An Extendable Platform for Routing Problem: Optimisation, Evaluation and Solution Visualisation

Chenhao Li*, Jiyuan Pei†, Qingquan Zhang‡, Jialin Liu and Xin Yao
Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation
Department of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen 518055, China
Email: *11611423@mail.sustech.edu.cn, †11613026@mail.sustech.edu.cn,‡qingquanzhangchn@gmail.com

***Abstract*—A variety of routing problems have been formalised based on the applications in real life and their specific constraints. Diverse solvers based on one or more exact or approximate algorithms have been designed separately for optimising such problems. In most of the existing works, only the quality of the best solution found within a certain computational time is reported, while its corresponding solution is omitted. However, when the difference between the solution quality is small, decision-makers are more interested in how the actual solutions are and their diversity in decision space. In this paper, we design and implement an online platform that (i) is webpage based, with a uniform online computation environment; (ii) includes a number of routing problem models, problem instances, and solvers; (iii) includes a list of evaluation metrics and allows the visualisation of solutions in different ways for an easier comparison between solvers or solutions; (iv) is extendable, thus, offers the functionality of adding new problems, instances, solvers and evaluation metrics. We also present a novel fast cluster-based genetic algorithm for large-scale travelling salesman problems and perform a study using the proposed platform.**

***Index Terms*—Routing problem, meta-heuristics, capacitated arc routing problem, travelling salesman problem**

I. Introduction

Routing problems widely exist in real life, examples include road clearing service, urban garbage collection, power line inspection, and postal delivery. Extracting the characteristics of real-life applications, some classic routing problems [1] are formulated, such as travelling salesman problems (TSPs) and capacitated arc routing problems (CARPs) [2], [3].

CARP and TSP are NP-hard combinatorial problems. In the past two decades, solvers for those problems have been continuously proposed and improved. Different focus and adaptation of those solvers make the measurement of a solver's performance much more complicated. In some specific cases, the performance is not static and depends on the available computational budget and the size of the problem instance. Furthermore, under uncertain contexts, such as optimising CARP with uncertainties (UCARP) [4], the goal is to find the most robust solution, rather than the optimal solution for any single deterministic realisation [5], which makes the evaluating of solutions or solvers harder. A more intuitive method is needed to demonstrate the performance of solvers for better measurement and comparison.

A standard benchmark which tests solvers within an identical environment and offers intuitive comparable measurement is a good solution. It would be better to run all computation processes on a same machine, instead of downloading the platform and run in different computation environments. However, thought there are already some offline platform designed for solving and researching routing problems [6], [7], currently there is no public online platform to meet this specific demand.

Therefore, in this paper, we design and implement an webpage-based algorithm visualisation platform. Besides, we propose a standard solution format of CARP for better comparison. Solution visualisation is also supported for some routing problems integrated in this system. The system supports uploading problem instances or selecting standard problem instances for testing, based on some standard built-in algorithms. In addition, based on the modular architecture design of the platform, other similar problems can be easily added to the platform. Through such a platform, we aim to help decision-makers quickly test and analyse problems or solvers for some routing problems, while TSP and CARP are already supported.

Besides, we present a novel fast cluster-based genetic algorithm (FCGA) for large-scale travelling salesman problems and perform a study using the proposed platform. The main steps of our genetic algorithm are clustering, inter-cluster optimisation, intra-cluster operation and global optimisation. Based on a K-means algorithm [8], we divide a TSP instance into several sub-TSP instances and find the optimal solution for each sub-TSP. Then, the solutions are connected as a global population, which will be optimised in following stage. The FCGA proposed in this paper only applies mutation operators to the global population for fast computation. Experimental studies on some TSP instances of different size verify the time efficiency of our algorithm.

This work was supported by the National Key R&D Program of China (Grant No. 2017YFC0804003), the National Natural Science Foundation of China (Grant No. 61906083, 61976111), the Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (Grant No. 2017ZT07X386), the Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531), the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. JCYJ20190809121403553) and the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008).

Corresponding author: Xin Yao (*xiny@sustech.edu.cn*).

This paper is organised as follows. Section II introduces the background. Section III describes our platform. Our FCGA is presented in Section IV. Finally, Section V concludes.

II. Background

There are several classic routing problems, such as travelling salesman problems (TSPs), vehicle routing problems (VRPs) and capacitated arc routing problem (CARPs) [1]. TSP is defined as a minimisation problem to find the shortest route that visits each city only once and returns to the original city. There are many real-world applications [9] of TSP. In CARP [2], each edge owns a certain cost, and some edges own a particular demand that needs to be served. The optimisation goal of CARP can be described as finding the minimum-total-cost routes set which served all tasks. Uncertain CARP (UCARP) [4], [5] considers uncertainties in the problem or constraints, such as the number of routes, costs of edges and demands. Some efficient evolutionary computation methods, such as memetic algorithms (MAs) [10] and estimation of distribution algorithms (EDAs), have been proposed for solving UCARP [11].

There are some public platforms designed for routing problems solving and researching, such as Routing-Solver [6] and OR-Tools [7]. In these platforms, the application interface of routing problem instances and solution analysis are provided.

However, to the best of our knowledge, currently there is no public platform which offers a standard computing environment for optimising routing problems. Therefore, we design and implement such an online platform.

III. A Platform for Optimising Routing Problems

This algorithm visualisation platform[1] is mainly designed for routing problems. The current algorithm performance measurement and solution representation are usually numeric and text-based, which is hard to understand for human. Thus, this platform provides a visualisation function for solution path and calculation process. By reconverting the representation of solutions into physical meanings, decision-makers can have an intuitive comparison of the quality of solutions and the effectiveness of different algorithms. In addition, to improve extensibility, the platform adopts the modular design. It packages and integrates different problems into different modules, and uses a unified interface to process data. In this way, users can freely assemble problems and algorithms. The standard interface and environment of computing also reduce the measurement error and give a more objective and realistic evaluation. Following we describe the design of this platform.

A. The Platform Architecture

The main architecture of the platform consists of three major layers: front-end, back-end and C++ algorithm library, demonstrated in Figure 1. The overall structure is a pipeline working mode, and communications are located in data layers.

The front-end is the display part of the platform, using an HTML soundtrack code for the overall layout. For the platform we use Bootstrap to render the page and use its proprietary grid layout for a more uniform global layout. In addition, the front-end also uses JQuery for scripting control. Because of the need of communication between the front-end and back-end, the interactive technology of Asynchronous JavaScript And XML (Ajax) is also introduced in the front-end. Based on it, the front-end only needs to exchange a small amount of data with the server to load the results asynchronously, without refreshing the entire web interface. Therefore, the response of pages is faster and more humane. At the same time, Ajax also has the function of encapsulating and parsing JSON, which is more convenient for front-end and back-end communication.

[1]http://118.178.121.200:9999/model/en/combinatorialoptimization_en.html

The back-end of the platform is developed based on the classic Springboot framework. Compared with other WEB frameworks, Springboot has the advantages of rapid development, automatic management dependence, and convenient export of various forms of APIs. For a lightweight WEB application, the Springboot framework is very suitable for developing this platform. The overall architectural design was inherited from the MVC software design pattern and made some platform-specific changes.

Because the C++ language is different from Java in architectural design and has inherent performance advantages, the algorithm integrated in this platform is written in C++ language instead of Java. In general, programs that use Java language to call C++ can use related functions of process management, but there are transfer problems caused by different parameter formats, so this platform uses the improved library Java Native Access based on the Java Native Interface. Compared with the native interface, the library does not need to write tedious and complicated data structure control functions, and can directly write the calling function according to the structure that has been specified.

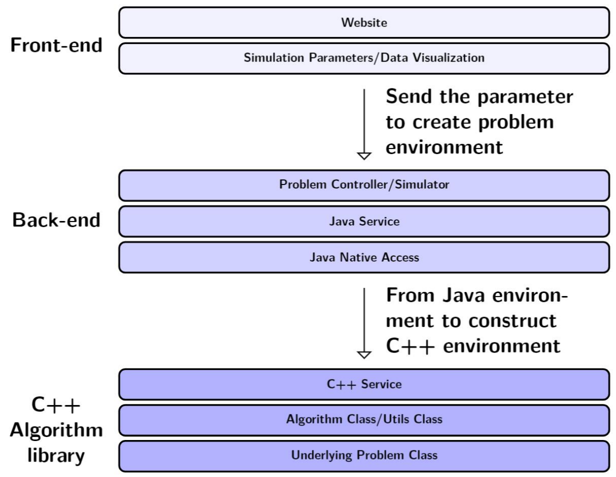

Front-end
Website
Simulation Parameters/Data Visualization
Send the parameter to create problem environment
Back-end
Problem Controller/Simulator
Java Service
Java Native Access
From Java environment to construct C++ environment
C++ Algorithm library
C++ Service
Algorithm Class/Utils Class
Underlying Problem Class

Fig. 1. The Platform Architecture.

Users select the desired problem on the web page. Parameters of a problem are selected or typed, and then sent to the simulator. The simulator layer analyzes parameters and creates a problem environment based on the specific problem, and then calls the Java Service of the problem.

In order to improve the execution efficiency of the algorithm, C++ is used to implement the algorithm instead of Java, which requires Java Native Access (JNA) to perform a transfer. Therefore, a C++ input interface suitable for a specific problem is created in the Java Service layer, and the C++ algorithm library is directly called through JNA. After receiving the problem environment of Java, it is first converted to the problem environment of C++, and the selected algorithm is called to calculate the problem.

An algorithm that inherits the underlying specific problem class (e.g., CARP class) in the algorithm library starts to execute. After getting the result, it is packaged into the corresponding solution structure (e.g., CARP solution structure) and then returned to the Java Service through JNA. Because the Java Service constructs the corresponding solution structure according to the rules of JNA, the data is parsed and then returned to the front-end through the controller. The front-end receives the solution data of the problem, organises the visualised data, and finally displays it on the web page.

B. *Problem Module*

In the current routing problem module, the two main problem modules are TSP module and CARP module. Different simulation parameters will be constructed under different problem environments, and the way of visualisation will also change. Under the framework of the platform, other problem modules can be easily extended, by adding the required problem instance parameters and designing visualisation function.

C. *Algorithm Module*

The Algorithm Module is implemented in C++ and communicates with the Java control layer through JNA. The library architecture is shown in Figure 2. At present, it contains three parts: the path scanning algorithm, the algorithm of utils and abstracted the problem class. The overall behavior is mainly controlled by JNAService, which analyses the received parameters and builds a C++ environment. For example, a selected main path scanning algorithm implements the abstract problem and its interface while inheriting routing algorithm, and then the result is encapsulated according to the solution structure in the abstract problem and returned to JNAService. Finally, JNAService returns it through JNA. Since the design is modular, users can freely add new algorithms to a certain problem, or add a new problem.

D. *Evaluation Metric Module*

The computational efficiency of the algorithm will be demonstrated in each problem module. For example, in the CARP problem in Figure 3, the execution time of the algorithm is calculated. It also shows some necessary information, including the number of trips, the cost of each trip, and detailed routes. In the follow-up work, a series of corresponding results exhibited by the genetic algorithm with increasing iteration will be added one after another.

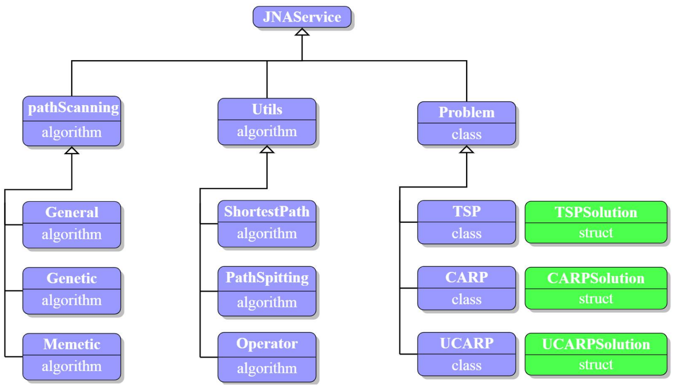
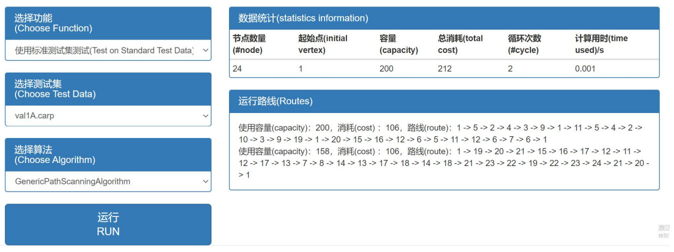


Fig. 2. The C++ Algorithm Library Architecture.


选择功能
(Choose Function)
使用标准测试集测试(Test on Standard Test Data)

选择测试集
(Choose Test Data)
val1A.carp

选择算法
(Choose Algorithm)
GenericPathScanningAlgorithm

运行
RUN

数据统计(statistics information)

节点数量 (#node)	起始点(initial vertex)	容量 (capacity)	总消耗(total cost)	循环次数 (#cycle)	计算用时(time used)/s
24	1	200	212	2	0.001

运行路线(Routes)

使用容量(capacity)：200，消耗(cost)：106，路线(route)：1 -> 5 -> 2 -> 4 -> 3 -> 9 -> 1 -> 11 -> 5 -> 4 -> 2 -> 10 -> 3 -> 9 -> 19 -> 1 -> 20 -> 15 -> 16 -> 12 -> 6 -> 5 -> 11 -> 12 -> 6 -> 7 -> 6 -> 1
使用容量(capacity)：158，消耗(cost)：106，路线(route)：1 -> 19 -> 20 -> 21 -> 15 -> 16 -> 17 -> 12 -> 11 -> 12 -> 17 -> 13 -> 7 -> 8 -> 14 -> 13 -> 17 -> 18 -> 14 -> 18 -> 21 -> 23 -> 22 -> 19 -> 22 -> 23 -> 24 -> 21 -> 20 -> 1

Fig. 3. Visualisation of CARP module.

E. *Solution visualisation Module*

Solution visualisation is important for analysing the solution quality and algorithm performance. Currently, most of the work on routing problems usually demonstrate the quality of the best solution only, like minimal total cost. However, for decision-makers facing the real-life routing problems, the evaluation is just one of the reference information. They may care more about the actual path sequence of the solution, or even the path graph, which, yet, is not provided from those works. Besides, without the actual path, it is hard to verify the solution, as well as the reliability of work.

The visualisation of solutions can be divided into two categories, path sequence and path graph. Normally the path graph is generated based on path sequence, together with other necessary information. Following we describe the solution visualisation for TSP and CARP on our platform.

1) *Solution visualisation for TSP:* The graph solution visualisation for TSP is illustrated in Figure 4. Combined the solution with the corresponding coordinate data, the path is drawn on a route map with a uniform scale. So the solutions can be demonstrated more intuitively.

2) *Solution visualisation for CARP:* Currently, as the CARP problem instances normally do not provide the coordinate of each vertex, solution graphic visualisation is not possible. We illustrate the solutions in sequence (Figure 3). Inspired by project HGS-CARP [12], we raise a novel standard solution format for CARP. The sequential solution is generated following the standard solution format, which is described as follows.

A formatted solution file consists of several lines, which store each route's information in the solution. The first part of each line is the overview, shown as in Table I. Then there are multiple items enclosed in brackets. Each item belongs to one of these three categories, defined as follows, donated as D (depot), S (serve) and P (pass).

TABLE I
A NOVEL STANDARD FORMAT FOR PRINTING SOLUTIONS FOR CARP.

depot	route id	served demand	cost of route	#travelled edges

- Depot:
  $(D, 0, depot, depot, served\ demand, cost\ so\ far)$
- Task:
  $(S, edge\ id, head, tail, served\ demand, cost\ so\ far)$
- Edge without task:
  $(P, edge\ id, head, tail, served\ demand, cost\ so\ far)$

In short, a route in a solution is a line of text formed by a 5-number overview prefix and a sequence of route component within brackets. In CARP, a solution consists of one or multiple routes. For gdb1 [13], a widely used problem instance for CARP algorithm, a complete solution example is as follows.

0 1 5 76 8 (D, 0, 1, 1, 0, 0) (P, 5, 1, 12, 0, 4) (P, 15, 12, 6, 0, 7) (P, 11, 6, 5, 0, 14) (S, 12, 5, 11, 1, 34) (S, 21, 11, 9, 2, 48) (S, 8, 9, 2, 3, 50) (S, 7, 2, 4, 4, 59) (S, 2, 4, 1, 5, 76) (D, 0, 1, 1, 5, 76)
0 2 5 60 7 (D, 0, 1, 1, 0, 0) (S, 5, 1, 12, 1, 4) (P, 15, 12, 6, 1, 7) (S, 14, 6, 7, 2, 11) (P, 16, 7, 8, 2, 19) (S, 19, 8, 11, 3, 29) (S, 22, 11, 10, 4, 41) (S, 4, 10, 1, 5, 60) (D, 0, 1, 1, 5, 60)
0 3 5 86 10 (D, 0, 1, 1, 0, 0) (P, 5, 1, 12, 0, 4) (S, 13, 12, 5, 1, 15) (P, 10, 5, 3, 1, 20) (S, 9, 3, 4, 2, 40) (P, 7, 4, 2, 2, 49) (S, 6, 2, 3, 3, 67) (S, 10, 3, 5, 4, 72) (S, 11, 5, 6, 5, 79) (P, 15, 6, 12, 5, 82) (P, 5, 12, 1, 5, 86) (D, 0, 1, 1, 5, 86)
0 4 5 53 8 (D, 0, 1, 1, 0, 0) (P, 5, 1, 12, 0, 4) (P, 15, 12, 6, 1, 7) (P, 14, 6, 7, 1, 11) (S, 16, 7, 8, 2, 19) (S, 18, 8, 10, 3, 22) (S, 20, 10, 9, 4, 38) (P, 8, 9, 2, 4, 40) (S, 1, 2, 1, 5, 53) (D, 0, 1, 1, 5, 53)
0 5 2 41 3 (D, 0, 1, 1, 0, 0) (P, 5, 1, 12, 0, 4) (S, 17, 12, 7, 1, 22) (S, 3, 7, 1, 2, 41) (D, 0, 1, 1, 2, 41)

Based on the solution format from HGS-CARP, we add more information about edge cost and 0-demand edges for better verifying and decision-making. Our format can completely record the information needed for both cost computation and route demonstration

F. Platform Layout

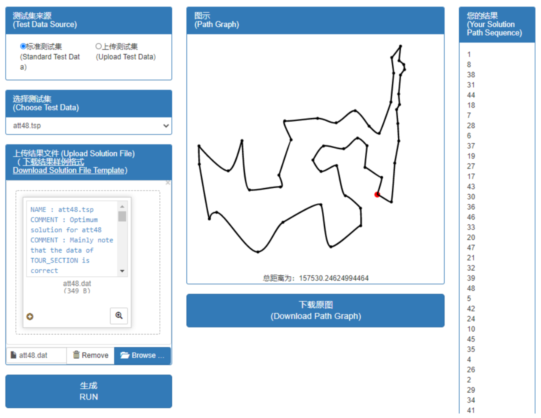
This platform is online and includes distinct webpages for each module, described as follows.

1) Main page: The main webpage contains TSP, CARP and UCARP (under construction). Users can redirect to a specific problem module page from the main page.

2) Module webpage: Each module webpage contains a problem description, an operation area for algorithm and problem instance uploading or selecting, and an area to visualise the solution with a functional button for downloading file templates and graphs.

a) CARP module webpage: Figure 3 illustrates the operation area for CARP page. The operation area is where the users input the instance parameters and the webpage displays the data visualisation results. This platform has some built-in standard problem instances for CARP. Users can choose built-in standard problem instances, or upload their own designed problem instances, which must follow the format. The display area demonstrates the statistical information of the optimisation process and the text solution visualisation, which is described above.

b) TSP module webpage: Figure 4 illustrates the operation area for TSP page. The design of TSP webpage is similar to the on of CARP. We add a route map visual area for the TSP. By combining the problem instance with the corresponding result, the page can draw a route map in this route map visual area, according to a uniform scale. The overall path is intuitively demonstrated.



```
NAME : att48.tsp
COMMENT : Optimum
solution for att48
COMMENT : Mainly note
that the data of
TOUR_SECTION is
correct
```

总距离为：157530.24624994464

```
1
8
38
31
44
18
7
28
6
37
19
27
17
43
30
36
46
33
20
47
21
32
39
48
5
42
24
10
45
35
4
26
2
29
34
41
```

Fig. 4. Visualisation of TSP Module.

IV. Fast Cluster-based Genetic Algorithm

A. Main Approach

The algorithms for solving TSP can be divided into two categories: exact algorithms and heuristic algorithms. Exact algorithms ensure that the final solution is optimal [14]. However, the high complexity of exact algorithms limits its application. From the raising of Simulated Annealing [15], significant progress is achieved on solving TSP by heuristic algorithms which are able to jump out from local optima.

Genetic Algorithm (GA) is an efficient way for solving the routing problem. A general GA framework for optimising TSP is described in Algorithm 1. Yet, as the size of problem instances increases, the computational cost of GA on TSPs increases rapidly. No duplicated city is allowed in TSP solutions. To handle the violence of constraints, the fixing operation after crossover is introduced. The Larger problem instance makes duplication more frequent, so the time cost increases.

Algorithm 1 Solving TSP based on classic GA.

```
Require: A TSP instance;
  Randomly initialise a population $P$ with $N$ individuals;
  repeat
    Evaluate all individuals;
    Choose the fittest individuals for crossover, get an offspring;
    Mutate the offspring;
    Add the offspring into the next generation according to a selection strategy;
  until Stop criterion is satisfied;
  return The shortest total distance individual of the population.
```

1) Cluster-based GA with two stages: An intuitive idea to reduce cost is reducing the size of the problem instance. However, normally the size is determined due to the target problem. Dividing a large problem instance into several smaller isolated parts is an alternative. From the real-life experience, the vertices in a dense area are more likely to be visited as an uninterrupted paths, which means each of them can be treated as a sub-TSP. Therefore, the idea of transfer a large map into smaller maps with clustering algorithm and then solving it with GA was raised to handle this situation [16]. The classic clustering algorithm, k-means [8] is suitable for the clustering of vertices. Vertices are divided into multiple groups, then by applying GA to find each optimal or approximate optimal sub-path of the corresponding group and linking them head-to-tail, the global solution path is generated. The link order is the the lowest-total-distance one of all the possible orders, found by traversal searching.

Nevertheless, considering the difference between objectives of clustering and routing, it is possible that the consecutive vertices of the optimal path are clustered in different group, which means the optimal path of a local group may not be the same path segment of those vertices in global optimal path. In this situation, linking local path will not lead to the global optimal solution. Therefore, we propose a two stages cluster-based GA. The first stage is clustering and local optimisation, which is the same as described above. In the second stage, we connect each sub-path according to an inter-cluster connecting strategy, instead of brute force searching. Then we add an extra GA process to optimise the global path.

The inter-cluster connecting strategy is as follows. (I) Each cluster is treated as a vertex located at the geometric center of the cluster, then the original TSP is transformed into a simplified TSP. (II) Applying classic GA on the simplified TSP to get the path of clusters. (III) Merge the clusters one by one according to the path. The distance between two clusters is calculated as the Euclidean distance of the two geometric centers of clusters. During merging, we find the closest pair of vertices of each cluster's shortest path and reconnect them by randomly breaking an origin connection. The origin connected vertices of this pair will also be connected to maintain the path integrity. Then we randomly reconnect the rest of the paths until the population reaches the required size. Illustrated as Figure 5, the closest pair of clusters is I and II, and the closest pair of vertex is A and B. Therefore, we randomly break a connection of A and a connection of B, and reconnect A with B. Then fix the close path. C and D are the same.

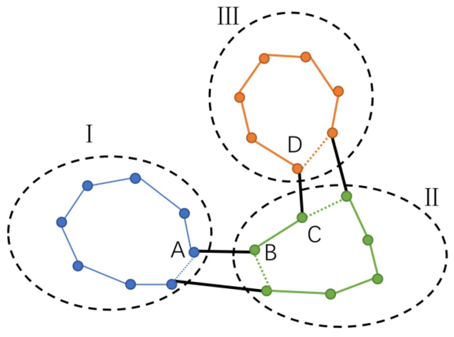


Fig. 5. Intra-cluster connection.

2) Replace classic GA with Fast GA: Due to the frequent fixing, crossover takes most of the time. Table II illustrates the time cost of operators when solving TSP on data Djibouti, Qatar and Uruguay [13]. In the experiment, we set population size as 100 and the stop criterion as 20000 generations. On average crossover takes about 85% of the time, while mutation takes about 10%.

TABLE II
TIME CONSUMED BY OPERATORS ON DJIBOUTI, QATAR AND URUGUAY. CROSSOVER (WITH FIXING) TAKE MOST OF THE TIME.

City	Crossover	Mutation	Other
Djibouti	82.3s(87%)	6.5s(7%)	6.3s(7%)
Qatar	264.3s(84%)	32.8.3s(10%)	18.7s(6%)
Uruguay	1290.8s(83%)	208.4s(13%)	55s(4%)

In this paper we propose a fast GA framework in which crossover operators are not included for improving the time efficiency. Assuming we have g (normally 3) different mutation operators, firstly we randomly initialise a population $P(0)$ with N individuals. Then in each generation t, all individuals are evaluated and randomly assigned into $N/(g+1)$ groups, while each group owns $(g+1)$ individuals. In each group, except for the high fitness individual p, each of the others will be replaced by an offspring generated by directly mutating p of high fitness. After that, the new generation of population $P(t+1)$ is formed. This procedure is repeated until the stop criterion is satisfied. The highest fitness individual of the whole process will be recorded and return as the final solution.

Besides, we apply a bias mutating location selection strategy, which boosts the performance of fast GA on TSP in our experiment. In this strategy, the first change location is selected under a specific probability distribution. The probability of choosing the i-th vertex is calculated as:

$$prob(i) = \frac{d(i) + d(i+1)}{2V}. \quad (1)$$

Where d is the minimal distance between the i-th vertex and all other vertices in other clusters, defined as:

$$d(i) = \frac{1}{\min_{q \in c_j, j \in \{1,...,k\}/\{u_j\}} dist(x_i, q)}, \qquad (2)$$

Where V is the a factor that makes $\sum_{j=1}^{N-1} prob(i) = 1$, and $dist(x_i, q)$ is the Euclidean distance of x_i and q. We also set a probability (0.2 in our work) to randomly choose the first vertex. The probability of the second location is proportional to the distance with the first location.

Combining the design described above, a cluster-based fast GA is formed. The algorithm has two main steps. The first step is clustering, and following intra-cluster optimisation with classic GA. In the second step, the intra-cluster solutions are connected and evolved by a global fast GA. The reasons of applying classic GA in the first step and fast GA on the second step are: (I) For small data in the first step, classic GA can keep more diversity and perform very well. (II) The similarity among sub-population in the same cluster is close to each other and there is no need to improve diversity by applying crossover. (III) Furthermore, mutation with bias distribution can achieve a significant performance in dealing with large data like the second step. Details are described in Section IV-B. Though our target problem is TSP, this algorithm is able to solve any problem which has similar characteristic as TSP.

Algorithm 2 Fast Cluster-based GA.

```
Require: A TSP instance;
  Cluster vertices of the instance by k-means;
  for Each cluster $C$ do
    Apply classic GA on $C$ with the objective of minimising the total distance;
    Record all individuals in the last generation, each individual is a sub-path;
  end for
  Find the shortest path $p_{shortest}$ passing all clusters' center by traversal searching;
  Connect the sub-paths following the order of $p_{shortest}$, generate new population $P$;
  Calculate the probability of mutation on each location;
  Apply Fast GA with $P$ as the initial population;
  return The best global path during the whole Fast GA process.
```

B. Experiment

1) Parameter setting: In our experiences, a solution is represented as a integer sequence with length as n, where n is the number of city and each integer represents a city, $0 \leq a \leq n$. A solution's fitness is the opposite number of the total length of the solution path. Longer path leads to smaller fitness. We use binary tournament selection and elitist selection as the selection strategy. Flip, swap and slide are chosen as the mutate operators with a mutate rate as 0.9. Especially, in classic GA, a solution is generated from partially mapped crossover (PMX) with 2 cutting points and mutate operators according to the mutate rate. Besides, for clustering, we choose k-means with cluster number k as $\lfloor \#city = 70c \rfloor$, which is the best value for better performance of final routing solution in our experiment. If not specified, the stop criterion of all experiments is that no better individual is generated within 500 generations. We think 500 large enough to avoid stopping the optimisation when the possibility of finding better solution is considerable.

2) Classic GA and fast GA: In this section we analyse the characteristic of classic GA and fast GA, then we explain the reason why we choose the specific combination of these two as mentioned above.

Firstly we compare the performance of classic GA with fast GA, with population size as 100 and 30 repeat run. Under this parameter setting we think the characteristics of fast GA and classic GA is observable. The problem instances [13] are listed in Table III. In all tables, distances are measured in meters, and time is in seconds.

TABLE III
INSTANCES USED IN OUR STUDY.

	Sahara	Djibouti	Qatar	Uruguay	Zimbabwe
#city	29	38	194	734	929
Lowest distance	27603	6656	9352	79114	95345
#Iteration	10000	10000	20000	20000	20000

TABLE IV
RESULT OF CLASSIC GA AND FAST GA ON 5 PROBLEM INSTANCES. FAST GA IS ABOUT 10 TIMES AS FAST AS CLASSIC GA WITH SAME MAX GENERATION NUMBER. DISTANCE IS MEASURED IN METERS, WHILE TIME IS IN SECOND.

City	Classic GA		Fast GA	
	Avg. distance±Std	Avg. time	Avg. distance±Std	Avg. time
Sahara	27645±71.3	34.2	27631±89.8	3.5
Djibouti	6803.7±186.17	43.3	6707.5±113.4	3.5
Qatar	10210±182.1	316.8	13352±394.2	56.9
Uruguay	142422±3926.2	1616.2	255460±3764.4	167.9
Zimbabwe	203840±5340	2325.6	246890±4864.6	328.0

Figure 6 and Table IV illustrate the experiment results. GA has a faster converging speed, but in relatively small problem instances, both of them can get good solutions. However, as the size of data increases, fast GA is more likely to have premature convergence, such as the results of Qatar, Uruguay and Zimbabwe. It leads to a worse solution. On these three problem instances, classic GA performs better than fast GA, though the time cost of classic GA is quite larger.

Figure 7 illustrates the entropy of the population during evaluation. It is obvious that the entropy of the population generated by fast GA reduces faster. As it only keeps the best individual based on which a population is generated. The diversity of fast GA reduces rapidly, so does the entropy. Another characteristic is that the entropy of classic GA finally converges but the entropy of fast GA is continuously oscillating. Next subsection discusses more about it.

As premature convergence is the main issue of fast GA, we believe that increasing the maximum number of generations will not significantly improve the performance. Therefore, We

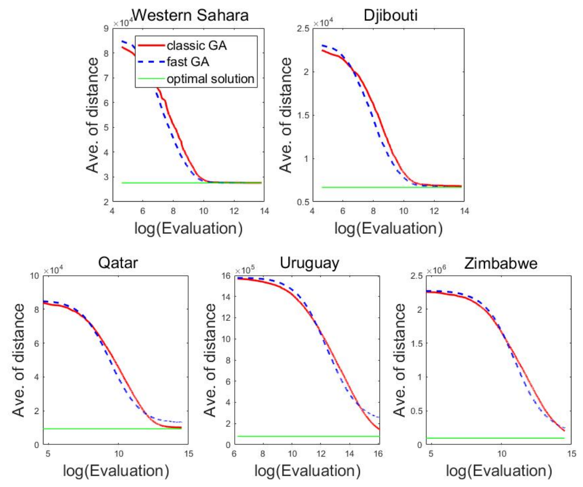


Fig. 6. Convergence curves of classic GA and fast GA on 5 problem instances. Fast GA converges faster but converges to higher-cost solutions.

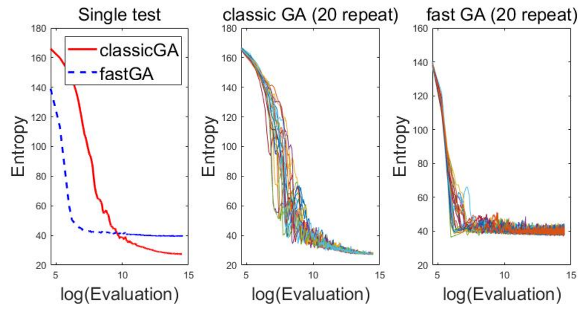


Fig. 7. The entropy of the population during the optimisation on Qatar. The entropy of Fast GA reduces more rapidly, but does not converge.

design a simple experiment to find out the effect of another way, increasing population size. The problem instances for testing are Qatar and Uruguay, and the population size is 500. The result is illustrated in Table V, which is as we expected.

TABLE V
RESULTS OF GA WITH DIFFERENT POPULATION SIZE. WITH LARGER POPULATION, FAST GA IS MORE EFFICIENT THAN CLASSIC GA.

	POP size	Qatar		Uruguay	
		Distance	Avg. time	Distance	Avg. time
Classic GA	100	10210	316.8	142420	1616.2.9
Fast GA	100	13352	56.9	255460	167.9
Fast GA	500	10010	85.4	153250	192.3

Increasing the population size has a good effect on the performance of fast GA. The distance of the best solution of fast GA with population size 500 is relatively close to classic GA with population size 100, and the time cost of the former is quite low.

Fast GA requires less computing power but the diversity is lower, which leads to a premature convergence. The population size can be increased to significantly improve the performance with an acceptable extra cost.

3) Verification of bias mutation location selecting: In this work we propose a biased mutation location selecting strategy to improve the efficiency of the cluster-based fast GA. The core idea is letting the mutation focus on those adjacent parts of clusters. After intra-cluster GA, solutions in each cluster are relatively well arranged and close to the corresponding part of the global optimal solution. Therefore, what needs to be changed more is the vertices where clusters are connected, which are the adjacent parts. The mutation operators operate on more than one node. By giving the vertices at adjacent parts higher mutation possibility and encouraging the next node to be selected near the first node, the mutated sequence is focused.

To verify the design, we test first the performance of uniform random location selection and bias selection on both classic GA and fast GA on data Uruguay. For both, the first part is the same, applying uniform location selection classic GA for intra-cluster optimisation. The difference is the GA in the second step. The location selection of crossover operator of classic GA is the same as its mutation location selection. The result is illustrated in Figure VI. The biased selection improves both classic GA and fast GA, while the improvement of fast GA is more noteworthy.

TABLE VI
RESULTS OF GAS WITH DIFFERENT LOCATION SELECTION STRATEGY ON URUGUAY. TIME COST OF STEP 1 FOR BOTH GA IS CLOSED. TIME COST OF STEP 2 FOR CLASSIC GA IS MUCH HIGHER THAN FAST GA.

	Classic GA		Fast GA	
	Uniform	Bias	Uniform	Bias
Distance	87971	87097	93868	86516
Distance Std	933.1	678.5	3178.0	832.8
Time of step 1	178.1	171.0	174.1	172.6
Time of step 2	1149.8	1238.7	22.2	28.7

As most part of solutions in the initial population of step 2 is duplicated, the expensive crossover operator is not efficient. It is unlikely for crossover to introduce diversity. However, three quarters of individuals in each generation are generated from mutation in fast GA. More mutation and no crossover introduce more diversity for fast GA under the situation of step 2. The entropy of the population in fast GA at later stage is higher than classic GA and continuously jittering, which is mentioned above. It proves our thought. Fast GA with a bias mutation location has significantly better performance at the optimisation of step 2. To eliminate the influence of stochastic factors, we test fast GA and classic GA with the same bias selection and initial population, which is generated from the step 1 process. The result of 30 repeats on Zimbabwe and Uruguay are shown in Table VII.

TABLE VII
RESULT OF BIAS SELECTION BY CLASSIC GA AND FAST GA. GA WITH BIAS SELECTION PERFORM BETTER.

City	Classic GA		Fast GA	
	Avg. distance±Std	Avg. time	Avg. distance±Std	Avg. time
Zimbabwe	107050±2188.0	997.6	106830±2210.3	25.2
Uruguay	86363±438.4	1151.2	86275±334.5	28.7

At last, we test classic GA without clustering, fast GA without clustering, and our FCGA. The result is illustrated in

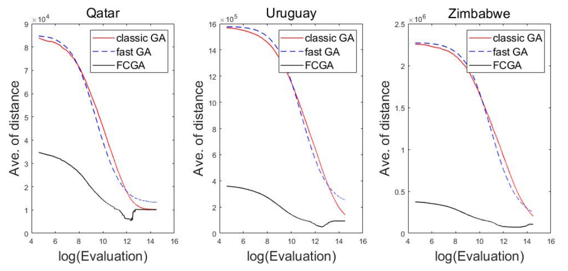


Fig. 8. Convergence curves of classic GA and fast GA on 3 problem instances. FCGA converges much better.

Table VIII and Figure 8. The reason why FCGA's population average distance is increased at each end stages in Figure 8 is that in step 2, most individuals generated by mutating have longer distance, which makes the average distance increased. However, there are still some individual is improved in step 2, in which the best one will be return as the solution.

TABLE VIII
RESULT OF BIAS SELECTION OF CLASSIC GA AND FAST GA. K-MEANS BASED FAST GA IS BETTER.

	Classic GA		Fast GA		K-means based fast GA.	
	Distance	Time	Distance	Time	Distance	Time
Qatar	10210	316.8	13352	56.9	10113	54.1
Uruguay	142420	161620	255460	167.9	86275	197.1
Zimbabwe	203840	2325.6	246890	328	106830	522.4

V. CONCLUSION

Because of the diversity of solvers for routing problems, a standard platform for comparing solvers and measuring solvers' performance and solution visualisation is necessary. In this paper, we design and implement a standard platform for TSP and CARP. Based on the technical design described in Section III, users can test various algorithms and problem instances, from general standard to their own design. Considering the similarity of other routing problems and combinatorial optimisation problems, our platform allows adding more problems. Besides, the platform has a set of well-designed tests and comparison visualisation methods that can help decision-makers analysing the results. In addition, we present a novel fast cluster-based genetic algorithm for large-scale TSP and perform a study using the proposed platform.

According to the design, some more functionalities are under construction. In the future work, we will expand the functionalities for TSP as CARP. Users can use the standard instances or upload their own data to test classic build-in algorithms, as well as their own solvers, like the CARP module. Furthermore, we will add more state-of-the-art algorithms and instances into the platform for better comparing.

New visualisation schemes will be explored. Besides improving the solution visualisation, we are also working on visualising the optimisation progress with convergence curves in real time.

REFERENCES

[1] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[2] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–315, 1981.

[3] S. Wøhlk, "A decade of capacitated arc routing," in *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008, pp. 29–48.

[4] Y. Mei, K. Tang, and X. Yao, "Capacitated arc routing problem in uncertain environments," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[5] J. Liu, K. Tang, and X. Yao, "Robust optimisation in uncertain capacitated arc routing problems: Progresses and perspectives," 2020.

[6] University of Antwerp, "RoutingSolver," [EB/OL], https://www.routing-solver.com/ Accessed September 30, 2020.

[7] Google, "Google OR-Tools," [EB/OL], https://developers.google.cn/optimization/ Accessed September 30, 2020.

[8] K. Wagstaff and C. Cardie, "Constrained k-means clustering with background knowledge," *Proceedings of the 17th International Conference on Machine Learning*, pp. 1103–1110, 01 2000.

[9] G. Gutin and A. Punnen, "The traveling salesman problem and its variations," *Paradigms of Combinatorial Optimisation Problems & New Approaches*, vol. 4, no. 2, p. 193–205, 2007.

[10] J. Wang, K. Tang, and X. Yao, "A memetic algorithm for uncertain capacitated arc routing problems," in *2013 IEEE Workshop on Memetic Computing (MC)*. IEEE, 2013, pp. 72–79.

[11] J. Wang, K. Tang, J. A. Lozano, and X. Yao, "Estimation of the distribution algorithm with a stochastic local search for uncertain capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 96–109, 2015.

[12] vidalt, "HGS-CARP," [EB/OL], https://github.com/vidalt/HGS-CARP/ Accessed September 9, 2020.

[13] The Department of Combinatorics and Optimisation at the University of Waterloo, "TSP World," [EB/OL], http://www.math.uwaterloo.ca/tsp/world/ Accessed September 9, 2020.

[14] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, June 1992.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[16] L. Tan, Y. Tan, G. Yun, and Y. Wu, "Genetic algorithms based on clustering for traveling salesman problems," in *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2016.