Expert Systems With Applications 67 (2017) 148–162

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Effective three-phase evolutionary algorithm to handle the large-scale colorful traveling salesman problem

Hassan Ismkhan*

Department of Computer Engineering, Engineering Faculty, University of Bonab, Bonab, Iran

ARTICLE INFO

Article history:
Received 29 April 2016
Revised 12 September 2016
Accepted 12 September 2016
Available online 15 September 2016

Index Terms:
The colorful TSP
Priority based crossover
Enhanced local search
Large-scale instances
H-3Ph-EA

ABSTRACT

This paper proposes a novel evolutionary algorithm to handle the colorful traveling salesman problem. The proposed evolutionary algorithm is a three-phase algorithm and utilizes two types of a new crossover and an enhanced local search. In the first phase, quality of population is increased by applying the first crossover. After the first phase, a priority value is computed to each color, based on the best tour of the first phase. During the second phase, priority values are updated when a solution with better cost than existing solutions in population is found. Priority values are used by the second type of crossover, which is used in the second phase. After the second phase, the problem is converted to the classical traveling salesman problem ingeniously, and finally, the third phase is a hybrid genetic algorithm, for which an enhanced local search algorithm is applied. Applying the second type of crossover and updating priority values are continued in the third phase. The proposed algorithm is effective and finds new bounds for many large-scale instances and outperforms other state-of-the-art heuristic in terms of accuracy and speed. In many cases, gaps between results obtained by the proposed algorithm and the other methods are high. For some large-scale instances, proposed algorithm is 44 times faster than the other state-of-the-art competitor, which is the current most efficient algorithm.

1. INTRODUCTION

Many types of the traveling salesman problem (TSP) have been proposed. TSP with time limitation on nodes (Edelkamp, Gath, Cazenave, & Teytaud, 2013; López-Ibáñez & Blum, 2010; López-Ibáñez, Blum, Ohlmann, & Thomas, 2013), dynamic TSP (Mavrovouniotis & Yang, 2011), physical TSP (Perez, Rohlfshagen, & Cowling, 2013), double TSP with stacks (Petersen & Archetti, 2010), bottleneck TSP (Ahmed, 2013) and pickup-and-delivery variants of TSP (Mladenović, Urošević, Hanafi, & Ilić, 2012) are famous and have many applications.

The colorful TSP (CTSP) or minimum labeling Hamiltonian cycle problem (MLHCP) is another type of the TSP. In the CTSP, we are given a set of labels (colors) and a graph in which a label is assigned for each edge. The objective is to find a Hamilton cycle with minimum number of labels. For example, suppose the set of labels is {1, 2, 3}, on which a five-node labeled graph is defined, as shown in Fig. 1. In this example, the number of labels used for Hamilton cycle 1-2-3-4-5 is 1, so the cost of this tour is 1.

We can define CTSP in a formal way (Jozefowiez, 2011). For undirected and un-weighted graph G = (V, E), where V is the vertex set, n = |V|, and E is the edge set. Let (e) be a label (or color) associated with edge e, and let C be the set of all labels. We define binary variables x_e equal to 1 if and only if $e \in E$ is used, and binary variables u_k equal to 1 if and only if $k \in C$ is used. The aim of CTSP is to minimize:

$$\sum_{k \in C} u_k \tag{1}$$

$$\sum_{e \in \omega(\{i\})} x_e = 2 \tag{2}$$

$$\sum_{e \in \omega(S)} x_e \geq 2 \quad (S \subset V, 3 \leq |S| \leq n-3) \tag{3}$$

$$x_e \in u_{\delta(e)} \quad (e \in E) \tag{4}$$

$$x_e \in \{0,\ 1\} \quad (e \in E) \tag{5}$$

$$u_k \in \{0,\ 1\} \quad (k \in C) \tag{6}$$

where, $\omega(S) = \{e = (i, j) \in E |\ i \in S, j \in V \backslash S\}$. Objective (1) minimizes the number of labels. Constraints (2) and (3) are classical degree and connectivity constraints. Constraints (4) link the variables x_e and u_k by ensuring that if an edge labeled by k is used, then k is

* Fax: +98 412 7240800.
E-mail addresses: H.Ismkhan@bonabu.ac.ir, Esmkhan@gmail.com

http://dx.doi.org/10.1016/j.eswa.2016.09.022
0957-4174/

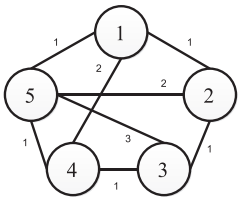


Fig. 1. A labeled graph.

counted in the objective function. Constraints (5) and (6) are integrality constraints.

The CTSP has applications in transportation (Xiong, Golden, & Wasil, 2007), planning a convoy trip, solving machine scheduling problem with sequential tasks (Silberholz et al., 2013) and minimizing the number of types of communication lines in telecommunication networks (Jozefowiez, 2011).

The CTSP is more difficult than Hamilton cycle problem (HCP)[1] which is NP-Complete (Garey & Johnson, 1979). It has been proved that the CTSP is NP-Hard (Cerulli, Dell'Olmo, Gentili, & Raiconi, 2006) which implies that runtime to obtain an exact solution for a large-dimension instance is not satisfactory, while in practice, we need to solve large-dimension instances. As we know, only few heuristics for the CTSP are available (Jozefowiez, 2011; Cerulli, Dell'Olmo, et al., 2006; Silberholz et al., 2013; Xiong et al., 2007). However, there are some papers about the similar topics like the spanning tree for a labeled graph (Broersma & Li, 1997; Cerulli, Fink, Gentili, & Voß, 2005, 2006; Chang & Leu, 1997; Krumke & Wirth, 1998; Wan, Chert, & Xu, 2002; Xiong, Golden, & Wasil, 2006; Xiong, Golden, & Wasil, 2005a,b), the minimum label path problem (Carr, Doddi, Konjevod, & Marathe, 2000), the minimum label Steiner tree problem discussed (Cerulli, Fink, et al., 2006), the labelled maximum matching problem (Carrabs, Cerulli, & Gentili, 2009), the labelled minimum path problem (Broersma, Li, Woeginger, & Zhang, 2005; Hassin, Monnot, & Segev, 2007) and the labelled max traveling salesman problem (Couëtoux, Gourvès, Monnot, & Telelis, 2008). There are some papers with similar titles (Li, Xing, Huaxuan, & MengChu, 2015; Li, Zhou, Sun, Dai, & Yu, 2014) which are different from CTSP considered in this paper. For instance, colored TSP in Li et al. (2014) is a type of TSP with muliple salseman, for which a color is assigned for each salesman,and it is also abbreviated by "CTSP", while for stated problem in our paper, for each edge a color is assigned.

The time complexity of branch-and-bound algorithm proposed in Jozefowiez (2011) is high and does not let apply it to large-scale instances. The heuristic solution in Xiong et al. (2007) loses its applicability when it is applied to uncompleted-graph. To solve this problem, some heuristics are proposed in Silberholz et al. (2013). Time complexity of these heuristics is better than branch-and-bound solution in Jozefowiez (2011), but these heuristics are still slow, and their runtimes, for instances with sizes of more than 500, are too high. In addition, time complexity of these heuristics increases for low edge-density graphs.

This paper proposes a three-phase genetic algorithm (GA). The proposed three-phase GA in this paper is different with other three phase algorithms in the literature like Jebari, moujahid, Bouroumi, and Ettouhami (2012), in which the problem is divided into some small sub-problems. For the proposed three-phase GA in this paper, a heuristic crossover is proposed for the first phase. Whereas for the second phase, another novel heuristic crossover is proposed which uses priority value of each color, and is based on the selection of colors with high priority.

[1] The aim of HCP is to determine whether a given graph contains a Hamilton cycle.

While many meta-heuristics have been developed to solve classic TSP (Adham & Bentley, 2014; Dorigo & Gambardella, 1997; Ismkhan & Zamanifar, 2010; Jolai & Ghanbari, 2010; Karaboga & Gorkemli, 2011; Nagata & Kobayashi, 2013; Ouaarab, Ahiod, & Yang, 2014), they have never been used in the CTSP solver heuristics. Among these solutions, types of Lin-Kernighan (LK) algorithm (Lin & Kernighan, 1973) have been successfully applied to the large-scale TSP (Applegate, Cook, & Rohe, 2003; Helsgaun, 2000, 2009). Therefore, the TSP solver heuristics are tempting to be used in the CTSP solver heuristic, yet it seems to be impossible.

This paper proposes an elegant algorithm which transforms the CTSP instance to the TSP instance. This algorithm is applied after the second phase of the proposed GA, then in the third phase, the LK is applied to the obtained TSP instance. The type of transformation proposed in this paper is different with other works in the literature like Diaz-Delgadillo, Montiel, and Sepúlved (2016) and Montiel, Diaz-Delgadillo, and Sepúlveda (2013), in which the size of problem is reduced. In the proposed transformation, the CTSP instance is converted to TSP instance, in order to utilize advantages of powerful TSP solvers.

With these descriptions, Section 2 reviews the CTSP solver heuristics, Section 3 reviews TSP solver heuristics and algorithm of the LK in particular. Section 4 deals with the proposed algorithm. Section 5 explains the experiments, and Section 6 summarizes the conclusions of the paper

2. Previous studies for the CTSP

Except Jozefowiez (2011), which proposes a branch-and-cut algorithm for the CTSP, almost all of the other references in this paper have been based on the path extension. In these methods, in each step, a new node is inserted to the path such that the number of the used colors does not increase. The reference Cerulli, Dell'Olmo, et al. (2006) proposes a type of path extension algorithm in which, to avoid local trap, a tabu-list is used to memorize the previous moves which leads to feasible solution.

Paper Xiong et al. (2007) proposes an effective heuristic which is also based on a type of path extension method in which four different cases are considered to insert next node into the existing path. To determine initial set of colors, this paper uses a type of GA proposed in Xiong et al. (2005).

GA in Xiong et al. (2005) is proposed to solve the minimum labelling spanning tree (MLST). The crossover of this GA applies union of color sets of both parents, sorts obtained set in decreasing order of the frequency of labels, and then adds edges and their corresponding labels to the set of child until a feasible solution is created. This algorithm beats another famous method named the maximum vertex covering algorithm (MVCA). Paper Xiong et al. (2006) improves both methods with minor changes.

Paper Silberholz et al. (2013) is a comparative review paper. Furthermore, in some cases, it proposes some new hybrid methods by combining existing methods with small changes. The proposed deconstruction and repair genetic algorithm (DRGA) outperforms the other heuristics in terms of accuracy.

3. TSP solver heuristics

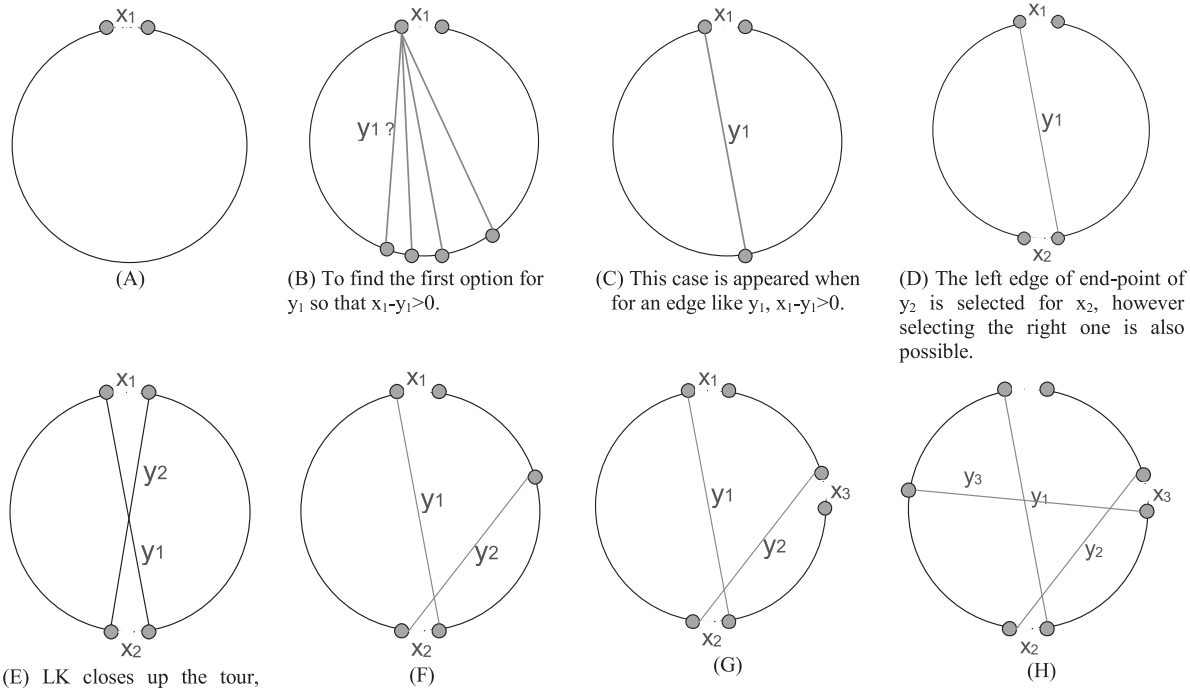
Many meta-heuristics have been proposed to solve the TSP. These meta-heuristics include the ant colony optimization (ACO) (Dorigo & Gambardella, 1997), the neural networks (NN) (Jolai & Ghanbari, 2010), the artificial bee colony (ABC) (Karaboga & Gorkemli, 2011) and the hybrid algorithms (Ismkhan & Zamanifar, 2010). These algorithms have a wide range of applications. The type of GA proposed in Nagata and Kobayashi (2013) solves the high-dimensional TSP instances.

The Start of LK
Input: tour
do
//The start of the step (each step is started after **do**)
For each edge as x_1 in the tour
//The start of the sub-step
Level-1:
Remove edge x_1 from the tour; //see Fig. 3 (A)
Find and add edge y_1 such that $|x_1|-|y_1| > 0$; //Fig. 3 (C)
If finding edge y_1 is impossible **goto For**;
Level-2:
Remove edge x_2 from tour; //see Fig. 3 (D)
Close up the tour and **goto For**, if it leads to reduce the cost;//Fig. 3 (E)
Find and add edge y_2 such that $|x_1|-|y_1|+|x_2|-|y_2| > 0$; //Fig. 3 (F)
If finding edge y_2 is impossible **goto** the Level-1 and check another edge y_1;
Level-3:
Remove edge x_3 from tour; //see Fig. 3 (G)
Close up the tour and **goto For**, if it leads to reduce the cost;
Find and add edge y_3 such that $|x_1|-|y_1|+|x_2|-|y_2|+|x_3|-|y_3| > 0$; //Fig. 3 (H)
If finding edge y_3 is impossible **goto** the Level-2 and check another edge y_2;
…
Level-K:
Remove edge x_k from tour;

/*In general, we have two choices for removing. For instance, in Fig. 3 (D), the left edge of end-point of y2 is selected for x2, however selecting the right one is also possible.*/

Close up the tour and **goto For**, if it leads to reduce the cost;
Find and add edge y_k such that $\sum_{i=1}^{K}|x_i| - \sum_{i=1}^{K}|y_i| > 0$;
If finding edge y_k is impossible **goto** the Level-(K-1) and check another edge y_{k-1};
End For
While the cost of the tour has reduced in current step

Fig. 2. The LK algorithm.



(A)
(B) To find the first option for y_1 so that x_1-y_1>0.
(C) This case is appeared when for an edge like y_1, x_1-y_1>0.
(D) The left edge of end-point of y_2 is selected for x_2, however selecting the right one is also possible.
(E) LK closes up the tour, when it leads to have a tour with a better cost.
(F)
(G)
(H)

Fig. 3. Some possible moves for LK in each sub-step.

Among meta-heuristics, the LK algorithm (Lin & Kernighan, 1973) is effective. Its variants can solve the TSP instances with thousands of nodes (Applegate et al., 2003; Helsgaun, 2009)

Definition 1. A tour is in k-optimal form, if it is not possible to decrease its cost by exchanging at most K edges.

Therefore, according to this definition, if for a tour T, it is possible to reduce its cost by removing k edges and adding another k edges, then tour T is not in k-optimal form.

In fact, the LK tries to convert an input tour to its possible k-optimal form. Briefly, the LK operates by applying the three words, "break", "link" and "consider some conditions" (Ismkhan & Zamanifar, 2014). Fig. 2 shows the LK algorithm. In each sub-step, the LK tries to reduce the cost of tour by some moves that are formed by deleting some edges from the tour and adding other possible edges to it. Fig. 3 shows some initial possible moves of the LK, which can be occurred in each sub-step. For instance, Fig. 3(B) indicates a case that the LK removes edge x1 and tries to find the first possible y1 so that $x1-y1 > 0$. After adding y1, as shown in Fig. 3(C),

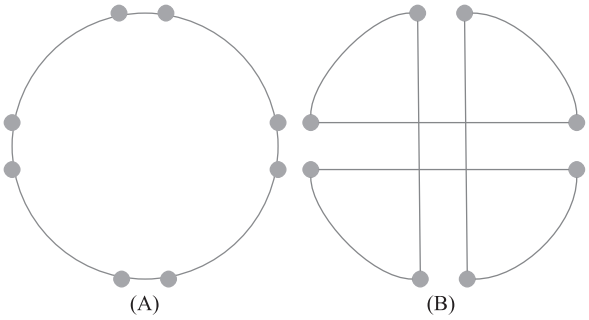


Fig. 4. Double-Bridge operator: This operator removes 4 random edges and then, reconnects remaining sub-paths as (B).

the LK has two choices for removing. Fig. 3(D) shows a case, where the left edge of the end-point of y2 is selected for x2. In Fig. 3(D), there is a path which covers the all nodes of the graph. In this case, like Fig. 3(E), the LK can close up the obtained path by inserting an edge between the two end-points of this path, only when it can lead to have a feasible tour with the lower cost than the initial cost. If closing up the tour is impossible, then the LK should start another level. Typically, in the levelK, after removing xK, if closing up the tour does not lead to obtain a feasible tour with the lower cost than original one, then the LK decides to return back to levelK – 1 and tests another possible edge for yK-1 (backtracking).

Above-mentioned LK considers sequential moves. For sequential moves, xi and yi share an end-point as well as yi and xi+1, where a non-sequential move, as shown in Fig. 4, does not satisfy these conditions. Considering non-sequential moves makes the LK too complicated. The original LK only probes a limited case of non-sequential moves (Lin & Kernighan, 1973). The proposed approach in Martin, Otto, and Felten (1991), to cover non-sequential moves, uses Double-Bridge operation (Johnson, 1990; Martin et al., 1991) which is shown in Fig. 4.

The proposed iterated variant of the LK in Helsgaun (2000), which is named LKH, limits the depth of search to five levels. To speed up the LK, a candidate-set is assigned to each node. Usually, a constant number of nearest-neighbors of each node are considered for its candidate-set. This causes the LK to focus on candidate-set in backtracking, whenever it wants to select an edge to add. Moreover, in $step_i$, only the edges which have been added in $step_{i-1}$ are considered as x_1.

Theorem 1. *The time complexity of the LK is O ($N \times steps \times |NN|^K + K \times |E|$), where N is the dimension of instance, steps is the number of repetitions of the do-while, |NN| is the size of candidate-set of each node, K is the depth of search, and |E| is the number of edges of the graph.*

Proof. The term $K \times |E|$ is the time complexity of computing the K number of the nearest neighbor for each node. In each step, we have the N number of edges which have to be considered in rule of x_1 in each sub-step. Each sub-step itself has K levels to search, where, in each level the number of branches is |NN|, so the time complexity of each sub-step is O ($|NN|^K$). In each step, each sub-step is applied to all edges, so the overall time complexity is O ($N \times steps \times |NN|^K$). □

4. Proposals

To utilize advantages of powerful TSP solver heuristic for CTSP, we have to assign a cost to each edge of CTSP instance. The proposed EA assigns a priority value to each color, then defines cost of each edge based on priority of its corresponding color, then utilizes the advantages of powerful TSP solver heuristic for CTSP. The main skeleton of proposed EA is shown in Fig. 5. This EA is formed by the three main phases. The proposed EA is referred as Hybrid Three-Phase EA (H-3Ph-EA) because all three phases are steady-state GA, and the third phase uses a type of LK which is a local search, so the third phase is a memetic algorithm. The first phase of this algorithm uses a heuristic crossover which is presented in the next sub-section. The aim of the first phase is to increase quality of initial solutions of population which are created randomly. The first phase approaches this goal using the first proposed heuristic crossover. In the second phase, a priority value is assigned for each color, and during the second phase (and also in the third phase) these values are updated, when a better solution is found. Proposed methods to compute and updating priority of colors are novel. Both second and third phases use the second type of heuristic crossover which is based on priority of colors. After the second phase, the CTSP instance is transformed to the TSP instance. In the third phase, an enhanced LK is applied to the TSP instance in iterations of GA. Proposed EA uses a type of the LK based on implementations in Ismkhan and Zamanifar (2014) with constant number of iterations.

The pseudo-code in Fig. 6 shows a form of steady-state GA which is used in each phase of H-3Ph-EA. For this type of GA, in each generation, at most one new solution is created. Using conditions in lines 2–4, H-3Ph-EA determines the active phase. In line 5, it decides to select one of crossover operator or mutation operator. If the generated random number, which is between 0 and 1, is smaller than crossover rate (XR) then crossover is selected to be operated and lines 8–14 are performed, otherwise, lines 16–25 are performed to mutate some solutions of population. It should be noted that for line 8, the first type of crossover is applied when first phase is active, and the second type of crossover is applied when the second or third phase are active. For line 11, LK is applied, when the third phase is active.

4.1. The first phase

The main goal of this phase is to increase quality of initial solutions in population. At the start of h-3Ph-EA, population is initialized by random-generated solutions, which have not enough quality. To approach this goal, we propose a crossover which can be applied to input instance of CTSP. In this phase a steady-state version

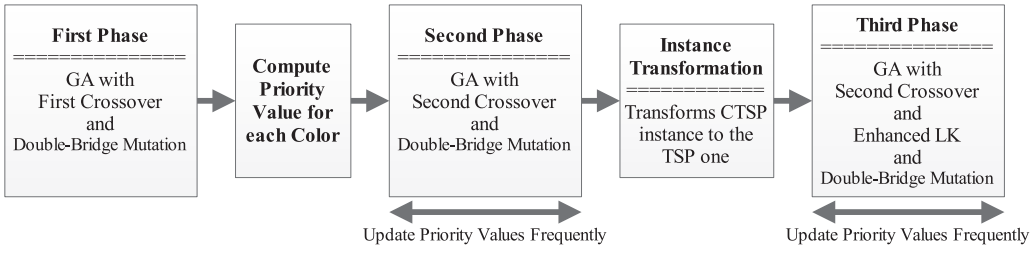


Fig. 5. Hybrid three-phase EA.

```
01) For I = 1 to Gen-Size
02)    If I ≤ Gen-Size1 Then it is the first phase;
03)    If I > Gen-Size1 and I ≤ Gen-Size2 Then it is the second phase;
04)    If I > Gen-Size2 Then it is the third phase;
05)    If Random (0, 1) < XR
06)       // XR is crossover rate which is a parameter
07)       // Random (0, 1) generates a random number between 0 and 1.
08)       Select parents using the proposed Linear-Selection in [40];
09)       Operate crossover and generate a new-solution;
10)       Apply LK to the new-solution, if type of instance is TSP;
11)       //LK is operated in the third phase only.
12)       If fitness (new-solution) > fitness (worst-parent)
13)          Replace the worst-parent with the new-solution;
14)       End If
15)    Else
16)       Mutation-Size = 5;
17)       If I > Gen-Size / 2 Then Mutation-Size = 2;
18)          For J = 1 to Mutation-Size
19)             Index = Linear-Selection;
20)             If Index < population-size / 2 and I > Gen-Size / 2 Then
21)                Index = Index + population-size / 2;
22)                /*after some generations (Gen-Size / 2), the mutation is
23)                only applied to the worst solutions of population*/
24)                Operate Double-Bridge mutation on population [Index];
25)          End For
26)    End If
27)    Sort population according to fitness, in descending order;
28) End For
```

Fig. 6. Algorithm of GA which is used in all phases.

of GA is applied, which is shown in Fig. 6. It should be noted that it is impossible to use the advantages of TSP-solver LK in this phase, because each edge of input-graph has a label, while LK can be applied on weighted graph in which a cost (a positive real number) is assigned to each edge. To escape from premature convergence and maintaining the diversity of population the Double-Bridge operator is applied to mutate population in this phase.

4.1.1. Heuristic crossover for the first phase

Two parents are needed for this crossover. As shown in Fig. 6, these parents are selected by the Linear-Selection (Whitley, 1989). In Linear-Selection, index of selected population is determined by:

$$index = \frac{|population| \times \frac{bias - \sqrt{bias - 4\rho(bias-1)}}{2}}{bias - 1}, \tag{7}$$

where bias is a parameter and usually set 1.25 or 1.5 and ρ is a random number between 0 and 1. This crossover makes an auxiliary double-linked list for each parent at first. It selects first node randomly. In each step, selected node is copied into the child tour and its corresponding nodes in both auxiliary double-linked lists are deactivated. This means that pointers from other nodes to this node are deactivated. This process is applied for each node which is added to the child tour. In each step, to select the next node, active pointed nodes by recently added node (say V) are considered. If there is a node (say N) which does not need a new color (color of edge NV has been added before), then N is selected. When the produced tour is not feasible, then its cost is set to be $+\infty$. Fig. 7 shows the algorithm of this crossover in details.

Example 1. Consider graph in Fig. 1. Example in Fig. 8 shows how heuristic crossover of the first phase produces child according to 5-3-4-1-2 and 1-5-2-3-4 which are parents' tours.

Theorem 2. *The time complexity of the first crossover is O (N), where N is the number of nodes of the graph.*

Proof. The number of steps is N where in each step some constant number of operations are applied. By using an array of pointers to hold the nodes of each double-linked list the time complexity of access to each node becomes O (1) then in overall, the time complexity becomes O(N) × O (1)=O (N). □

4.2. Priority computation of colors

After the first phase, an algorithm to calculate priority value for each label (color) is applied. This algorithm is based on the best tour found in the first phase.

$$priority[L] = \begin{cases} \frac{1}{C_{best}}, & L \in Colors_{best} \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

In this equation, *priority [L]* points to the priority value of color L, c_{best} is the cost of the best tour obtained in the first phase of algorithm and $Colors_{best}$ is the set of colors which are used in the best tour.

4.3. The second phase

Increasing quality of population is the aim of all phases of H-3Ph-EA, however, the second phase tries to do this by utilizing another techniques different with the first phase. This phase uses the second type of the proposed crossover which is similar to the first one. However, despite first crossover which never uses additional heuristic information, the second crossover gives priority to edges with frequent colors used in tours with the highest quality. So, this crossover uses priority values which are computed at the start of this phase. In addition, priority values are updated during this phase, when a new better solution is found. Therefore, the rest of this sub-section explains how this phase updates priority values, and how the second crossover does work.

```
The Start of Crossover Function
    Input: parent-1 and parent-2
    Output: child (produced tour)
    DLL-1←Make an auxiliary double-linked list for parent-1;
    DLL-2←Make an auxiliary double-linked list for parent-2;
    Dim ← the size of instance;
    Node X ← Random node;
    Add X to child;
    In DLL-1, disable two possible pointers which point to X;
    In DLL-2, disable two possible pointers which point to X;
    Mark X as inactive node;
    Colors ← {};
    For I = 2 to Dim
        Node V ← X;
        S ← {};
        S ← S ∪ set of active nodes which are pointed by V from DLL-1;
        S ← S ∪ set of active nodes which are pointed by V from DLL-2;
        If for a node N∈S, corresponding label of edge VN∈ Colors
            X ← N;
        Else
            X ← randomly select from S;
            //This crossover may produce infeasible tour
            //When produced tour is infeasible then its cost becomes +∞
            If VX is in set of edges of the graph
                Add corresponding label (color) of edge VX to Colors;
            End If
        End If
        Add X to the end of child path;
        In DLL-1, disable two possible pointers which point to X;
        //these pointers are from its active neighbors.
        Change DLL-1 such that active neighbors of X point to each other;
        In DLL-2, disable two possible pointers which point to X;
        //these pointers are from its active neighbors.
        Change DLL-1 such that active neighbors of X point to each other;
        Mark X as inactive node;
    End For
The End of Crossover
```

Fig. 7. The algorithm of the crossover of the first phase.

4.3.1. A method for updating priority

Priority method is applied after the first phase. During the run of the second and third phases, when a new best tour is found, then the priority values of colors are updated via the following equation.

$$priority[L] = \begin{cases} priority[L] + \frac{1}{C_{best}}, & L \in Colors_{best} \\ priority[L], & \text{otherwise} \end{cases} \quad (9)$$

4.3.2. Heuristic crossover for the second phase

This crossover is similar to the crossover of the first phase. The main difference between two crossovers is in the selection of the next node. In the second crossover, the active nodes pointed by the recently selected node are considered, and among these edges, corresponding edge of the color with the highest priority is selected. If two or more colors have a same priority, then a node which does not cause a new label is selected.

Example 2. Suppose, as shown in Fig. 9, node 1 has been selected and added to the child tour in recent step and it has been deactivated in both auxiliary double-linked lists. Now node 1 points nodes 4, 2 and 5 in these lists, so colors of edges 1–2, 1–4 and 1–5 are considered, and the corresponding edge of the color with the highest priority is selected.

Theorem 3. *The time complexity of the first crossover is O (N), where N is the number of nodes of the graph.*

Proof. This crossover is similar to the first crossover so, see Theorem 2. □

4.4. The Transformation of CTSP to the TSP

In order to utilize the advantages of the LK in solving the CTSP instance, there may be two possible choices. The first one is to change and enhance the LK. This way seems to be impossible, because the LK uses the cost of edges in each sub-step, while for the CTSP instances there is no defined cost for edges. The second way, which is proposed in this algorithm, is to transform the CTSP instance to the TSP one. This way is not impossible. Specially, with proposals in this paper it seems to be easy where for each color the priority value is defined.

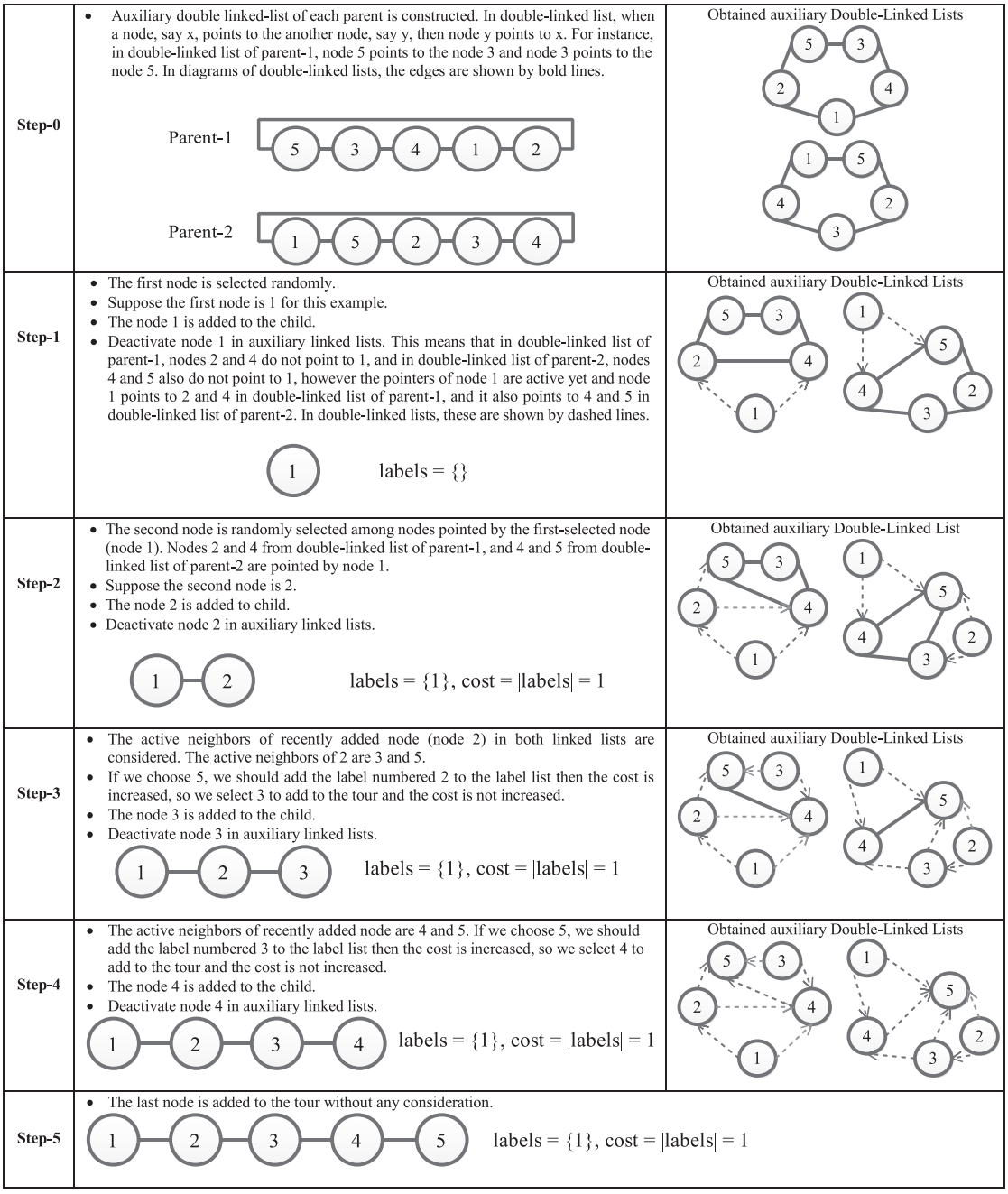


Fig. 8. An example for heuristic crossover operator of the first phase.

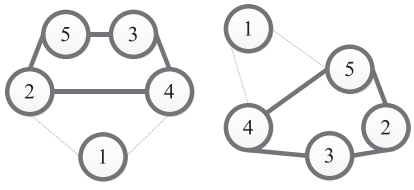


Fig. 9. An example for the second crossover.

With these descriptions, we define cost value for each edge such as XY in the CTSP instance with the following equation.

$$cost(X,\ Y) = \begin{cases} \frac{1}{priority[L(X,Y)]}, & L(X,\ Y) \in Colors_{best} \\ +\infty, & priority[L(X,\ Y)] = 0 \text{ or } XY \notin E \end{cases} \quad (10)$$

In this equation, L(X, Y) is the corresponding color of the edge XY and E is the set of all edges. In this definition, the cost of XY becomes infinite, when XY does not belong to E or priority of its corresponding edge is 0.

4.5. The third phase

After that a cost is assigned to each edge, then H-3Ph-EA can utilize advantages of the LK TSP-solver. In this phase, like the second phase, the second type of crossover is used, priority values are updated, and then the cost of each edge is also updated. To preserve speed of the H-3Ph-EA in this phase, in each iteration, a light version of LK is applied. Line 10 of algorithm of Fig. 6 shows that LK is applied to the new-solution which is generated by the second type of crossover in the line 9.

4.6. The time-complexity of H-3Ph-EA

In our implementations, in order to hold linear time complexity, we limit the number of steps and number of the nearest-neighbors (|NN|) to a constant number. The depth of search in our implementation is 5, because we use an enhanced variant of the LKH, and it considers five levels of search.

Theorem 4. *The time complexity of the H-3Ph-EA is O (Gen-Size* $\times N + r \times K \times |E|$*), where Gen-Size is the sum of generation-sizes of all three phases, N is the number of nodes of the graph, |E| is the number of edges, and r indicates how many times the candidate sets are updated during the algorithm.*

Proof. Except the third phase which uses the LK algorithm (and the second type of crossover), the first and second phases use crossovers with time complexity of O (N) (see Theorems 2 and 3). As above-mentioned descriptions, the LK algorithm which is used in the third phase is limited to the constant number of levels of search, the constant number of the nearest-neighbors and the constant number of steps. So, the time complexity of H-3Ph-EA becomes O (Gen-Size $\times$ N). However, we should add the term $r \times K \times |E|$ to the time complexity, because during the algorithm, the candidate-sets are updated r times. Therefore, in overall, the time complexity becomes O (Gen-Size $\times N + r \times K \times |E|$). □

The *r* is not a parameter; candidate-sets computation is applied, when a new tour with better cost is found, and priority values are updated.

5. Empirical Results

5.1. Accuracy

To validate performance of H-3Ph-EA, the proposed DRGA (Silberholz et al., 2013) is selected for the competition. According to experiments in Silberholz et al. (2013), the results show that DRGA outperforms the other state-of-the-arts in terms of accuracy and speed.

Experiments were run on Intel® Core™ i3-4150 CPU @ 3.50 GHz with 4 GB of random access memory. The operating system was 64-bit windows 7. Programs were written by C++ codes. These codes were compiled with MSVC++ 2012. The source codes of DRGA have been provided by the first author of Silberholz et al. (2013). Original DRGA was recompiled again on MSVC++ environment. The parameter setting for H-3Ph-GA is as following:

For GA:

- Generation-Size = 1000,000
- Gen-Size$_1$ (See Fig. 6) = Generation-Size/10
- Gen-Size$_2$ (See Fig. 6) = Generation-Size/5
- Population-Size = 10
- XR (crossover rate) = 0.85
- In implementation of both crossover operators, the order of active neighbors of recently added node P is as: 1) left node of P in the first double-linked list, 2) right node of P in the first double-linked list, 3) left node of P in the second double-linked list, 4) right node of P in the second double-linked list. It is obvious that, in this mentioned order, when a node is not active, then it is not considered. For the both types of proposed crossovers, if there are more than one node which have the same conditions and can be added to the child, then the mentioned order is considered.

For the LK:

- Implementation of the LK is based on Ismkhan and Zamanifar (2014). Paper Ismkhan and Zamanifar (2014) provides a TSP solver package of some recent meta-heuristics. This package includes the LK which is based on the LKH implementation.
- |K| (search depth) = 5
- |NN| (number of nearest-neighbors for each node) = 5
- steps = 20

For Linear-Selection:

- bias = 1.25

In these experiments, both DRGA and H-3Ph-EA solved instances with dimensions of 280, 532 and 1000. Instances with the same dimension differ from each other by their types and density of existing edges. The density of each instance is a number between 0 and 1 which 1 indicates that the instance is complete graph with maximum number of edges. The types are LC, LR, RC and RR (Chen et al., 2008; Silberholz et al., 2013). These types differ from each other in how edges are selected and how a label is assigned to each edge. There are two ways to select edges: length-based frequency distribution and random frequency distribution. After deciding which edges are in graph, there are two way to assign a color to each edge: random label selection and clustered label selection. Each type shows how the instance has been created, and each latter in the name of type points to the corresponding method of how edges have been created and how colors have been assigned to edges. For example, the LC indicate that instance is created using length-based frequency distribution for selection of edges and clustered label selection for assigning colors to edges. These instances are available at http://josilber.scripts.mit.edu/.

Table 1 shows the results obtained from 30 runs by each algorithm for instances with dimension of 280. Table 2 shows the results obtained from 30 runs by each algorithm for instances with dimension of 532. Table 3 shows the results obtained from 30 runs by each algorithm for instances with dimension of 1000. In these tables, the best, the worst, average, and standard deviations are shown for both algorithms. In these tables, column WSRT shows Wilcoxon signed ranks test (WSRT) at confidence level of 5% (if the P-value is less than 0.05, then the difference is significant) between H-3P-EA and DRGA. The sign '+' shows H-3Ph-EA is significantly better than DRGA, the sign '−' shows DRGA is significantly better than H-3Ph-EA, and '∼' shows there is no significant difference between H-3Ph-EA and DRGA.

Table 3 shows the difference between DRGA and H-3Ph-EA is significant for dimension of 1000. In the many cases, the worst cost of H-3Ph-EA is better than the best cost of DRGA. For all instances, by considering average cost, it can be easily concluded that H-3Ph-EA outperforms DRGA with significant gap. For verification purpose, the best tour for Inst-1000-1, obtained by H-3Ph-EA is shown in Appendix I. With these results, the results of runtimes are more surprising where for some instances with dimension of 1000, the average runtimes of H-3Ph-EA are 9 times faster than DRGA. For Inst-532-13, H-3Ph-EA is 44 times faster than DRGA.

5.2. Emperical view of the time complexities

Based on Theorem 4, the time complexity of H-3Ph-EA is O (Gen-Size $\times$ N $\times$ r $\times$ K $\times$ |E|). The time complexity of DRGA is O (P2 $\times$ M $\times$ N4) where M is the number of labels, and P is the number of generations (Silberholz et al., 2013). It is obvious that time complexity of H-3Ph-EA is better than DRGA. The obtained runtimes of experiments also certify this fact. Tables 4–6 show results of runtimes of each algorithm for dimension of 280, 532 and

Table 1
Results obtained from 30 runs for the instances with dimension of 280.

		Number	Density		DRGA				H-3Ph-EA				
	Dimension	of Colors	of Edges	Type	Worst	Average	STDV	Best	Worst	Average	STDV	Best	WSRT
Inst-280-1	280	784	0.2	LC	70	67.9	1.18	65	71	**66.07**	2.27	62	+
Inst-280-2	280	784	0.2	LR	32	30.87	0.82	28	33	**29.93**	1.82	27	+
Inst-280-3	280	784	0.2	RC	69	66.9	1.32	64	66	**62.43**	1.76	58	+
Inst-280-4	280	784	0.2	RR	30	27.83	0.87	26	30	**27.23**	1.33	25	+
Inst-280-5	280	784	0.5	LC	43	41.67	0.99	40	43	**39.87**	1.72	36	+
Inst-280-6	280	784	0.5	LR	16	**15.23**	0.5	14	19	16.93	1.26	14	−
Inst-280-7	280	784	0.5	RC	43	40.37	1.22	38	42	**39.03**	1.47	35	+
Inst-280-8	280	784	0.5	RR	15	**14**	0.59	13	17	15	1.05	13	−
Inst-280-9	280	784	0.8	LC	35	33.07	0.74	32	35	**31.67**	1.79	28	+
Inst-280-10	280	784	0.8	LR	11	**9.9**	0.4	9	14	11.83	1.05	10	−
Inst-280-11	280	784	0.8	RC	33	31.83	0.65	31	35	**31.13**	1.68	28	+
Inst-280-12	280	784	0.8	RR	10	**9.57**	0.5	9	13	11.17	1.21	8	−
Inst-280-13	280	784	1.0	LC	31	28.93	0.78	28	31	**27.1**	1.54	25	+
Inst-280-14	280	784	1.0	LR	9	**8.03**	0.32	7	11	9.6	0.86	8	−
Inst-280-15	280	784	1.0	RC	31	28.93	0.78	27	30	**27.27**	1.46	24	+
Inst-280-16	280	784	1.0	RR	9	**7.83**	0.46	7	12	9.77	0.94	8	−

Table 2
Results obtained from 30 runs for the instances with dimension of 532.

		Number	Density		DRGA				H-3Ph-EA				
	Dimension	of Colors	of Edges	Type	Worst	Average	STDV	Best	Worst	Average	STDV	Best	WSRT
Inst-532-1	532	2830	0.2	LC	165	161.87	2.15	157	170	**157.77**	6.58	146	+
Inst-532-2	532	2830	0.2	LR	75	72.33	1.12	70	75	**68.43**	2.8	62	+
Inst-532-3	532	2830	0.2	RC	162	154.67	2.9	150	161	**150.33**	5	140	+
Inst-532-4	532	2830	0.2	RR	80	78.1	1.3	76	75	**70.13**	2.45	66	+
Inst-532-5	532	2830	0.5	LC	116	110.83	2.1	107	107	**100.37**	2.71	97	+
Inst-532-6	532	2830	0.5	LR	43	**40.2**	1.03	38	45	40.77	2.19	35	~
Inst-532-7	532	2830	0.5	RC	116	112.07	1.98	108	120	**105.53**	3.75	99	+
Inst-532-8	532	2830	0.5	RR	45	42.6	0.97	41	43	**39.8**	1.47	37	+
Inst-532-9	532	2830	0.8	LC	99	95.53	2	92	97	**89.77**	3.26	83	+
Inst-532-10	532	2830	0.8	LR	30	**29.3**	0.65	28	35	30.33	1.9	27	−
Inst-532-11	532	2830	0.8	RC	101	96.27	2.41	91	99	**93.1**	3.73	86	+
Inst-532-12	532	2830	0.8	RR	31	29.93	0.91	28	32	**29.47**	1.41	26	~
Inst-532-13	532	2830	1.0	LC	101	95.5	2.67	90	98	**91.23**	3.6	83	+
Inst-532-14	532	2830	1.0	LR	26	**25.13**	0.68	24	29	26.43	1.41	24	−
Inst-532-15	532	2830	1.0	RC	100	95.27	2.07	91	107	**91.23**	3.99	86	+
Inst-532-16	532	2830	1.0	RR	26	**25**	0.53	24	28	25.63	1.3	23	−

Table 3
Results obtained from 30 runs for the instances with dimension of 1000.

		Number	Density		DRGA				H-3Ph-EA				
	Dimension	of Colors	of Edges	Type	Worst	Average	STDV	Best	Worst	Average	STDV	Best	WSRT
Inst-1000-1	1000	10,000	0.2	LC	275	268.97	3.67	262	256	**240.83**	6.27	227	+
Inst-1000-2	1000	10,000	0.2	LR	186	181.7	2.85	174	158	**149.67**	4.25	140	+
Inst-1000-3	1000	10,000	0.2	RC	265	258.97	3.03	252	252	**227.5**	7.53	216	+
Inst-1000-4	1000	10,000	0.2	RR	185	179.87	2.36	175	159	**147.43**	3.95	142	+
Inst-1000-5	1000	10,000	0.5	LC	152	148.2	1.81	145	140	**132.53**	3.05	126	+
Inst-1000-6	1000	10,000	0.5	LR	105	102.4	1.43	99	92	**86.8**	2.59	82	+
Inst-1000-7	1000	10,000	0.5	RC	145	141.8	1.99	137	136	**126.03**	3.26	120	+
Inst-1000-8	1000	10,000	0.5	RR	103	99.93	1.39	98	90	**84.93**	2.64	79	+
Inst-1000-9	1000	10,000	0.8	LC	109	106.7	1.62	103	103	**96.6**	2.92	92	+
Inst-1000-10	1000	10,000	0.8	LR	73	71.17	1.26	68	67	**63.33**	1.86	60	+
Inst-1000-11	1000	10,000	0.8	RC	107	103.77	1.61	100	102	**94.5**	3.72	89	+
Inst-1000-12	1000	10,000	0.8	RR	72	70.4	1.28	67	68	**62.33**	2.34	58	+
Inst-1000-13	1000	10,000	1.0	LC	93	90	1.66	87	89	**82.27**	2.1	79	+
Inst-1000-14	1000	10,000	1.0	LR	61	58.8	0.96	57	60	**55.87**	1.91	53	+
Inst-1000-15	1000	10,000	1.0	RC	92	89.9	1.09	88	88	**82.23**	2.39	78	+
Inst-1000-16	1000	10,000	1.0	RR	61	58.97	1.00	57	60	**55.43**	2.18	50	+

1000, respectively. The second and third columns of these tables point to the average runtimes of DRGA and H-3Ph-EA, respectively, and the fourth column shows how many times H-3Ph-EA is faster than DRGA. Except six benchmarks with dimension of 280, for all other benschmarks, Tables 4–6 show that H-3Ph-EA is faster than DRGA. For Inst-532-13, H-3Ph-EA is 44 times faster than DRGA.

When the dimension is increased, the speed of H-3Ph-EA becomes unreachable in comparison to DRGA. Fig. 10 may give us a better view of how runtimes of DRGA and H-3Ph-EA increase when the dimension is increased. Fig. 10 includes diagrams of

Table 4
Runtime results of 30 runs.

	Average runtime of DRGA	Average runtime of H-3Ph-EA	Speed H-3Ph-EA/ DRGA*
Inst-280-1	36.7	14.27	2.57
Inst-280-2	16.83	12.7	1.33
Inst-280-3	35.02	14.13	2.48
Inst-280-4	15.84	12.62	1.26
Inst-280-5	25.93	14.74	1.76
Inst-280-6	11.31	13.57	0.83
Inst-280-7	24.9	14.73	1.69
Inst-280-8	10.99	13.55	0.81
Inst-280-9	22.49	13.77	1.63
Inst-280-10	9.45	12.72	0.74
Inst-280-11	21.6	13.67	1.58
Inst-280-12	9.36	12.71	0.74
Inst-280-13	20.61	12.61	1.63
Inst-280-14	8.74	11.34	0.77
Inst-280-15	20.64	12.61	1.64
Inst-280-16	8.74	11.66	0.75

* This column indicates how much H-3Ph-EA is faster than DRGA.

Table 5
Runtime results of 30 runs.

	Average runtime of DRGA	Average runtime of H-3Ph-EA	Speed H-3Ph-EA/ DRGA*
Inst-532-1	273.41	31.3	8.74
Inst-532-2	87.69	26.23	3.34
Inst-532-3	241.76	29.37	8.23
Inst-532-4	94.77	26.37	3.59
Inst-532-5	222.84	31.48	7.08
Inst-532-6	63.34	27.89	2.27
Inst-532-7	273.55	32.25	8.48
Inst-532-8	65.71	27.36	2.4
Inst-532-9	310.59	30.29	10.25
Inst-532-10	54.82	24.88	2.2
Inst-532-11	549.26	30.73	17.87
Inst-532-12	55.73	24.88	2.24
Inst-532-13	1270.87	28.77	44.17
Inst-532-14	52.28	22.73	2.3
Inst-532-15	1209.69	28.93	41.81
Inst-532-16	52.4	22.9	2.29

* This column indicates how much H-3Ph-EA is faster than DRGA.

type LC with 0.2 edge-density, RC with 0.2 edge-density, LR with 1.0 edge-density and RR with 1.0 edge-density, only. However, by considering Tables 4–6 , we figure out that diagrams of the other types, which are not included in Fig. 10, would also be similar to diagrams in Fig. 10. Fig. 10 presents an easy view to see that time-complexity of H-3Ph-EA is obviously better than DRGA.

With these descriptions, the following question should be justified: why H-3Ph-EA shows linear time-complexity, while its exact time-complexity is O (Gen-Size $\times N \times r \times K \times |E|$). For instances with density of edges of 1, like Inst-$*$-13, Inst-$*$-14, Inst-$*$-15 and Inst-$*$-16, we have $|E| = O(N^2)$. This means that when the dimension is increased by 2 then the runtime should be increased at least by 4, because for these instances, the time complexity becomes O (Gen-Size $\times N \times r \times K \times N^2$). However in practice we observe that the run-time increment of H-3Ph-EA is linear. For instance, for Inst-280-16, Inst-532-16 and Inst-1000-16 the run-times are 11.66, 22.9 and 55.06, respectively. This case can be easily justified. The term $r \times K \times |E|$ of time complexity is to compute K nearest-neighbors of each node for r times during the algorithm.

Table 6
Runtime results of 30 runs.

	Average runtime of DRGA	Average runtime of H-3Ph-EA	Speed H-3Ph-EA/ DRGA*
Inst-1000-1	1093.72	63.08	17.34
Inst-1000-2	594.76	62.52	9.51
Inst-1000-3	1053.34	63.75	16.52
Inst-1000-4	592.24	60.57	9.78
Inst-1000-5	667.86	66.56	10.03
Inst-1000-6	401.56	67.72	5.93
Inst-1000-7	650.73	67.74	9.61
Inst-1000-8	429.27	64.84	6.62
Inst-1000-9	528.58	63.23	8.36
Inst-1000-10	352.47	60.15	5.86
Inst-1000-11	505.99	63.98	7.91
Inst-1000-12	355.94	59.91	5.94
Inst-1000-13	481.66	58.72	8.2
Inst-1000-14	327.6	55.92	5.86
Inst-1000-15	488.36	57.89	8.17
Inst-1000-16	325.25	55.06	5.86

* This column indicates how much H-3Ph-EA is faster than DRGA.

However, it should be considered that this computation does not consider all edges, it considers edges to which a priority has been assigned in the first phase, so the density of edges with priority is too light, and then the speed of computing K nearest-neighbors is increased.

5.3. The effects of parameters setting

We set up several other experiments to show the effects of varying important parameters of the evolutionary algorithm on the performance of H-3Ph-EA. In these experiments, H-3Ph-EA with different parameters is used to solve instances with dimension of 1000, 30 times.

The first parameter is Population-Size. We consider the results of three types of H-3Ph-EA including type-1, type-2 and type-3. The Population-Size is 5, 10 and 15 for type-1, type-2 and type-3, respectively. The other parameters are same as those used for H-3Ph-EA in the earlier sub-section. Table 7 shows the results of 30 runs. The three Average columns of this table show that type-1 has better accuracy than other types. Type-1, H-3Ph-EA with Population-Size of 5 achieves the best average cost on 14 instances, while type-2 achieves the best average cost on only two instances, and type-3 never achieves the best average cost on an instance. However, for standard-deviation, the three STDV columns show that type-2 may be more stable as it achieves the minimum standard deviation across the 30 replications on eight problem instances. The author believes that, the results of Table 7 were predictable even without these experiments, because when we increase the size of population, diversity of the population increases, so its convergence postpones, and it needs more generations to reach local-optima.

The second important parameter is the rate of crossover (XR). We design another experiment in which types of H-3Ph-EA with different values of XR are applied to instances with dimension of 1000 for 30 times. The XR values for type-4, type-5 and type-6 of H-3Ph-EA are set 0.8, 0.85 and 0.9, respectively. The other remaining parameters are similar to H-3Ph-EA in the first sub-section. Table 8 shows the results of this experiment. The three Average columns of this table reveal that type-4 of H-3Ph-EA is slightly better than type-5 and type-6.

In order to consider effects of Generation-Size, similar experiment is set up. For type-7, type-8 and type-9 of H-3Ph-

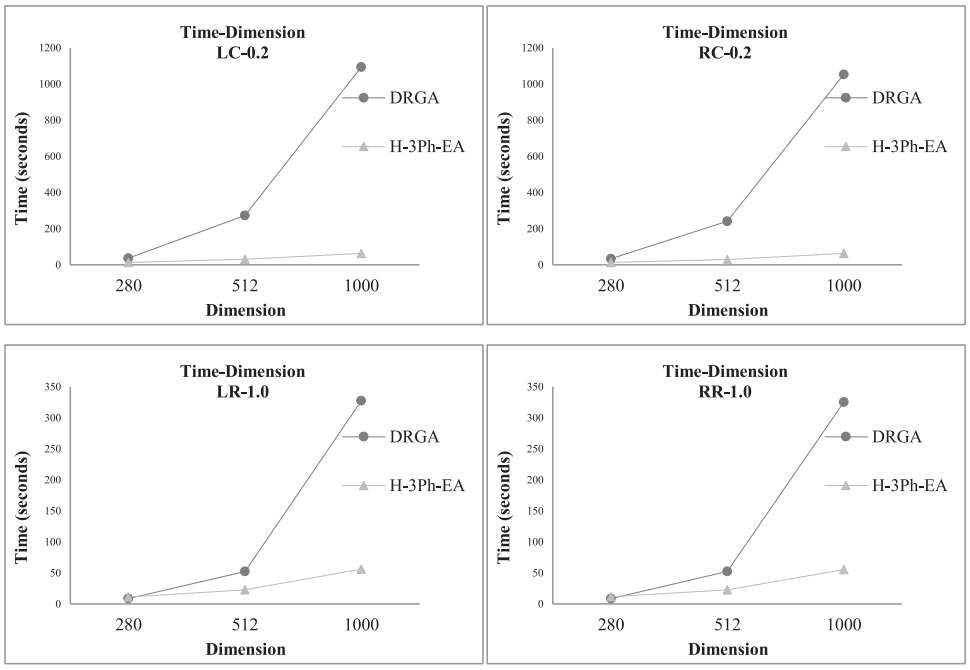


Fig. 10. Time-dimension diagrams of DRGA and H-3Ph-EA.

Table 7
Results obtained from 30 runs for the instances with dimension of 1000.

	Type-1 Population-Size = 5				Type-2 Population-Size = 10				Type-3 Population-Size = 15			
	Best	Average	STDV	Worst	Best	Average	STDV	Worst	Best	Average	STDV	Worst
Inst-1000-1	232	244.9	5.5	256	231	**241.1**	6.68	267	230	242.73	8.89	273
Inst-1000-2	138	**146.33**	3.75	154	145	149.7	3.24	157	147	155.3	5.39	170
Inst-1000-3	223	234.17	5.74	249	214	**227.6**	6.33	243	219	228.27	6.34	246
Inst-1000-4	136	**144.17**	4.19	154	141	148.37	2.86	155	146	151.77	3.56	158
Inst-1000-5	122	**128.93**	3.74	139	127	131.93	3.23	142	129	135.83	3.5	143
Inst-1000-6	80	**85.47**	2.75	91	80	87.1	2.75	93	84	88.57	2.08	92
Inst-1000-7	114	**122.7**	3.78	130	118	124.6	2.99	131	124	130.1	2.98	138
Inst-1000-8	78	**82.9**	2.23	88	79	84.6	2.43	90	80	86.47	2.75	92
Inst-1000-9	87	**92.97**	3.37	102	92	97.2	2.89	107	93	99.53	3.33	108
Inst-1000-10	56	**62.77**	2.85	67	60	63.4	2.14	68	59	64.67	2.15	69
Inst-1000-11	83	**90.9**	3.18	95	89	93.9	3.13	100	90	98.63	3.86	107
Inst-1000-12	58	**61.47**	1.89	66	59	62.9	1.9	66	59	63.87	2.37	69
Inst-1000-13	74	**80.5**	2.99	84	78	82.3	2.83	88	78	83.73	2.83	92
Inst-1000-14	49	**54.57**	2.37	61	53	55.63	1.75	59	51	55.73	2.13	61
Inst-1000-15	74	**80.9**	2.99	89	77	82.33	3.26	91	79	84.73	3.49	93
Inst-1000-16	50	**54.3**	2.38	58	50	55.5	2.56	61	52	55.73	1.8	59

Table 8
Results obtained from 30 runs for the instances with dimension of 1000.

	Type-4 XR = 0.80				Type-5 XR = 0.85				Type-6 XR = 0.90			
	Best	Average	STDV	Worst	Best	Average	STDV	Worst	Best	Average	STDV	Worst
Inst-1000-1	224	**239.2**	9.52	272	231	241.1	6.68	267	230	243.3	5.22	253
Inst-1000-2	140	**148.97**	3.63	155	145	149.7	3.24	157	145	152.47	4.21	167
Inst-1000-3	215	**226.7**	6.13	244	214	227.6	6.33	243	221	231.3	5.95	244
Inst-1000-4	142	148.83	5.26	162	141	**148.37**	2.86	155	141	148.67	4	156
Inst-1000-5	127	132.6	3.79	141	127	**131.93**	3.23	142	124	133.2	3.26	139
Inst-1000-6	82	**86.7**	2.44	91	80	87.1	2.75	93	84	88.07	2.63	94
Inst-1000-7	119	125.3	3.88	136	118	**124.6**	2.99	131	117	126	3.98	133
Inst-1000-8	78	**84.1**	2.35	87	79	84.6	2.43	90	81	85.8	2.67	93
Inst-1000-9	92	96.87	2.75	102	92	97.2	2.89	107	91	**96.67**	3.73	104
Inst-1000-10	57	**63.13**	2.42	66	60	63.4	2.14	68	61	65.3	2.05	70
Inst-1000-11	87	**93.8**	4.14	103	89	93.9	3.13	100	89	94.23	2.45	99
Inst-1000-12	58	**62.23**	2.11	66	59	62.9	1.9	66	59	62.9	1.95	66
Inst-1000-13	74	**80.8**	3.28	87	78	82.3	2.83	88	78	83	2.38	87
Inst-1000-14	52	**54.5**	1.76	58	53	55.63	1.75	59	52	56.2	1.99	61
Inst-1000-15	76	**81.73**	2.74	87	77	82.33	3.26	91	77	83.13	2.53	88
Inst-1000-16	50	**54.27**	1.74	57	50	55.5	2.56	61	52	55.87	2.11	61

Table 9
Results obtained from 30 runs for the instances with dimension of 1000.

	Type-7 Population-Size = 20 Generation-Size = 1.000.000				Type-8 Population-Size = 20 Generation-Size = 1.500.000				Type-9 Population-Size = 20 Generation-Size = 2.000.000			
	Best	Average	STDV	Worst	Best	Average	STDV	Worst	Best	Average	STDV	Worst
Inst-1000-1	233	242.23	5.75	253	229	237.17	4.58	245	220	**234.23**	8.08	252
Inst-1000-2	148	155.2	4.34	166	144	152.13	4.02	161	144	**151.23**	4.67	162
Inst-1000-3	220	231.73	6.38	250	212	224.93	5.39	237	215	**223.6**	5.55	239
Inst-1000-4	147	152.5	3.45	161	145	151.4	3.67	158	142	**149.67**	4.04	163
Inst-1000-5	135	140.4	3.74	150	127	136.37	3.56	142	125	**133.97**	4.83	145
Inst-1000-6	81	89.57	3.08	94	80	88.13	3.28	95	81	**86.43**	2.1	92
Inst-1000-7	125	133.6	4.01	140	124	131.4	3.42	138	125	**129.83**	3.23	137
Inst-1000-8	81	87.43	2.57	92	81	85.6	2.51	90	79	**84.9**	2.64	89
Inst-1000-9	96	102.1	3.4	108	92	**100.33**	3.37	106	94	100.67	3.07	108
Inst-1000-10	62	65.27	1.66	70	59	64.03	2.28	67	58	**63.2**	2.16	68
Inst-1000-11	95	100.3	3.26	111	92	99.33	3.52	107	92	**97.87**	3.31	107
Inst-1000-12	58	63.6	2.25	70	58	62.67	2.09	66	58	**61.93**	2.05	65
Inst-1000-13	79	86.07	3.46	92	81	86.23	2.81	93	78	**84.17**	3.7	93
Inst-1000-14	53	55.37	1.54	58	50	54.8	2.25	59	51	**53.93**	1.7	59
Inst-1000-15	82	87.07	3.22	93	78	84.47	2.71	90	75	**83.2**	3.43	89
Inst-1000-16	53	56.03	2.04	60	52	55.03	1.47	57	50	**54.23**	1.76	58

Table 10
Average runtime (second) of 30 runs for the instances with dimension of 1000.

	Type-1	Type-2	Type-3	Type-4	Type-5	Type-6	Type-7	Type-8	Type-9
Inst-1000-1	57.63	66.69	76.17	67.77	66.69	65.74	80.8	120.63	159.99
Inst-1000-2	56.32	64.12	70.98	65.39	64.12	63.22	77.29	115.81	153.1
Inst-1000-3	57.35	65.93	71.31	66.92	65.93	64.99	80.67	119.37	158.54
Inst-1000-4	55.12	64.12	68.91	64.88	64.12	63.69	77.38	114.7	152.5
Inst-1000-5	60.18	70.58	74.99	71.83	70.58	69.62	84.46	125.23	164.47
Inst-1000-6	58.02	68.56	71.94	69.53	68.56	67.46	79.85	120.91	159.22
Inst-1000-7	59.51	70.13	75.46	71.04	70.13	69.61	84.21	124.28	165.13
Inst-1000-8	58.44	67.88	72.77	68.93	67.88	67.22	80	119.71	160.22
Inst-1000-9	56.31	66.56	71.81	67.46	66.56	66.3	80.33	120.37	159.87
Inst-1000-10	55.32	64.43	68.31	64.67	64.43	64.21	76.65	113.41	151.42
Inst-1000-11	57.09	66.63	72.26	67.79	66.63	65.91	80.3	119.14	158.75
Inst-1000-12	54.03	65.8	68.26	65.11	65.8	63.75	77	114.14	153.47
Inst-1000-13	53.55	62.85	67.79	63.86	62.85	62.85	75.9	112.43	150.08
Inst-1000-14	52.08	60.78	63.78	61.56	60.78	60.23	72.05	107.53	143.37
Inst-1000-15	53.72	63.45	68.63	64.41	63.45	62.78	75.89	112.89	149.24
Inst-1000-16	57.63	66.69	76.17	67.77	66.69	65.74	80.8	120.63	159.99

EA, Generation-Sizes are 1.000.000, 1.500.000 and 2.000.000, respectively, and for all these types Population-Size is 20. Table 9 shows results obtained from 30 runs. It is obvious that type-9 is definitely more accurate than the other types. Only for Inst-1000-9, type-8 is slightly better than type-9. Considering the Best column of type-9 reveals that it has new bounds for some instances. With enough Population-Size, by increasing Generation-Size, we reach to high quality tours; however, the runtime increases. Fortunately, because we use steady-state type of genetic algorithm, the runtime increment is linear. The average runtime of all types are introduced in Table 10. The last three columns of this table are for type-7, type-8 and type-9. The average runtime of tye-9 is twice of type-7, where the Generation-Size of type-9 is also twice of type-7. This case is occurred similarly, when we compare runtime of type-7 and type-8 respect to their Generation-Size. Therefore, we can easily conclude that for the linear increment of Generation-Size, runtime increment becomes linear.

6. Conclusion

This paper proposes a hybrid three-phase evolutionary algorithm which is named H-3Ph-EA. H-3Ph-EA is formed from three phases. Each phase of H-3Ph-EA is a GA, and the third phase is a memetic algorithm, in which GA is hybridized with a type of local search, namely LK. The LK is one of the most efficient local searches for classical TSP. H-3Ph-EA uses two novel heuristic crossovers and after converting the CTSP to the TSP problem, utilizes advantages of the LK algorithm which is one of the most powerful TSP solver.

H-3Ph-EA is accurate and results of experiments confirm this, where, this algorithm significantly outperforms DRGA, while experiments in Silberholz et al. (2013) show that DRGA itself is more accurate and faster than the recent state-of-the-arts. H-3Ph-EA is also faster than DRGA and for some instances, it is 16 times faster than DRGA.

In addition an idea of H-3Ph-EA, defining new cost for each edge, may be applicable to other combinatorial optimization problems, where we can define new heuristic costs, and replace new costs in the end of each deterministic period (some generations), repeatedly. Therefore, for this case, after performing parameter-tuning process again, we may be able to apply H-3Ph-EA to other combinatorial optimization problems like variants of TSP, variants of dynamic vehicle routing problem and etc.

Appendix I

1-742-48-176-950-890-730-858-206-911-848-631-495-853-603-229-349-612-283-544-268-352-431-860-233-844-100-403-237-519-422-769-304-810-308-274-236-450-863-9-817-132-254-568-87-563-136-73-658-16-595-906-531-453-444-875-483-587-230-433-311-355-570-429-13-154-197-187-189-315-137-713-765-834-543-186-29-879-916-732-6-489-907-638-643-654-683-816-425-173-490-211-466-854-635-641-818-628-712-1000-985-999-990-992-976-978-159-970-964-265-177-75-198-102-331-437-903-402-169-80-366-447-445-849-932-33-428-47-427-99-85-330-300-365-185-158-556-204-933-475-689-244-692-617-751-419-430-253-552-227-350-130-493-362-463-839-724-535-504-101-235-215-435-223-530-685-779-699-11-942-394-966-578-479-161-952-292-894-804-583-707-573-691-58-465-46-656-975-120-306-191-195-559-744-775-416-184-579-454-144-584-377-285-830-364-439-406-112-934-929-986-959-998-962-991-332-196-386-757-392-862-378-156-953-271-49-569-754-715-134-231-368-371-888-456-408-776-339-826-739-842-341-35-358-503-367-316-510-755-551-877-344-44-861-39-64-560-870-621-634-34-718-514-107-640-815-171-193-668-821-118-580-874-335-395-12-259-25-240-7-342-27-565-38-180-20-841-773-764-673-667-768-541-792-784-824-676-814-455-222-528-527-390-93-247-56-696-529-347-524-60-515-774-919-270-533-122-441-115-614-469-387-957-637-505-98-24-897-32-340-10-940-687-680-695-820-720-571-743-905-593-770-789-678-731-626-484-606-296-599-662-829-666-623-703-591-781-886-15-898-388-396-474-806-405-443-252-89-357-95-160-359-299-282-918-440-131-941-937-30-398-23-648-74-972-963-960-974-562-965-251-420-245-313-218-298-79-329-147-305-52-276-536-783-598-852-558-149-289-117-704-657-36-663-787-753-451-501-496-837-575-210-152-401-523-542-823-709-697-766-812-645-381-549-828-525-324-467-76-642-258-553-710-693-723-622-360-857-219-777-418-577-256-923-109-632-411-449-361-399-833-106-334-65-19-55-646-54-384-59-250-478-798-756-234-797-228-788-982-423-989-984-973-981-26-967-625-988-534-994-971-997-945-507-895-843-706-661-908-811-548-800-586-114-241-884-847-127-924-92-277-607-288-486-943-880-516-318-476-799-711-803-572-417-468-309-557-255-421-762-290-3-927-17-639-51-925-337-297-708-801-747-752-909-690-864-728-655-281-819-400-887-412-284-436-968-561-741-633-652-677-672-665-321-409-143-376-946-786-624-838-538-644-550-750-726-522-372-462-40-286-68-174-893-141-45-630-878-574-896-930-414-18-540-4-194-432-539-611-225-123-146-77-170-500-43-472-749-585-669-702-615-679-734-714-205-325-91-275-338-610-506-526-426-78-448-312-217-931-208-310-481-470-322-589-179-949-119-581-736-866-885-28-926-139-748-434-190-226-537-84-212-142-547-202-303-910-182-172-287-373-21-145-954-153-42-348-31-604-922-148-588-383-851-808-600-955-243-458-62-701-518-794-597-104-302-917-263-199-958-629-684-996-995-825-554-759-83-951-868-135-892-482-345-94-601-512-608-636-521-671-717-681-291-686-498-249-353-881-785-889-128-351-125-987-948-379-97-2-126-346-567-242-555-221-627-807-705-761-782-899-735-664-694-650-566-188-651-727-688-674-873-746-613-758-508-609-314-590-738-592-983-582-57-140-369-103-356-835-336-915-660-513-480-682-389-150-82-264-90-956-41-200-53-939-14-546-5-457-307-594-8-670-832-370-175-167-413-836-721-871-293-602-224-477-698-902-407-865-214-796-944-791-460-869-928-61-260-151-488-333-138-977-375-494-936-487-722-809-716-827-382-993-183-980-63-485-737-778-238-913-725-273-912-266-232-116-165-947-813-845-272-66-675-872-647-882-790-771-719-850-733-517-326-492-473-267-891-239-327-81-323-280-209-564-831-181-162-938-124-133-178-509-471-497-70-620-914-822-576-464-213-745-619-37-961-69-969-86-979-164-659-729-415-605-380-295-520-461-700-438-71-248-763-262-328-883-900-772-920-491-301-363-867-410-105-168-113-129-391-354-319-22-805-88-96-261-67-343-499-50-121-155-653-320-110-207-278-904-269-72-935-374-317-111-424-446-616-802-793-532-459-294-397-220-545-855-192-859-760-385-442-921-257-452-203-795-649-767-780-846-393-246-279-108-216-502-856-157-840-201-876-596-740-618-166-901-163-404-511

The best tour obtained by H-3Ph-EA for Inst-1000-1 (Type: LC, 0.2, Dimension: 1000).

Acknowledgment

The author thanks editor and anonymous referees for reviewing this paper and their constructive comments and suggestions. This work is supported by University of Bonab with contract number: 94/D/AP/543.

References

Adham, M. T., & Bentley, P. J. (2014). An artificial ecosystem algorithm applied to the travelling salesman problem. *2014 Conference companion on genetic and evolutionary computation companion*.

Ahmed, Z. H. (2013). A hybrid genetic algorithm for the bottleneck traveling salesman problem. *ACM Transactions on Embedded Computing Systems, 12*(1) 9:1-9:10.

Applegate, D., Cook, W., & Rohe, A. (2003). Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing, 15*(1), 82–92.

Broersma, H., & Li, X. (1997). Spanning trees with many or few colors in edge-colored graphs. *Discussiones Mathematicae Graph Theory, 17*(2), 259–269.

Broersma, H., Li, X., Woeginger, G., & Zhang, S. (2005). Paths and cycles in colored graphs. *Australasian Journal on Combinatorics, 31*, 299–311.

Carr, R. D., Doddi, S., Konjevod, G., & Marathe, M. (2000). On the red-blue set cover problem. *The eleventh annual ACM-SIAM symposium on Discrete algorithms*.

Carrabs, F., Cerulli, R., & Gentili, M. (2009). The labeled maximum matching problem. *Computers and Operations Research, 36*(6), 1859–1871.

Cerulli, R., Dell'Olmo, P., Gentili, M., & Raiconi, A. (2006). Heuristic approaches for the minimum labelling hamiltonian cycle problem. *Electronic Notes in Discrete Mathematics, 25*(1), 131–138.

Cerulli, R., Fink, A., Gentili, M., & Voß, S. (2005). Metaheuristics comparison for the minimum labelling spanning tree problem. In *The next wave in computing, optimization, and decision technologies* (pp. 93–106). US: Springer.

Cerulli, R., Fink, A., Gentili, M., & Voß, S. (2006). Extensions of the minimum labelling spanning tree problem. *Journal of Telecommunication and Information Technology, 4*, 39–45.

Chang, R.-S., & Leu, S.-J. (1997). The minimum labeling spanning trees. *Information Processing Letters, 63*(5), 277–282.

Chen, Y., Cornick, N., Hall, A. O., Shajpal, R., Silberholz, J., Yahav, I., & Golden, B. L. (2008). Comparison of heuristics for solving the Gmlst problem. In *Telecommunications modeling, policy, and technology* (pp. 191–217). Springer.

Couëtoux, B., Gourvès, L., Monnot, J., & Telelis, O. A. (2008). On labeled traveling salesman problems. The 19th international symposium on algorithms and computation..

Diaz-Delgadillo, F. J., Montiel, O., & Sepúlveda, R. (2016). Reducing the size of traveling salesman problems using vaccination by fuzzy selector. *Expert Systems with Applications, 49*, 20–30.

Dorigo, M., & Gambardella, A. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.

Edelkamp, S., Gath, M., Cazenave, T., & Teytaud, F. (2013). Algorithm and knowledge engineering for the TSPTW problem. *IEEE symposium on computational intelligence in scheduling (SCIS), 2013*.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York: W. H. Freeman & Co..

Hassin, R., Monnot, J., & Segev, D. (2007). Approximation algorithms and hardness results for labeled connectivity problems. *Journal of Combinatorial Optimization, 14*(4), 437–453.

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research, 126*(1), 106–130.

Helsgaun, K. (2009). General k-opt submoves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation, 1*(2–3), 119–163.

Ismkhan, H., & Zamanifar, K. (2010). Using ants as a genetic crossover operator in GLS to solve STSP. *International conference of soft computing and pattern recognition*.

Ismkhan, H., & Zamanifar, K. (2014). Developing programming tools to handle traveling salesman problem by three object oriented languages. *Applied Computational Intelligence and Soft Computing*.

Jebari, K., moujahid, A. E., Bouroumi, A., & Ettouhami, A. (2012). Unsupervised fuzzy clustering-based genetic algorithms to traveling salesman problem. *International conference on multimedia computing and systems (ICMCS)*.

Johnson, D. S. (1990). Local optimization and the traveling salesman problem. *the 17th international colloquium on automata, languages and programming (ICALP 90)*.

Jolai, F., & Ghanbari, A. (2010). Integrating data transformation techniques with Hopfield neural networks for solving travelling salesman problem. *Expert Systems with Applications, 37*(7), 5331–5335.

Jozefowiez, N. (2011). A branch-and-cut algorithm for the minimum labeling Hamiltonian cycle problem and two variants. *Computers and Operations Research, 38*(11), 1534–1542.

Karaboga, D., & Gorkemli, B. (2011). A combinatorial Artificial Bee Colony algorithm for traveling salesman problem. *2011 International symposium on innovations in intelligent systems and applications (INISTA)*: 2.

Krumke, S. O., & Wirth, H.-C. (1998). On the minimum label spanning tree problem. *Information Processing Letters, 66*(2), 81–85.

Li, J., Xing, D., Huaxuan, L., & MengChu, Z. (2015). A decomposition approach to colored traveling salesman problems. *IEEE international conference on automation science and engineering (CASE)*.

Li, J., Zhou, M., Sun, Q., Dai, X., & Yu, X. (2014). Colored traveling salesman problem. *IEEE Transactions on Cybernetics, 45*(11), 2390–2401.

Lin, S., & Kernighan, B. W. (1973). An Effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21*(2), 498–516.

López-Ibáñez, M., & Blum, C. (2010). Beam-ACO for the travelling salesman problem with time windows. *Computers and Operations Research, 37*(9), 1570–1583.

López-Ibáñez, M., Blum, C., Ohlmann, J. W., & Thomas, B. W. (2013). The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing, 13*(9), 3806–3815.

Martin, O., Otto, S. W., & Felten, E. W. (1991). Large-step Markov chains for the traveling salesman problem. *Complex Systems, 5*(3), 299–326.

Mavrovouniotis, M., & Yang, S. (2011). A memetic ant colony optimization algorithm for the dynamic travelling salesman problem. *Soft Computing, 15*(7), 1405–1425.
Mladenović, N., Urošević, D., Hanafi, S., & Ilić, A. (2012). A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research, 220*(1), 270–285.
Montiel, O., Diaz-Delgadillo, F. J., & Sepúlveda, R. (2013). Combinatorial complexity problem reduction by the use of artificial vaccines. *Expert Systems with Applications, 40*, 1871–1879.
Nagata, Y., & Kobayashi, S. (2013). A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *Informs Journal on Computing, 25*(2), 346–363.
Ouaarab, A., Ahiod, B., & Yang, X.-S. (2014). Discrete cuckoo search algorithm for the travelling salesman problem. *Neural Computing and Applications, 24*(7-8), 1659–1669.
Perez, D., Rohlfshagen, P., & Cowling, P. I. (2013). Solving the physical travelling salesman problem: Tree search and macro-actions. *IEEE Transactions on Computational Intelligence and AI in Games, 6*(1), 31–45.
Petersen, H. L., & Archetti, C. S. (2010). Exact solutions to the double travelling salesman problem with multiple stacks. *Networks, 56*(4), 229–243.
Silberholz, J., Raiconi, A., Cerulli, R., Gentili, M., Golden, B., & Chen, S. (2013). Comparison of heuristics for the colourful travelling salesman problem. *International Journal of Metaheuristics, 2*(2), 141–173.
Wan, Y., Chert, G., & Xu, Y. (2002). A note on the minimum label spanning tree. *Information Processing Letters, 84*(2), 99–101.
Whitley, D. (1989). The genitor algorithm and selective pressure: why rank-based allocation of reproductive trials is best. *3rd International conference on Genetic Algorithm*.
Xiong, Y., Golden, B., & Wasil, E. (2005). A one-parameter genetic algorithm for the minimum labeling spanning tree problem. *IEEE Transactions on Evolutionary Computation, 9*(1), 55–60.
Xiong, Y., Golden, B., & Wasil, E. (2005). Worst-case behavior of the MVCA heuristic for the minimum labeling spanning tree problem. *Operations Research Letters, 33*(1), 77–80.
Xiong, Y., Golden, B., & Wasil, E. (2006). Improved heuristics for the minimum labelling spanning tree problem. *IEEE Transactions on Evolutionary Computation, 10*(6), 700–703.
Xiong, Y., Golden, B., & Wasil, E. (2007). The Colorful Traveling Salesman Problem. In *Advances in computing, optimization, and decision technologies* (pp. 115–123). US: Springer.

Hassan Ismkhan joined University of Bonab as instructor, after he got his master degree in computer engineering in 2011. His teaching courses include data structures, algorithm design, information retrieval, and theory of formal languages and automata. His research interests include Social Networks, Information Retrieval, Clustering Algorithms, Evolutionary Algorithms and Operational Research.