group in which individuals are generated using EHM

Z_n the group in which individuals are generated using NHM

$g_{\max}$ the maximum number of iterations

g the identifier of iteration.

3.2 EDA1

3.2.1 Initialisation

In this paper, the initial G_{NHM} is related to the state growth index of each task. It can be initialised as follows:

$$G_{NHM} = \left[G_{NHM}(i, j)\right]_{n\times n} = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \end{bmatrix}. \tag{13}$$

3.2.2 Updating NHM

It is necessary to introduce F_n as the frequency matrix to save frequency data of Z_b. F_n can be initialised as follows:

$$F_n = \left[F_n(i, j)\right]_{n\times n} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{14}$$

$I_n^k(i, j)$ is the indicator function corresponding to the k^{th} individual in Z_b:

$$I_n^k(i,\ j)=\begin{cases}1, \text{if task } i \text{ is the } j^{\text{th}} \text{ task finished in the } k^{\text{th}} \text{ individual in } Z_b\\ 0, \text{otherwise}\end{cases}. \tag{15}$$

The procedure of collecting information is showed in equation (16):

$$F_n(i,\ j)=\sum_{k=1}^{N_b} I_n^k(i,\ j). \tag{16}$$

The formula of updating G_{NHM} is as follows:

$$G_{NHM}(i,\ j)=(1-\mu_n)\cdot G_{NHM}(i,\ j)+\mu_n\cdot\frac{F_n(i,\ j)}{N_b} \tag{17}$$

where $\mu_n \in (0,\ 1)$ denotes the learning rate and $i, j = 1, 2, \ldots, n$.

3.2.3 Sampling method

The access sequence of each individual is obtained by the roulette wheel. The procedure of generating a new individual based on G_{NHM} is as follows:

Step 1 Set $i = 1$.

Step 2 Select task point k according to the elements in the line i of G_{NHM} by the roulette wheel. Let $\mathbf{z}(i) = k$.

Step 3 Set all the elements of the column k in G_{NHM} to zero.

Step 4 Let $i = i + 1$ and repeat Steps 2 and 3 until all of the task points are selected.

3.3 EDA2

3.3.1 Initialisation

In this case, the travelling time dominates the total time and EHM suits this case. G_{EHM} can be initialised as follows:

$$G_{EHM}=\left[G_{EHM}(i,\ j)\right]_{n\times n}=\frac{1}{D_{i,j}}. \tag{18}$$

3.3.2 Updating EHM

Introduce F_e as the frequency matrix to save the frequency information of Z_b. F_e can be initialised as follows:

$$F_e=\left[F_e(i,\ j)\right]_{n\times n}=\begin{bmatrix}0 & 0 & \cdots & 0\\ 0 & 0 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 0\end{bmatrix}. \tag{19}$$

$I_e^k(i,\ j)$ is the indicator function corresponding to the k^{th} individual in Z_b:

$$I_e^k(i,\ j)=\begin{cases}1, \text{ if task } i \text{ is before task } j \text{ in the } k^{\text{th}} \text{ individual in } Z_b\\ 0, \text{ otherwise}\end{cases}. \tag{20}$$

The procedure of collecting information is showed in equation (21):

$$F_e(i,\ j)=\sum_{k=1}^{N_b}\left\{I_e^k(i,\ j)+I_e^k(j,i)\right\}. \tag{21}$$

The formula of updating G_{EHM} is as follows:

$$G_{EHM}(i,\ j)=(1-\mu_e)\cdot G_{EHM}(i,\ j)+\mu_e\cdot\frac{F_e(i,\ j)}{N_b} \tag{22}$$

where $\mu_e \in (0,\ 1)$ denotes the learning rate and $i, j = 1, 2, \ldots, n$.

3.3.3 Sampling method

The roulette wheel is used to sample the probability model to get the access sequence of each individual. The procedure of generating a new individual based on G_{EHM} is as follows:

Step 1 Set $i = 1$.

Step 2 Select task point k according to the elements in the line i of G_{EHM} by the roulette. Let $\mathbf{z}(i) = k$.

Step 3 Set all the elements of the column k in G_{EHM} to zero.

Step 4 Let $i = k$ and repeat Steps 2 and 3 until all of the task points are selected.

3.4 EDA3

In EDA3, both NHM and EHM are employed to collect the information of individuals and generate new population in which the number of individuals is fixed. EDA3 uses the methods of updating and sampling in EDA1 and EDA2 for reference.

However, because of the coexistence of NHM and EHM, it is necessary to determine the number of individuals generated from NHM and EHM. Therefore, the coefficient $\lambda(g)$, $\lambda \in [0.05, 0.95]$ denotes the selection ratio of NHM to EHM in the g^{th} iteration. This updating method assumes that $\lambda(0) = 0.5$. It means that without the prior knowledge, the chance of selecting NHM and EHM is equal. The updating rule is showed as follows:

$$\lambda(g+1)=(1-\mu_\lambda)\cdot\lambda(g)+\mu_\lambda\cdot\frac{\dfrac{N_e}{\lambda(g)}}{\dfrac{N_n}{1-\lambda(g)}+\dfrac{N_e}{\lambda(g)}} \tag{23}$$

where μ_λ denotes the learning rate which is designed to avoid premature convergence. N_n is the number of individuals in Z_n, and N_e is the number of individuals in Z_e.

4 GA for ARP-MPDT

4.1 Overview of GA

GA is a random parallel search method which is based on natural selection and genetic inheritance (Zhou et al., 2018; Larrañaga et al., 1999). In GA, individuals (solutions) are represented by character strings or matrices which are often referred as chromosomes. GA relies on the loop of the selection operator, the crossover operator and the mutation operator to find the individual with the best genetic material from the search space.

The procedure of the basic GA is showed as following:

Step 1 Initialise the population of chromosomes.

Step 2 Evaluate the fitness of each chromosome.

Step 3 Select parents from the current population via the selection operator.

Step 4 Choose a pair of parents to execute the crossover operator to create offspring.

Step 5 Execute the mutation operator according to the mutation probability.

Step 6 Judge the termination condition. If the condition is satisfied, stop the algorithm and output the results. Otherwise, go to Step 2.

In this paper, the tournament selection operator is applied to select the population of parents. Through comparing the fitness values of two individuals selected randomly from the population, the better individual will be selected into the population of parents to generate new individuals.

4.2 Crossover operator

In order to increase the population diversity in early iteration of GA, the crossover operator should be carried out for more individuals to enhance the global searching ability. However, at the later stage of iteration, in order to keep the better individuals, the crossover probability should be reduced. Therefore, an adaptive adjustment strategy of the crossover operator of each chromosome is introduced. The i^{th} chromosome in the g^{th} iteration can be expressed as $S_i(g) = (S_{i1}(g), S_{i2}(g), \ldots, S_{in}(g))$.

We set the fitness function as $fi_i(g) = 1 / f(S_i(g))$, and the larger fitness means the better chromosome.

The crossover probability of each chromosome changes as follows:

$$Pc_i(g) = \begin{cases} Pc_{\max} - (Pc_{\max} - Pc_{\min}) \cdot \left(\dfrac{g}{2 \cdot g_{\max}} + \dfrac{fi_i(g) - ave\_fi(g)}{2 \cdot (\max\_fi_i(g) - ave\_fi(g))} \right), & fi_i(g) < ave\_fi(g) \\ Pc_{\max}, & fi_i(g) < ave\_fi(g) \end{cases} \tag{24}$$

where $Pc_i(g)$ is the crossover probability of the i^{th} chromosome, $Pc_{\max}$ and $Pc_{\min}$ are the maximum value and the minimum value of probabilities. $ave\_f_i(g)$ is the average value of all the fitness values and $\max\_f_i(g)$ is the maximum value of all the fitness in the g^{th} iteration. In this way, the crossover probability of each chromosome in the g^{th} iteration can be calculated.

Select two chromosomes $S_i(g)$, $S_j(g)$ randomly and judge whether the crossover operator will be carried out or not according to the crossover probabilities of both chromosomes.

The crossover operator can be described as follows:

Step 1 Select a continuous gene segment Ss_i randomly from $S_i(g)$.

Step 2 Replace the gene segment of $S_j(g)$ with Ss_i at the same position. The new chromosome completed the crossover operator is the new chromosome called $Sc_j(g)$.

Step 3 It is hard to ensure that $Sc_j(g)$ is a permutation. Therefore, it is necessary to adjust the genes of $Sc_j(g)$.

Step 3.1 Check each gene of $Sc_j(g)$ which is not in Ss_i whether is the same as the genes in Ss_i.

Step 3.2 If k^{th} gene conflict with the gene in Ss_i and the same gene is located at the p^{th} position in $S_i(g)$, replace k^{th} gene in $Sc_j(g)$ with the p^{th} gene in $S_j(g)$.

4.3 Mutation operator

An adaptive mutation operator is introduced to improve the population diversity. The population diversity will decrease with iteration. Therefore, the mutation probability of each increases with iteration. The mutation probability changes as follows:

$$Pm(g) = Pm_{\min} + (Pm_{\max} - Pm_{\min}) \cdot \frac{g}{g_{\max}} \tag{25}$$

where $Pm(g)$ is the mutation probability in the g^{th} iteration, $Pm_{\max}$ and $Pm_{\min}$ are the maximum value and the minimum value of probabilities, respectively. Each chromosome is determined whether to be mutated or not according to the mutation probability. Two genes in the chromosome will switch their positions according to the mutation operator.

5 PSO for ARP-MPDT

5.1 Overview of PSO

PSO was introduced by Kennedy and Eberhart (1995). PSO is a popular presently swarm inspired method in computational intelligence. The inspiration comes from the study of social behaviour of birds flocking. PSO regards each individual as a particle without mass and volume. These particles are located in the n-dimensional space and the positions of the particles are potential solutions to the problem. These particles move at a changing speed in search space to find the best solution. The speed of each particle will be adjusted adaptively through the motion experiences from itself and population. In this paper, a random key technique is applied to transform the search space of continuous PSO to permutation-based space of ARP-MPDT.

The position of particle i after the g^{th} iteration is $X_i(g) = (X_{i1}(g), X_{i2}(g), \ldots, X_{i3}(g))$, and the speed is $V_i(g) = (V_{i1}(g), V_{i2}(g), \ldots, V_{i3}(g))$. The particle i will update its speed and position according to the following formulas.

$$V_i(g+1) = \omega \times V_i(g) + c_1 \times r_1 \times \left(pbest_i - X_i(g)\right) + c_2 \times r_2 \times \left(gbest - X_i(g)\right) \tag{26}$$

$$X_i(g+1) = X_i(g) + V_i(g) \tag{27}$$

where ω is the inertia weight. c_1, c_2 are learning rates and r_1, r_2 are random numbers. $pbest_i$ is the best position currently found so far of particle i and $gbest$ is the best position currently found so far of all the particles. In PSO, c_1 and c_2 are important in searching.

The procedure of the basic PSO is shown as following:

Step 1 Initialise randomly the position and speed of each particle.

Step 2 Calculate the fitness value of each particle and find out $pbest_i$ and $gbest$.

Step 3 Update the speed and the position according to equations (26) and (27).

Step 4 Judge the termination condition. If the condition is satisfied, stop the algorithm and output the results. Otherwise, go to Step 2.

5.2 Random key

Initially, PSO was introduced for continuous search spaces. However, in this paper, the PSO algorithm is focused on finding the best permutation in discrete space. Therefore, random key decoder is introduced to support the application of PSO to permutation optimisation problem.

The random key creates a unidirectional mapping from the continuous space of PSO to the discrete solution space of ARP-MPDT. That means the solution to ARP-MPDT can be represented by the position of a particle of PSO.

Define the transformation from continuous space to permutation space as follows:

$$Z = R(X) \tag{28}$$

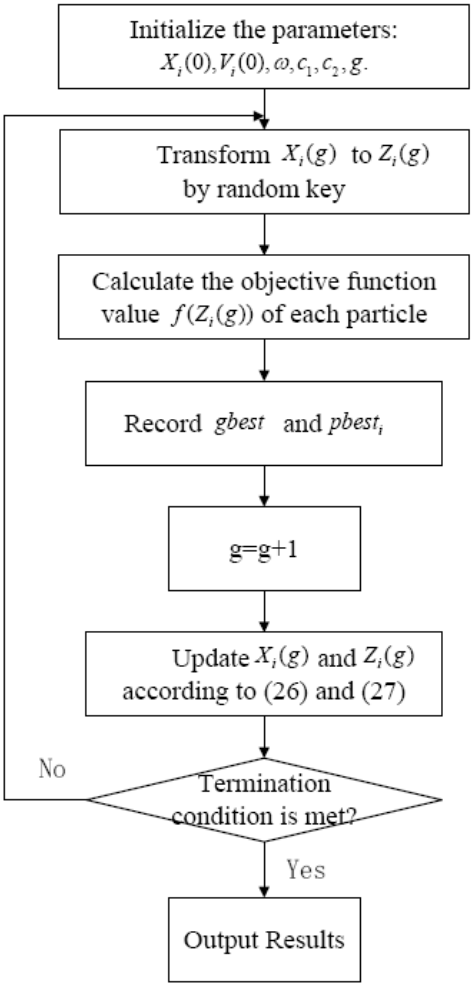
where Z is a solution to ARP-MPDT in discrete space and X is the position of a particle in continuous space. The transformation strategy of the random key will sort the elements in X according to the size of each element in ascending order. For example, if X_i = (1.2, 3.5, 0.9, 0.5, 2.8), after sorting, the new sequence will be (0.5, 0.9, 1.2, 2.8, 3.5). From this sequence, it is easy to find that 1.2 in X_i is at the third place in new sequence. Therefore, 3 corresponds to 1.2, and similarly 5 corresponds to 3.5. In this way, the permutation which reflects the position in new sequence of each elements in X_i is Z = (3, 5, 2, 1, 4).

5.3 PSO with random key

The flowchart of PSO algorithm employing the random key technique for ARP-MPDT is illustrated in Figure 1.

In this way, PSO algorithm will search the better positions in continuous space. These positions will be transformed to permutations to calculate the objective function values. Then, $pbest_i$ and $gbest$ can be found out via the objective function to update the positions and speeds of all the particles.

Figure 1 Flowchart of PSO for ARP-MPDT



6 Experimental studies

In the following section, we introduce the setup of the experiments and the analysis of the results. Table 2 shows the execution parameters set of the EDAs. Table 3 shows the execution parameters set of the GA. Table 4 shows the parameters set of PSO. In this paper, sensitivity analyses of key parameters are executed to enhance the performance in the following section, and the other parameters are set empirically. In order to ensure the fairness of design of experiments, the value of the common parameters in these algorithms is kept the same.

Eleven instances are randomly generated to compare the five algorithms. Each instance has different agent parameters and different task parameters. The agent parameters include the initial position, the capability parameter and the speed. The task

parameters consist of the number of the tasks, the position, the initial state, and the state growth index of each task. The data have been uploaded on https://github.com/oksaisai/IJAAC.

Table 2 Parameters setting of the EDAs

Parameter	*Value*	*Parameter*	*Value*
Population size	$10 \cdot n$	Learning rate μ	0.2
Selection type	Ranking selection method	Learning rate μ_λ in EDA3	0.2
Offspring size	$10 \cdot n$	Selection size	n
Stopping criterion	1,000		

Table 3 Parameters setting of the GA

Parameter	*Value*
Population size	$10 \cdot n$
Inertia weight	0.7
$Pc_{\max}$	0.9
$Pc_{\min}$	0.1
$Pm_{\max}$	0.5
$Pm_{\min}$	0

Table 4 Parameters setting of the PSO

Parameter	*Value*
Population size	$10 \cdot n$
Inertia weight	0.7
Learning rate c_1	3
Learning rate c_2	0.6

For each algorithm and problem instance, 20 runs have been completed. These tests were carried out on a computer configured as Inter(R) Xeon(R) CPU E5-2620 v4, 32 GB RAM, Windows 7 operation system.

6.1 Parametric sensitivity analysis

In this section, several tests are carried out to analyse the sensitivity of different algorithms to part of their key parameters. An instance where the number of tasks is 8 was tested by different algorithms with different parameters.

EDAs whose learning rate is set to the member in (0.1, 0.2, …, 0.9) successively are used to solve the instance. The optimisation results of EDAs are shown in Figure 2, Figure 3 and Figure 4. It is easy to find that a smaller learning rate of EDAs makes them perform better.

PSO using different c_1 and c_2 is applied to solving the instance. The results are shown as the three-dimensional graph in Figure 5. From Figure 5, we can find that most areas with better objective values are located at the place where c_1 is larger and c_2 is smaller; for instance, $c_1 = 3$ and $c_2 = 0.6$.

Figure 2 Sensitivity analysis in EDA1 (see online version for colours)

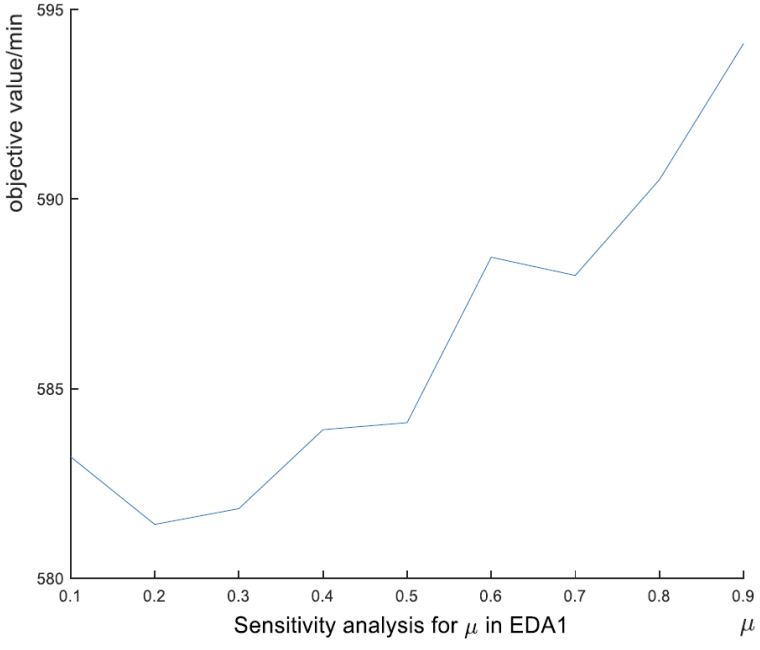


Figure 3 Sensitivity analysis in EDA2 (see online version for colours)

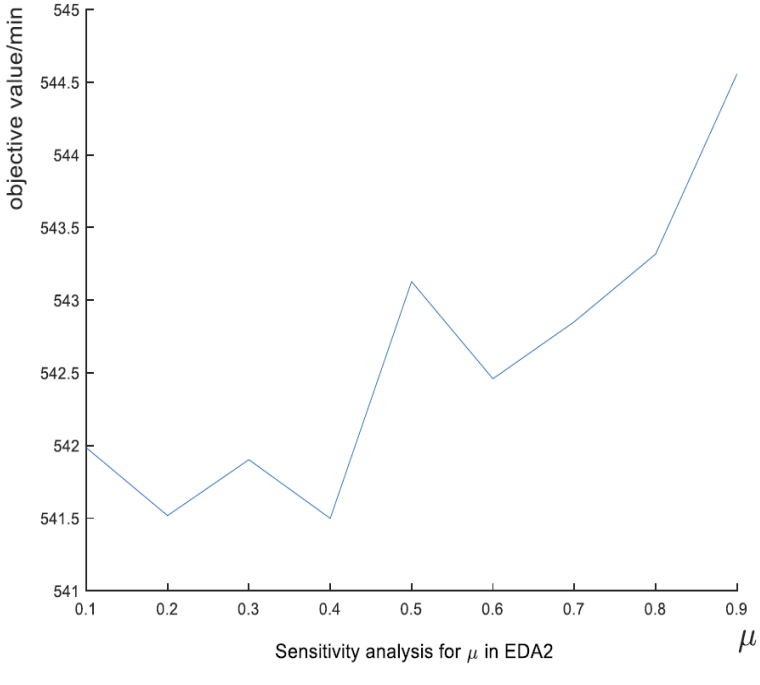


Figure 4 Sensitivity analysis in EDA3 (see online version for colours)

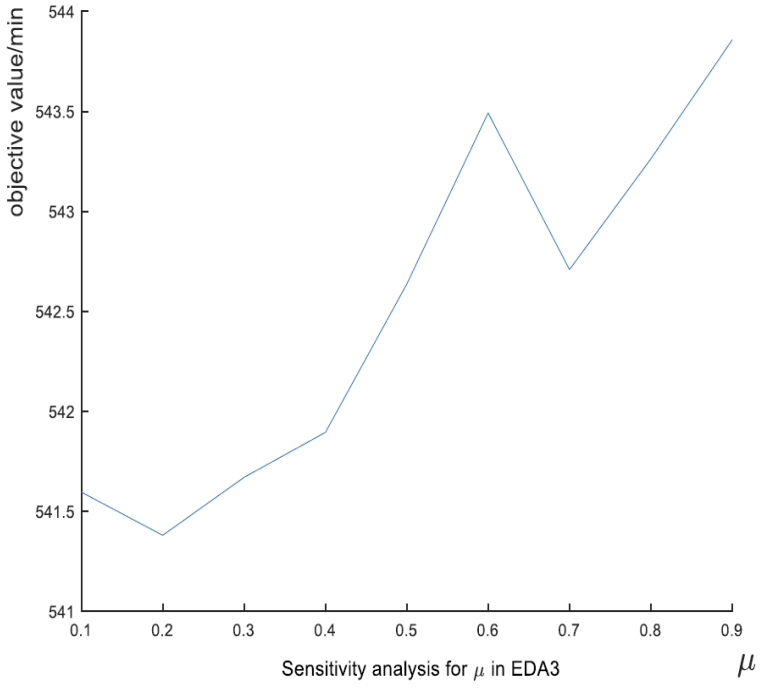
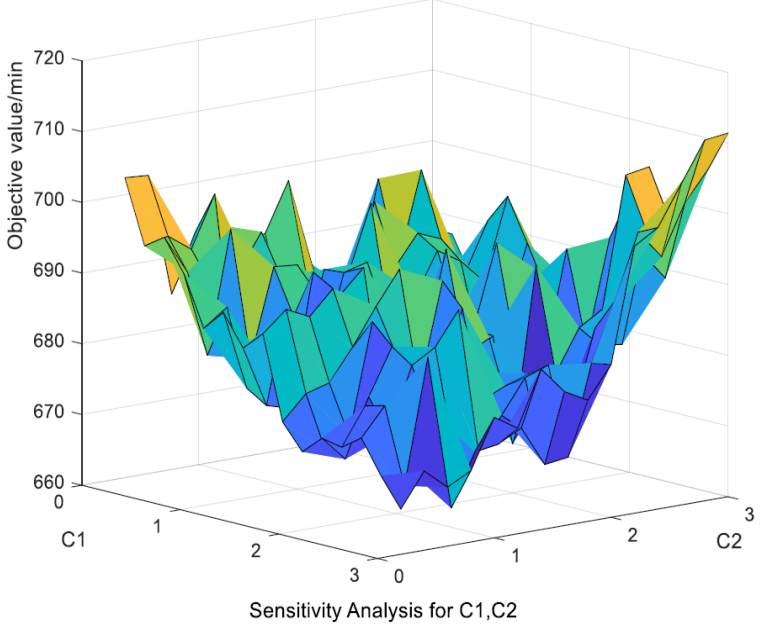


Figure 5 Sensitivity analysis in PSO (see online version for colours)



6.2 Comparative experiments

In this section, a large number of experiments are carried out. The five algorithms are applied to solving the eleven instances. Table 5 shows the average value and standard deviation of the objective value and running time of each experiment. The value before '±' of the first data in each unit is the average objective value and the value after '±' is the standard deviation. The value before '±' of the second data in each unit is the average time consumption and the value after '±' is the standard deviation.

In Table 5, from the data of EDAs, we can find that EDA1 performs better than EDA2 in the larger-scale tests. However, the performance of EDA1 in small-scale tests is worse than EDA2. EDA3 which employs the EHM and NHM has a better search performance. EDA3 can follow closely the best objective value of the other EDAs or performs better than them. The adaptability of the EDA employing single probability model for different instances is poor. From the above analysis, we know that the objective function value consists of the execution time and the travelling time. NHM and EHM are suitable for different instances. This is the reason why the optimisation effects of EDA1 and EDA2 for different instances are different. EDA3 with both of them can adapt to these instances well than the others.

Comparing EDAs, GA and PSO, we can find that for the small-scale instances, EDAs have better performance than the others. However, GA performs well in the large-scale tests. PSO has the worst performance in these algorithms. Compared with GA, EDAs rely on the probability model and the probability model needs a lot of the samples of the better solutions. So, the speed of the convergence will decline. In small-scale tests, the travelling time dominates the whole time, and that results in good effects of EDA2 which considers the edge information. As the scale of the problem increases, the task execution time gradually dominates the total time. So, the performance of EDA1 gradually improved than EDA2. Considering the two factors comprehensively, EDA3 showed strong adaptability in many instances. Because of the lack of mutation operation, it is difficult for EDAs to jump out of the local optimal solution in the large-scale tests. Compared to the other algorithms, PSO applying the random key technique performs worse because of the high inherent redundancy, though consuming shorter running time.

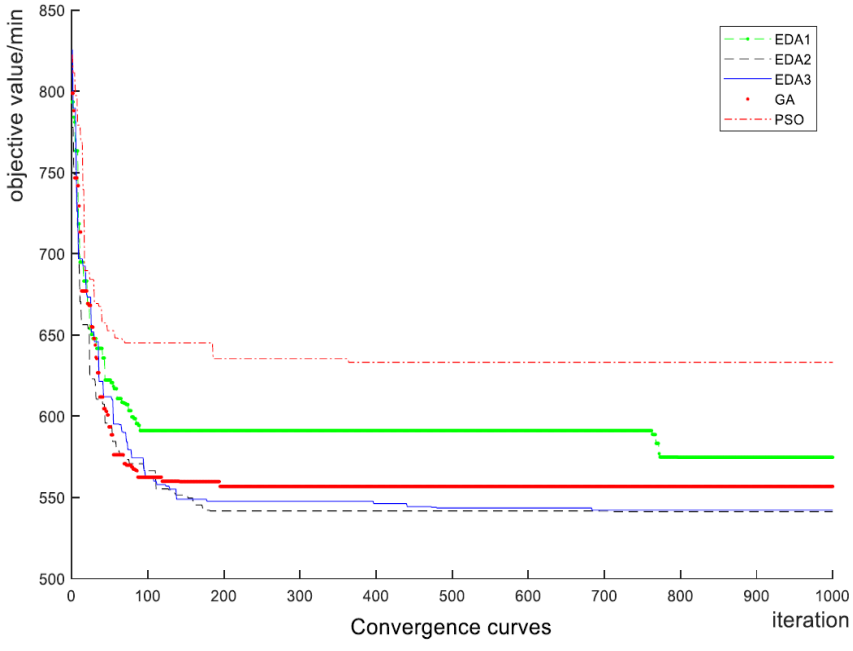
In order to observe the optimisation process and the dynamic changes of key parameters in the algorithms in more detail, the following graphs are shown. The convergence curves of objective value via different algorithms are shown in Figure 6. Comparing the two graphs, it is obvious that EDA2 whose effect is better in Figure 6(a) was unable to adapt to the instance in Figure 6(b). EDA3 and GA had better performances in different instance. In EDA3, an adaptive adjustment strategy is designed and the selection ratio λ is the key parameter. The change curve of λ is shown in Figure 7. We can find that λ tends to 1 which means EDA3 will use EHM more because the individuals generated from EHM have better objective values. In GA, the crossover probability is also adaptively adjusted and its change curve is shown in Figure 8. Overall, from these graphs, EDA3 showed great adaptability and GA had a faster convergence speed.

Table 5 Statistical results

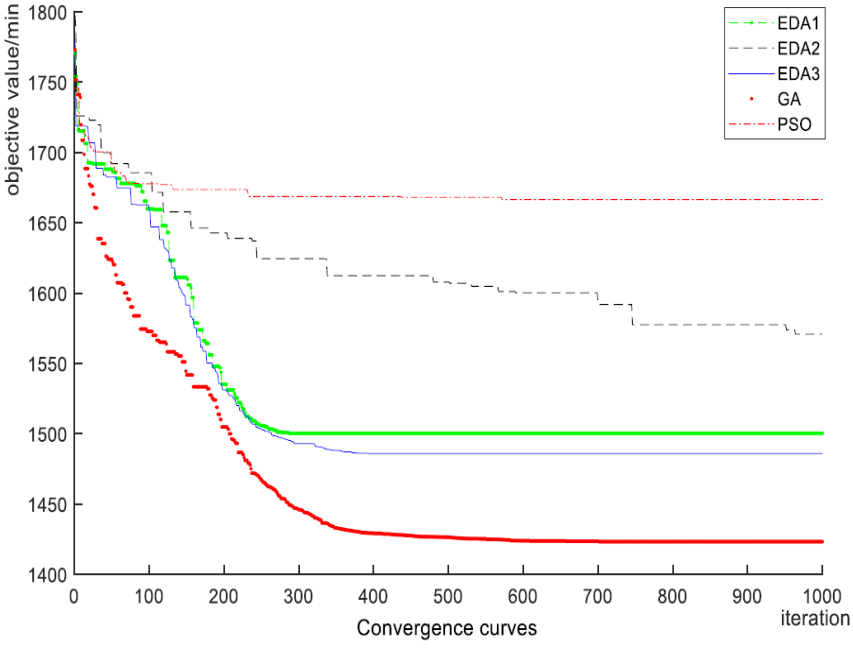
No.	*n*	*Ave ± Std*	*EDA1*	*EDA2*	*EDA3*	*GA*	*PSO*
1	4	Obj	*46.12 ± 0*	*46.12 ± 0*	*46.12 ± 0*	*46.12 ± 0*	*46.12 ± 0*
		Rt	2.82 ± 0.43	2.68 ± 0.15	2.72 ± 0.22	5.89 ± 0.24	2.47 ± 0.13
2	8	Obj	*97.09 ± 0*	*97.09 ± 0*	*97.09 ± 0*	98.05 ± 1.22	99.47 ± 1.42
		Rt	6.77 ± 0.31	7.23 ± 1.26	7.02 ± 0.14	12.74 ± 0.34	5.15 ± 0.75
3	30	Obj	582.63 ± 5.53	*541.67 ± 1.84*	542.01 ± 1.47	553.23 ± 5.00	668.38 ± 18.53
		Rt	56.20 ± 0.81	57.38 ± 0.76	56.22 ± 1.14	71.73 ± 6.13	26.93 ± 0.37
4	30	Obj	1,813.6 ± 18.08	*1,708.3 ± 1.87*	1,710.1 ± 1.53	1,735.3 ± 16.01	2,100.6 ± 48.50
		Rt	59.6 ± 0.92	59.2 ± 1.41	60.7 ± 1.14	70.2 ± 2.61	27.1 ± 0.40
5	50	Obj	544.27 ± 2.41	*533.89 ± 2.25*	541.26 ± 4.79	535.9 ± 2.65	603.02 ± 8.12
		Rt	156.63 ± 7.66	158.18 ± 5.79	155.78 ± 5.51	189.28 ± 6.26	60.22 ± 0.98
6	50	Obj	534.81 ± 6.21	*477.70 ± 1.93*	477.93 ± 3.12	496.90 ± 7.45	641.59 ± 8.98
		Rt	155.33 ± 3.32	155.36 ± 3.38	155.73 ± 3.00	182.14 ± 5.76	60.3 ± 0.54
7	100	Obj	1,493.5 ± 6.54	1,568.5 ± 10.18	1,486.47 ± 7.56	*1,427.8 ± 3.18*	1,655.0 ± 24.45
		Rt	609.6 ± 5.95	6,623.5 ± 10.21	619.4 ± 42.59	564.9 ± 15.55	197.5 ± 1.20
8	100	Obj	980.83 ± 4.20	1,001.3 ± 4.92	964.81 ± 3.94	*931.97 ± 3.42*	1,091.3 ± 11.32
		Rt	662.69 ± 7.75	646.0 ± 7.88	654.15 ± 7.33	558.13 ± 15.97	191.6 ± 5.22
9	150	Obj	1,737.0 ± 9.96	1,877.2 ± 13.87	1,723.5 ± 11.92	*1,613.4 ± 4.50*	2,038.8 ± 26.67
		Rt	1,588.3 ± 35.56	1,557.8 ± 33.88	1,599.0 ± 37.53	1,373.8 ± 98.44	422.8 ± 2.45
10	150	Obj	1,697.8 ± 8.09	1,842.4 ± 12.23	1,695.4 ± 11.72	*1,588.6 ± 5.03*	1,966.8 ± 27.68
		Rt	1,561.5 ± 21.67	1,543.4 ± 22.56	1,548.6 ± 20.65	1,314.0 ± 96.37	423.1 ± 3.60
11	250	Obj	4,550.5 ± 22.21	5,360.5 ± 31.43	4,537.1 ± 29.30	*4,148.9 ± 13.88*	5,664.7 ± 101.9
		Rt	5,001.4 ± 111.93	5,163.8 ± 95.34	5,083.7 ± 100.28	3,583.9 ± 237.66	1,213.1 ± 104.5

Note: Ave: average value, Std: standard deviation, Obj: objective value (min.) and Rt: runtime (sec).

Figure 6 Convergence curves in different instances, (a) $n = 30$ (b) $n = 100$ (see online version for colours)



(a)



(b)

Figure 7 The curve of λ in EDA3 (see online version for colours)

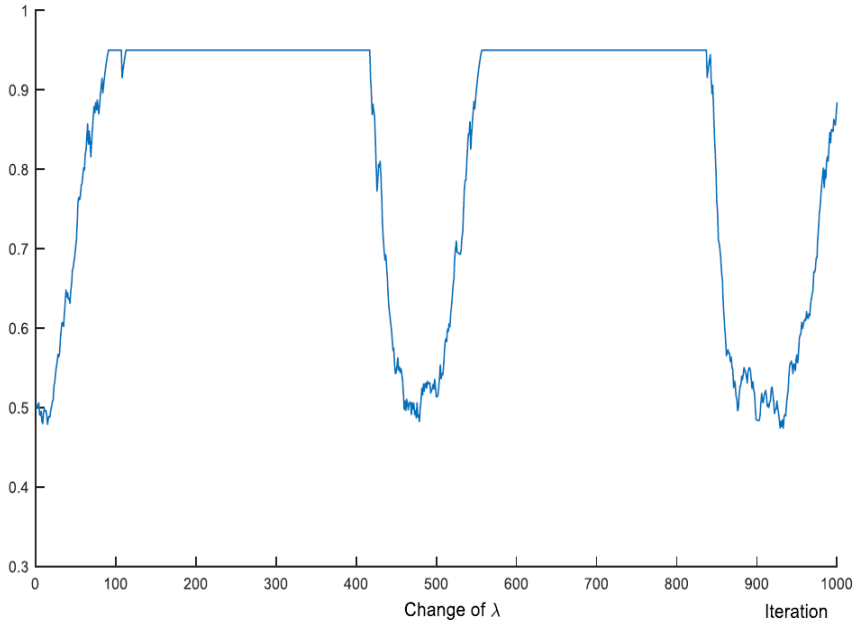
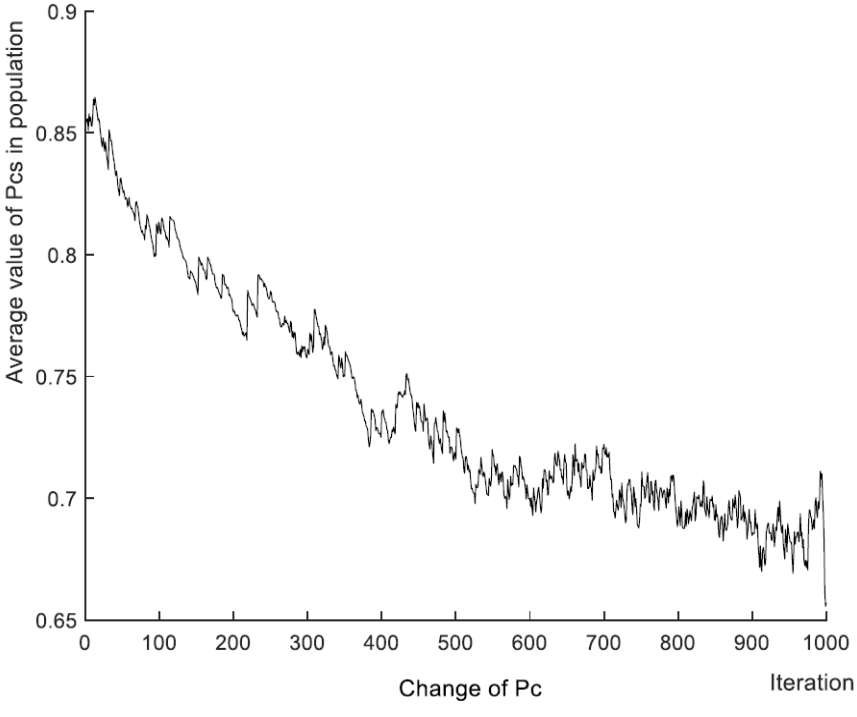


Figure 8 The curve of crossover probability in GA



7 Conclusions

In this paper, five algorithms are designed to solve the novel problem, ARP-MPDT. The objective function value is given by the travelling time and the dynamic execution time. After being tested in 11 instances, the EDA3 employing NHM and EHM is demonstrated the better adaptability to ARP-MPDT than the other EDAs. The selection proportion of NHM and EHM contributes much to the better performance. GA has a stronger competitiveness in large-scale instances. PSO has poor effect in searching the solution space. In the future, for EDAs, the adoption of local search and knowledge should be taken into consideration.

Acknowledgements

This work was supported in part by the National Key R&D Program of China (2018YFB1308000), in part by the National Outstanding Youth Talents. Support Program under Grant 61822304, in part by the National Natural Science Foundation of China under Grant 61673058, in part by the NSFC-Zhejiang Joint Fund for the Integration of Industrialization and Informatization under Grant U1609214, in part by the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant 61621063, and in part by the Projects of Major International (Regional) Joint Research Program of NSFC under Grant 61720106011.

References

Abderrezek, H., Aissa, A. and Harmas, M.N. (2019) 'Modified PSO-based nonlinear controllers applied to a DC-DC converter', *International Journal of Automation and Control*, Vol. 13, No. 1, pp.1–16.

Baioletti, M., Milani, A. and Santucci, V. (2017) 'Algebraic particle swarm optimization for the permutations search space', *IEEE Congress on Evolutionary Computation (CEC)*, pp.1587–1594.

Bhaduri, R. and Banerjee, S. (2011) 'Optimisation of controller parameters by genetic algorithm for an electromagnetic levitation system', *International Journal of Automation and Control*, Vol. 5, No. 3, pp.219–244.

Ceberio, J., Irurozki, E., Mendiburu, A. and Lozano, J.A. (2012) 'A review on estimation of distribution algorithm in permutation-based combinatorial optimization problems', *Progress in Artificial Intelligence*, Vol. 1, No. 1, pp.103–117.

Chaudhary, N., Raj, R., Kiran, K., Nema, S. and Padhy, P.K. (2015) 'Design of multivariable PID controller using DE-PSO', *International Journal of Automation and Control*, Vol. 9, No. 3, pp.173–185.

Chen, S.M. and Chen, C.Y. (2011) 'Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques', *Expert Systems with Applications*, Vol. 38, No. 12, pp.14439–14450.

Gong, W.Y., Wang, Y., Cai, Z.H. and Wang, L. (2018) 'Finding multiple roots of nonlinear equation system via a repulsion-based adaptive differential evolution', *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp.1–15, DOI: 10.1109/TSMC.2018.2828018.

Kennedy, J. and Eberhart, R. (1995) 'Particle swarm optimization', *International Conference on Neural Networks*, pp.1942–1948.

Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I. and Dizdarevic, S. (1999) 'Genetic algorithm for the travelling salesman problem: a review of representations and operators', *Artificial Intelligence Review*, Vol. 13, No. 2, pp.129–170.

Liao, Z.W., Gong, W.Y., Yan, X.S., Wang, L. and Hu, C.Y. (2018) 'Solving nonlinear equations system with dynamic repulsion-based evolutionary algorithms', *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp.1–12, DOI: 10.1109/TSMC.2018.2852798.

Lozano, J.A., Larrañaga, P., Inza, I. and Bengoetxea, E. (2006) *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*, Springer, Berlin.

Lu, S., Xin, B., Dou, L.H. and Wang, L. (2018) 'A multi-model estimation of distribution algorithm for agent routing problem in multi-point dynamic task', *2018 37th Chinese Control Conference (CCC)*, pp.2468–2473.

Panigrahi, C.K., Chattopadhyay, P.K. and Chakrabarti, R. (2007) 'Load dispatch and PSO algorithm for DED control', *International Journal of Automation and Control*, Vol. 1, Nos. 2/3, pp.182–194.

Tang, J., Ma, Y. and Guan, J. (2013) 'A max-min ant system for the split delivery weighted vehicle routing problem', *Expert Systems with Applications*, Vol. 40, No. 18, pp.7468–7477.

Wang, L. and Fang, C. (2012) 'An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem', *Computers and Operations Research*, Vol. 39, No. 2, pp.449–460.

Wang, S.Y., Wang, L., Xu, Y. and Liu, M. (2013) 'An effective estimation of distribution algorithm for the flexible job-shop scheduling problem with fuzzy processing time', *International Journal of Production Research*, Vol. 51, No. 12, pp.3778–3793.

Zhong, J.H., Zhang, J. and Fan, Z. (2010) 'MP-EDA: a robust estimation of distribution algorithm with multiple probabilistic models for global continuous optimization', *8th International Conference of Simulated Evolution and Learning*, Vol. 6457, pp.85–94.

Zhou, H., Song, M. and Pedrycz, W. (2018) 'A comparative study of improved GA and PSO in solving multiple traveling salesmen problem', *Applied Soft Computing*, Vol. 64, pp.564–580.

Zhou, S.D. and Sun, Z.Q. (2007) 'Survey on estimation of distribution algorithm', *Acta Automatica Sinica*, Vol. 33, No. 2, pp.113–124.