Manuscript version: Author's Accepted Manuscript

The version presented in WRAP is the author's accepted manuscript and may differ from the published version or Version of Record.

Persistent WRAP URL:

http://wrap.warwick.ac.uk/112794

How to cite:

Please refer to published version for the most recent bibliographic citation information.
If a published version is known of, the repository item page linked to above, will contain details on accessing it.

Copyright and reuse:

The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions.

Publisher's statement:

Please refer to the repository item page, publisher's statement section, for further information.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk.

Xiao-Bing Hu[a,b,c*], Hai-Lin Zhang[d], Chi Zhang[b], Ming-Kong Zhang[e] , Hang Li[a] , Mark S. Leeson[c]

a China-France Joint Research Center of Applied Mathematics for Air Traffic Management, Tianjin Key Laboratory for Advanced Signal Processing, College of Electronic Information and Automation, Civil Aviation University of China, Tianjin, 300300, China
b Collaborative Innovation Center of eTourism, Beijing Union University, Beijing, China
c School of Engineering, University of Warwick, Coventry, CV4 7AL, UK
d China TransInfo Technology Co. Ltd, Beijing, 100185, China
e Academy of Disaster Reduction and Emergence Management, Faculty of Geographical Science, Beijing Normal University, Beijing, 100875, China

ABSTRACT: Due to the complexity of multi-objective optimization problems (MOOPs) in general, it is crucial to test MOOP methods on some benchmark test problems. Many benchmark test problem toolkits have been developed for continuous parameter/numerical optimization, but fewer toolkits reported for discrete combinational optimization. This paper reports a benchmark test problem toolkit especially for multi-objective path optimization problem (MOPOP), which is a typical category of discrete combinational optimization. With the reported toolkit, the complete Pareto front of a generated test problem of MOPOP can be deduced and found out manually, and the problem scale and complexity are controllable and adjustable. Many methods for discrete combinational MOOPs often only output a partial or approximated Pareto front. With the reported benchmark test problem toolkit for MOPOP, we can now precisely tell how many true Pareto points are missed by a partial Pareto front, or how large the gap is between an approximated Pareto front and the complete one.

1. Introduction

In reality, optimization often has to be carried out subject to not just one but multiple objectives, so come multi-objective optimization problems (MOOPs). Such multiple objectives in MOOPs are often conflictive to each other, which means improving one objective might cause some other objectives to degrade as costs. Therefore, a major goal of resolving MOOPs is to achieve a good balance between multiple objectives. An ideal balance can be represented by a Pareto optimal solution, which in definition means there exists no solution that has at least objective improved without sacrificing other objectives [1–4]. Since not just one but many Pareto optimal solutions exist for a MOOP, this often makes it more complicated and more difficult to resolve a MOOP than a SOOP (single-objective optimization problem). The projections of all such Pareto optimal solutions in the objective space compose the Pareto front of the MOOP. Basically, resolving a MOOP demands effective methods to calculate or find out the Pareto front of the MOOP. Because of the importance of MOOP to both theoretical research and daily life applications, many methods have been reported to resolve various MOOPs. Roughly speaking, most existing methods for MOOPs can be classified into three categories: aggregate objective function (AOF) based methods, constrained objective function (COF) based methods, and Pareto-compliant ranking-based (PCR) methods [4,12]. An AOF method needs to construct a single aggregate objective function which combines all of the original objectives in an MOOP, and then optimize the single aggregate objective function by resolving the single-objective optimization problem (SOOP) [2,5–10]. The COF methods are also based on SOOPs because, in a COF method, only one single objective is optimized while all other objectives are treated as extra constraints [11–14]. The PCR methods, by favoring nondominated solutions and employing population-based evolutionary approaches (such as a genetic algorithm, particle swarm optimization, and ant colony optimization), generate and operate on a pool of candidate solutions, and therefore are capable of identifying multiple Pareto non-dominated solutions [15–29]. It should

be noted that AOF and COF methods are much less popular than PCR methods these years. However, many existing MOOP methods mainly aim to find a partial or approximated Pareto front (AOF and COF methods may find some Pareto points with theoretical guarantee, while PCR methods often output just some non-dominated solutions), and there are not many theoretical results on finding out the complete Pareto front [4,12]. It is often difficult to tell if they have found the complete Pareto front [12]. In particular, very few results are available on the quality of the approximation of the Pareto front for discrete MOOPs [4]. Due to the stochastic nature of PCR methods and their wide applications to discrete MOOPs [27–29], relevant approaches to evaluate the performance of PCR methods for discrete MOOPs are highly demanded. To evaluate the performance of such MOOP methods, it is crucial to find out how partial the calculated Pareto front is, or how big the gap is between the approximated Pareto front and the true one. To conduct such performance evaluation for MOOP methods, we need to know the complete Pareto front in the first place. There are two strategies to get the information about complete Pareto front: (i) we design optimization methods capable of finding out complete Pareto front with theoretical guarantee of optimality, or (ii) we design MOOPs whose complete Pareto front can be preset or deduced manually. The first strategy looks ideal, but it is difficult to achieve (otherwise, there would have not been so many MOOP methods that can only find a partial or approximated Pareto front). Compared with the enormous number of results on generating incomplete Pareto front, there exist a very handful of publications talking about how to find complete Pareto front to some specific MOOPs e.g., shortest path problem [30–38], spanning tree problem [39,40], knapsack problem [41,42], and integer program problem [43–45]. Basically, once a specific MOOP is resolved by these methods, we can use that MOOP as a case study problem to evaluate those MOOP methods that can only find a partial or approximated Pareto front. However, the first strategy has a serious practicability issue, because (i) there is no control on any features of Pareto front (e.g., assume we want a concave Pareto front with many Pareto points on it, but resolving a specific MOOP could output a convex Pareto front with only a few Pareto points on it), and (ii) those methods in Refs. [30–45] only works on small problem scale. The second strategy is actually to design benchmark test problem toolkit. In this strategy, we are supposed to have a good control on many important features of generated benchmark MOOPs, e.g., the shape of Pareto front and the problem scale. By adjusting such features, the generated benchmark MOOPs can be used to make a comprehensive evaluation on the performance of a MOOP method under test. For example, can the MOOP method find both convex and concave Pareto front? Will the MOOP method be easily entrapped into major clusters of Pareto points and thus miss out some niche clusters? Does the MOOP method have a good scalability (e.g., extending to many-objective problems)? It should be noted that a test problem toolkit is different from a case study problem. The former can generate many test problems, and the most important, the Pareto front of a test problem generated by toolkit is controllable and pre-known before we use it to test MOOP methods. The latter is just a single test problem, its Pareto front is usually not controllable, and what is worse, the Pareto front of a complicated case study problem is often unknown. This means that, if we use such a case study problem, we can only tell relative performance difference between MOOP methods, but cannot make any absolute performance evaluation for a MOOP method in terms of finding the complete Pareto front. By employing some carefully-designed mathematical functions as objectives, there have already been some benchmark test problem toolkits reported, and they have significantly boosted the MOOP research [46–53]. However, these MOOP toolkits are mainly for continuous numerical optimization, and there still lacks effective benchmark test problem toolkit for discrete combinational MOOPs. As is well known, resolving discrete combinational optimization often demands highly purpose-designed methods, and those continuous numerical optimization methods can hardly apply. For example, some classical designs of mutation and crossover for genetic algorithm are very effective for continuous numerical

optimization, but they will simply cause serious infeasibility issues to discrete combinational optimization [54,55]. In other words, a MOOP method working well on continuous numerical optimization toolkit does not necessarily mean it will also work well on discrete combinational MOOPs. Therefore, we need benchmark test problem toolkit to evaluate methods for discrete combinational MOOPs. This paper makes an attempt to develop a benchmark test problem toolkit especially for multi-objective path optimization problem (MOPOP), which is a typical discrete combinational MOOP. In existing literature on MOPOPs, focus was usually put on how to design effective methods to resolve MOPOPs, and such methods were often tested on case study problems [30–38]. Attention has relatively much less been paid to develop test problem toolkits for evaluating MOPOP methods. In this paper, Section 2 gives the mathematical description of MOPOP. Section 3 explains to how to design a test problem toolkit for MOPOP. Section 4 makes some analyses on the reported toolkit. Experimental results are reported in Section 5, and the paper ends with some conclusions in Section 6.

2. Problem description of MOPOPs

In this paper, we mainly study one-to-one static MOPOP, i.e., finding all Pareto optimal paths between a pair of origin and destination nodes, denoted as $[O, D]$, in a static route network. Assume a route network $G(V, E)$ is composed of node set V and link set E. V has N_N different nodes including the origin and the destination, and E has N_L links between nodes. This route network can be recorded as an $N_N \times N_L$ adjacent matrix A. The matrix entry $A(i, j) = 1, i = 1, \dots, N_N; , j = 1, \dots, N_N$, defines a link from node i i to node j. Otherwise, $A(i, j) = 0$ means no link. We assume $A(i, i) = 0$, i.e., no self-connecting link is allowed in this study. There are N_{Obj} costs, i.e., $C_k(i, j), k = 1, \dots, N_{Obj}$, associated with each link $A(i, j)$, and $C_k(i, j)$ will be used to calculate the kth objective of a path. Here, the route network $G(V, E)$ is called static, because both $A(i, j)$, and $C_k(i, j)$ are fixed and will not change during the procedure of path optimization.

In MOPOP, a pair of origin and destination nodes $[O, D]$ are specified, and then the goal is to find Pareto optimal paths from O O to D. Suppose a candidate path is recorded as an integer vector whose element $P(i) = j$ means node j is the ith node in the path, , $i = 1, \dots, N_P$ and $j = 1, \dots, N_N$, where N_P tells how many nodes, including the origin and the destination, are included in the path. Obviously, $P(1)$ is the origin and $P(N_P)$ is the destination, i.e., $P(1) = O$ and $P(N_P) = D$. Since no loop is allowed in this study, no node can appear in a path for more than once, i.e.,

$$P(i) \neq P(j), if \ i \neq j, \forall i, j \tag{1}$$

For a given path P, we can calculate its total cost from the origin to the destination in terms of each objective

$$f_k(P) = \sum_1^{N_P - 1} C_k[P(i), P(i + 1)], k = 1, \dots, N_{Obj} \tag{2}$$

where $f_k(P)$ is the kth objective of an MOPOP. Please note that the problem description of this study is in a general form, and depending on what real world MOPOP is to be resolved in an application, the objectives $f_k(P), k = 1, \dots, N_{Obj}$, may be given relevant definitions of real-world meanings, such as travelling time, physical distance, fuel consumption and road service charge.

Then, a general mathematical formulation of MOPOP can be given as follows:

$$\min_P \left[f_1(P), f_2(P), \dots, f_{N_{Obj}}(P) \right]^T \tag{3}$$

subject to Eq. (1)

$$P \in \Omega_P \tag{4}$$

where Ω_P is the set of all possible paths connecting the given pair of $[O, D]$.

A Pareto-optimal path P^* to the above problem is such that there exists no P that makes

$$f_i(P) \leq f_i(P^*), \forall\, i = 1, \dots, N_{\text{Obj}} \tag{5}$$

$$f_j(P) < f_j(P^*), \text{for at least one } j \in \left[1, \dots, N_{\text{Obj}}\right] \tag{6}$$

The projection of such a P^* in the objective space, i.e., the point $\left[f_1(P^*), f_2(P^*), \dots, f_{N_{\text{Obj}}}(P^*)\right]$, is called a Pareto point. For the above MOPOP, there is usually a set of Pareto-optimal paths, and the projection of this set in the objective space is called the Pareto front. Assume there are N_{PP} Pareto-optimal paths to the above MOPOP, then, the complete Pareto front will have N_{PP} Pareto points. Although it is possible that some Pareto points might happen to have exactly the same coordinates in the objective space (i.e., some Pareto optimal paths share the same $\left[f_1(P^*), f_2(P^*), \dots, f_{N_{\text{Obj}}}(P^*)\right]$, in this study, we still view them as different Pareto points as long as they are associated with different Pareto optimal paths.

Basically, MOPOP is a typical category of discrete combinational optimization problem and finding the complete Pareto front of MOPOP is a challenging task. Although there are many methods which may be used to resolve the above MOPOP, most of them can only find a partial of approximated Pareto front. Fig. 1 gives an example about the performance of some methods for MOPOP in terms of finding Pareto optimal paths. As shown in Fig. 1, the well-known AOF method can only find a partial Pareto front composed of 4 convex Pareto points, and misses out many other Pareto points, which are probably better tradeoff between two objectives for decision-makers. NSGA-II, because of its stochastic nature, outputs an approximated Pareto front where 4 associated solutions are not Pareto optimal at all. According to the theoretical proves of [30], we know the results of method in Ref. [30] should be the true, complete Pareto front to the MOPOP of the left-hand side subplot. However, the MOPOP in Fig. 1 has a very small problem scale, e.g., it has only $N_{\text{N}} = 49$ nodes and $N_{\text{Obj}} = 2$ objectives. It is not clear how the method of [30] could perform in large-scale MOPOP, but it does seem to have an issue of computational efficiency if applied to large-scale MOPOP.

Actually, in the existing literature, very few results have been reported on testing methods on large-scale MOPOP, partially because we do not have any proper large-scale MOPOP whose complete Pareto front we have precisely known in advance. Without such knowledge on complete Pareto front, testing a method on large-scale MOPOP does not make much sense in terms of performance assessment. Therefore, there is a need for benchmark test problems of large-scale MOPOP, whose complete Pareto front must be known before running any MOPOP method. Such benchmark test problems will put us in a good position to evaluate various methods for those discrete combinational problems that can be converted into MOPOP.

3. Benchmark test problem toolkit

The core or engine of benchmark test problem toolkit for MOPOP is a network model, which should be able to generate a network with any N_{N}, N_{L} (of course N_{L} should be subject to certain conditions depending on N_{N}, e.g., $N_{\text{L}} \leq N_{\text{N}} \times (N_{\text{N}} - 1)/2$ if no more than one link can be set up between a

pair of nodes), N_{Obj}, and N_{PP}, and the most important, all the N_{PP} Pareto optimal paths between a given $[O, D]$ pair in a generated network must be easily and manually deduced out without the help of any MOPOP method.

Here, suppose we need to generate a route network with N_{NN} nodes and N_{L} links, each link has N_{Obj} costs $C_{\text{k}}(i, j)$, and between a given pair of nodes, say $[O, D]$, there should exactly exist N_{PP} Pareto optimal paths in terms of N_{Obj} objective functions, which are calculated based on $C_{\text{k}}(i, j)$, according to Eq. (2).

Assume N_{PP} sets of $\left[f_{1,\text{m}}, f_{2,\text{m}}, \dots, f_{N_{\text{Obj}},\text{m}}\right]$ values are given, $m = 1, \dots N_{\text{PP}}$. For any $m \in [1, \dots, N_{\text{PP}}]$, there exists no $n \in [1, \dots N_{\text{PP}}]$, such that

$$f_{i,n} \leq f_{i,m}, \forall\, i = 1, \dots N_{\text{Obj}} \tag{7}$$

$$f_{i,n} < f_{i,m}, \text{ for at least one } j \in \left[1, \dots N_{\text{Obj}}\right] \tag{8}$$

In other words, these N_{PP} sets of $\left[f_{1,\text{m}}, f_{2,\text{m}}, \dots, f_{N_{\text{Obj}},\text{m}}\right]$ are not Pareto dominated by each other. So, these N_{PP} sets of $\left[f_{1,\text{m}}, f_{2,\text{m}}, \dots, f_{N_{\text{Obj}},\text{m}}\right]$ will be used as the objective values of N_{PP} Pareto optimal paths in a network to be generated by the model of this paper.

Under the above specified requirements, our model will generate a somehow random network by three main steps.

Step 1: Set up two nodes first, one as O, and the other as D; set up N_{PP} direct dummy links between $[O, D]$; each dummy link has N_{Obj} costs, and for the mth dummy link, $m = 1, \dots N_{\text{PP}}$, set its kth cost equal to $f_{k,\text{m}}$, $k = 1, \dots N_{\text{Obj}}$, i.e., the mth dummy link has a costs vector $\left[f_{1,\text{m}}, f_{2,\text{m}}, \dots, f_{N_{\text{Obj}},\text{m}}\right]$.

Step 2: Randomly insert some nodes into each of these N_{PP} direct dummy links between $[O, D]$, in order to divide each dummy link into some true links for route network. Assume we want to insert $N_{\text{ISN}} > 0$ nodes into the mth dummy link to generate $(N_{\text{ISN}} + 1)$ true links. Then, we assign random positive values to $\left[c_1(l), c_2(l), \dots, c_{\text{Obj}}(l)\right]$ for the lth true link resulting from inserting nodes, $l = 1, \dots N_{\text{ISN}} + 1$, and make sure

$$\sum_{l=1}^{N_{\text{ISN}}+1} c_k(l) = f_{k,m}, k = 1, \dots N_{\text{Obj}} \tag{7}$$

Then, we update $A(i, j)$ and $C_{\text{k}}(i, j)$ accordingly.

Step 3: Assume after inserting nodes into dummy links, we still need N_{EN} extra nodes and N_{EL} extra links, so that the generated network will have N_{N} nodes and N_{L} links in total. To add an extra link, we randomly choose two nodes between which there is no direct link. Assuming nodes i and j are chosen, then we set $A(i, j) = 1$, and assign a value to $C_{\text{k}}(i, j)$ according to certain rules that can guarantee any path going through link $A(i, j)$ will definitely not be Pareto optimal (we will give and explain some examples of such rules in Sub-section 4.3). To add an extra node, first we randomly choose an existing link, and then insert a node to divide it into two new links. If the chosen link comes from a dummy link, then spread the costs of the chosen link over the two new links. If the chosen link is an extra link newly added, then set up the costs of the two new links according to certain rules to avoid resulting in any new Pareto optimal path. At last, update $A(i, j)$ and $C_{\text{k}}(i, j)$ accordingly.

The model above can generate a route network with N_N nodes and N_L links, and between node pair $[O, D]$ there are exactly N_{PP} Pareto optimal paths, and the mth Pareto optimal path has pre-given objective values of $\left[f_{1,m}, f_{2,m}, \dots, f_{N_{Obj,m}} \right]$ as required. Fig. 2 gives a simple example to illustrate how the proposed network model can generate a MOPOP for the purpose of testing various path optimization methods. Fig. 3 summarizes the toolkit as a flowchart, which particularly shows the toolkit has a good control on key features of generated test case of MOPOP. For example, those nodes added in Step 2 mainly aim to adjust certain features of N_{PP} Pareto optimal paths, while those nodes generated in Step 3 will not increase or reduce the number of Pareto optimal paths, and will not cause any change to the shape of Pareto front, either. In next section, we will explain in detail how to control and tune such key features of generated MOPOP. Please note that, in this study of toolkit, like some other theoretical studies on algorithms [30–37], MOPOP is defined in an abstract way, not referring to any specific real-world applications. If we want to use the reported toolkit to study certain real-world MOPOP, we need to introduce some problem-specific measures and/or constraints to modify the reported network model. For example, in many real transportation systems, a node usually has no more than 4 links, a link may be directed, and the costs of a link may be within certain ranges according to the physical meanings of objectives.

4. Analysis of the toolkit

4.1. Scalability of toolkit

The problem scale of MOPOP is largely dependent on N_N, N_L, N_{Obj} and N_{PP}. Basically, the larger the values of these, the larger the problem scale. Fortunately, the network model in Section 3 uses N_N, N_L, N_{Obj} and N_{PP} as major model parameters, and with a given set of N_N, N_L, N_{Obj} and N_{PP} values, the model will generate a MOPOP with relevant problem scale. In particularly, many-objective problems (where $N_{Obj} \gg 3$) rather than multi-objective problems (where N_{Obj} is often 2 or 3, barely larger than 10) have now become the focus of MOOP research. The reported toolkit can easily generate test problems with $N_{Obj} > 100$. No matter how large the problem scale is, no need of help from any MOPOP method, we know for sure that between $[O, D]$ in the generated route network, there are N_{PP} Pareto optimal paths, which are composed by those links resulted from dividing N_{PP} dummy links, and have pre-given objective values $\left[f_{1,m}, f_{2,m}, \dots, f_{N_{Obj,m}} \right]$, $m = 1, \dots N_{PP}$. Therefore, the reported toolkit has very good scalability. It should be noted that there might be some limits for setting up N_N, N_L, N_{Obj} and/or N_{PP}, depending on which application area is targeted. For example, if we want to generate test networks similar to city route networks, then N_N might be about thousands, and N_L is about $2N_N$ to $3N_N$, because in a city route network, a node usually has 2 to 4 links. If we want to generate test networks to study free-flight network of civil aviation in a country, then N_N should be the number, or a few times of the number of airports in the country, and N_L could be up to $N_N(N_N - 1)/2$. Such application-dependent setup (or limit) of N_N and N_L is somehow out of the scope of this study, whose focus is a general model of generating test networks.

4.2. Shape of generated complete pareto front

As is well known, the shape of Pareto front is a main factor determining how difficult a MOOP is [46–52]. Therefore, a good multi-objective test problem toolkit is supposed to have a good control in the shape of Pareto front for generated test problems. For continuous numerical optimization, there have been quite a few toolkits that can generate test problems with purpose-designed, sophisticated Pareto front shape, in order to fully explore and evaluate the capability and performance of MOOP methods. Unfortunately, for discrete combinational optimization, such as MOPOP, there lacks such a toolkit to generate a test problem with certain desired Pareto front shape. For some kinds of

discrete combinational MOOPs, such as knapsack problem and path optimization, some researchers have reported some methods that can find the complete Pareto front [30–45], and thus their case studies may be used as benchmark problems to test and evaluate other methods. However, their case study problems have little control in Pareto front shape. Actually, it is not clear at all if they paid any attention to control Pareto front shape when setting up their case study problems [30–45]. Basically, it is something like "whatever shape, as long as there is a test problem". Differently, the toolkit of Section 3 has a full control in Pareto front shape of generated MOPOP. Basically, any shape we can imagine and want, convex, concave, smooth, sharp, zigzag, piece-wise, with local attractions evenly distributed or clustered, the reported toolkit is always capable generating MOPOP with the desired Pareto front shape. This is achieved in a very straightforward way, because desired Pareto front shape is represented by objective values $\left[f_{1,\mathrm{m}}, f_{2,\mathrm{m}}, \dots, f_{N_{\mathrm{Obj,m}}}\right], m = 1, \dots N_{\mathrm{PP}}$, which are parameters to the network model of Section 3. In other words, we sample a desired Pareto front shape for N_{PP} times, in order to get N_{PP} sets of objective values $\left[f_{1,\mathrm{m}}, f_{2,\mathrm{m}}, \dots, f_{N_{\mathrm{Obj,m}}}\right]$, and then use such values to set up those N_{PP} Pareto optimal paths in the generated route network.

For continuous numerical optimization, many typical challenging Pareto front shapes have been suggested with detailed mathematical descriptions [46–51]. The toolkit of Section 3 for MOPOP can take advantage and make full use of such results of continuous numerical MOOP. If we want MOPOP to have a Pareto front shape reported in Refs. [46–51], we simply use the associated mathematical formula to generate such a shape. Then, we choose N_{PP} points on the shape, and use their coordinates to set up our network model parameters $\left[f_{1,\mathrm{m}}, f_{2,\mathrm{m}}, \dots, f_{N_{\mathrm{Obj,m}}}\right], m = 1, \dots N_{\mathrm{PP}}$. Then, according to the model of Section 3, the projections of those N_{PP} Pareto optimal paths of the generated MOPOP will have the desired shape in the objective space.

Fig. 4 and Fig. 5 gives two simple examples, one has a concave shape, and the other has a convex shape. Basically, for a MOPOP with N_{Obj} objectives, we say its Pareto front is concave if there exists at least one Pareto point, for which there exist another N_{Obj} Pareto points such that this Pareto point is above the hyper-plane determined by these N_{Obj} Pareto points. Otherwise, we say the Pareto front is convex. In the example of Fig. 4, $N_{\mathrm{Obj}} = 2$ and the Pareto front is composed 4 Pareto points. Pareto point (8,9) is above the straight line determined by Pareto points (3,12) and (12,3). So, the Pareto front is concave. In Fig. 5, $N_{\mathrm{Obj}} = 2$ and the Pareto front is also composed 4 Pareto points. But no Pareto point is above any straight line determined by a pair of other Pareto points. So, the Pareto front is convex. Fig. 6 further illustrates how to get a desirable Pareto front shape for generated MOPOP.

4.3. Distance between pareto points and pareto dominated points

Besides problem scale and Pareto front shape, how closely Pareto points and Pareto dominated points are mixed together in the objective space can also influence the complexity of MOOP. Basically, if most Pareto dominated points are far away from the Pareto front, this often implies there could be a good chance to employ certain heuristic rules to filter out such Pareto dominated points at an early stage of optimization, in order to effectively narrow down search space, which is good news for MOOP methods. Otherwise, i.e., if Pareto points and Pareto dominated points are closely mixed, this often implies a high chance for MOOP methods to be distracted by local optimal solutions. For example, in the MOPOP of Fig. 4, assuming we employ a heuristic rule of choosing the link with the smallest costs to add to a path, then we can avoid all Pareto non-optimal paths. However, this heuristic rule will not work well in the MOPOP of Fig. 5. Assume we have travelled along path $O - 3 - 7$, and now we need to choose which way to go next, link $(7, D)$ or link $(7,4)$? If

$C_1(i,j)$ is our major concern, then we will be distracted to link (7,4), which will lead to a Pareto non-optimal path.

A good test problem toolkit is supposed to also have a good control in the average distance between Pareto points and Pareto dominated points. Fortunately, the network model of Section 3 has such a capability by introducing necessary rules in Step 3 when assigning value to the $C_k(i,j)$ of extra links. By employing different rules in Step 3, we can roughly control the average distance between Pareto points and Pareto dominated points. Here are 3 rules for example to set up the costs of extra link.

Rule 1: $C_k(i,j)$ can be any value that satisfies

$$C_k(i,j) \leq \max(f_{k,1},\dots,f_{k,N_{PP}}), k = 1,\dots,N_{Obj} \tag{10}$$

Rule 2: $C_k(i,j)$ can be set as small as possible, as long as $\left[C_1(i,j),\dots,C_{N_{Obj}}(i,j)\right]$ is dominated by at least one Pareto point, i.e., there exists at least one $m \in [1,\dots N_{PP}]$, so that

$$C_k(i,j) \geq f_{k,m}, k = 1,\dots,N_{Obj} \tag{11}$$

$$C_h(i,j) > f_{h,m}, \text{for at least one } h \in \left[1,\dots N_{Obj}\right] \tag{12}$$

Rule 3: Find the shortest single-objective paths in terms of each of the N_{Obj} objective function between node i and node O, and between node j and node D; also find the shortest single-objective paths between node i and node D, and between node j and node O; then set up $C_k(i,j)$ as small as possible, making sure combining link (i,j)) with those shortest single objective paths between $[O,i]$ and $[j,D]$, and between $[O,j]$ and $[i,D]$, will not generate any new Pareto optimal path between $[O,D]$. In particular, we have the follow Theorem below for Rule 3.

Theorem. *Suppose among all paths that are constructed by combining those single-objective optimal sub-paths and the extra link to connect O and D, $f_{SO,k}$ is the smallest value for the kth objective, $k = 1,\dots,N_{Obj}$. If the point $\left[f_{SO,1},\dots,f_{SO,N_{Obj}}\right]$ is dominated by at least one of those N_{PP} Pareto points set up in Step 1 and Step 2 of the network model, then, adding the extra link will not introduce any new Pareto points, i.e., any path travelling through the extra link newly added cannot be Pareto optimal.*

Proof. *Assume a path travels through the extra link from O to D, and it has costs $\left[f_1,\dots,f_{N_{Obj}}\right]$. Since $f_{SO,1},\dots,f_{SO,N_{Obj}}$ are associated with those paths that are constructed by combining those single-objective optimal sub-paths and the extra link to connect O and D, clearly, one has $f_k \geq f_{SO,k}, k = 1,\dots,N_{Obj}$. Because point $\left[f_{SO,1},\dots,f_{SO,N_{Obj}}\right]$ is dominated by at least one of those N_{PP} Pareto points, then point $\left[f_1,\dots,f_{N_{Obj}}\right]$ must also be dominated by the same Pareto points. Therefore, the path travelling through the extra link cannot be Pareto optimal.*

Fig. 7 gives a simple illustration about the above Theorem for Rule 3. In Fig. 7, $N_{Obj} = 2$, and $N_{PP} = 4$. P_A and P_B are associated with the paths which are constructed by combining single-objective optimal sub-paths and the extra link, and have the smallest 1st objective and the smallest 2nd objective, respectively. Obviously, any path travelling through the extra link can be located only within Zone II of Fig. 7. If P_C is dominated by at least one Pareto point, i.e., P_C is located outside of Zone I in Fig. 7, then, any point in Zone II will also be dominated. Thus, all paths that travel through the extra link will not be Pareto optimal. Basically, Rule 1 will generate MOPOP with Pareto dominated points far away from Pareto front. In the case of $N_{Obj} = 2$, Pareto dominated points under Rule 1 will be distributed in Zone I of Fig. 8.

In theory, under Rule 2, Pareto dominated points could be in any place of Zone I to Zone IV in Fig. 8, assuming $N_{\text{Obj}} = 2$ for example. In practice, they are more likely to be scattered in Zone I, Zone III and Zone IV. Compared with points in Zone I, a point in Zone III or Zone IV is more difficult for MOOP methods to distinguish from Pareto points. However, if we can find out those best single-objective solutions, which is much easier than resolving MOOP, we can then introduce a heuristic rule to help MOOP methods to filter out all points in Zone III and Zone IV. For instance, in Fig. 8, if we know Pareto points P_1 and $P_{N_{\text{PP}}}$, then we know any point with $f_2 > f_{2,1}$ or $f_2 > f_{2,N_{\text{PP}}}$, is not Pareto point. A more general description of the above heuristic rule is as follows. Assume S_k is the best solution in terms of the kth objective function, $k = 1, \ldots, N_{\text{Obj}}$, and S_k has a projection of $\left[f_{1,k}^*, \ldots, f_{1,N_{\text{Obj}}}^* \right]$ in the objective space. Then, if a solution for at least one $i \epsilon [1, \ldots, N_{\text{Obj}}]$, has its f_i satisfying

$$f_i > \max_{j \neq i, j \epsilon [1, \ldots, N_{\text{Obj}}]} f_{i,j}^* \tag{13}$$

the solution is not Pareto optimal.

Obviously, if a Pareto dominated point is within Zone II of Fig. 8, then the above heuristic rule can help nothing, which means it is the most difficult for MOOP methods to distinguish such a point from Pareto points. Under Rule 3 for Step 3 of our network model in Section 3, it is more likely to generate MOPOP with more Pareto non-optimal paths projected into Zone II, given $N_{\text{Obj}} = 2$ for the sake of simple illustration. Therefore, with Rule 1 to Rule 3, the toolkit of Section 3 can have a control to some extent in the average distance between Pareto points and Pareto dominated points. Furthermore, we may develop some more relevant rules for Step 3, in order to achieve a better control, which is no doubt a direction for improving the reported toolkit in future research.

4.4. Distribution of pareto points on pareto front

The distribution of Pareto points on Pareto front may also influence the complexity of MOOP. In general, if Pareto points are distributed more smoothly and evenly on Pareto front (see Fig. 9(a) for example), then it could be relatively easier for MOOP methods to find most Pareto points that are important and/or representative. If Pareto points are highly clustered and unevenly distributed on Pareto front (see Fig. 9(b) for example), then, it might be difficult for MOOP methods to find any Pareto point located in a niche cluster. For continuous numerical MOOP, the distribution of Pareto points on Pareto front might seem like the same issue as Pareto front shape. However, for discrete combinational MOOP, such as MOPOP, they are two issues quite different from each other, because a same Pareto front shape can give many different sets of N_{PP} sample points. Fortunately, because the network model of Section 3 allows us to sample a desirable Pareto front in any way we like to set up model parameters $\left[f_{1,m}, \ldots, f_{N_{\text{Obj}},m} \right], m = 1, \ldots N_{\text{PP}}$, we can have a good control in the distribution of Pareto points on Pareto front. For example, for the case of Fig. 9(b), we can particularly sample a few points on the niche cluster, so that we can generate suitable MOPOP to test a method about its capability of searching for niche clusters on Pareto front.

One might argue that a niche cluster on Pareto front does not necessarily mean it is associated with less Pareto optimal solutions. This is true because a Pareto point is the projection of one or multiple Pareto optimal solutions (if they have the same objective values) in objective space, and if a niche cluster is associated with many Pareto optimal solutions, then a MOOP method usually stands a good chance to find one or a few Pareto points in this niche cluster. Fortunately, this does not affect the practicability of the proposed MOPOP toolkit at all. Basically, we want a toolkit to generate complex test problems, as simple test problems can hardly distinguish MOOP methods in terms of

their performance. Clearly, a niche cluster associated with many Pareto optimal solutions implies less complexity, so, it is less useful from the viewpoint of evaluating MOOP methods.

4.5. Pareto optimal paths similar in appearance

Please note that the network model of Section 3 will generate N_{PP} Pareto optimal paths completely separated from each other, i.e., the generated N_{PP} Pareto optimal paths share no common nodes (except O and D) or links with each other.

However, sometimes we might want a test problem to study the local searching capability of a MOOP method. For example, it is important for genetic algorithm (GA) to have a good capability of identifying, inheriting and protecting good common genes in chromosomes. Those N_{PP} Pareto optimal paths generated by the network model of Section 3 simply share no common genes.

First, despite the above demerit, the network model of Section 3 is still useful, because N_{PP} Pareto optimal paths sharing no common nodes or links often means more difficult to find complete Pareto front, i.e., the generated test problem is a more challenging MOPOP. As is well known, in path optimization, it is a popular practice to modify already-found optimal/good paths in order to find more other optimal/good paths. For example, in the k-shortest paths problem, the first j shortest paths are often used to calculate the $(j-1)$th shortest path, $1 \le j < k$ [56,57], and in dynamical path optimization, the optimal path of current time instant is often used to calculate the optimal path of next time instant [58,59]. The MOPOP generated by the model of Section 3 can reveal the disadvantage of such a popular practice in path optimization, and therefore demands developing more effective path optimization methods.

Second, for the purpose of testing local searching capability, Step 1 of the network model in Section 3 can be modified to some extent, in order to allow some Pareto optimal paths to share some common nodes (besides O and D) and links. For instance, assume we want the ith Pareto optimal path and the jth Pareto optimal path share some common nodes and links. Then, in Step 1 of modified network model, we divide the objective values of each path into three parts as follows,

$$f_{k,\text{i}} = x_{\text{k}} + y_{\text{k}} + z_{\text{k,i}}, k = 1, \dots, N_{\text{Obj}} \tag{14}$$

$$f_{k,\text{j}} = x_{\text{k}} + y_{\text{k}} + z_{\text{k,j}}, k = 1, \dots, N_{\text{Obj}} \tag{15}$$

Now, we set up 4 path segments between O and D, one path segment connects O, one path segment connects D, and the other two path segments are parallel in the middle. We let the path segment connecting O have objective values of $\left[x_1, \dots x_{N_{\text{Obj}}}\right]$, the path segment connecting D have objective values of $\left[y_1, \dots y_{N_{\text{Obj}}}\right]$, and the other two path segments have $\left[z_{1,\text{i}}, \dots z_{N_{\text{Obj}},\text{i}}\right]$ and $\left[z_{1,\text{j}}, \dots z_{N_{\text{Obj}},\text{j}}\right]$, respectively. Then, we go on to Step 2 of the model in Section 3. Fig. 10 illustrates the network model with modified Step 1.

So, Step 1 of the model in Section 3 can be modified to allow Pareto optimal paths to share common nodes and links. Actually, the model of Section 3 may be further modified to allow more complicated situations of sharing common nodes and links between Pareto optimal paths, and this is no doubt a direction worth further investigation in future.

4.6. Global optima and local optima

In the study of MOPOP, global optima are N_{PP} Pareto optimal paths, while a local optimum may be defined as a path which has a sub-path that can dominate a relevant sub-path in a Pareto optimal path. Such a local optimum might distract a searching method from the true Pareto optimal path.

Basically, in the network model of Section 3, (i) global optima are generated in Step 1 and Step 2, and Step 1 and Step 2 can generate only global optima; (ii) Step 3 can generate only non-Pareto-optimal paths, and local optima may be generated in and only in Step 3.

A simple way of generating a local optimum is described as follows. After an extra link (whose costs are such that any path travelling through this link will be dominated by at least one of the N_{PP} Pareto optimal paths generated in Step 1 and Step 2) is added in Step 3, we can add an extra node to the link, so, two new links appear. Then, we divide the costs of the original link so that a new link will have very small costs. In this way, local optima may be generated, as illustrated in Fig. 11. There are $N_{\mathrm{PP}} = 2$ Pareto optimal paths in Fig. 11, i.e., path $O \rightarrow 1 \rightarrow 3 \rightarrow D$ and path $O \rightarrow 2 \rightarrow 4 \rightarrow D$, and their costs are [6,9] and [9,6], respectively. First, an extra link is added between node 2 and node 3, and its costs are set as (5,5). Clearly, any path that includes this link is not Pareto optimal in Fig. 11. Then, an extra node, node 5 is added to this link, diving it into two new links, one new link is given costs (1,1), and the other new link (4,4). Thus, for a searching method which favors such links with small costs, it will be more likely to be distracted by going from node 2 to node 5 rather than to node 4.

Please note that the definition of local optima in MOPOP is somehow dependent of searching method under test. The major effect of local optima is to distract a searching method from global optima. From a width-first searching method's point of view, the example of Fig. 11 has no local optima. But if we add an extra link directly between node 1 and node D, we will then get a local optimum. In other words, depending on what kind of searching methods are tested, we may adopt relevant measures for adding extra links and nodes in Step 3 of the reported network model.

4.7. Number of intermediate nodes in pareto optimal paths

The percentage of N_{N} nodes used to insert into dummy links in Step 2 of Section 3 will also influence the complexity of generated MOPOP. Basically, if a smaller proportion of N_{N} nodes are inserted into dummy links in Step 2, then on average, a Pareto optimal path will have fewer intermediate nodes. Thus, a heuristic rule (e.g., depth-first search) that favors those nodes closer to the destination in terms of intermediate nodes could bring an advantage to problem resolving. If a larger proportion of N_{N} nodes are inserted into dummy links in Step 2, then in general, more of N_{L} links are included in Pareto optimal paths. Thus, a heuristic rule (e.g., crossover in genetic algorithm) that explore common links could lead to fast convergence.

A difficult test problem should be challenging to both heuristic rules. One way that might lead to generate such a challenging test problem is to restrict the total number of intermediate nodes in all Pareto optimal paths, and at the same time, to allow the number of intermediate nodes in a single Pareto optimal path to vary within a wide range. Therefore, the total number of intermediate nodes in all Pareto optimal paths and the range for the number of intermediate nodes in a single Pareto optimal path are two model parameters to control the complexity of generated test problem.

4.8. Complexity of the reported network model

Basically, Step 1 and Step 2 of the reported network model in Section 3 are straightforward, i.e., set up N_{PP} Pareto optimal paths, and then randomly insert a given number of nodes to these N_{PP} Pareto optimal paths. Here, we mainly analyze the complexity of Step 3. Depending on the rule

adopted to assign costs to an extra link, Step 3 will consume quite different computational time and then have different complexity.

Suppose $\alpha \times N_\mathrm{N}$ nodes and $\beta \times N_\mathrm{L}$ are included in those N_PP Pareto optimal paths in Step 1 and Step 2, $0 < \alpha < 1, 0 < \beta < 1_\mathrm{N}$, then we need to add $(1 - \alpha) \times N_\mathrm{N}$ extra nodes and $(1 - \beta) \times N_\mathrm{L}$ extra links in Step 3. The most challenging part is to assign costs to extra links, which provides a guarantee that the generated network has and only has those N_PP paths set up in Step 1 and Step 2 as Pareto optimal paths.

If we adopt Rule 1 in Sub-section 4.3 to assign costs to extra links, then we simply generate $N_\mathrm{Obj} \times (1 - \beta) \times N_\mathrm{L}$ random values which satisfy Condition (10). So, the associated complexity may be assessed as $O(N_\mathrm{Obj} \times N_\mathrm{L})$.

If we adopt Rule 2 in Sub-section 4.3 to assign N_Obj costs to an extra link, then for a set of randomly generated costs, we need to make $N_\mathrm{Obj} \times N_\mathrm{PP}$ comparisons according to Conditions (11) and (12) in the worst case, in order to guarantee those N_PP Pareto optimal paths set up in Step 1 and Step 2 will not be dominated. Suppose on average, we need to generate N_SRC sets of random costs before we can find a set which satisfies Conditions (11) and (12). So, for $((1 - \beta) \times N_\mathrm{L}$ extra links, the associated computational burden may be assessed as $O(N_\mathrm{Obj} \times N_\mathrm{PP} \times N_\mathrm{SRC} \times N_\mathrm{L})$.

If we adopt Rule 3 in Sub-section 4.3 to assign N_Objj costs to an extra link, then first we need to find 4NObj single-objective optimal sub-paths in the so-far network. Depending on which algorithm is employed, the computational complexity to find a single-objective optimal sub-path may vary. Here assuming Dijkstra's algorithm is employed and the so-far network has $N_\mathrm{N,SF}$ nodes and $N_\mathrm{L,SF}$ links, then, the complexity of finding a single-objective optimal sub-path can be assessed as $O(N_\mathrm{L,SF} + N_\mathrm{N,SF} \times \log N_\mathrm{N,SF})$ [60]. Since $N_\mathrm{N,SF} \leq N_\mathrm{N}$ and $N_\mathrm{L,SF} \leq N_\mathrm{L}$ for all so-far networks, we may use $O(N_\mathrm{L} + N_\mathrm{N} \log N_\mathrm{N})$ as an upper-bound for finding a single-objective optimal sub-path. Assume after combining those single-objective optimal sub-paths with the extra link, the associated smallest value for the kth objective is $f_\mathrm{SO,k}, k = 1, \dots, N_\mathrm{Obj}$. Then, we need to compare point $[f_\mathrm{SO,1}, \dots, f_\mathrm{SO,N_\mathrm{Obj}}]$ with those N_PP Pareto points, to check if it is dominated by at least one Pareto point. Again, assume on average we need to generate NSRC sets of random costs for the extra link before point $[f_\mathrm{SO,1}, \dots, f_\mathrm{SO,N_\mathrm{Obj}}]$ can be dominated by at least one Pareto point. So, there are $N_\mathrm{Obj} \times N_\mathrm{PP} \times N_\mathrm{SRC}$ comparisons for each extra link. Plus the computational burden of finding single-objective optimal sub-paths, thus for $(1 - \beta) \times N_\mathrm{L}$ extra links, the total computational burden of Step 3 can be assessed as $O\left(\left(N_\mathrm{Obj} \times N_\mathrm{PP} \times N_\mathrm{SRC} + 4N_\mathrm{Obj} \times (N_\mathrm{L} + N_\mathrm{N} \log N_\mathrm{N})\right) \times N_\mathrm{L}\right)$.

Clearly, on one hand, adopting Rule 3 makes Step 3 the most timeconsuming when compared with Rule 1 or Rule 2. On the other hand, according to Sub-section 4.3, adopting Rule 3 may generate much more challenging test problems.

The complexity of Rule 2 and Rule 3 can be reduced if the network model is acceptable with a low degree of randomness. In other words, we do not try completely random values as costs for extra links. For example, under Rule 2, we randomly choose one of those N_PP Pareto points, add small random positive values to the costs of the chosen Pareto point, and then use the new values as the costs for an extra link. In this way, we always have $N_\mathrm{SRC} = 1$, so, the complexity of Step 3 under Rule 2 becomes $O(N_\mathrm{Obj} \times N_\mathrm{PP} \times N_\mathrm{L})$ Similarly, under Rule 3, we can make $N_\mathrm{SRC} = 1$ if the costs of an extra link are chosen only from certain ranges that are determined by those N_PP Pareto points and the minimal objective values of those single-objective optimal sub-paths. Then, the complexity of

Step 3 under Rule 3 becomes $O\left(\left(N_{\mathrm{Obj}} \times N_{\mathrm{PP}} + 4N_{\mathrm{Obj}} \times (N_{\mathrm{L}} + N_{\mathrm{N}} \log N_{\mathrm{N}})\right) \times N_{\mathrm{L}}\right)$ given that $N_{\mathrm{PP}} \ll 4(N_{\mathrm{L}} + N_{\mathrm{N}} \log N_{\mathrm{N}})$, then we have $O\left(N_{\mathrm{Obj}} \times (N_{\mathrm{L}} + N_{\mathrm{N}} \log N_{\mathrm{N}}) \times N_{\mathrm{L}}\right)$.

It should be noted that the complexity of generating a random test problem is different from the complexity of resolving a given test problem. The former is a concern of this manuscript, while the latter is a focus of algorithm design. For a given test problem, different algorithms (e.g., width-first-search, depth-first-search, best-first-search methods) may have rather different complexities, which is somehow out of the scope of this study.

5. Experimental results

5.1. MOPOP methods for test

In this section, we will demonstrate that the reported test problem toolkit can be used to evaluate the performance of methods for MOPOP. For demonstrating purposes, we will test three categories of MOPOP methods. The first category is capable of finding all Pareto optimal paths with a theoretical guarantee of optimality, and we choose the method reported in Ref. [30] as a representative. We denote this method as COM because it can find the complete Pareto front. COM can help to verify the correctness of the reported toolkit, i.e., if the results of COM are the same as those manually deduced Pareto fronts, then we know the toolkit works properly. Otherwise, there could be something wrong with the toolkit, theoretically or technically.

The second category is capable of finding some true Pareto points (i.e., a partial Pareto front), and we develop a simple AOF (aggregate objective function) method as a representative. In the AOF, we combine the N_{Obj} objectives into a single one as follows

$$f_{AOF} = \sum_{k=1}^{N_{\mathrm{Obj}}} w_k f_k \tag{16}$$

where $0 \leq w_k \leq 1$, and $0 < w_k$ for at least one k. In this study, once a MOPOP is given by the toolkit, we randomly generate 100 sets of $\left[w_1, \ldots, w_{N_{\mathrm{Obj}}}\right]$, in order to get 100 different aggregate objective functions f_{AOF}. Based on each f_{AOF}, we apply Dijkstra's algorithm of [60] to resolve a single-objective path optimization problem (SOPOP), so, we will get a best path in terms of the given f_{AOF}, and such a best path is a Pareto-optimal path to the original MOPOP. It often happens that some different sets of $\left[w_1, \ldots, w_{N_{\mathrm{Obj}}}\right]$ may result in a same Pareto-optimal path, so, the AOF method will usually output less than 100 Pareto-optimal paths.

The third category employs Pareto-compliant ranking-based (PCR) techniques to approximate Pareto front, and we choose genetic algorithm (GA) as a representative. Since PCR methods are nowadays the most popular for resolving various MOOPs, particularly, discrete combinational MOOPs, here, we develop three simple GAs for MOPOP, in order to better demonstrate how the reported toolkit can help to evaluate PCR methods.

All three GAs have the same chromosome structure, where the ith gene in a chromosome records the ith node in the path represented by the chromosome. In the first GA, denoted as GA1, a chromosome is always randomly initialized, i.e., when appending a new link to the end of so-far

path, we simply make a random choice between all those links that start from the end of so-far path. GA1 only adopts a mutation operator. In the mutation of GA1, we randomly choose two non-successive genes in a chromosome to be mutated, and if there exists a link in the route network between those two nodes recorded by those two non-successive genes, then we delete all genes between those two non-successive genes in the chromosome. GA1 has no crossover operator, because, as is well known, for many discrete combinational problems, crossover, if not properly designed, will be more destructive than helpful to convergence [54,55]. Fig. 12 illustrates the mutation operator in GA1.

The second GA, denoted as GA2, has exactly the same design as GA1, except that when initializing a chromosome, we introduce a heuristic rule that if a link, once appended to the end of so-far path, can achieve Pareto non-dominance when compared with other links that start from the end of so-far path, then this link will stand a better chance to be chosen to attach to the end of so-far path. Fig. 13 illustrates the heuristic rule in GA2 for initializing a chromosome.

The third GA, denoted as GA3, also has exactly the same design as GA1, except that a crossover operator is introduced. In the crossover of GA3, we randomly choose two chromosomes as parents first. Then we use the binary representation techniques in Ref. [55] to identify common gene sections. A common gene section means at least two successive genes in two chromosomes are the same. Then, we swap non-common gene sections between two parent chromosomes in order to produce two offspring chromosomes. Basically, if a common gene section often appears in many good chromosomes, it is very likely that this gene section is useful to construct Pareto optimal paths. The crossover operator here can well identify, inherit and protect such a useful gene section, and therefore leads to a good convergence performance [55]. Fig. 14 illustrates the crossover operator in GA3. One might argue that the above crossover operator could cause a node to appear more than once in a path, and therefore lead to an infeasible path. In this study, for the sake of simplicity, once an infeasible path is produced, we will discard it and replace it with a randomly initialized new chromosome. Actually, the possibility of causing infeasible chromosomes by the above crossover is not very high, because a node close to the source/destination in a rout network is not likely to appear as a node close to the destination/source in a path, which means the above crossover is not likely to cause such a node to appear twice in an offspring chromosome. For example, in Fig. 14, because node 7 is closer (in terms of number of intermediate links) to the destination in the route network, it is very likely that many initialized chromosomes reach node 6 before reaching node 7. Therefore, crossing over such parent chromosomes will not cause node 7 to appear twice in offspring chromosomes.

These three GAs employ the following common measures. When choosing a link to attach to the end of so-far path in chromosome initialization, if a link connects to the destination, then this link stands a better chance to be chosen. Besides, when initializing a chromosome, a node that needs to pass fewer intermediate nodes to reach the destination will be favored with a better chance of attaching to the end of so-far path. Elitism strategy is adopted in GAs, which means some best chromosomes in a generation will be directly passed on to the next generation. In contrast to an elitism strategy, an elimination strategy is used to replace some worst chromosomes in a generation by new, randomly initialized chromosomes. In each GA, when sorting and ranking those individuals in a generation, the techniques of NGSA − II in Ref. [19] were referred to.

It should be emphasized that this study is not concerned with developing effective methods for resolving MOPOP. It is possible that one could develop a much more effective GA than GA1 to GA3 by introducing more sophisticated designs, for example, introducing the heuristic rule of GA2 and the crossover operator of GA3 into GA1 simultaneously, and taking some feasibility-enforcing

measures in the crossover operator of GA3 (in order to avoid a node appearing more than once in a path). Instead, this study simply aims to demonstrate that the reported test problem toolkit can well distinguish differences in performance of different methods. Here GA1 to GA3 have quite different designs, which could be suitable for some route networks, but not very effective to some other route networks. With the reported test problem toolkit, we are possible to show the merits and demerits of those design techniques adopted by GA1 to GA3, respectively. In other words, this study is not to prove that GA1, GA2 or GA3 is effective to MOPOP, but to demonstrate that the reported test problem toolkit can examine whether a GA design technique for MOPOP is effective to what extent. When evaluate the performance of different GAs, usually, people compare those Pareto non-dominated solutions found by different GAs. However, due to the stochastic nature of GAs, we do not know if such Pareto non-dominated solutions are true Pareto optimal solutions, and what is worse, we even have no clue how many true Pareto optimal solutions are missed by GAs under test. Now, fortunately with the help of the reported test problem toolkit, we will know exactly which Pareto non-dominated solutions found by different GAs are true Pareto optimal solutions, and how many true Pareto optimal solutions are missed out. This allows us to evaluate the performance of different GAs (or GA design techniques) in a much more precise manner. In the following experiments, for each MOPOP, we apply COM once, and apply AOF, GA1, GA2 and GA3 for 100 times, respectively. For each GA, the population size is 200, the number of evolving generations is 500, the mutation probability is 0.2, 10% of best chromosomes in a generation will directly passed on to the next generation, and 10% of worst chromosomes in a generation will be replaced by randomly initialized new chromosomes. For GA2, a link with non-dominated costs is 3 times more likely to be chosen than a link with dominated costs. For GA3, the crossover probability is 0.4.

5.2. Comparison between different categories of MOPOP methods

In this sub-section, we aim to use the reported MOPOP toolkit to test three categories of MOPOP methods, and they are COM, AOF and GA1, capable of outputting complete, partial and approximate Pareto fronts, respectively. Since COM has a scalability issue [30], here we use the reported toolkit to generate MOPOPs with relatively small problem scales. We fix $N_{\mathrm{N}} = 40, N_{\mathrm{L}} = 120, N_{\mathrm{PP}} = 30$, and allow $N_{\mathrm{Obj}} = 2,4,6,8,10$. For each N_{Obj} value, we generate 100 MOPOPs. The average results of COM, AOF and GA1 are given in Table 1, where CT, N_{PPF} and R_{FCPF} mean computational time (in second), number of true Pareto points found by a method, and the rate of a method for finding the complete Pareto front (i.e., in how many of the 100 route networks of a given N_{Obj}, the method has found the complete Pareto front), respectively. From Table 1, one may have the following observations.

● COM can always find all of those $N_{\mathrm{PP}} = 30$ Pareto optimal paths for every MOPOP. COM has a theoretical guarantee to find complete Pareto front [30]. In this sub-section, the toolkit is supposed to generate MOPOPs with $N_{\mathrm{PP}} = 30$ Pareto optimal paths. Therefore, the results of COM prove that the reported toolkit works properly as supposed.

● However, the computational burden of COM soars up significantly as N_{Obj} increases. In the most complicated case of $N_{\mathrm{Obj}} = 10$, COM is the most computationally expensive. This, in some sense, implies the difficulty in designing an efficient method that is capable of finding complete Pareto front, and therefore explains why those methods of finding partial or approximate Pareto fronts are more popular in practice of MOOPs.

● The most computational efficient method is AOF. For a given MOPOP, AOF just needs to resolve 100 SOPOPs, which Dijkstra's algorithm of [60] can resolve very fast. The CT of AOF is also the most stable, as it only changes slightly as N_{Obj} increases.

● However, based on the linear sum of N_{Obj} original objectives according to Eq. (16), AOF cannot find any Pareto points located on those concave parts of Pareto front [4,12]. In a MOPOP generated by the toolkit with NPP sets of random $\left[f_{1,\text{m}}, \ldots, f_{N_{\text{Obj}},\text{m}}\right]$ values, the Pareto front often has some and even many concave parts, covering most Pareto points (see Fig. 2 for example). This explains why AOF finds the least true Pareto points, and stands the smallest chance the find complete Pareto front (it is possible only when the Pareto front is completely convex).

● It seems that GA1 could make a relatively good tradeoff between computational time and solution quality, i.e., in the most complicated case of $N_{\text{Obj}} = 10$, GA1 takes less CT than COM, and finds more true Pareto points than AOF. This might help to explain why PCR methods are the most popular in real-world MOOP applications.

● It should be noted that the experiments in this study mainly aim to demonstrate that the reported MOPOP toolkit can help to evaluate different MOPOP methods. To analyze a specific MOPOP method in more depth, more experiments will be needed. For example, we can change the way of choosing weights w_{k} of Eq. (16) for AOF (e.g., from randomly setting to evenly sampling), and we can also change the number of sets of w_{k} (basically, more sets of w_{k} could result in more true Pareto points to be found by AOF). For GA1, we may change the population size, the number of generations to evolve, and the probability of mutation, and then study how the computational time and solution quality might change accordingly. However, due to limited space, in this study, we will not go deeper with every specific MOPOP method, and in next sub-section, we will use the toolkit only to analyze some GA designing techniques, because of the popularity of PCR methods.

5.3. Comparison between 3 GAs

In this sub-section, we further carry out 4 sets of experiments to compare three different GAs, i.e., GA1, GA2 and GA3 given in Sub-section 5.1, in order to demonstrate that the reported toolkit may help to evaluate different GA designing techniques. In each set of experiments, 100 MOPOPs are randomly generated by the reported toolkit with $N_{\text{N}} = 200,300,400,500,600$, respectively. In all experiments, $N_{\text{L}} = 3N_{\text{N}}$, $N_{\text{Obj}} = 2$ and $N_{\text{PP}} = 100$. In a route network of experiment set 1 (ES1), $N_{\text{PP}} = 100$ Pareto optimal paths share no common node except O and D, and a link that does not belong to any Pareto optimal path has its costs set up according to Rule 2, Eq. (11) and Eq. (12) in Sub-section 4.3. In a route network of experiment set 2 (ES2), $N_{\text{PP}} = 100$ Pareto optimal paths again share no common node except O and D but a link that does not belong to any Pareto optimal path has its costs set up according to Rule 3 and Eq. (13) in Sub-section 4.3. In a route network of experiment set 3 (ES3), $N_{\text{PP}} = 100$ Pareto optimal paths may share some common nodes besides O and D by following the modification described in Sub-section 4.5, and a link that does not belong to any Pareto optimal path has its costs set up according to Rule 2, Eq. (11) and Eq. (12) in Sub-section 4.3. In a route network of experiment set 4 (ES4), $N_{\text{PP}} = 100$ Pareto optimal paths again may share some common nodes, but a link that does not belong to any Pareto optimal path has its costs set up according to Rule 3 and Eq. (13) in Sub-section 4.3. Table 2 summarizes the key features of these four experiment sets.

For the sake of simplicity but without jeopardizing demonstrating purposes, we set up other model parameters as simple as possible. In particular, we insert all N_{N} nodes (except O and D) into dummy links in Step 2 of network model, so, we only need to add some extra links in Step 3 of network model. According to Sub-section 4.6, a test problem generated in this way might not be very difficult to resolve. This is actually what we want here, because GA1 to GA3 are not professional, purpose-designed methods, but simply developed only for demonstrating the usefulness of the reported

toolkit. If we test GA1 $-$ GA3 on very difficult problems, we could end up seeing no difference between these three GAs.

In other words, the goal of this study is not to design difficult MOPOPs to beat all methods (or to design effective methods to resolve all MOPOPs), but to demonstrate that the reported toolkit can be properly tuned, in order to test certain features of a method. This study uses the reported test problem toolkit mainly to examine how many true Pareto optimal paths a GA could find. The mean results and standard deviation (SD) in terms of the number of true Pareto optimal paths found by GA1, GA2 and GA3 in four experiment sets are given in Tables 3–6, respectively, from which one may have the following observations.

● In ES1 of Table 3, GA1 has the worst performance, GA3 is better, and GA2 is the best. This is explainable. In a route network of ES1, the costs of a link that does not belong to any Pareto optimal path are set up according to Eq. (11) and Eq. (12), which mean the link itself is dominated by Pareto optimal paths, let alone by links in Pareto optimal paths. GA2 employs a heuristic rule that favors nondominated links. Therefore, when initializing chromosomes, GA2 stands a better chance to choose those links belonging to Pareto optimal paths, and as a result, GA2 achieves the best performance in three GAs. Actually, Table 3 shows that GA2 can almost find out all of the $N_{\mathrm{PP}} = 100$ Pareto optimal paths. Although the quality of an initialized chromosome in GA3 is as poor as in GA1, the crossover operator employed in GA3 can help, through evolution generation by generation, to identify, inherit and protect some of those links belonging to Pareto optimal paths. Therefore, GA3 has a better performance than GA1 in ES1.

● Actually, in all tests, GA3 outperforms GA1. One reason for this is because, as explained in the experimental setup, all N_{N} nodes (except O and D) are inserted into dummy links in Step 2 of network model. Thus, according to Sub-section 4.6, a generated MOPOP could be relatively more friendly to common-link-oriented measures such as the crossover operator of GA3.

● When comparing Table 4 with Table 3, the performance of GA2 drops significantly. This is mainly because in ES2 of Table 4, the costs of a link that does not belong to any Pareto optimal path are set up according to Eq. (13), which means such a link is much less likely dominated by those links belonging to Pareto optimal paths. Therefore, the heuristic rule of GA2 is no longer an advantage in ES2, but instead becomes a disadvantage as it largely misleads GA2 to explore those links that do not belong to any Pareto optimal path. As a result, sometimes in ES2, the quality of initialized chromosomes in GA2 is worse than that in either GA1 or GA3, so is the performance of GA2. The performance of either GA1 or GA3 is relatively stable, but still degrades when compared with ES1 of Table 3. This is because, as explained in Sub-section 4.3, Pareto points are much closer to Pareto dominated points in ES2 than in ES1, which means the complexity of MOPOP is much higher in ES2.

● When comparing ES3 of Table 5 with ES1 of Table 3, and comparing ES3 of Table 6 with ES1 of Table 4, we can see the performance of GA1 remains similar, which means whether Pareto optimal paths share common nodes or not does not influence GA1 much. GA2 performs a little better in ES3 than in ES1, but a little worse in ES4 than in ES2. This can be explained again based on the heuristic rule of GA2. The setup that Pareto optimal paths share common nodes besides O and D means they share some common links. In other words, there are relatively less (more) links that do (do not) belong to Pareto optimal paths in ES3/ES4 than in ES1/ES2. In ES3, the heuristic rule enables GA2 to focus more on those less links belonging to Pareto optimal paths, and then stands a better chance to find more Pareto optimal paths. In ES4, the heuristic rule makes GA2 more likely to be distracted by those more links that do not belong to Pareto optimal paths, and then ends up with finding less Pareto optimal paths. In the case of GA3, its performance is obviously boosted in ES3 and ES4,

because the crossover operator of GA3 is particularly designed for dealing with common links, and more common links shared by Pareto optimal paths imply a better chance for GA3 to find more Pareto optimal paths.

● When comparing the performance of different GAs for discrete combinational MOOPs, researchers often compare the number of Pareto non-dominated solutions found by different GAs. However, a Pareto non-dominated solution is not necessarily Pareto optimal. Assume that one GA finds hundreds of Pareto non-dominated solutions but none of them are Pareto optimal, while another GA only finds a few Pareto non-dominated solutions but they are all Pareto optimal. Then, which GA is better? Obviously, comparing the number of Pareto non-dominated solutions found by GAs is not an ideal way to assess the performance of GAs. Differently, with the help of the reported test problem toolkits, here we can compare the number of true Pareto optimal solutions found by GAs, and then precisely evaluate and even rank the performance of GAs or GA design techniques.

● Based on Tables 3–6 as well as these 4 experiment sets, we can give some discussions such as how problem scale might influence the performance of different GAs, and how fast each GA might converge. However, such discussions can also be made without the reported test problem toolkit. Therefore, to highlight the unique contribution of this paper, we will not go on with such things that other test problem toolkits may also contribute. In other words, those discussions made in previous bullet points are largely based on the capability of generating test problems with pre-given/known, desirable Pareto fronts. It is very difficult, if not impossible, for existing test problem toolkits for discrete combinational MOOP to achieve such a capability. Thanks to the reported toolkit of this paper, such a capability is now no longer a luxury we cannot have when evaluating the performance of methods for discrete combinational MOOP.

● In the comparative experiment based on the reported test problem toolkit, we only use one performance indicator, i.e., the number of true Pareto points found by a method. It should be noted that there may be some other performance indicators to evaluate MOPOP methods. For example, in theory we may use the gap (or the distance) between calculated Pareto front (approximate Pareto front) and the exact Pareto front as a performance indicator. Basically, the gap between calculated Pareto front and the exact Pareto front may be defined in different ways. Once a definition of gap is given, we can usually calculate it precisely based on the information of exact Pareto front thanks to the reported toolkit. A possible definition of gap is the hyper-volume of the polyhedron determined by both the calculated Pareto front and the exact Pareto front. However, if the number of calculated Pareto points is less than N_{Obj}, or the polyhedron is not enclosed, this definition will become difficult to apply. Another possible definition is the average Euclidean distance between each calculated Pareto point and its closest true Pareto point. However, sometimes this does not make a good indicator of MOPOP method performance. For example, assume $N_{\text{PP}} = 100$, method 1 only outputs 3 calculated Pareto points and they are all true Pareto points, while method 2 outputs 100 calculated Pareto points, and none of them is true Pareto point, but they are all close to those $N_{\text{PP}} = 100$ true Pareto points. In this case, the gap does not make much sense to compare two methods. Due to the above drawbacks of gap definitions, in the experiments of this section, we simply count and compare the number of true Pareto points found by each method. The definition of the gap between calculated Pareto front and the exact Pareto front is a technique needed after a test problem is given. This paper is more concerned with how to generate test problems before we need such a gap definition. In Section 6, further evaluating techniques (e.g., how to design effective performance indicators) based on the reported test problem toolkit are mentioned as future work.

6. Conclusions

This paper reports a benchmark test problem toolkit for multiobjective path optimization problems (MOPOPs). The reported benchmark test problem toolkit has a very good scalability in terms of numbers of objectives, nodes, links and Pareto points. The complexity of a generated test problem can also be well controlled and adjusted, for example, the Pareto front of a generated test problem can be of any possible shape (e.g., convex or concave), and Pareto points can be set close or far away from Pareto dominated points. The most important, the complete Pareto front of a generated test problem can be deduced and found out manually, no matter how large the problem scale is. Comprehensive experimental results demonstrate that the reported benchmark test problem toolkit can help to tell how many true Pareto points are missed by a partial method, and how large the gap is between the true Pareto front and the result of an approximate method. Compared with the strategy of testing a MOPOP method on some case study problems whose complete Pareto front is not known or adjustable, the reported toolkit enables a better performance evaluation.

Test problem toolkits for continuous numerical multi-objective optimization problems (MOOPs) have significantly contributed to performance evaluation of relevant MOOP methods. There is a need for test problem toolkits particularly for discrete combinational MOOPs. This study makes an attempt by targeting MOPOP, which is a typical category of discrete combinational MOOPs. Future research may be conducted to explore the full potentials of the reported toolkit. For example, how to define effective performance metrics (e.g., the gap between true Pareto front and a partial or approximated one) needs to be studied for discrete combinational MOOPs. It is also worth efforts how to make problem specific modifications to the reported toolkit, in order to better apply

to some real-world MOPOPs. Further attentions may be paid to investigate the possibility of extending the basic idea and techniques of this study to develop test problem toolkits for more other discrete combinational MOOPs.

References

[1] N. Barr, Economics of the Welfare State, Oxford University Press, New York, USA), 2004.

[2] R.E. Steuer, Multiple Criteria Optimization: Theory, Computations, and Application, John Wiley & Sons, Inc., New York, 1986.

[3] D.A. Van Veldhuizen, G.B. Lamont, Multiobjective evolutionary algorithms: analyzing the state-of-the-art, Evol. Comput. 8 (2) (2000) 125–147.

[4] J. Figueira, S. Greco, Multiple Criteria Decision Analysis: State of the Art Surveys, Matthias Ehrgottc, 2005.

[5] Y. Sawaragi, H. Nakayama, T. Tanino, Theory of Multiobjective Optimization (Vol. 176 of Mathematics in Science and Engineering), Academic Press Inc., Orlando, FL, 1985.

[6] I. Das, J.E. Dennis, Normal-boundary intersection: a new method for generating the pareto surface in nonlinear multicriteria optimization problems, SIAM J. Optim. 8 (1988) 631–657.

[7] I. Das, J. Dennis, Normal-boundary Intersection: an Mlternate Method for Generating Pareto Optimal Points in Multicriteria Optimization Problems, 1996. NASA Contractor Report 201616, ICASE Report No. 96-62.

[8] A. Messac, A. Ismail-Yahaya, C.A. Mattson, The normalized normal constraint method for generating the Pareto front, Struct. Multidiscip. Optim. 25 (2) (2003) 86–98.

[9] A. Messac, C.A. Mattson, Normal constraint method with guarantee of even representation of complete Pareto front, AIAA J. 42 (10) (2004) 2101–2111.

[10] T. Erfani, S.V. Utyuzhnikov, Directed search domain: a method for even generation of pareto front in multiobjective optimization, Eng. Optim. 43 (5) (2011) 467–484.

[11] D. Craft, T. Halabi, H. Shih, T. Bortfeld, Approximating convex Pareto surfaces in multiobjective radiotherapy planning, Med. Phys. 33 (9) (2006) 3399–3407.

[12] R.T. Marler, J.S. Arora, Survey of multi-objective optimization methods for engineering, Struct. Multidiscip. Optim. 26 (2004) 369–395.

[13] Y.Y. Haimes, L.S. Lasdon, D.A. Wismer, On a bicriterion formulation of the problems of integrated system identification and system optimization, IEEE Trans. Syst. Man Cybern. 1 (1971) 296–297.

[14] W. Stadler, J.P. Dauer, Multicriteria optimization in engineering: a tutorial and survey, in: M.P. Kamat (Ed.), Structural Optimization: Status and Promise, American Institute of Aeronautics and Astronautics, Washington, DC, 1992, pp. 211–249.

[15] A. Konak, D.W. Coit, A.E. Smith, Multi-objective optimization using genetic algorithms: a tutorial, Reliab. Eng. Syst. Saf. 91 (9) (2006) 992–1007.

[16] L. Vanneschi, R. Henriques, M. Castelli, Multi-objective genetic algorithm with variable neighbourhood search for the electoral redistricting problem, Swarm Evol. Comput. 36 (2017) 37–51.

[17] D.F. Jones, S.K. Mirrazavi, M. Tamiz, Multiobjective meta-heuristics: an overview of the current state-of-the-art, Eur. J. Oper. Res. 137 (1) (2002) 1–9.

[18] N. Srinivas, K. Deb, Multiobjective optimization using nondominated sorting in genetic algorithms, Evol. Comput. 2 (3) (1994) 221–248.

[19] K. Deb, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[20] W.F. Leong, G.G. Yen, PSO-based multiobjective optimization with dynamic population size and adaptive local archives, IEEE Transact. Syst. Man Cybern. B 38 (5) (2008) 1270–1293.

[21] J.D. Knowles, D.W. Corne, Approximating the nondominated front using the Pareto archived evolution strategy, Evol. Comput. 8 (2) (2000) 149–172.

[22] S. Bandyopadhyay, S.K. Pal, B. Aruna, Multiobjective GAs, quantitative indices, and pattern classification, IEEE Transact. Syst. Man Cybern. B 34 (5) (2004) 2088–2099.

[23] A. Saad, S.M. Khan, A. Mahmood, A multi-objective evolutionary artificial bee colony algorithm for optimizing network topology design, Swarm Evol. Comput. 38 (2018) 187–201.

[24] K. Deb, Multi-objective Optimization Using Evolutionary Algorithms, Wiley, New York, 2001.

[25] X.F. Zou, Y. Chen, M.Z. Liu, L.S. Kang, A new evolutionary algorithm for solving many-objective optimization problems, IEEE Trans. Syst. Man Cybern. B 38 (5) (2008) 1402–1412.

[26] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Evol. Comput. 8 (2) (2000) 173–195.

[27] A. Zhou, B.Y. Qu, H. Li, S.Z. Zhao, P.N. Suganthan, Q.F. Zhang, Multiobjective evolutionary algorithms: a survey of the state of the art, Swarm Evol. Comput. 1 (1) (2011) 32–49.

[28] Y.Y. Han, D.W. Gong, Y.C. Jin, Q.K. Pan, Evolutionary multi-objective blocking lotstreaming flow shop scheduling with machine breakdowns, IEEE Trans. Cybern. 99 (2017) 1–14.

[29] J.Q. Li, H.Y. Sang, Y.Y. Han, C.G. Wang, K.Z. Gao, Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions, J. Clean. Prod. 181 (2018) 584–598.

[30] X.B. Hu, M. Wang, E. Di Paolo, Calculating complete and exact pareto front for multi-objective optimization: a new deterministic approach for discrete problems, IEEE Trans. Syst. Man Cybern. B 43 (3) (2013) 1088–1101.

[31] E.Q.V. Martins, On a multicriteria shortest path problem, Eur. J. Oper. Res. 16 (1984) 236–245.

[32] C.T. Tung, K.L. Chew, A multicriteria pareto-optimal path algorithm, Eur. J. Oper. Res. 62 (1992) 203–209.

[33] F. Guerriero, R. Musmanno, Label correcting methods to solve multicriteria shortest path problems, J. Optim. Theor. Appl. 111 (1) (2001) 589–613.

[34] A. Raith, M. Ehrgott, A comparison of solution strategies for biobjective shortest path problems, Comput. Oper. Res. 36 (2009) 1299–1331.

[35] A. Raith, M. Ehrgott, A two-phase algorithm for the biobjective integer minimum cost flow problem, Comput. Oper. Res. 36 (2009) 1945–1954.

[36] F.J. Pulido, L. Mandow, J.L. Perez-la-Cruz, Dimensionality reduction in multiobjective shortest path search, Comput. Oper. Res. 64 (2015) 60–70.

[37] A. Sedeno, A. Raith, A Dijkstra-like method computing all extreme supported nondominated solutions of the biobjective shortest path problem, Comput. Oper. Res. 57 (2015) 83–94.

[38] D. Duque, L. Lozano, A. Medaglia, An exact method for the biobjective shortest path problem for large-scale road networks, Eur. J. Oper. Res. 242 (2015) 788–797.

[39] R.M. Ramos, S. Alonso, J. Sicilia, C. Gonzales, The problem of the optimal biobjective spanning tree, Eur. J. Oper. Res. 111 (1998) 617–628.

[40] S. Steiner, T. Radzik, Computing all efficient solutions of the biobjective minimum spanning tree problem, Comput. Oper. Res. 35 (1) (2008) 198–211.

[41] M. Laumanns, L. Thiele, E. Zitzler, An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method, Eur. J. Oper. Res. 169 (2006) 932–942.

[42] G. Mavrotas, J.R. Figueira, A. Antoniadis, Using the idea of expanded core for the exact solution of bi-Objective multi-dimensional knapsack problems, J. Global Optim. 49 (4) (2011) 589–606.

[43] J. Sylva, A. Crema, A method for finding the set of nondominated vectors for multiple objective integer linear programs, Eur. J. Oper. Res. 158 (2004) 46–55.

[44] A. Przybylski, X. Gandibleux, M. Ehrgott, A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives, Discrete Optim. 7 (3) (2010) 149–165.

[45] B. Lokman, M. Koksalan, Finding all nondominated points of multi-objective integer programs, J. Global Optim. 57 (2013) 347–365.

[46] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable Test Problems for Evolutionary Multiobjective Optimization, Technical Report Technical report 112, Computer Engineering and Networks Laboratory, ETH Zurich, 2001.

[47] S. Huband, P. Hingston, L. Barone, R.L. While, A review of multiobjective test problems and a scalable test problem toolkit, IEEE Trans. Evol. Comput. 10 (5) (2006) 477–506.

[48] V.L. Huang, A.K. Qin, K. Deb, E. Zitzler, P.N. Suganthan, J.J. Liang, M. Preuss, S. Huband, Problem Definitions for Performance Assessment of Multi-objective Optimization Algorithms, Technical Report Technical Report, Nanyang Technological University, NTU, Singapore, 2007.

[49] Q. Zhang, A. Zhou, S.Z. Zhao, P.N. Suganthan, W. Liu, S. Tiwari, Multiobjective Optimization Test Instances for the CEC 2009 Special Session and Competition, Technical Report Technical Report CES-887, University of Essex and Nanyang Technological University, NTU, Singapore, 2007.

[50] H. Li, Q. Zhang, Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II, IEEE Trans. Evol. Comput. 12 (2) (2009) 284–302.

[51] H. Li, Q. Zhang, J.D. Deng, Multiobjective Test Problems with Complicated Pareto Fronts: Difficulties in Degeneracy, Congress on Evolutionary Computation, 2014, pp. 2156–2163.

[52] H.L. Liu, L. Chen, Q. Zhang, K. Deb, Adaptively allocating search effort in challenging many-objective optimization problems, IEEE Trans. Evol. Comput. (2017), https://doi.org/10.1109/TEVC.2017.2725902 in press.

[53] H. Li, K. Deb, Q.F. Zhang, P.N. Suganthan, Challenging Novel Many and Multiobjective Bound Constrained Benchmark Problems, http://www.ntu.edu.sg/home/epnsugan/index_files/SWEVO-ManyObjSI.htm.

[54] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer Verlag, Berlin, 2003.

[55] X.B. Hu, E. Di Paolo, Binary representation based genetic algorithm for aircraft arrival sequencing and scheduling, IEEE Trans. Intell. Transport. Syst. 9 (2) (2008) 301–310.

[56] H. Aljazzar, S. Leue, A heuristic search algorithm for finding the k shortest paths, Artif. Intell. 175 (18) (2011) 2129–2154.

[57] K. Mohanta, B. Poddar, Comprehensive study on computational methods for k shortest paths problem, Int. J. Comput. Appl. 40 (14) (2012) 22–26.

[58] D. Frigioni, A. Marchetti-Spaccamela, U. Nanni, Fully dynamic algorithms for maintaining shortest paths trees, J. Algorithm 22 (3) (2000) 251–281.

[59] R. Bauer, D. Wagner, Batch Dynamic Single-origin Shortest-path Algorithms: an Experimental Study, Experimental Algorithms, 2009, pp. 51–62.

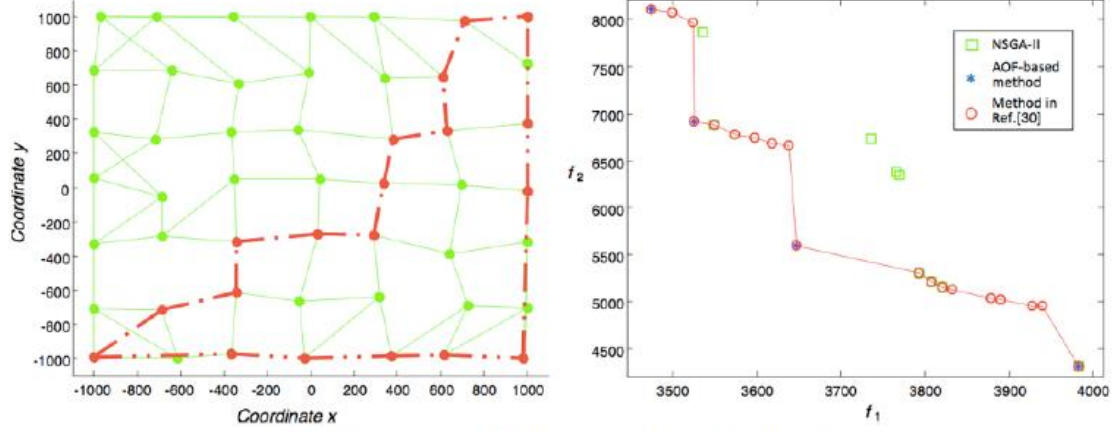[60] M.T. Goodrich, R. Tamassia, Algorithm Design: Foundation, Analysis and Internet Examples, Wiley, New York, 2006.

Fig. 1. Example of solutions by different methods ($N_N = 49$ and $N_{Obj} = 2$).
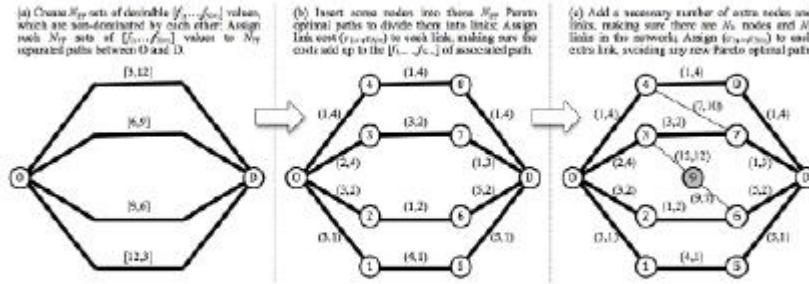

Fig. 2. Example of generating a MOPOP with $N_{Obj} = 2$, $N_{PP} = 4$, $N_N = 11$ and $N_L = 15$.
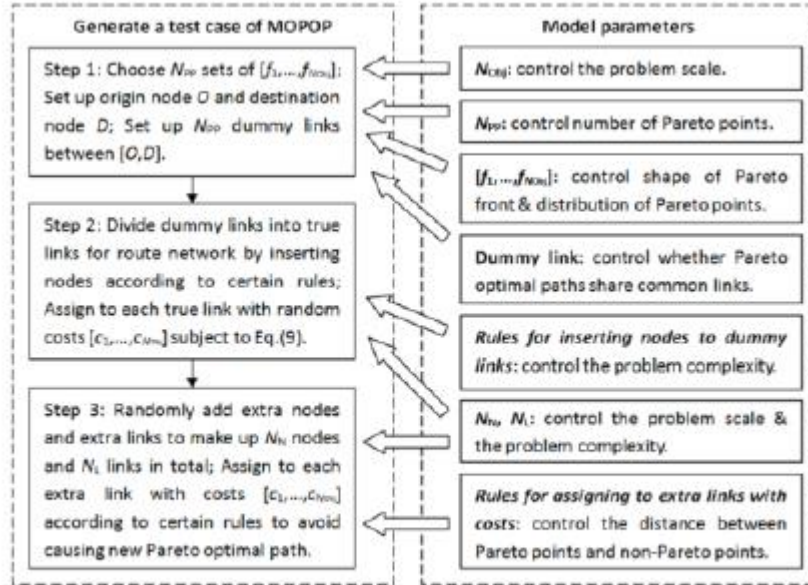

Fig. 3. Flowchart of generating test case of MOPOP with desirable features.
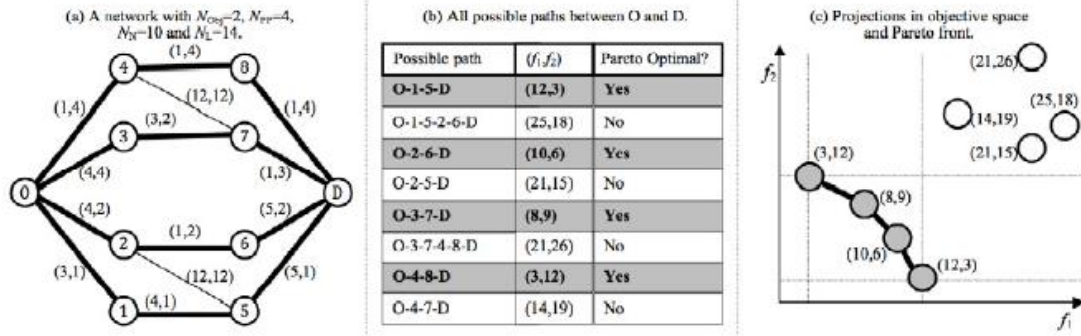
Fig. 4

(a) A network with $N_{Obj}=2$, $N_{PP}=4$, $N_N=10$ and $N_L=14$.

Network link labels: (1,4), (12,12), (1,4), (1,4), (3,2), (1,4), (4,4), (1,3), (4,2), (5,2), (1,2), (3,1), (12,12), (5,1), (4,1)

(b) All possible paths between O and D.

Possible path	(f_1,f_2)	Pareto Optimal?
O-1-5-D	(12,3)	Yes
O-1-5-2-6-D	(25,18)	No
O-2-6-D	(10,6)	Yes
O-2-5-D	(21,15)	No
O-3-7-D	(8,9)	Yes
O-3-7-4-8-D	(21,26)	No
O-4-8-D	(3,12)	Yes
O-4-7-D	(14,19)	No

(c) Projections in objective space and Pareto front.

Points: (21,26), (25,18), (14,19), (21,15), (3,12), (8,9), (10,6), (12,3)

Fig. 4. Example of a MOPOP with concave Pareto front well separated from dominated points.

Fig. 5

(a) A network with $N_{Obj}=2$, $N_{PP}=4$, $N_N=10$ and $N_L=14$.

Network link labels: (1,4), (1,6), (1,3), (1,5), (2,2), (1,3), (2,3), (4,2), (3,1), (1,2), (3,1), (1,7), (5,1), (4,1)

(b) All possible paths between O and D.

Possible path	(f_1,f_2)	Pareto Optimal?
O-1-5-D	(12,3)	Yes
O-1-5-2-6-D	(12,12)	No
O-2-6-D	(8,5)	Yes
O-2-5-D	(10,10)	No
O-3-7-D	(5,8)	Yes
O-3-7-4-8-D	(6,18)	No
O-4-8-D	(3,12)	Yes
O-4-7-D	(4,14)	No

(c) Projections in objective space and Pareto front.

Points: (6,18), (3,12), (4,14), (12,12), (5,8), (10,10), (8,5), (12,3)
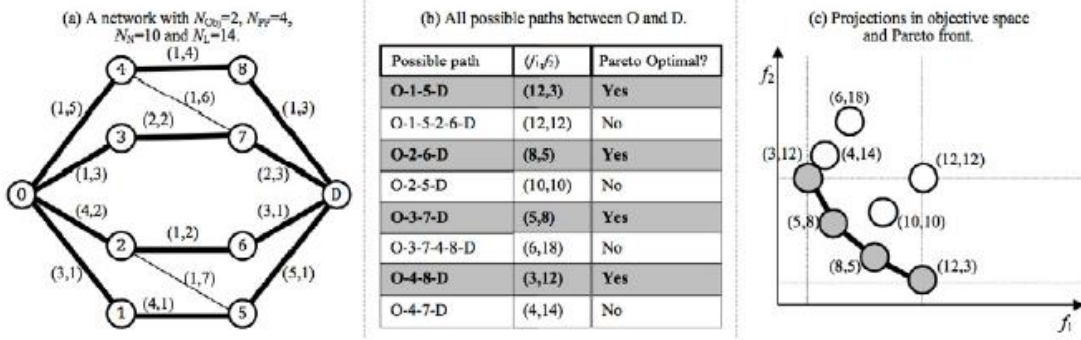
Fig. 5. Example of a MOPOP with convex Pareto front close to dominated points.

Fig. 6

(a) Generate a desirable shape according to study on Pareto front of continuous numerical MOOP.

(b) Sample N_{PP} points on the desired shape to set up N_{PP} sets of $[f_{1,m},...,f_{N_{Obj},m}]$ values, $m=1,...,N_{PP}$.

Sample points: P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8

(c) Based on the $[f_{1,m},...,f_{N_{Obj},m}]$ values of N_{PP} sample points, set up N_{PP} paths in Step 1 of model.

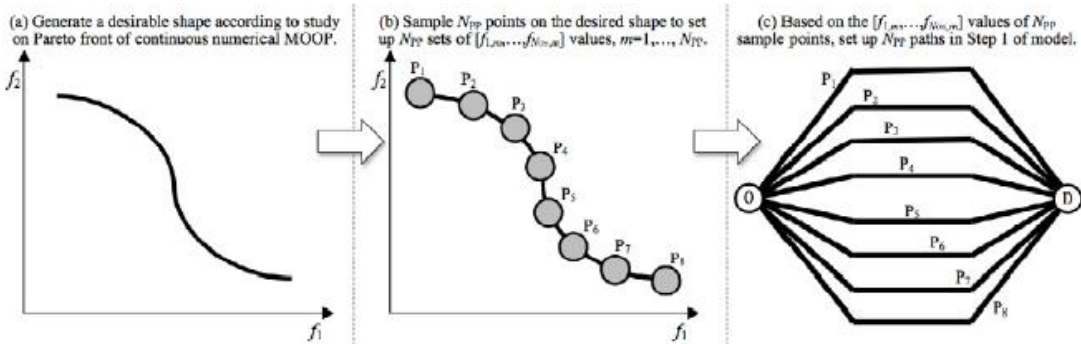Paths from O to D: P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8

Fig. 6. Example of how to get a desirable Pareto front shape for MOPOP.
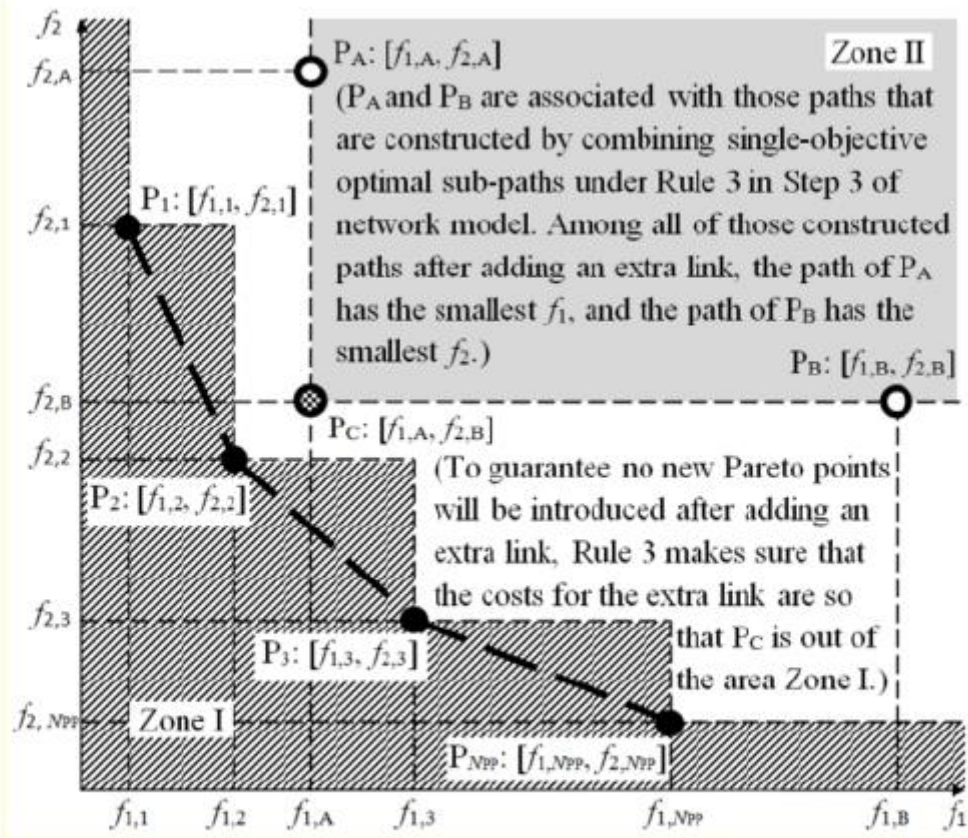
Fig. 7. Example about why assigning costs to an extra link under Rule 3 will not affect those N_{PP} Pareto optimal paths ($N_{\text{obj}} = 2$, $N_{\text{PP}} = 4$).
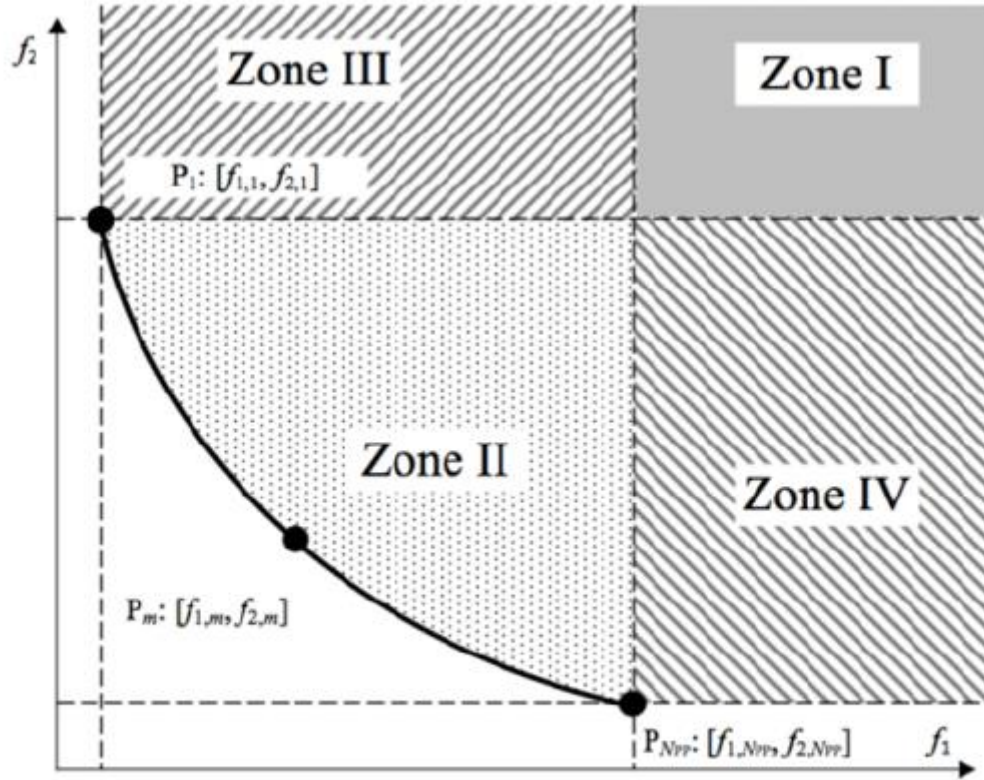
Fig. 8. Example of zones that determines the average distance between Pareto points and Pareto dominated points.
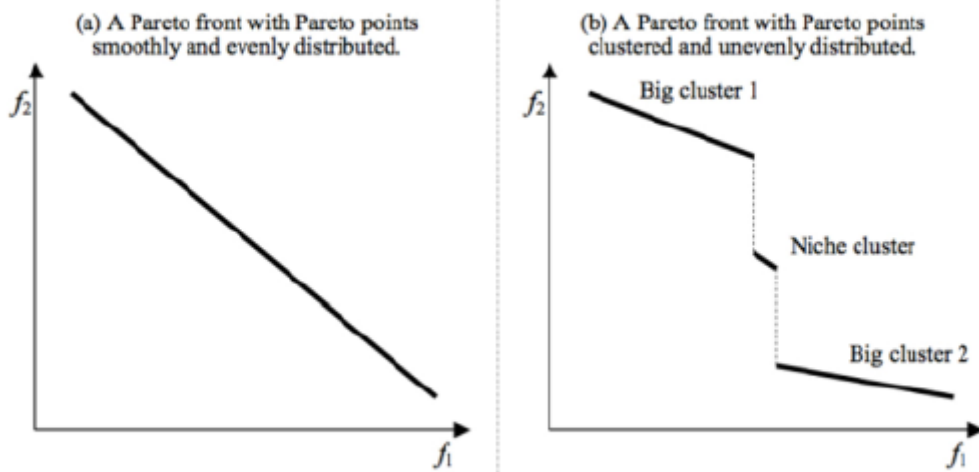

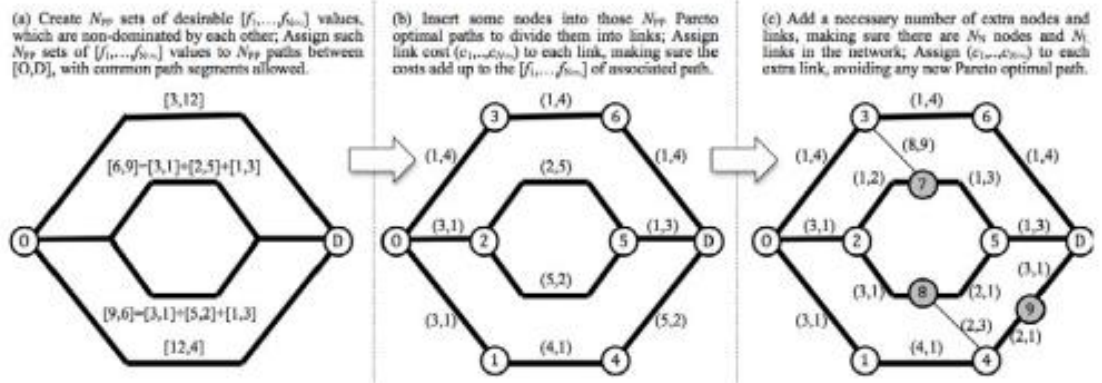**Fig. 9.** Examples of different distributions of Pareto points on Pareto front.

Fig. 10. Example of generating a MOPOP with Pareto optimal paths sharing common nodes and links ($N_{Obj} = 2$, $N_{PP} = 4$, $N_N = 11$ and $N_L = 14$).
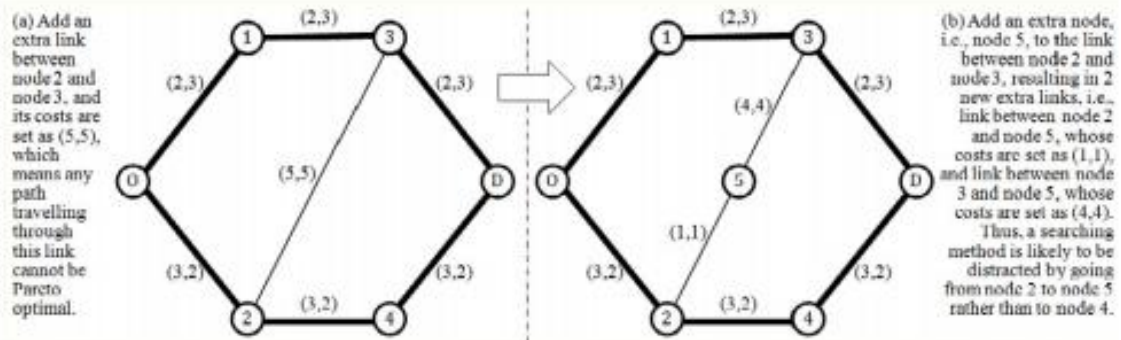

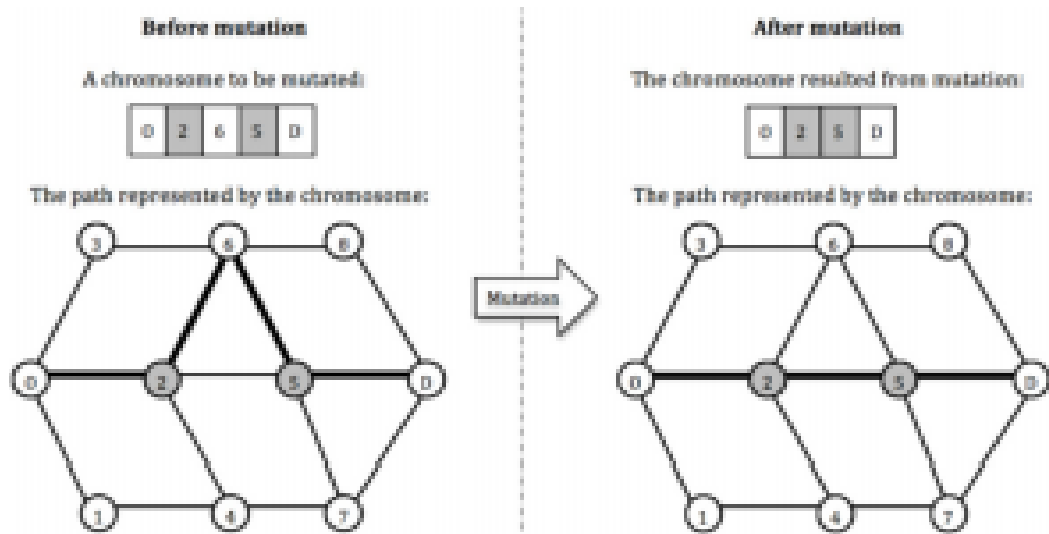**Fig. 11.** Example of generating a local optimum in Step 3 ($N_{Obj} = 2$, $N_{PP} = 2$).

Before mutation

A chromosome to be mutated:

| 0 | 2 | 6 | 5 | D |

The path represented by the chromosome:

After mutation

The chromosome resulted from mutation:

| 0 | 2 | 5 | D |

The path represented by the chromosome:

Fig. 12. An illustration of the mutation operator in GA1.

Without heuristic rule for choosing link in chromosome initialization

Randomly choose between link 1➔2 and link 1➔3:

50% to choose link 1➔3   50% to choose link 1➔2

With heuristic rule for choosing link in chromosome initialization

More likely to choose a link with non-dominated costs:
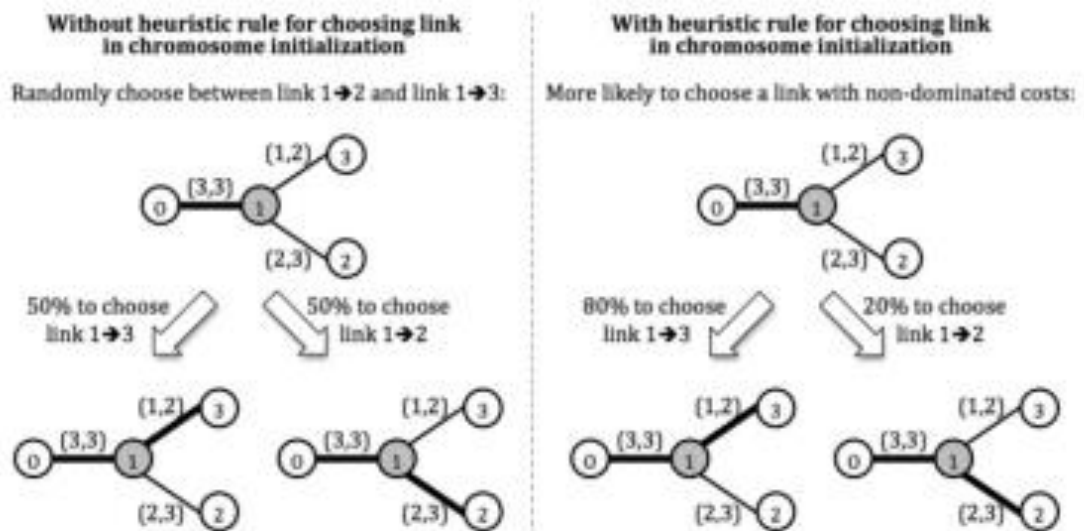
80% to choose link 1➔3   20% to choose link 1➔2

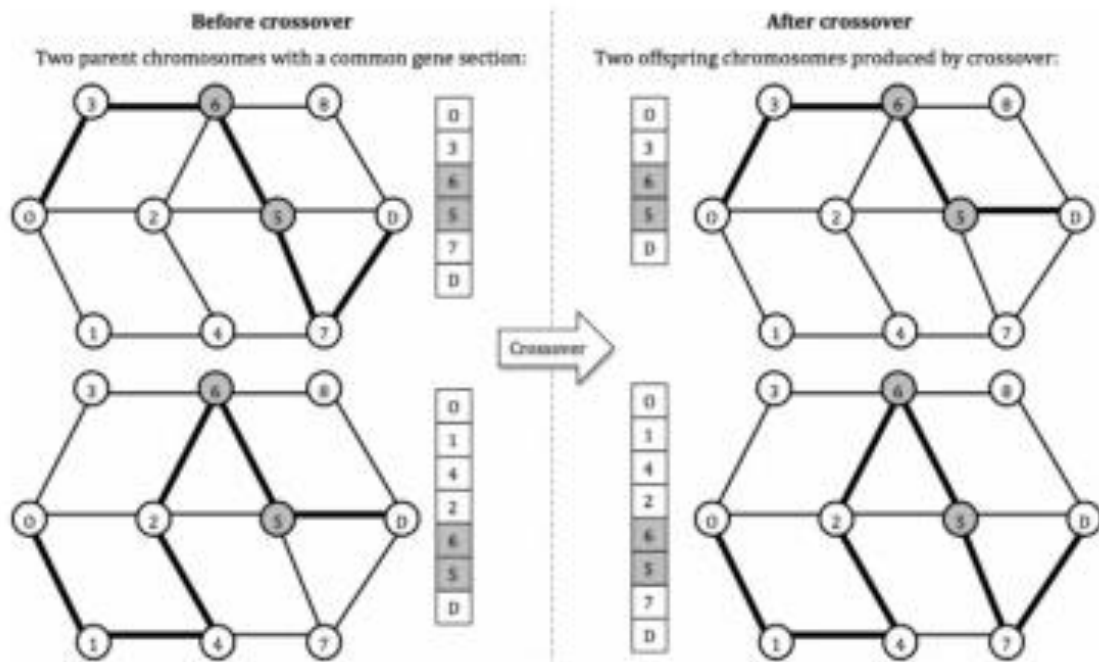Fig. 13. An illustration of the heuristic rule in GA2.

Fig. 14. An illustration of the crossover operator in GA3.

Table 1
Comparative results between three categories of MOPOP methods.

		COM	AOF	GA1
$N_{Obj} = 2$	CT	25.28	1.59	101.37
	N_{PPF}	30	5.06	19.20
	R_{FCPF}	1.00	0.03	0.12
$N_{Obj} = 4$	CT	39.74	1.61	116.66
	N_{PPF}	30	4.85	18.57
	R_{FCPF}	1.00	0.03	0.13
$N_{Obj} = 6$	CT	65.22	1.65	136.72
	N_{PPF}	30	5.13	16.70
	R_{FCPF}	1.00	0.02	0.11
$N_{Obj} = 8$	CT	147.53	1.73	158.33
	N_{PPF}	30	5.63	13.06
	R_{FCPF}	1.00	0.02	0.08
$N_{Obj} = 10$	CT	262.85	1.85	186.95
	N_{PPF}	30	4.58	9.62
	R_{FCPF}	1.00	0.01	0.06

Table 2
Key features of four experiment sets (i.e., ES1 to ES4).

	Costs of an extra link is set up according to Rule 2, Eq. (11) and Eq. (12) in Sub-section 4.3	Costs of an extra link is set up according to Rule 3 and Eq. (13) in Sub-section 4.3
Pareto optimal paths share no common nodes except O and D	ES1	ES2
Pareto optimal paths may share some common nodes besides O and D	ES3	ES4

Table 3
The number of true Pareto optimal paths found by 3 GAs in ES1.

	GA1		GA2		GA3	
	Mean	SD	Mean	SD	Mean	SD
$N_N = 200$	35.21	5.66	99.06	0.58	43.80	6.89
$N_N = 300$	27.51	6.33	93.32	4.13	35.98	7.43
$N_N = 400$	14.86	3.85	84.35	6.19	23.54	6.39
$N_N = 500$	7.75	1.74	70.79	5.29	14.34	3.34
$N_N = 600$	4.58	1.10	61.52	4.19	10.15	2.23

Table 4
The number of true Pareto optimal paths found by 3 GAs in ES2.

	GA1		GA2		GA3	
	Mean	SD	Mean	SD	Mean	SD
$N_N = 200$	31.92	6.17	45.93	6.05	40.03	7.93
$N_N = 300$	24.19	7.07	36.31	5.91	34.61	9.27
$N_N = 400$	12.79	4.58	22.42	6.27	21.75	6.92
$N_N = 500$	6.79	1.80	11.54	3.81	12.74	3.46
$N_N = 600$	4.27	1.16	7.61	2.16	8.39	2.13

Table 5
The number of true Pareto optimal paths found by 3 GAs in ES3.

	GA1		GA2		GA3	
	Mean	SD	Mean	SD	Mean	SD
$N_N = 200$	37.21	6.33	99.10	0.48	55.27	7.46
$N_N = 300$	26.42	6.01	93.98	3.62	46.73	8.19
$N_N = 400$	14.22	3.80	87.61	4.96	34.18	6.42
$N_N = 500$	7.68	2.14	73.79	5.91	20.09	3.77
$N_N = 600$	5.05	1.06	65.89	5.33	12.76	2.67

Table 6
The number of true Pareto optimal paths found by 3 GAs in ES4.

	GA1		GA2		GA3	
	Mean	SD	Mean	SD	Mean	SD
$N_N = 200$	33.16	6.48	40.87	6.06	46.91	7.49
$N_N = 300$	25.03	6.45	33.71	6.55	40.47	8.18
$N_N = 400$	12.09	4.01	19.96	5.59	26.26	6.89
$N_N = 500$	6.81	1.77	10.67	3.65	15.34	3.57
$N_N = 600$	4.11	1.20	6.39	1.94	9.69	2.10