Soft Comput
DOI 10.1007/s00500-014-1487-2

METHODOLOGIES AND APPLICATION

On investigation of interdependence between sub-problems of the Travelling Thief Problem

Yi Mei · Xiaodong Li · Xin Yao

Abstract In this paper, the interdependence between sub-problems in a complex overall problem is investigated using a benchmark problem called Travelling Thief Problem (TTP), which is a combination of Travelling Salesman Problem (TSP) and Knapsack Problem (KP). First, the analysis on the mathematical formulation shows that it is impossible to decompose the problem into independent sub-problems due to the non-linear relationship in the objective function. Therefore, the algorithm for TTP is not straightforward although each sub-problem alone has been investigated intensively. Then, two meta-heuristics are proposed for TTP. One is the Cooperative Co-evolution (CC) that solves the sub-problems separately and transfers the information between them in each generation. The other is the Memetic Algorithm (MA) that solves TTP as a whole. The comparative results showed that MA consistently obtained much better results than both the standard and dynamic versions of CC within comparable computational budget. This indicates the importance of considering the interdependence between sub-problems in an overall problem like TTP.

Communicated by V. Loia.

Y. Mei (✉) · X. Li
School of Computer Science and Information Technology, RMIT University, Melbourne, VIC 3000, Australia
e-mail: yi.mei@rmit.edu.au

X. Li
e-mail: xiaodong.li@rmit.edu.au

X. Yao
School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK
e-mail: x.yao@cs.bham.ac.uk

Keywords Combinatorial optimization · Evolutionary computation · Cooperative Co-evolution · Travelling Thief Problem · Interdependence

1 Introduction

Real-world problems often involve a large number of decision variables and constraints, making it impossible to find the global optimal solution within the given time budget. When tackling large-scale optimization problems, the divide-and-conquer approach is commonly adopted to decompose the overall problem into smaller sub-problems (Boyd et al. 2007; Omidvar et al. 2014; Mei et al. 2014a). For many real-world problems, the sub-problems are naturally defined. For example, in supply chain management (Thomas and Griffin 1996; Stadtler 2005; Melo et al. 2009), each stage or operation such as procurement, production and distribution can correspond to a sub-problem. However, it is often inevitable that such sub-problems are still interdependent on each other. As mentioned in Michalewicz (2012), one of the main complexity of real-world problems is the interdependence between sub-problems, which makes many conventional approaches ineffective. As a result, even if each sub-problem has been intensively investigated, it is still an open question how to integrate the high-quality partial solutions for the sub-problems to obtain a global optimum or at least a high-quality solution for the overall problem. Therefore, it is important to investigate how to tackle the interdependence between sub-problems.

To facilitate such investigation, Bonyadi et al. (2013) recently defined a benchmark problem called Travelling Thief Problem (TTP). TTP is a combination of two well-known combinatorial optimization problems, i.e., Travelling Salesman Problem (TSP) and Knapsack Problem (KP).

Published online: 17 October 2014

Specifically, a thief is to visit a set of cities and pick some items from the cities to put in a rented knapsack. Each item has a value and a weight. The knapsack has a limited capacity that cannot be exceeded by the total weight of the picked items. In the end, the thief has to pay the rent for the knapsack, which depends on the travel time. TTP aims to find a tour for the thief to visit all the cities exactly once, pick some items along the way and finally return to the starting city, so that the benefit of the visit, which is the total value of the picked items minus the rent of the knapsack, is maximized. Since TSP and KP have been intensively investigated, TTP facilitates to concentrate on the interdependence between sub-problems.

An example of potential relevant real-world applications of TTP is the capacitated arc routing problem (Dror 2000) with service profit. Although there have been extensive studies for solving various forms of the capacitated arc routing problem depending on different practical scenarios [e.g., the classic model (Mei et al. 2009a, b; Tang et al. 2009; Fu et al. 2010), the multi-objective model (Mei et al. 2011a), the stochastic model (Mei et al. 2010), the periodic model (Mei et al. 2011b) and the large-scale model (Mei et al. 2013, 2014a, b)], two important practical issues have been overlooked so far. One is the service profit, which is the profit that can be gained by serving the customers. Each customer may have a different profit/demand ratio. Thus, given the limited capacity of the vehicle, one may need to serve only a subset of the customers with higher profit/demand ratios to maximize the final benefit. The other factor is the dependency of the travel cost of the vehicle on its load. Obviously, a heavier load of the vehicle leads to a higher consumption of petrol, and thus a higher travel cost. In this case, it would be more desirable to serve the customers with a higher demand first to save the travel cost of the subsequent route. With the above factors taken into account, the resultant arc routing problem can be modelled as a TTP.

In this paper, the interdependence of TSP and KP in TTP is investigated both theoretically and empirically. First, the mathematical formulation of TTP is developed and analysed to show how the two sub-problems interact with each other. Then, a Cooperative Co-evolution algorithm (CC) (including a standard and a dynamic version) and a Memetic Algorithm (MA) are developed. CC solves TSP and KP separately, and transfers the information between them in each generation. MA solves TTP as a whole. Standard crossover and mutation operators are employed. The proposed algorithms were compared on the benchmark instances proposed in Bonyadi et al. (2013), and the results showed that MA managed to obtain much better solutions than CC for all the test instances. In other words, with the same crossover and mutation operators for each sub-problem, a more proper way of integrating the optimization process of the sub-problems can result in a significantly better solution. This demonstrates that considering the interdependence between sub-problems is important for obtaining high-quality solution for the overall problem. Moreover, the theoretical analysis establishes the fundamental understanding of the problem.

The rest of the paper is organized as follows: TTP is formulated and analysed in Sect. 2. After that, CC and MA are depicted in Sect. 3. Then, the experimental studies are carried out in Sect. 4. Finally, the conclusion and future work are described in Sect. 5.

2 Travelling Thief Problem

In this section, TTP is introduced. The mathematical formulation is first described in Sect. 2.1 and then analysed in Sect. 2.2, particularly in terms of the interdependence between the TSP and KP decision variables in the objective function.

2.1 Mathematical formulation

TTP is a combination of TSP and KP. In TSP, n cities with the distance matrix of $D_{n\times n}$ are given, where d_{ij} is the distance from city i to j. In KP, there are m items. Each item i has a weight w_i, a value b_i and a set of available cities A_i. For example, $A_i = \{1, 2, 5\}$ implies that item i can only be picked from city 1, 2 or 5. A thief aims to visit all the cities exactly once, pick items on the way and finally come back to the starting city. The thief rents a knapsack to carry the items, which has a capacity of Q. The rent of the knapsack is R per time unit. The speed of the thief decreases linearly with the increase of the total weight of carried items and is computed by the following formula:

$$v = v_{\max} - (v_{\max} - v_{\min})\frac{\bar{w}}{Q}, \tag{1}$$

where $0 \le \bar{w} \le Q$ is the current total weight of the picked items. When the knapsack is empty ($\bar{w} = 0$), the speed is maximized ($v = v_{\max}$). When the knapsack is full ($\bar{w} = Q$), the speed is minimized ($v = v_{\min}$). Then, the benefit gained by the thief is defined as the total value of the picked items minus the rent of the knapsack.

Figure 1 illustrates an example of a TTP solution that travels through the path A–B–C–D–A, picking items 1 and 2 at cities A and C, respectively. The weights and values of the items are $w_1 = b_1 = 2$ and $w_2 = b_2 = 1$. The numbers associated with the arcs indicate the distances between the cities. The total value of the picked items is $b_1 + b_2 = 3$. The travel speeds between each pair of cities are $v_{AB} = v_{BC} = 4 - 3 \cdot 2/3 = 2$ and $v_{CD} = v_{DA} = 1$. Then, the total travel time is $(2+1.5)/2+(1+1.5)/1 = 4.25$. Finally, the benefit of the travel is $3 - 1 \cdot 4.25 = -1.25$ (a loss of 1.25).

To develop a mathematical formulation for TTP, the 0–1 decision variables of x_{ij} $(i, j = 1, \ldots, n)$, y_i $(i = 1, \ldots, n)$

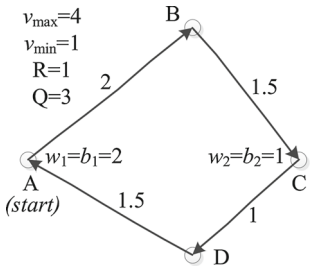

Fig. 1 An example of a TTP solution

and z_{ij} $(i = 1, \ldots, m;\ j = 1, \ldots n)$ are defined. The TSP decision variables x_{ij} takes 1 if there is a path from city i to j, and 0 otherwise. The starting city decision variables y_i equals 1 if city i is the starting city, and 0 otherwise. The KP decision variables z_{ij} takes 1 if item i is picked in city j, and 0 otherwise. Then, TTP can be formulated as follows:

$$\max \quad \mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z}) \tag{2}$$

$$s.t.: \sum_{i=0, i\neq j}^{n} x_{ij} = 1, \quad j = 1, \ldots, n \tag{3}$$

$$\sum_{j=0, j\neq i}^{n} x_{ij} = 1, \quad i = 1, \ldots, n \tag{4}$$

$$u_i - u_j + n x_{ij} \leq n - 1, \quad 1 \leq i \neq j \leq n \tag{5}$$

$$\sum_{i=1}^{n} y_i = 1 \tag{6}$$

$$\sum_{j\in A_i} z_{kj} \leq 1, \quad k = 1, \ldots, m \tag{7}$$

$$\sum_{j\notin A_i} z_{kj} = 0, \quad k = 1, \ldots, m \tag{8}$$

$$\sum_{k=1}^{m}\sum_{j=1}^{n} w_i z_{kj} \leq Q \tag{9}$$

$$x_{ij}, y_i, z_{kj} \in \{0, 1\}, u_i \geq 0, \quad i, j = 1, \ldots n;\ k = 1, \ldots, m \tag{10}$$

The objective (2) is to maximize the benefit $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z})$, whose definition is complex and thus will be described in details later. The constraints (3)–(5) are standard constraints that ensure the validity of the TSP solution. In Eq. (5), the u_i's $(i = 1, \ldots, n)$ are non-negative artificial variables to avoid solutions with sub-tours. The constraint (6) indicates that the tour has exactly one starting city. The constraints (7)–(9) imply that each item is picked at most once from its set of available cities, and the total weight of the picked items cannot exceed the capacity of the knapsack. The constraint (10) defines the domain of the variables.

In Eq. (2), $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is defined as follows:

$$\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{i=1}^{m}\sum_{j=1}^{n} b_i z_{ij} - R \cdot T, \tag{11}$$

where T is the total travelling time that is calculated as

$$T = \sum_{j=1}^{n} y_j T_j \tag{12}$$

$$T_j = \sum_{l=1}^{n} \frac{P_{jl} - P_{j(l-1)}}{v_{\max} - (v_{\max} - v_{\min})\bar{w}_{jl}/Q} \tag{13}$$

$$P_{jl} = \sum_{\substack{k_1,\ldots,k_l=1 \\ k_1\neq\cdots\neq k_l}}^{l} (d_{jk_1} + d_{k_1k_2} + \cdots + d_{k_{l-1}k_l}) \times x_{jk_1} x_{k_1k_2} \ldots x_{k_{l-1}k_l} \tag{14}$$

$$\bar{w}_{jl} = \sum_{\substack{k_1,\ldots,k_r=1 \\ k_1\neq\cdots\neq k_r}}^{l} \left(\sum_{r=1}^{l}\sum_{i=1}^{m} w_i z_{ik_r}\right) x_{jk_1} x_{k_1k_2} \ldots x_{k_{l-1}k_l} \tag{15}$$

In Eq. (12), T is defined as the total travelling time T_j starting from city j where $y_j = 1$. In Eq. (13), P_{jl} stands for the distance of the path with l links starting from city j, and $\bar{w}_{jl}$ is the current total weight of the picked items after visiting l cities excluding the starting city j. They are calculated by Eqs. (14) and (15), respectively. Note that $x_{jk_1} x_{k_1k_2} \ldots x_{k_{l-1}k_l}$ equals 1 if $x_{jk_1} = x_{k_1k_2} = \cdots = x_{k_{l-1}k_l} = 1$, i.e., the solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ includes an l-length path $(j, k_1, \ldots, k_l)$, and 0 otherwise. Therefore, Eq. (14) only counts in the total distance of the existing path in the solution $(\mathbf{x}, \mathbf{y}, \mathbf{z})$, and Eq. (15) only sums up the weights of the items picked along such path.

2.2 Problem analysis

From Eqs. (3)–(10), one can see that in the constraints, the decision variables $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are independent of each other. Obviously, Eqs. (3)–(5) only consist of $\mathbf{x}$, Eq. (6) only includes $\mathbf{y}$, and Eqs. (7)–(9) solely involve $\mathbf{z}$. However, as shown in Eqs. (11)–(15), there is a non-linear relationship between the variables in the objective $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z})$. For example, Eq. (15) includes the product of the z_{ij}'s and x_{ij}'s, and Eq. (13) involves the quotient of the x_{ij}'s and z_{ij}'s. In the above formulation, it is difficult to find an additively separation of $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z})$, if not impossible. That is, one cannot find the functions $\mathcal{G}_1(\mathbf{x})$, $\mathcal{G}_2(\mathbf{y})$ and $\mathcal{G}_3(\mathbf{z})$ such that $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathcal{G}_1(\mathbf{x}) + \mathcal{G}_2(\mathbf{y}) + \mathcal{G}_3(\mathbf{z})$. In other words, it is impossible to decompose the overall problem $\mathcal{P}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ into independent sub-problems $\mathcal{P}_1(\mathbf{x})$, $\mathcal{P}_2(\mathbf{y})$ and $\mathcal{P}_3(\mathbf{z})$ such that $\mathcal{OBJ}(\mathcal{P}) = \mathcal{OBJ}(\mathcal{P}_1) + \mathcal{OBJ}(\mathcal{P}_2) + \mathcal{OBJ}(\mathcal{P}_3)$, where $\mathcal{OBJ}(\mathcal{P})$ stands for the objective function of the problem $\mathcal{P}$.

The above analysis enables us to better understand the reason why solving the sub-problems individually can hardly lead to high-quality solutions. Take TTP as an example, in

Bonyadi et al. (2013), a simple decomposition of TTP into TSP and KP was designed by setting $\mathcal{G}_1(\mathbf{x}, \mathbf{y}) = \mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{0}) \cdot v_{\max}/R = -td(\mathbf{x})$ and $\mathcal{G}_3(\mathbf{z}) = \sum_{i=1}^{m} \sum_{j=1}^{n} b_i z_{ij}$, where $td(\mathbf{x})$ stands for the total distance of the TSP tour $\mathbf{x}$, which is independent of the starting city decision variables $\mathbf{y}$. In other words, TTP was decomposed into TSP and KP with standard objective functions (minimizing total distance for TSP and maximizing total value for KP). However, the preliminary experimental results showed that such decomposition cannot lead to good TTP solutions. Based on the above analysis, the reason is that the original objective $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is not the summation of the $\mathcal{G}_1(\mathbf{x}, \mathbf{y})$ and $\mathcal{G}_3(\mathbf{z})$. Thus, optimizing $\mathcal{G}_1(\mathbf{x}, \mathbf{y})$ and $\mathcal{G}_3(\mathbf{z})$ is not directly related to optimizing $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ itself.

To summarize, the mathematical formulation of TTP shows that the objective $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is not additively separable. Therefore, one cannot expect that solving the TSP and KP sub-problems individually will obtain competitive TTP solutions since their objectives are not fully correlated. In this paper, each solution is evaluated directly with respect to the original objective $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ provided that there is no TSP and KP objective functions strongly correlated to $\mathcal{G}(\mathbf{x}, \mathbf{y}, \mathbf{z})$ so far.

3 Solving TTP with meta-heuristics

According to the mathematical formulation described in Sect. 2.1, it is seen that TTP is a complex nonlinear integer optimization problem. It is also obvious that TTP is NP-hard, since it can be reduced to the TSP when $w_i = b_i = 0, \forall i = 1, \ldots, m$, which has been proved to be NP-hard (Papadimitriou 1977). In this situation, meta-heuristics are good alternatives as it has been demonstrated to be able to obtain competitive solutions within a reasonable computational budget for various NP-hard combinatorial optimization problems (Mei et al. 2009a, 2011a, b; Tang et al. 2009; Fuellerer et al. 2010; Bolduc et al. 2010; De Giovanni and Pezzella 2010; Sbihi 2010). In the following, two meta-heuristic approaches are proposed for solving TTP. The former is a Cooperative Co-evolution algorithm (CC) (Potter and De Jong 1994) that optimizes the TSP and KP decision variables separately and exchange the information between them regularly. The latter is a Memetic Algorithm (MA) (Moscato 1989) which considers TTP as a whole and optimizes all the decision variables simultaneously. Next, the two algorithms are described respectively. Then, their computational complexities are analysed.

3.1 Cooperative Co-evolution

The two sub-problems of TTP, i.e., TSP and KP, are both well-known combinatorial optimization problems. They have been investigated intensively, and various algorithms have been proposed for solving them (Lin and Kernighan 1973; Dorigo and Gambardella 1997; Horowitz and Sahni 1974; Fidanova 2007). However, the algorithm for TTP is not straightforward due to the interdependence between the TSP and KP decision variables in the objective. In this case, an intuitive approach is to optimize the TSP and KP decision variables separately and transfer the information between them during the optimization. The Cooperative Co-evolution (CC) (Potter and De Jong 1994) is a standard approach to this end. It decomposes the decision variables into a number of subcomponents and evolves them separately. The transfer of information is conducted by the collaboration between the subcomponents occurring in evaluation. When evaluating an individual of a subcomponent, it is combined with the collaborators (e.g., the individual with the best fitness value) that are selected from the other subcomponents. Then, its fitness is set corresponding to that of the combined individual(s) of the overall problem.

As mentioned in Wiegand et al. (2001), when selecting the collaborators, there are three main issues that affect the performance of the CC: *collaborator selection pressure*, *collaboration pool size* and *collaboration credit assignment*. They are described as follows:

- *Collaborator selection pressure*: The degree of greediness of selecting a collaborator. In general, if the subcomponents are independent from each other, then one should set the strongest selection pressure, i.e., select the best-so-far individuals as collaborators. On the other hand, for the non-linearly interdependent subcomponents, a weak selection pressure is more promising, e.g., selecting the collaborators randomly (Wiegand et al. 2001). Another empirical study (Stoen 2006) also showed that proportional selection performs better than random selection.
- *Collaboration pool size*: The number of collaborators selected from each other subcomponent. A larger pool size leads to a more comprehensive exploration of the solution space and thus a better final solution quality. However, it induces a higher time complexity since it requires more fitness evaluations to obtain the fitness of an individual. A better alternative is to adaptively change the pool size during the optimization process (Panait and Luke 2005).
- *Collaboration credit assignment*: The method of assigning the fitness value based on the objective values obtained together with the collaborators. The empirical studies (Wiegand et al. 2001) showed that the optimistic strategy that assigns the fitness of an individual as the objective value of its best collaboration generally leads to the best results.

Algorithm 1 The CC for TTP

1: **procedure** CC- TTP(k)
2: Randomly initialize the subpopulations $\mathbf{X}^{(0)} = \{\mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_N^{(0)}\}$ for TSP and $\mathbf{Z}^{(0)} = \{\mathbf{z}_1^{(0)}, \ldots, \mathbf{z}_N^{(0)}\}$ for KP;
3: Randomly select $\mathbf{CX}^{(0)} = \{\mathbf{cx}_1^{(0)}, \ldots, \mathbf{cx}_k^{(0)}\} \subseteq \mathbf{X}^{(0)}$;
4: Randomly select $\mathbf{CZ}^{(0)} = \{\mathbf{cz}_1^{(0)}, \ldots, \mathbf{cz}_k^{(0)}\} \subseteq \mathbf{Z}^{(0)}$;
5: Set $g = 0$;
6: **while** Stopping criteria are not met **do**
7: $(\mathbf{X}^{(g+1)}, \mathbf{CX}^{(g+1)}) = \text{solveTSP}(\mathbf{X}^{(g)}, \mathbf{CZ}^{(g)})$;
8: $(\mathbf{Z}^{(g+1)}, \mathbf{CZ}^{(g+1)}) = \text{solveKP}(\mathbf{Z}^{(g)}, \mathbf{CX}^{(g)})$;
9: $g \leftarrow g + 1$;
10: **end while**
11: **return** $(\mathbf{x}_1^{(g)}, \mathbf{z}_1^{(g)})$;
12: **end procedure**

Based on the previous studies, the collaboration strategy in the proposed CC for TTP is set as follows: when evaluating an individual of TSP (KP, resp.), the best k individuals of KP (TSP, resp.) are selected to be collaborators. Then, the fitness of the individual is set as the best objective value among the k objective values.

The issue of collaboration in CC has been overlooked so far, and most of the limited studies are focused on continuous optimization problems (Potter and De Jong 1994; Potter 1997; Wiegand et al. 2001; Bull 2001; Panait and Luke 2005; Stoen 2006). For the combinatorial optimization problems, Bonyadi and Moghaddam (2009) proposed a CC for multiprocessor task scheduling, in which the collaboration pool size equals the population size, i.e., all the individuals are selected as collaborators. Ibrahimov et al. (2012) proposed a CC for a simple two-silo supply chain problem, which selects the best individual plus two random individuals for collaboration.

The CC for TTP is depicted in Algo. 1. In lines 7 and 8, solveTSP() and solveKP() are described in Algos. 2 and 3, which have the same framework. First, two parents are selected randomly and the crossover operator is applied to them. Then, the local search process is conducted with the probability of P_{ls}. Finally, all the generated offsprings are combined with the original population and the best N (k, resp.) individuals are selected to form the new population (collaborators, resp.). In the algorithm, the fitness function $\mathcal{F}()$ returns the best objective value of all the collaborations, i.e.,

$$\mathcal{F}(\mathbf{x}, \mathbf{CZ}) = \max_{l \in \{1,\ldots,k\}} \{\mathcal{G}(\mathbf{x}, \mathbf{cz}_l)\} \tag{16}$$

$$\mathcal{F}(\mathbf{z}, \mathbf{CX}) = \max_{l \in \{1,\ldots,k\}} \{\mathcal{G}(\mathbf{cx}_l, \mathbf{z})\} \tag{17}$$

Note that in Eqs. (16) and (17), the objective function $\mathcal{G}(\mathbf{x}, \mathbf{z})$ does not take the starting city decision variables $\mathbf{y}$ into account. This is because in the algorithm, a TTP solution

Algorithm 2 Solve the TSP for one generation

1: **procedure** SOLVETSP($\mathbf{X}$, $\mathbf{CZ}$)
2: $\mathbf{X}' \leftarrow \mathbf{X}$;
3: **for** $i = 1 \rightarrow N_{off}$ **do** ▷ N_{off} is the number of offsprings
4: $\mathbf{Y} = \emptyset$;
5: Randomly pick two individuals $\mathbf{x}_{p_1}$ and $\mathbf{x}_{p_2}$ from $\mathbf{X}$;
6: $\mathbf{x}_{xover} = \text{OX}(\mathbf{x}_{p_1}, \mathbf{x}_{p_2})$;
7: **if** $\mathbf{x}_{xover}$ is different from all the individuals in $\mathbf{X}'$ **then**
8: $\mathbf{Y} = \{\mathbf{x}_{xover}\}$;
9: **end if**
10: Randomly sample r between 0 and 1;
11: **if** $r < P_{ls}$ **then**
12: $\mathbf{x}_{ls} = \mathbf{x}_{xover}$;
13: **repeat**
14: $\mathbf{x}_{nt} = \arg\max\{\mathcal{F}(\mathbf{x}_{nb}, \mathbf{CZ}) | \mathbf{x}_{nb} \in \text{2-opt}(\mathbf{x}_{ls})\}$;
15: **if** $\mathcal{F}(\mathbf{x}_{nt}, \mathbf{CZ}) > \mathcal{F}(\mathbf{x}_{ls}, \mathbf{CZ})$ **then**
16: $\mathbf{x}_{ls} \leftarrow \mathbf{x}_{nt}$;
17: **end if**
18: **until** there is no improvement on $\mathbf{x}_{ls}$
19: **if** $\mathbf{x}_{ls}$ is different from all the individuals in $\mathbf{X}'$ **then**
20: $\mathbf{Y} = \{\mathbf{x}_{ls}\}$;
21: **end if**
22: **end if**
23: $\mathbf{X}' \leftarrow \mathbf{X}' \cup \mathbf{Y}$;
24: **end for**
25: Sort $\mathbf{X}'$ in the decreasing order of $\mathcal{F}(\cdot)$ to obtain $\mathbf{X}''$;
26: $\mathbf{X}_{nt} = \{\mathbf{x}_1'', \ldots, \mathbf{x}_N''\}$, $\mathbf{CX}_{nt} = \{\mathbf{x}_1'', \ldots, \mathbf{x}_k''\}$;
27: **return** $(\mathbf{X}_{nt}, \mathbf{CX}_{nt})$;
28: **end procedure**

Algorithm 3 Solve the KP for one generation

1: **procedure** SOLVEKP($\mathbf{Z}$, $\mathbf{CX}$)
2: $\mathbf{Z}' \leftarrow \mathbf{Z}$;
3: **for** $i = 1 \rightarrow N_{off}$ **do** ▷ N_{off} is the number of offsprings
4: $\mathbf{Y} = \emptyset$;
5: Randomly pick two individuals $\mathbf{z}_{p_1}$ and $\mathbf{z}_{p_2}$ from $\mathbf{Z}$;
6: $\mathbf{z}_{xover} = \text{OPX}(\mathbf{z}_{p_1}, \mathbf{z}_{p_2})$;
7: **if** $\mathbf{z}_{xover}$ is different from all the individuals in $\mathbf{Z}'$ **then**
8: $\mathbf{Y} = \{\mathbf{z}_{xover}\}$;
9: **end if**
10: Randomly sample r between 0 and 1;
11: **if** $r < P_{ls}$ **then**
12: $\mathbf{z}_{ls} = \mathbf{z}_{xover}$;
13: **repeat**
14: $\mathbf{z}_{flip} = \arg\max\{\mathcal{F}(\mathbf{z}_{nb}, \mathbf{CX}) | \mathbf{z}_{nb} \in \text{Flip}(\mathbf{z}_{ls})\}$;
15: $\mathbf{z}_{ex} = \arg\max\{\mathcal{F}(\mathbf{z}_{nb}, \mathbf{CX}) | \mathbf{z}_{nb} \in \text{EX}(\mathbf{z}_{ls})\}$;
16: $\mathbf{z}_{nt} = \arg\max\{\mathcal{F}(\mathbf{z}_{flip}, \mathbf{CX}), \mathcal{F}(\mathbf{z}_{ex}, \mathbf{CX})\}$;
17: **if** $\mathcal{F}(\mathbf{z}_{nt}, \mathbf{CX}) > \mathcal{F}(\mathbf{z}_{ls}, \mathbf{CX})$ **then**
18: $\mathbf{z}_{ls} \leftarrow \mathbf{z}_{nt}$;
19: **end if**
20: **until** there is no improvement on $\mathbf{z}_{ls}$
21: **if** $\mathbf{z}_{ls}$ is different from all the individuals in $\mathbf{Z}'$ **then**
22: $\mathbf{Y} = \{\mathbf{z}_{ls}\}$;
23: **end if**
24: **end if**
25: $\mathbf{Z}' \leftarrow \mathbf{Z}' \cup \mathbf{Y}$;
26: **end for**
27: Sort $\mathbf{Z}'$ in the decreasing order of $\mathcal{F}(\cdot)$ to obtain $\mathbf{Z}''$;
28: $\mathbf{Z}_{nt} = \{\mathbf{z}_1'', \ldots, \mathbf{z}_N''\}$, $\mathbf{CZ}_{nt} = \{\mathbf{z}_1'', \ldots, \mathbf{z}_k''\}$;
29: **return** $(\mathbf{Z}_{nt}, \mathbf{CZ}_{nt})$;
30: **end procedure**

Algorithm 4 Calculation of the benefit of a TTP solution $\mathcal{G}(\mathbf{x}, \mathbf{z})$

1: **procedure** $\mathcal{G}(\mathbf{x}, \mathbf{z})$
2: Set $\bar{w} = 0, \bar{T} = 0, \bar{b} = 0$;
3: **for** $i = 1 \rightarrow n-1$ **do**
4: **for** $j = 1 \rightarrow m$ **do**
5: **if** $z_j = i$ **then**
6: $\bar{w} \leftarrow \bar{w} + w_j, \bar{b} \leftarrow \bar{b} + b_j$;
7: **end if**
8: **end for**
9: $\bar{T} \leftarrow \bar{T} + d_{x_i, x_{i+1}}/(v_{\max} - (v_{\max} - v_{\min})\bar{w}/Q)$;
10: **end for**
11: $\bar{T} \leftarrow \bar{T} + d_{x_n, x_1}/(v_{\max} - (v_{\max} - v_{\min})\bar{w}/Q)$;
12: $\mathcal{G}(\mathbf{x}, \mathbf{z}) = \bar{b} - R \cdot \bar{T}$;
13: **return** $\mathcal{G}(\mathbf{x}, \mathbf{z})$;
14: **end procedure**

$(\mathbf{x}, \mathbf{z})$ is represented as the combination of a TSP tour $\mathbf{x} = (x_1, \ldots, x_n)$ and a KP picking plan $\mathbf{z} = (z_1, \ldots, z_m)$. $\mathbf{x}$ is a permutation of the n cities, with $x_i \in \{1, \ldots, n\}, \forall i = 1, \ldots, n$, and $z_i \in A_i \cup \{0\}, \forall i = 1, \ldots, m$ indicates the city to pick the item i. $z_i = 0$ implies that item i is not picked throughout the way. The TSP tour naturally starts from city x_1. Thus, the starting city is implicitly determined by $\mathbf{x}$, and $\mathbf{y}$ can be eliminated. Given a TTP solution $(\mathbf{x}, \mathbf{z})$, its benefit is computed by Algo. 4. The computational complexity of $\mathcal{G}(\mathbf{x}, \mathbf{z})$ is $O(nm)$.

Conventional crossover and mutation operators for the TSP and KP are adopted here. Specifically, the ordered crossover (Oliver et al. 1987) and 2-opt (Croes 1958) operators are used for the TSP, and the traditional one-point crossover, flip and exchange operators are used for the KP. They are described in details as follows:

Ordered Crossover (OX): Given two tours $\mathbf{x}_1 = (x_{11}, \ldots, x_{1n})$ and $\mathbf{x}_2 = (x_{21}, \ldots, x_{2n})$, two cutting positions $1 \leq p \leq q \leq n$ are randomly selected, and $(x_{1p}, \ldots, x_{1q})$ is copied to the corresponding positions of the offspring $(x'_p, \ldots, x'_q)$. After that, $\mathbf{x}_2$ is scanned from position $q+1$ to the end and then from beginning to position q. The unduplicated elements are placed one after another in $\mathbf{x}'$ from position $q+1$ to the end, and then from beginning to position $p-1$. The complexity of the OX operator is $O(n)$.

2-opt: Given a tour $\mathbf{x} = (x_1, \ldots, x_n)$, two cutting positions $1 \leq p < q \leq n$ are chosen and the sub-tour in between is inverted. The offspring is $\mathbf{x}' = (x_1, \ldots, x_{p-1}, x_q, x_{q-1}, \ldots, x_p, x_{q+1}, \ldots, x_n)$. During the local search, the neighbourhood size defined by the 2-opt operator is $O(n^2)$.

One-Point Crossover (OPX): Given two picking plans $\mathbf{z}_1 = (z_{11}, \ldots, z_{1m})$ and $\mathbf{z}_2 = (z_{21}, \ldots, z_{2m})$, a cutting position $1 \leq p \leq m$ is picked, and then the offspring is set to $\mathbf{z}' = (z_{11}, \ldots z_{1(p-1)}, z_{2p}, \ldots z_{2m})$. The OPX operator has a computational complexity of $O(m)$.

Algorithm 5 The MA for solving the overall TTP

1: **procedure** MA- TTP
2: Randomly initialize $(\mathbf{X}^{(0)}, \mathbf{Z}^{(0)})$=$\{(\mathbf{x}_1^{(0)}, \mathbf{z}_1^{(0)}), \ldots, (\mathbf{x}_N^{(0)}, \mathbf{z}_N^{(0)})\}$;
3: Set $g = 0$;
4: **while** Stopping criteria are not met **do**
5: $(\mathbf{X}^{(g+1)}, \mathbf{Z}^{(g+1)}) = \text{solveTTP}(\mathbf{X}^{(g)}, \mathbf{Z}^{(g)})$;
6: $g \leftarrow g + 1$;
7: **end while**
8: **return** $(\mathbf{x}_1^{(g)}, \mathbf{z}_1^{(g)})$;
9: **end procedure**

Flip: Given a picking plan $\mathbf{z} = (z_1, \ldots, z_m)$, a position $1 \leq p \leq m$ is selected, and z_p is replaced by a different value $z'_p \in A_p \cup \{0\}$. During the local search, the neighbourhood size defined by the Flip operator is $O(\prod_{i=1}^{m} |A_i|) = O(nm)$.

Exchange (EX): Given a picking plan $\mathbf{z} = (z_1, \ldots, z_m)$, two positions $1 \leq p < q \leq m$ are selected, and the values of z_p and z_q are exchanged. To keep feasibility, it is required that $z_q \in A_p \cup \{0\}$ and $z_p \in A_q \cup \{0\}$. During the local search, the neighbourhood size defined by the EX operator is $O(m^2)$.

Besides the above CC, which will be referred to as the *Standard CC (SCC)* for the sake of clarity, a variation named the *Dynamic CC (DCC)* that dynamically updates the collaborators within each generation is developed. From lines 7 and 8 of Algo. 1, one can see that the collaborators are updated after all the sub-problems have been solved. Therefore, within each generation, the latter sub-problem (i.e., KP) cannot use the updated collaborators obtained by the former sub-problem (i.e., TSP). To increase efficiency, the DCC simply replaces line 8 with the following codes:

$$(\mathbf{Z}^{(g+1)}, \mathbf{CZ}^{(g+1)}) = \text{solveKP}(\mathbf{Z}^{(g)}, \mathbf{CX}^{(g+1)});$$

In other words, the old collaborators $\mathbf{CX}^{(g)}$ is replaced by the updated ones $\mathbf{CX}^{(g+1)}$.

3.2 Memetic Algorithm

Based on the above crossover and local search operators, a MA is proposed for the overall problem. In the MA, the TSP and KP are solved together by combining the aforementioned operators. To be specific, the crossover of a TTP solution is conducted by applying the OX and OPX operators to its tour and picking plan simultaneously. Then, during the local search, the neighbourhood of the current solution is defined as the union of the neighbourhoods induced by all the 2-opt, Flip and EX operators.

The framework of the proposed MA is described in Algo. 5. In line 5, solveTTP() is described in Algo. 6. The only difference between solveTTP() and solveTSP() or solveKP() is in lines 6–7 and lines 15–18, which are the crossover and neighbourhood definition during the local search, respectively.

Algorithm 6 Solve TTP for one generation

procedure SOLVETTP($\mathbf{X}, \mathbf{Z}$)
  $(\mathbf{X}', \mathbf{Z}') \leftarrow (\mathbf{X}, \mathbf{Z})$;
  for $i = 1 \rightarrow N_{off}$ **do** ▷ N_{off} is the number of offsprings
    $\mathbf{Y} = \emptyset$;
    Randomly pick $(\mathbf{x}_{p_1}, \mathbf{z}_{p_1})$ and $(\mathbf{x}_{p_2}, \mathbf{z}_{p_2})$ from $(\mathbf{X}, \mathbf{Z})$;
    $\mathbf{x}_{xover} = \text{OX}(\mathbf{x}_{p_1}, \mathbf{x}_{p_2})$;
    $\mathbf{z}_{xover} = \text{OPX}(\mathbf{z}_{p_1}, \mathbf{z}_{p_2})$;
    if $(\mathbf{x}_{xover}, \mathbf{z}_{xover})$ is not a clone in $(\mathbf{X}', \mathbf{Z}')$ **then**
      $\mathbf{Y} = \{(\mathbf{x}_{xover}, \mathbf{z}_{xover})\}$;
    end if
    Randomly sample r between 0 and 1;
    if $r < P_{ls}$ **then**
      $(\mathbf{x}_{ls}, \mathbf{z}_{ls}) = (\mathbf{x}_{xover}, \mathbf{z}_{xover})$;
      repeat
        $\mathbf{x}_{opt} = \arg\max\{\mathcal{G}(\mathbf{x}_{nb}, \mathbf{z}_{ls}) | \mathbf{x}_{nb} \in \text{2-opt}(\mathbf{x}_{ls})\}$;
        $\mathbf{z}_{flip} = \arg\max\{\mathcal{G}(\mathbf{x}_{ls}, \mathbf{z}_{nb}) | \mathbf{z}_{nb} \in \text{Flip}(\mathbf{z}_{ls})\}$;
        $\mathbf{z}_{ex} = \arg\max\{\mathcal{G}(\mathbf{x}_{ls}, \mathbf{z}_{nb}) | \mathbf{z}_{nb} \in \text{EX}(\mathbf{z}_{ls})\}$;
        $(\mathbf{x}_{nt}, \mathbf{z}_{nt}) = \arg\max\{\mathcal{G}(\mathbf{x}_{opt}, \mathbf{z}_{ls}), \mathcal{G}(\mathbf{x}_{ls}, \mathbf{z}_{flip}), \mathcal{G}(\mathbf{x}_{ls}, \mathbf{z}_{ex})\}$;
        if $\mathcal{G}(\mathbf{x}_{nt}, \mathbf{z}_{nt}) > \mathcal{G}(\mathbf{x}_{ls}, \mathbf{z}_{ls})$ **then**
          $(\mathbf{x}_{ls}, \mathbf{z}_{ls}) \leftarrow (\mathbf{x}_{nt}, \mathbf{z}_{nt})$;
        end if
      until there is no improvement on $(\mathbf{x}_{ls}, \mathbf{z}_{ls})$
      if $(\mathbf{x}_{ls}, \mathbf{z}_{ls})$ is not a clone in $(\mathbf{X}', \mathbf{Z}')$ **then**
        $\mathbf{Y} = \{(\mathbf{x}_{ls}, \mathbf{z}_{ls})\}$;
      end if
    end if
    $(\mathbf{X}', \mathbf{Z}') \leftarrow (\mathbf{X}', \mathbf{Z}') \cup \mathbf{Y}$;
  end for
  Sort $(\mathbf{X}', \mathbf{Z}')$ in the decreasing order of $\mathcal{G}(\cdot)$ to obtain $(\mathbf{X}'', \mathbf{Z}'')$;
  $\mathbf{X}_{nt} = \{\mathbf{x}''_1, \ldots, \mathbf{x}''_N\}$, $\mathbf{Z}_{nt} = \{\mathbf{z}''_1, \ldots, \mathbf{z}''_N\}$;
  return $(\mathbf{X}_{nt}, \mathbf{Z}_{nt})$;
end procedure

3.3 Computational complexity analysis

The computational complexities of the proposed algorithms are as follows:

$$O(\text{CC}) = g_{\max}(O(\text{solveTSP}) + O(\text{solveKP})) \tag{18}$$

$$O(\text{solveTSP}) = N_{off}(O(\text{OX}) + O(\mathcal{F}) + P_{ls}L_1S_1O_{ls}(\mathcal{F})) + O(\text{sort}) \tag{19}$$

$$O(\text{solveKP}) = N_{off}(O(\text{OPX}) + O(\mathcal{F}) + P_{ls}L_2S_2O_{ls}(\mathcal{F})) + O(\text{sort}) \tag{20}$$

$$O(\text{MA}) = g_{\max}O(\text{solveTTP}) \tag{21}$$

$$O(\text{solveTTP}) = N_{off}(O(\text{OX}) + O(\text{OPX}) + 2O(\mathcal{G}) + P_{ls}L_3S_3O_{ls}(\mathcal{G})) + O(\text{sort}), \tag{22}$$

where $g_{\max}$ is the maximal number of generations, and N_{off} is the number of offsprings generated in each generation. L_1, L_2 and L_3 stand for the average number of local search steps for TSP, KP and TTP, and S_1, S_2 and S_3 are the neighbourhood sizes of the local search processes in TSP, KP and TTP, respectively. $O(\cdot)$ stands for the complexity of the corresponding algorithm or operation, and $O_{ls}(\cdot)$ indicates the complexity of evaluating a neighbouring solution with respect to $\mathcal{F}$ or $\mathcal{G}$ during the local search. Given the current solution $\mathbf{s}$ and $\mathcal{G}(\mathbf{s})$, the evaluation for each neighbouring solution may be much faster by computing the difference on $\mathcal{G}$ caused by the modification, i.e., $\mathcal{G}(\mathbf{s}') = \mathcal{G}(\mathbf{s}) + \Delta\mathcal{G}(\mathbf{s}, \mathbf{s}')$. For example, when applying the 2-opt operator to TSP that minimizes the total distance, we have $O_{ls}(tc(\mathbf{s}')) = O(\Delta tc(\mathbf{s}, \mathbf{s}')) = O(1)$, which is much lower than $O(tc(\mathbf{s})) = O(n)$. However, in TTP, $O_{ls}(\mathcal{G}) = O(\mathcal{G}) = O(nm)$, since it is still necessary to calculate the speed between each pair of adjacent cities in the tour. Then, based on Eqs. (16) and (17), we have $O(\mathcal{F}) = kO(\mathcal{G}) = kO(nm)$ and $O_{ls}(\mathcal{F}) = kO_{ls}(\mathcal{G}) = kO(nm)$.

Besides, we already have

$$S_1 = S(\text{2-opt}) = O(n^2) \tag{23}$$

$$S_2 = S(\text{Flip}) + S(\text{EX}) = O(nm) + O(m^2) \tag{24}$$

$$S_3 = S(\text{2-opt}) + S(\text{Flip}) + S(\text{EX}) = O(n^2) + O(nm) + O(m^2) \tag{25}$$

It is also known that $O(\text{OX}) = O(n)$, $O(\text{OPX}) = O(m)$ and $O(\text{sort}) = O(N_{off} \log N_{off})$. Clearly, the complexities of the algorithms are dominated by that of the local search. Then, we have

$$O(\text{CC}) = kg_{\max}N_{off}P_{ls}(L_1O(n^3m) + L_2O(n^2m^2) + L_2O(nm^3))) \tag{26}$$

$$O(\text{MA}) = g_{\max}N_{off}P_{ls}(L_3O(n^3m) + L_3O(n^2m^2) + L_3O(nm^3))) \tag{27}$$

Under the assumption that L_1, L_2 and L_3 are nearly the same, the computational complexity of CC is approximately k times as that of MA. In other words, when $k = 1$, CC and MA is expected to have comparable computational complexity. This will be verified in the experimental studies.

4 Experimental studies

In this section, the proposed CC and MA are compared on the TTP benchmark instances to investigate their performance.

4.1 Experimental settings

A representative subset of the TTP benchmark instances generated by Bonyadi et al.[1] is selected to compare the performance of the proposed algorithms. The benchmark set includes instances with various features with the number of cities n from 10 to 100 and number of items m from 10 to 150. For each parameter setting of the problem, 10 instances were generated randomly. As a result, there are totally 540

[1] The benchmark instances can be downloaded from http://cs.adelaide.edu.au/~ec/research/ttp.php.

instances. For the sake of simplicity, for each parameter setting with $10 \leq n, m \leq 100$, only the first instance is chosen from the 10 generated instances as a representative. The selected subset consists of 39 instances. Note that for some instances, the benefit may be negative due to the insufficient values of the items compared to the knapsack rent.

The complete parameter settings of the compared algorithms are given in Table 1. The population size, number of offsprings and probability of local search are set in the standard way that has been verified to be effective on sim-

Table 1 The parameter settings of the compared algorithms

Parameter	Description	Value
k	Number of collaborators in CC	1, 3
N	Population (subpopulation for CC) size	30
N_{off}	Number of offsprings	$6 \cdot psize$
P_{ls}	Probability of local search	0.2
$g_{\max}$	Maximal generations	100 for MA; $100/k$ for CCs

Table 2 Mean and standard deviation of the benefits obtained by the 30 independent runs of the proposed algorithms on the benchmark instances from 10-10-1-25 to 20-30-1-75

Instance	$k = 1$		$k = 3$		MA
	SCC	DCC	SCC	DCC	
10-10-1-25					
Mean	−17,115.3	−17,192.8	−16,820.7	−16,773.1	**−16,566.3**
SD	461.5	427.1	406.6	322.5	0.0
10-10-1-50					
Mean	1,887.2	1,775.1	1,925.8	1,904.8	**1,994.5**
SD	134.7	192.5	66.1	59.2	0.0
10-10-1-75					
Mean	−2,258.4	−2,629.4	−1,925.7	−1,980.2	**−1,877.6**
SD	402.5	417.1	137.9	223.0	0.0
10-15-1-25					
Mean	217.6	111.7	349.2	260.7	**389.4**
SD	154.6	196.6	65.2	115.3	0.0
10-15-1-50					
Mean	1,028.6	846.7	1,188.2	1,119.6	**1,295.1**
SD	270.7	287.6	144.2	197.1	0.0
10-15-1-75					
Mean	−6,525.7	−6,680.0	−6,332.4	−6,341.8	**−6,261.8**
SD	390.0	345.1	97.3	92.5	0.0
20-10-1-25					
Mean	721.3	620.6	683.8	662.5	**901.6**
SD	141.5	99.4	47.9	70.1	0.0
20-10-1-50					
Mean	1,995.5	1,955.7	1,939.4	1,933.5	**2,238.2**
SD	148.9	180.4	87.6	96.6	1.0
20-10-1-75					
Mean	−1,935.1	−2,105.2	−1,729.8	−1,859.7	**−1,596.7**
SD	249.2	306.4	180.5	256.3	0.0
20-20-1-25					
Mean	−1,777.8	−1,920.3	−1,644.1	−1,800.5	**−1,581.7**
SD	202.8	361.8	106.1	192.3	0.0
20-20-1-50					
Mean	−2,518.9	−2,635.6	−2,048.0	−2,037.7	**−1,685.8**
SD	603.7	612.4	394.9	469.6	0.0
20-20-1-75					
Mean	−44,352.9	−45,522.7	−44,058.3	−44,309.7	**−43,541.8**
SD	933.4	1,386.7	542.3	756.7	0.0
20-30-1-25					
Mean	−1,624.3	−1,814.8	−1,350.0	−1,515.0	**−1,219.5**
SD	543.6	500.9	111.8	329.9	4.8
20-30-1-50					
Mean	−1,013.0	−989.3	−598.6	−760.0	**−337.0**
SD	708.6	803.3	316.8	409.6	0.0
20-30-1-75					
Mean	−18,494.8	−19,204.0	−18,393.9	−18,640.2	**−17,226.9**
SD	1,186.5	1,443.7	852.4	1,171.0	135.1

The result of the algorithm that performed significantly better than the other compared algorithms is marked in bold

Table 3 Mean and standard deviation of the benefits obtained by the 30 independent runs of the proposed algorithms on the benchmark instances from 50-15-1-25 to 50-75-1-75

Instance	$k = 1$		$k = 3$		MA
	SCC	DCC	SCC	DCC	
50-15-1-25					
Mean	−1,266.1	−1,493.3	−1,326.6	−1,358.1	**−1,151.7**
SD	136.1	217.5	112.9	142.9	15.6
50-15-1-50					
Mean	−1,476.9	−1,775.1	−1,474.4	−1,709.2	**−1,100.2**
SD	232.4	347.4	222.7	297.8	62.9
50-15-1-75					
Mean	−23,999.8	−24,523.5	−24,104.9	−24,372.0	**−23,221.5**
SD	431.7	525.1	305.5	470.5	0.0
50-25-1-25					
Mean	−12,569.7	−12,964.8	−12,393.4	−12,363.0	**−11,701.4**
SD	523.7	599.8	415.4	412.2	0.0
50-25-1-50					
Mean	−153,764.3	−154,996.7	−153,233.6	−154,297.0	**−150,781.9**
SD	1,766.8	943.0	1,962.8	1,746.6	210.8
50-25-1-75					
Mean	−27,582.0	−27,816.8	−27,460.2	−27,311.9	**−26,022.0**
SD	667.3	884.9	708.3	814.9	153.0
50-50-1-25					
Mean	−20,895.9	−21,536.8	−20,599.1	−21,280.3	**−19,495.6**
SD	1,002.3	1,120.2	971.7	980.2	283.6
50-50-1-50					
Mean	−125,718.1	−126,632.7	−124,665.5	−125,709.3	**−123,097.5**
SD	1,759.2	3,111.9	1,347.8	2,246.2	344.1
50-50-1-75					
Mean	−258,700.5	−262,492.2	−257,906.2	−259,926.5	**−253,588.4**
SD	3,610.6	4,451.3	4,150.0	4,269.2	930.4
50-75-1-25					
Mean	−57,809.0	−59,733.5	−57,730.3	−58,590.6	**−56,247.7**
SD	1,651.9	1,631.1	1,507.8	2,077.8	615.9
50-75-1-50					
Mean	−11,871.2	−13,018.4	−11,549.8	−12,519.2	**−8,988.6**
SD	1,899.1	2,085.6	2,094.8	2,028.2	47.4
50-75-1-75					
Mean	17,035.7	15,965.1	17,174.0	16,964.9	**18,931.6**
SD	1,445.8	1,772.3	870.0	948.1	73.6

The result of the algorithm that performed significantly better than the other compared algorithms is marked in bold

ilar combinatorial optimization problems (Tang et al. 2009; Mei et al. 2011a). For CC, $k = 1$ and $k = 3$ are tested to investigate the effect of k on the performance of CC. The number of generations is set to 100 for MA and CC with $k = 1$. For CC with $k = 3$, the number of generations is set to $100/k = 34$ to make the compared algorithms have similar total number of fitness evaluations. Each algorithm is run 30 times independently.

4.2 Results and discussions

First, the average performance of the proposed algorithms are compared. Tables 2, 3, 4 show the mean and standard deviation of the final benefits obtained by the 30 independent runs of SCC, DCC and MA on the benchmark instances, whose features are included in their names. For an instance named n–m–ID–τ, n and m stand for the number of cities and items, ID is the identity of the instance (all are 1's here, since they are the first instance in each category), and τ indicates the tightness of the capacity constraint, which is the capacity of the knapsack over the total weight of the items. For each instance, the result of the algorithm that performed significantly better than the other compared algorithms using the Wilcoxon's rank sum test (Wilcoxon 1945) under the confidence level of 0.05 is marked in bold.

It can been seen that MA obtained significantly better results than SCC and DCC with both $k = 1$ and $k = 3$ on all the 39 benchmark instances, with larger mean and smaller standard deviation. This implies that MA can obtain better solutions more reliably. For both tested k values, SCC generally obtained better solutions than DCC, which indicates that it is better to update the collaborators after solving all the sub-problems. This is because updating the collaborators too frequently will mislead the search to a local optimum quickly and make it difficult to jump out of the local optimum due to the strong selection pressure.

Table 4 Mean and standard deviation of the benefits obtained by the 30 independent runs of the proposed algorithms on the benchmark instances from 100-10-1-25 to 100-100-1-75

Instance	$k = 1$		$k = 3$		MA
	SCC	DCC	SCC	DCC	
100-10-1-25					
Mean	−1,598.6	−1,603.8	−1,521.1	−1,550.6	**−1,452.0**
SD	104.0	81.1	58.4	71.8	8.7
100-10-1-50					
Mean	−1,708.5	−1,919.0	−1,940.0	−1,965.5	**−1,620.0**
SD	77.0	202.0	90.2	108.0	36.0
100-10-1-75					
Mean	−9,974.2	−10,270.7	−9,838.7	−9,835.8	**−9,420.8**
SD	318.8	326.5	254.7	266.9	38.1
100-25-1-25					
Mean	−17,731.5	−17,990.0	−17,534.1	−17,734.0	**−16,916.2**
SD	440.9	705.2	320.3	414.5	92.3
100-25-1-50					
Mean	−12,558.3	−12,861.6	−12,474.2	−12,642.6	**−11,708.5**
SD	450.9	593.0	321.2	422.2	147.1
100-25-1-75					
Mean	−83,477.8	−84,017.5	−83,679.1	−83,612.2	**−81,099.3**
SD	765.9	1,064.0	754.0	1,319.4	597.7
100-50-1-25					
Mean	−89,396.4	−89,761.7	−89,699.0	−89,785.8	**−87,898.0**
SD	1,090.1	956.5	1,241.0	1,232.4	477.6
100-50-1-50					
Mean	−26,801.8	−27,615.3	−26,980.4	−27,363.8	**−25,571.8**
SD	745.9	1,262.0	652.4	969.4	95.2
100-50-1-75					
Mean	−47,060.8	−47,577.0	−47,162.0	−47,789.6	**−44,965.7**
SD	1,046.4	1,492.4	1,255.3	1,107.5	296.5
100-100-1-25					
Mean	1,222.8	1,148.5	957.8	829.8	**2,282.0**
SD	795.3	845.4	429.9	505.3	150.1
100-100-1-50					
Mean	−66,873.1	−67,377.2	−66,590.0	−67,453.6	**−62,986.7**
SD	1,885.5	2,244.6	1,776.7	2,199.2	729.8
100-100-1-75					
Mean	−141,786.2	−141,796.0	−141,958.2	−142,773.6	**−135,169.7**
SD	3,118.6	3,503.9	3,435.2	3,688.5	1,237.0

The result of the algorithm that performed significantly better than the other compared algorithms is marked in bold

Among the proposed CC algorithms, SCC and DCC with $k = 3$ outperformed the ones with $k = 1$ for all the instances except the large ones ($n = 100$ and $m \geq 50$). This shows that for the instances with small or medium solution space, a larger k can lead to a better result since it has a wider neighborhood and thus is stronger in exploration. On the other hand, for the large-scale instances, a smaller k is a better option to allow more generations given a fixed total number of fitness evaluations.

Tables 5, 6, 7 show the benefits of the best solution and average number of fitness evaluations of the proposed algorithms on the benchmark instances. The best benefits among the compared ones are marked in bold. During the local search, each computation of the objective value of the neighbouring solutions is considered as a complete fitness evaluation, given that there is no simplified evaluation as in TSP or KP alone.

From the tables, one can see that the best performance of the algorithms is consistent with the average performance. MA performed the best. It managed to obtain the best solutions on all the benchmark instances. SCC with $k = 1$ comes next, obtaining the best solutions on 25 out of the 39 instances. DCC with $k = 1$ performed worse than the corresponding SCC, only achieving the best solutions on 15 instances. Both SCC and DCC with $k = 3$ obtained the best solutions on 13 instances. In terms of computational effort, one can see that the compared algorithms have comparable average number of fitness evaluations when the problem size is not large. This is consistent with the analysis in Eqs. (26) and (27) and indicates that the average number of local search steps L_1, L_2 and L_3 are nearly the same for the small- and medium-sized instances. For the larger instances ($m, n \geq 50$), SCCs require much more fitness evaluations than the other compared algorithms. Note that DCC generally needs less fitness evaluations than SCC, especially on the larger instances. This is because the dynamic change of the collaborators speeds up the convergence of the search process and thus reduces the number of steps (L_1 and L_2

Table 5 The benefits of the best solution and average number of fitness evaluations of the proposed algorithms from 10-10-1-25 to 20-30-1-75

Instance	$k = 1$		$k = 3$		MA
	SCC	DCC	SCC	DCC	
10-10-1-25					
Benefit	**−16,566.3**	**−16,566.3**	**−16,566.3**	**−16,566.3**	**−16,566.3**
No. eval	1.82e+06	2.08e+06	1.59e+06	1.64e+06	1.78e+06
10-10-1-50					
Benefit	**1,994.5**	**1,994.5**	1,963.6	1,963.6	**1,994.5**
No. eval	1.88e+06	1.88e+06	1.57e+06	1.47e+06	2.17e+06
10-10-1-75					
Benefit	**−1,877.6**	**−1,877.6**	**−1,877.6**	**−1,877.6**	**−1,877.6**
No. eval	2.28e+06	2.62e+06	1.89e+06	1.70e+06	2.41e+06
10-15-1-25					
Benefit	**389.4**	**389.4**	**389.4**	**389.4**	**389.4**
No. eval	3.10e+06	3.39e+06	2.57e+06	2.39e+06	3.53e+06
10-15-1-50					
Benefit	**1,295.1**	1,241.8	**1,295.1**	1,241.8	**1,295.1**
No. eval	3.53e+06	3.99e+06	2.51e+06	2.25e+06	3.41e+06
10-15-1-75					
Benefit	**−6,261.8**	**−6,261.8**	**−6,261.8**	−6,317.3	**−6,261.8**
No. eval	4.57e+06	5.61e+06	3.27e+06	2.67e+06	5.24e+06
20-10-1-25					
Benefit	**901.6**	828.7	716.8	709.2	**901.6**
No. eval	4.74e+06	4.44e+06	5.05e+06	4.68e+06	5.03e+06
20-10-1-50					
Benefit	**2,238.4**	**2,238.4**	2,064.8	2,064.8	**2,238.4**
No. eval	6.18e+06	5.73e+06	5.66e+06	5.22e+06	6.53e+06
20-10-1-75					
Benefit	**−1,596.7**	**−1,596.7**	**−1,596.7**	**−1,596.7**	**−1,596.7**
No. eval	5.82e+06	5.88e+06	5.56e+06	4.87e+06	8.71e+06
20-20-1-25					
Benefit	**−1,581.7**	**−1,581.7**	**−1,581.7**	**−1,581.7**	**−1,581.7**
No. eval	8.63e+06	1.12e+07	8.34e+06	7.54e+06	1.17e+07
20-20-1-50					
Benefit	**−1,685.8**	**−1,685.8**	**−1,685.8**	**−1,685.8**	**−1,685.8**
No. eval	9.41e+06	8.90e+06	1.05e+07	7.40e+06	1.16e+07
20-20-1-75					
Benefit	**−43,541.8**	**−43,541.8**	**−43,541.8**	**−43,541.8**	**−43,541.8**
No. eval	1.24e+07	1.49e+07	1.11e+07	8.43e+06	1.26e+07
20-30-1-25					
Benefit	**−1,218.3**	−1,237.3	**−1,218.3**	**−1,218.3**	**−1,218.3**
No. eval	1.63e+07	2.21e+07	1.47e+07	1.04e+07	2.22e+07
20-30-1-50					
Benefit	**−337.0**	**−337.0**	**−337.0**	**−337.0**	**−337.0**
No. eval	1.85e+07	2.31e+07	1.73e+07	1.17e+07	1.34e+07
20-30-1-75					
Benefit	**−17,191.4**	**−17,191.4**	**−17,191.4**	**−17,191.4**	**−17,191.4**
No. eval	2.42e+07	1.78e+07	2.41e+07	1.14e+07	1.28e+07

The best benefits among the compared ones are marked in bold

in Eq. (26)) to reach the local optimum. Besides, given the same number of generations, the number of fitness evaluations increases significantly with the increase of n and m, which is mainly induced by the increase of the neighbourhood sizes $S_1 = O(n^2)$, $S_2 = O(nm) + O(m^2)$ and $S_3 = O(n^2) + O(nm) + O(m^2)$.

The convergence curves of the compared algorithms on selected representative instances are shown in Figs. 2, 3, 4, 5, 6, 7, 8, where the x-axis and y-axis stand for the fitness evaluations and the average benefit of the best-so-far solutions of different runs, respectively. The selected instances include the following four diversified categories: (1) small n and m; (2) small n and large m; (3) large n and small m and (4) large n and m. Obviously, MA performed significantly better than the CC algorithms. In almost all the instances, the curve of MA is consistently above that of the other compared algorithms. Since MA solves TTP as a whole, its outperformance over the CC algorithms verifies the importance of considering the interdependence between the sub-problems of TTP.

Between the CC algorithms, one can see that DCC converges much faster, but generally obtained worse final results

Table 6 The benefits of the best solution and average number of fitness evaluations of the proposed algorithms from 50-15-1-25 to 50-75-1-75

Instance	$k = 1$		$k = 3$		MA
	SCC	DCC	SCC	DCC	
50-15-1-25					
Benefit	**−1,136.1**	−1,160.6	−1,213.4	−1,236.9	**−1,136.1**
No. eval	4.12e+07	2.92e+07	4.12e+07	3.36e+07	4.30e+07
50-15-1-50					
Benefit	**−1,059.5**	−1,189.2	−1,194.9	−1,294.1	**−1,059.5**
No. eval	4.85e+07	3.10e+07	4.43e+07	3.24e+07	4.36e+07
50-15-1-75					
Benefit	**−23,221.5**	−23,542.6	−23,347.0	**−23,221.5**	**−23,221.5**
No. eval	3.94e+07	3.32e+07	3.63e+07	3.10e+07	4.26e+07
50-25-1-25					
Benefit	**−11,701.4**	**−11,701.4**	−11,893.3	−11,893.3	**−11,701.4**
No. eval	5.19e+07	4.26e+07	5.52e+07	3.89e+07	5.90e+07
50-25-1-50					
Benefit	**−150,705.8**	−154,033.0	−150,729.4	−150,729.4	**−150,705.8**
No. eval	3.73e+07	3.01e+07	3.99e+07	3.64e+07	3.60e+07
50-25-1-75					
Benefit	−26,017.8	−26,259.9	−26,017.8	**−25,911.7**	**−25,911.7**
No. eval	6.07e+07	5.47e+07	5.40e+07	4.12e+07	7.47e+07
50-50-1-25					
Benefit	**−19,391.0**	−19,410.1	−19,646.6	−19,674.5	**−19,391.0**
No. eval	1.18e+08	5.73e+07	1.10e+08	5.62e+07	8.86e+07
50-50-1-50					
Benefit	−123,524.3	−122,964.5	**−122,793.1**	**−122,793.1**	**−122,793.1**
No. eval	1.30e+08	5.26e+07	1.47e+08	6.65e+07	9.26e+07
50-50-1-75					
Benefit	**−253,204.8**	−254,506.7	−253,247.2	−253,247.2	**−253,204.8**
No. eval	1.53e+08	5.03e+07	1.33e+08	6.39e+07	7.39e+07
50-75-1-25					
Benefit	−55,895.0	−56,600.1	−56,005.5	−55,931.8	**−55,836.8**
No. eval	1.92e+08	5.08e+07	1.91e+08	6.79e+07	1.04e+08
50-75-1-50					
Benefit	**−8,961.4**	**−8,961.4**	−9,441.9	−9,403.1	**−8,961.4**
No. eval	3.64e+08	9.42e+07	3.53e+08	9.55e+07	1.52e+08
50-75-1-75					
Benefit	**18,952.0**	**18,952.0**	17,998.9	17,998.9	**18,952.0**
No. eval	4.37e+08	4.01e+08	4.74e+08	1.25e+08	1.43e+08

The best benefits among the compared ones are marked in bold

than the corresponding SCC. This implies that the combination of $k = 1$ and dynamic update of the collaborators leads to such a strong selection pressure that the search process become stuck in a local optimum at very early stage and can hardly jump out of it. In most of the instances, the CC algorithms with $k = 3$ converged slower than the ones with $k = 1$ at the earlier stage of the search. This is due to the much larger number of fitness evaluations (nearly k times) within each generation. However, their curves intersect the ones with $k = 1$ (e.g., Figs. 4, 7), and finally outperformed the CCs with $k = 1$.

In summary, the competitiveness of the proposed MA sheds a light on developing algorithms for complex real-world problems consisting of interdependent sub-problems. First, by solving TTP as a whole, MA can be seen as considering the interdependence between the sub-problems more comprehensively than CC. Second, the properly designed framework and employed operators leads to a comparable computational complexity with CC. In other words, MA explores the solution space more effectively than CC by choosing better "directions" during the search process. This is similar to the ideas of the numerical optimization methods that use the gradient information such as the steepest descent and Quasi-Newton methods, and CMA-ES (Hansen 2006) in the evolutionary computation field. This implies that when tackling the interdependence between the sub-problems, the major issues should be designing a proper measure that can reflect the gradient or dependence of the objective value on the change of decision variables in the complex combinatorial solution space, based on which one can find the best "direction" during the search process.

5 Conclusion

This paper investigates the interdependence between sub-problems of a complex problem in the context of TTP, which is a simple but representative benchmark problem.

Table 7 The benefits of the best solution and average number of fitness evaluations of the proposed algorithms from 100-10-1-25 to 100-100-1-75

Instance	$k = 1$		$k = 3$		MA
	SCC	DCC	SCC	DCC	
100-10-1-25					
Benefit	−1,450.3	−1,448.0	−1,445.4	−1,440.1	**−1,437.2**
No. eval	1.65e+08	1.11e+08	1.94e+08	1.64e+08	1.40e+08
100-10-1-50					
Benefit	**−1,589.3**	−1,619.8	−1,805.4	−1,800.5	**−1,589.3**
No. eval	2.10e+08	1.29e+08	2.34e+08	1.95e+08	1.69e+08
100-10-1-75					
Benefit	−9,423.4	−9,663.8	−9,485.5	−9,460.3	**−9,331.3**
No. eval	1.91e+08	1.39e+08	2.51e+08	2.16e+08	1.99e+08
100-25-1-25					
Benefit	−16,866.6	−16,858.7	−17,039.0	−17,008.3	**−16,817.0**
No. eval	2.10e+08	1.31e+08	2.19e+08	1.86e+08	1.92e+08
100-25-1-50					
Benefit	−11,710.2	−11,624.1	−11,910.4	−11,878.1	**−11,562.8**
No. eval	2.53e+08	1.50e+08	2.69e+08	2.09e+08	2.34e+08
100-25-1-75					
Benefit	−82,287.8	−82,290.6	−82,378.5	−80,833.9	**−80,596.2**
No. eval	2.41e+08	1.26e+08	2.51e+08	1.93e+08	2.35e+08
100-50-1-25					
Benefit	−87,315.8	−87,931.1	−87,674.8	−87,933.0	**−87,229.0**
No. eval	2.73e+08	1.37e+08	3.36e+08	2.03e+08	2.29e+08
100-50-1-50					
Benefit	−25,511.2	−25,590.5	−25,751.7	−25,719.4	**−25,504.9**
No. eval	4.30e+08	2.13e+08	4.26e+08	2.29e+08	3.90e+08
100-50-1-75					
Benefit	−44,720.3	−45,194.4	−44,990.5	−45,700.9	**−44,524.1**
No. eval	4.67e+08	2.25e+08	4.47e+08	2.31e+08	3.43e+08
100-100-1-25					
Benefit	2,044.5	2,199.2	1,498.4	1,577.6	**2,434.0**
No. eval	9.27e+08	6.11e+08	1.09e+09	3.82e+08	6.92e+08
100-100-1-50					
Benefit	−63,072.6	−63,160.9	−63,343.3	−64,201.6	**−61,957.9**
No. eval	1.25e+09	2.39e+08	1.25e+09	3.66e+08	6.66e+08
100-100-1-75					
Benefit	−134,104.9	−135,781.0	−134,622.6	−135,440.8	**−133,676.2**
No. eval	9.48e+08	2.76e+08	1.10e+09	4.17e+08	7.48e+08

The best benefits among the compared ones are marked in bold

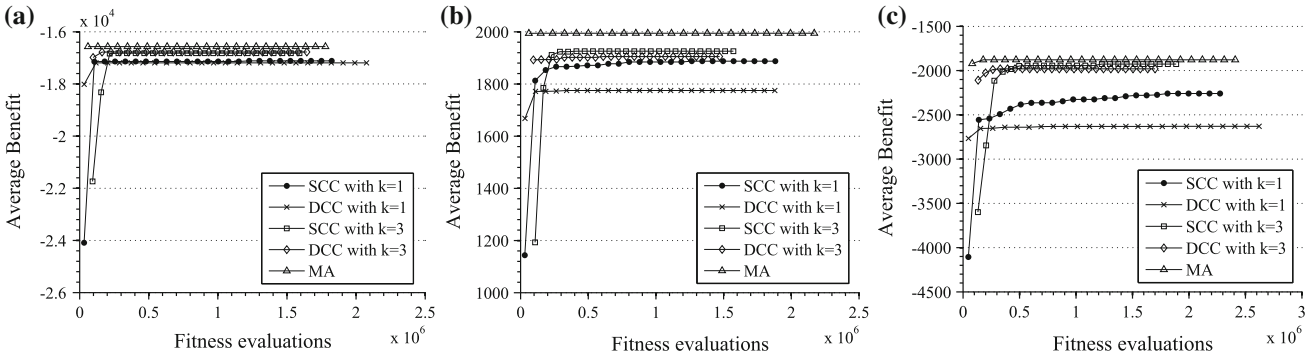

Fig. 2 Convergence curves of the compared algorithms on the TTP instances with $n = 10$ and $m = 10$

The analysis is conducted both theoretically and empirically. At first, the mathematical formulations of TTP show that the non-linear interdependence of the sub-problems lying in the objective function makes it difficult to decompose the problem into independent sub-problems, if not impossible. The NP-hardness also makes the exact methods only applicable for small-sized instances. Then, a CC, which further consists of a standard and a dynamic version, and a MA is proposed to solve the problem approximately. The former optimizes the sub-problems separately and exchanges the information in each generation, while the latter solves the problem as a whole. The outperformance of MA over CC on the bench-

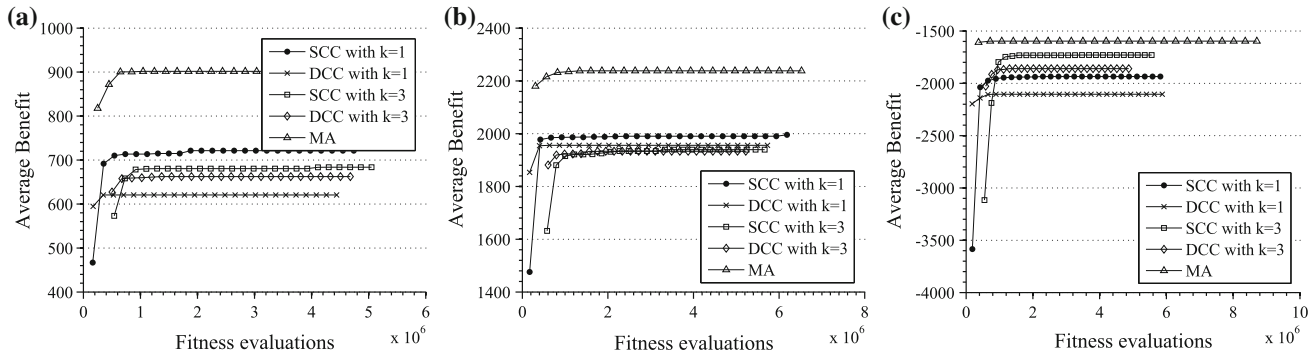

Fig. 3 Convergence curves of the compared algorithms on the TTP instances with $n = 20$ and $m = 10$

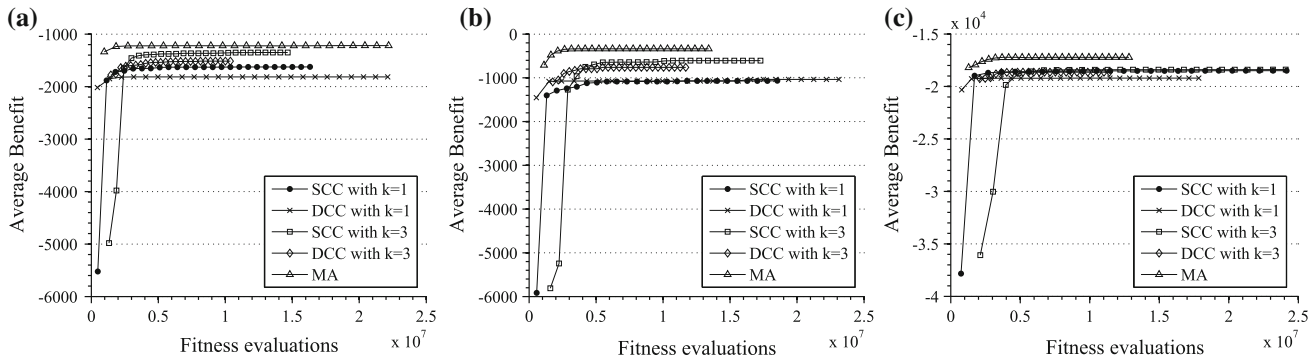

Fig. 4 Convergence curves of the compared algorithms on the TTP instances with $n = 20$ and $m = 30$

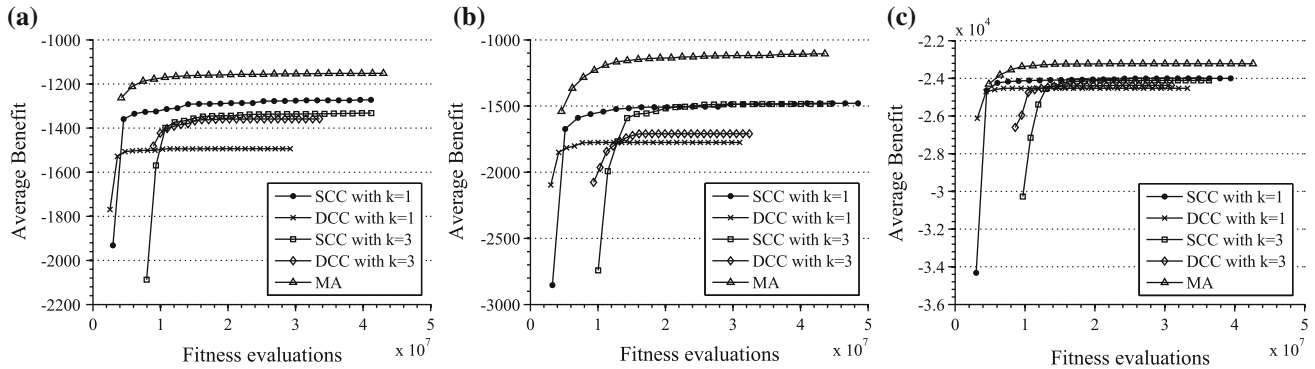

Fig. 5 Convergence curves of the compared algorithms on the TTP instances with $n = 50$ and $m = 15$

mark instances illustrates the importance of considering the interdependence between sub-problems. The significance of the research reported here may go beyond just TTP because there are other similar problems that are composed of two or more sub-problems, each of which is an NP-hard problem. For example, Gupta and Yao (2002) described a combined Vehicle Routing with Time Windows and Facility Location Allocation Problem, which is composed of Vehicle Routing with Time Windows and Facility Location Allocation. The research in this paper will help to understand and solve the above problem as well.

In the future, more sophisticated operators such as the 3-opt and Lin–Kernighan (LK) heuristic Lin and Kernighan (1973) can be employed in an attempt to enhance the search capability of the algorithm. More importantly, measures that take the interdependence between the sub-problems into account to reflect the dependence of the objective value on the change of the decision variables are to be designed so that frameworks can be developed more systematically by identifying the best "direction" during the optimization process rather than heuristically.

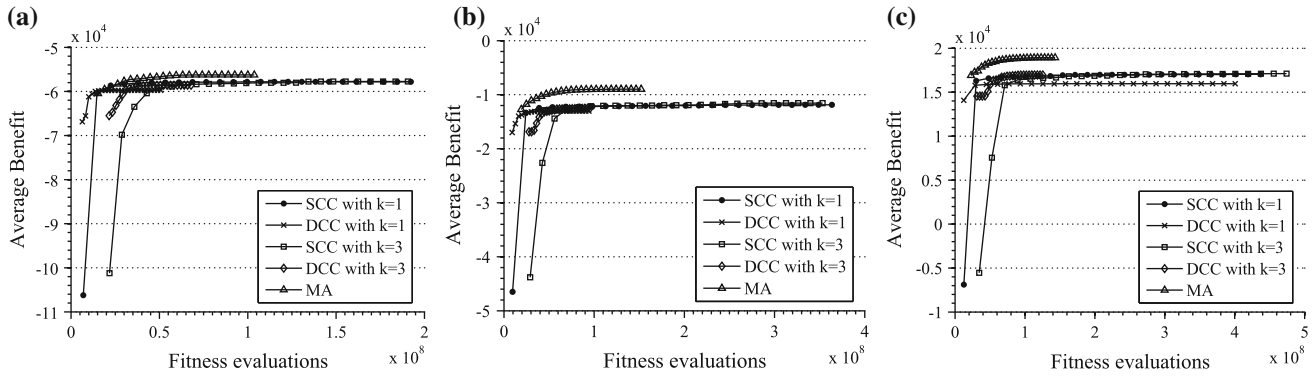

Fig. 6 Convergence curves of the compared algorithms on the TTP instances with $n = 50$ and $m = 75$

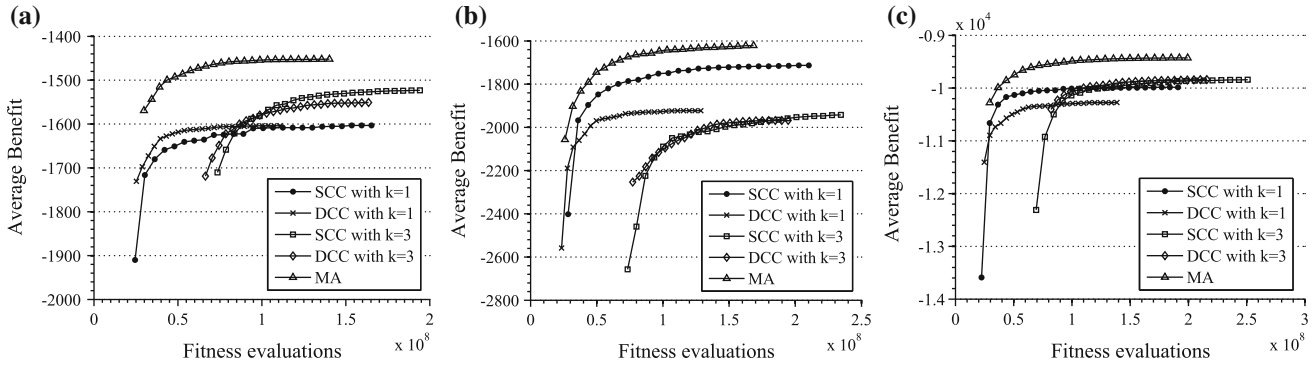

Fig. 7 Convergence curves of the compared algorithms on the TTP instances with $n = 100$ and $m = 10$

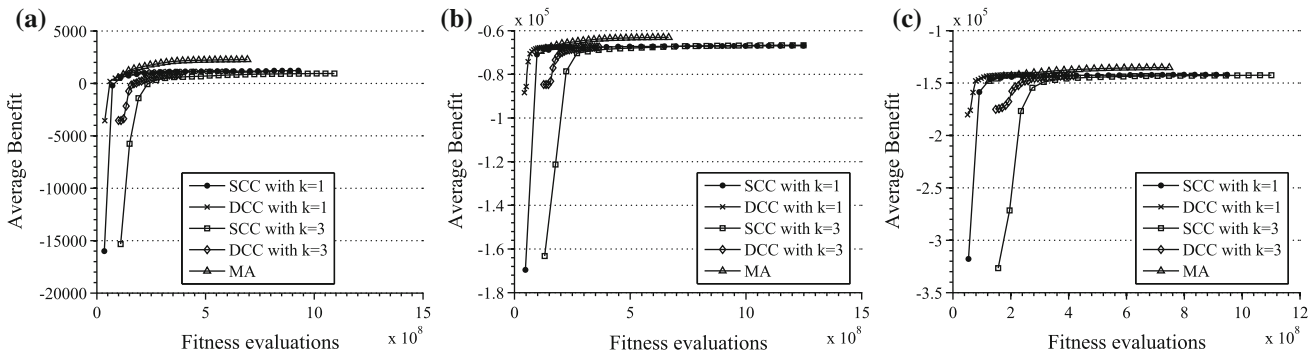

Fig. 8 Convergence curves of the compared algorithms on the TTP instances with $n = 100$ and $m = 100$

Acknowledgments This work was supported by an ARC Discovery Grant (No. DP120102205) and an EPSRC Grant (No. EP/I010297/1). Xin Yao is supported by a Royal Society Wolfson Research Merit Award.

References

Bolduc M, Laporte G, Renaud J, Boctor F (2010) A tabu search heuristic for the split delivery vehicle routing problem with production and demand calendars. Eur J Oper Res 202(1):122–130

Bonyadi M, Moghaddam M (2009) A bipartite genetic algorithm for multi-processor task scheduling. Int J Parallel Program 37(5):462–487

Bonyadi M, Michalewicz Z, Barone L (2013) The travelling thief problem: the first step in the transition from theoretical problems to realistic problems. In: Proceedings of the 2013 IEEE congress on evolutionary computation, Cancun, Mexico, pp 1037–1044

Boyd S, Xiao L, Mutapcic A, Mattingley J (2007) Notes on decomposition methods. Notes for EE364B, Stanford University

Bull L (2001) On coevolutionary genetic algorithms. Soft Comput 5(3):201–207

Croes G (1958) A method for solving traveling-salesman problems. Oper Res 6(6):791–812
De Giovanni L, Pezzella F (2010) An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. Eur J Oper Res 200(2):395–408
Dorigo M, Gambardella L (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66
Dror M (2000) Arc routing: theory, solutions and applications. Kluwer Academic Publishers, Boston
Fidanova S (2007) Ant colony optimization and multiple knapsack problem. In: Handbook of research on nature inspired computing for economics and management, pp 498–509
Fuellerer G, Doerner K, Hartl R, Iori M (2010) Metaheuristics for vehicle routing problems with three-dimensional loading constraints. Eur J Oper Res 201(3):751–759
Fu H, Mei Y, Tang K, Zhu Y (2010) Memetic algorithm with heuristic candidate list strategy for capacitated arc routing problem. In: Proceedings of the 2010 IEEE congress on evolutionary computation (CEC). IEEE, pp 1–8
Gupta K, Yao X (2002) Evolutionary approach for vehicle routing problem with time windows and facility location allocation problem. In: Bullinaria JA (ed) Proceedings of the 2002 UK workshop on computational intelligence (UKCI'02). Birmingham, UK
Hansen N (2006) The cma evolution strategy: a comparing review. In: Towards a new evolutionary computation. Springer, Berlin, pp 75–102
Horowitz E, Sahni S (1974) Computing partitions with applications to the knapsack problem. J ACM (JACM) 21(2):277–292
Ibrahimov M, Mohais A, Schellenberg S, Michalewicz Z (2012) Evolutionary approaches for supply chain optimisation: part i: single and two-component supply chains. Int J Intell Comput Cybern 5(4):444–472
Lin S, Kernighan B (1973) An effective heuristic algorithm for the traveling-salesman problem. Oper Res 21(2):498–516
Mei Y, Tang K, Yao X (2009a) A global repair operator for capacitated Arc routing problem. IEEE Trans Syst Man Cybern Part B Cybern 39(3):723–734
Mei Y, Tang K, Yao X (2009b) Improved memetic algorithm for capacitated arc routing problem. In: Proceedings of the 2009 IEEE congress on evolutionary computation, pp 1699–1706
Mei Y, Tang K, Yao X (2011a) Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem. IEEE Trans Evol Comput 15(2):151–165
Mei Y, Tang K, Yao X (2011b) A memetic algorithm for periodic capacitated Arc routing problem. IEEE Trans Syst Man Cybern Part B Cybern 41(6):1654–1667
Mei Y, Li X, Yao X (2014a) Cooperative co-evolution with route distance grouping for large-scale capacitated arc routing problems. IEEE Trans Evol Comput 18(3):435–449
Mei Y, Li X, Yao X (2014b) Variable neighborhood decomposition for large scale capacitated arc routing problem. In: Proceedings of the 2014 IEEE congress on evolutionary computation (CEC2014). IEEE, pp 1313–1320
Mei Y, Li X, Yao X (2013) Decomposing large-scale capacitated arc routing problems using a random route grouping method. In: Proceedings of 2013 IEEE congress on evolutionary computation (CEC). IEEE, pp 1013–1020
Mei Y, Tang K, Yao X (2010) Capacitated arc routing problem in uncertain environments. In: Proceedings of the 2010 IEEE congress on evolutionary computation, pp 1400–1407
Melo M, Nickel S, Saldanha-Da-Gama F (2009) Facility location and supply chain management—a review. Eur J Oper Res 196(2):401–412
Michalewicz Z (2012) Quo vadis, evolutionary computation? On a growing gap between theory and practice. In: Advances in computational intelligence. Lecture notes in computer science, vol. 7311. Springer, Berlin, pp 98–121
Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech concurrent computation program, C3P Report 826
Oliver I, Smith D, Holland J (1987) A study of permutation crossover operators on the traveling salesman problem. In: Grefenstette JJ (ed) Proceedings of the second international conference on genetic algorithms on genetic algorithms and their application. L. Erlbaum Associates Inc., pp 224–230
Omidvar M, Li X, Mei Y, Yao X (2014) Cooperative co-evolution with differential grouping for large scale optimization. IEEE Trans Evol Comput 18(3):378–393
Panait L, Luke S (2005) Time-dependent collaboration schemes for cooperative coevolutionary algorithms. In: AAAI fall symposium on coevolutionary and coadaptive systems
Papadimitriou C (1977) The euclidean travelling salesman problem is np-complete. Theor Comput Sci 4(3):237–244
Potter M (1997) The design and analysis of a computational model of cooperative coevolution. Ph.D. thesis, George Mason University
Potter M, De Jong K (1994) A cooperative coevolutionary approach to function optimization. In: Parallel problem solving from nature (PPSN), pp 249–257
Sbihi A (2010) A cooperative local search-based algorithm for the multiple-scenario max–min knapsack problem. Eur J Oper Res 202(2):339–346
Stadtler H (2005) Supply chain management and advanced planning—basics, overview and challenges. Eur J Oper Res 163(3):575–588
Stoen C (2006) Various collaborator selection pressures for cooperative coevolution for classification. In: International conference of artificial intelligence and digital communications, AIDC
Tang K, Mei Y, Yao X (2009) Memetic algorithm with extended neighborhood search for capacitated Arc routing problems. IEEE Trans Evol Comput 13(5):1151–1166
Thomas D, Griffin P (1996) Coordinated supply chain management. Eur J Oper Res 94(1):1–15
Wiegand R, Liles W, De Jong K (2001) An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In: Proceedings of the genetic and evolutionary computation conference (GECCO), pp 1235–1242
Wilcoxon F (1945) Individual comparisons by ranking methods. Biom Bull 1(6):80–83