Accepted Manuscript

Title: A New Cooperative Framework for Parallel
Trajectory-Based Metaheuristics

Author: Jialong Shi Qingfu Zhang

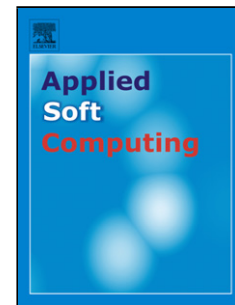Please cite this article as: Jialong Shi, Qingfu Zhang, A New Cooperative Framework
for Parallel Trajectory-Based Metaheuristics, <![CDATA[Applied Soft Computing
Journal]]> (2018), https://doi.org/10.1016/j.asoc.2018.01.022

This is a PDF file of an unedited manuscript that has been accepted for publication.
As a service to our customers we are providing this early version of the manuscript.
The manuscript will undergo copyediting, typesetting, and review of the resulting proof
before it is published in its final form. Please note that during the production process
errors may be discovered which could affect the content, and all legal disclaimers that
apply to the journal pertain.

- A new cooperative framework is proposed for parallel trajectory-based metaheuristics.
- A parallel variant of Guided Local Search (GLS) is designed using the proposed framework.
- Extensive experiments have been conducted on the Tianhe-2 supercomputer to test the designed parallel GLS algorithm.

A New Cooperative Framework for Parallel Trajectory-Based Metaheuristics

Jialong Shi[a], Qingfu Zhang[b,c]

[a]School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an, China
[b]Department of Computer Science, City University of Hong Kong, Hong Kong
[c]The City University of Hong Kong Shenzhen Research Institute, Shenzhen, China

Abstract

In this paper, we propose the Parallel Elite Biased framework (PEB framework) for parallel trajectory-based metaheuristics. In the PEB framework, multiple search processes are executed concurrently. During the search, each process sends its best found solutions to its neighboring processes and uses the received solutions to guide its search. Using the PEB framework, we design a parallel variant of Guided Local Search (GLS) called PEBGLS. Extensive experiments have been conducted on the Tianhe-2 supercomputer to study the performance of PEBGLS on the Traveling Salesman Problem (TSP). The experimental results show that PEBGLS is a competitive parallel metaheuristic for the TSP, which confirms that the PEB framework is useful for designing parallel trajectory-based metaheuristics.

Keywords: Combinatorial Optimization, Parallel Metaheuristics, Algorithm Design, Guided Local Search

1. Introduction

Metaheuristics are often used to find nearly optimal solutions of hard optimization problems within a reasonable amount of time. There are two main categories of metaheuristics [1, 2]: trajectory-based metaheuristics and population-based ones. A trajectory-based metaheuristic iteratively improves a single solution and forms a search trajectory in the solution space.

Email addresses: shi.jl@outlook.com (Jialong Shi), qingfu.zhang@cityu.edu.hk (Qingfu Zhang)
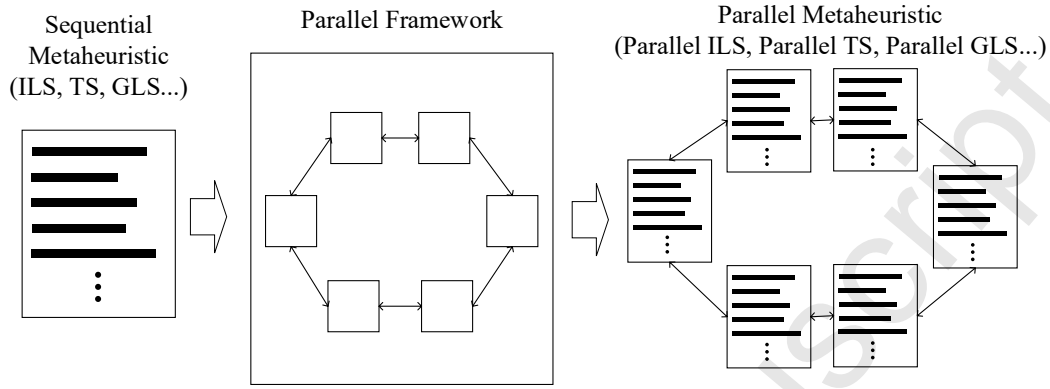
Figure 1: Following a parallel framework, one can design the parallel variants of different sequential metaheuristics

Examples of trajectory-based metaheuristics include Simulated Annealing (SA), Tabu Search (TS), Iterated Local Search (ILS) and Guided Local Search (GLS) [3]. In population-based metaheuristics, a population of solutions is processed by several operators at each iteration (generation). The members of the population are replaced by new ones so that the solution space can be explored. Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Artificial Bee Colony (ABC) [4] are some widely-used population-based metaheuristics.

With the increasing popularity of multi-processor and multi-core platforms, parallelism has become ubiquitous in today's computer technologies. Hence, parallel metaheuristics have attracted a lot of research effort. Here we argue that designing a *parallel framework* is more essential than designing a parallel metaheuristic. A parallel framework is a universal model for designing the parallel variants of a certain kind of metaheuristics. As sketched in Figure 1, a parallel framework defines how multiple metaheuristic processes cooperate with each other and one can apply different sequential metaheuristics to this framework to design different parallel metaheuristics. In this paper we propose a parallel framework which can be used to design the parallel variants of trajectory-based metaheuristics including TS, GLS, etc.

In [5], we have proposed a Parallel Elite Biased Tabu Search (PEBTS) algorithm for the Unconstrained Binary Quadratic Programming (UBQP) problem. We generalize the parallelism strategies of PEBTS and propose the Parallel Elite Biased framework (PEB framework) in this paper. Using

2

the PEB framework, we design a new parallel variant of GLS, called Parallel Elite Based GLS (PEBGLS). Extensive experiments are conducted to study the performance and behavior of PEBGLS using the symmetric Traveling Salesman Problem (TSP) as the test suite. We hope that our study can provide a new possible direction for designing parallel metaheuristics.

This paper is structured as follows. Section 2 reviews the related works. In Section 3, the PEB framework is presented and discussed. In Section 4 we design a parallel variant of GLS following the PEB framework. Section 5 gives the experimental studies on the TSP. Section 6 concludes this paper.

2. Related Works

The existing parallelism strategies of trajectory-based metaheuristics fall into the following two categories.

- *Low-level parallelism* or *acceleration strategy*. This strategy aims at speeding up a sequential metaheuristic. It does not change the behavior of the original sequential metaheuristic. The implementations of this strategy usually use the master-slave topology. The master controls the main procedure, dispatching tasks to the slaves. The tasks can be evaluating moves, or evaluating partial solutions. After collecting and integrating the results returned by the slaves, the master proceeds to the subsequent procedure.

- *High-level parallelism* or *multiple search strategy*. In this strategy, multiple search processes are executed simultaneously. Each process makes a unique trajectory in the search space. The heuristic methods and initial solutions of the search processes can be the same or different. They may run independently and communicate at the end to identify the best overall solution, or they may exchange useful information during the search.

A number of high-level parallel trajectory-based metaheuristics have been proposed for various problems. Table 1 summarizes the related works on parallel trajectory-based metaheuristics.

As shown in Table 1, in most of the existing parallel trajectory-based metaheuristics, the parallel processes exchange solutions with each other. The solutions can be the best solutions found so far or some elite solutions. In Table 1, the centralized communication method is used by

3

Table 1: Literature Review

Related Work	Algorithm	Problem	Information Type	Information Exchanging Method	Information Utilizing Method
Garcia-Lopez et al. 2002 [6]	Parallel VNS	P-median problem	Best found solutions	Centralized	Restart
Bortfeldt et al. 2003 [7]	Parallel TS	Container loading problem	Best found solutions	Distributed	Restart
Attanasio et al. 2004 [8]	Parallel TS	Dynamic multi-vehicle dial-a-ride problem	Best found solutions & Visiting frequencies	Distributed	Restart & Refer to the frequencies
Banos et al. 2004 [9]	Parallel SA-TS	Graph partitioning	Best found solutions	Distributed	Restart
Crainic et al. 2004 [10]	Parallel VNS	P-median problem	Best found solutions	Centralized	Restart
Blazewicz et al. 2004 [11]	Parallel TS	2-dimensional cutting	Best found solutions	Centralized	Restart
Le Bouthillier and Crainic 2005 [12]	Parallel TS, EA & Post-optimization	VRPTW	Elite solutions	Centralized	Path-relinking
Le Bouthillier, et al. 2005 [13]	Parallel TS, EA, Post-optimization & Pattern identification	VRPTW	Elite solutions & Solution attributes	Centralized	Restart & Fix or prohibit the attributes
Talbi and Bachelet 2006 [14]	COSEARCH (GA, Kick Operator & TS)	QAP	Elite solutions & Global frequencies	Centralized	Restart & Refer to the frequencies
Fischer and Merz 2005 [15] 2007 [16]	Parallel Chained Lin-Kernighan	TSP	Best found solutions	Distributed	Restart
Lukasik et al. 2007 [17]	Parallel SA	Graph coloring problem	Best found solutions	Centralized	Restart
Ribeiro and Rosseti 2007 [18]	Parallel GRASP	2-path network design problem	Elite solutions	Centralized	Path-relinking
Araujo et al. 2007 [19]	Parallel GRASP-ILS	Mirrored traveling tournament problem	Elite solutions	Centralized	Restart
Aydin and Sevkli 2008 [20]	Parallel VNS	Job shop scheduling	Best found solutions	Distributed	Restart
Polacek et al. 2008 [21]	Parallel VNS	MDVRPTW	Best found solutions	Centralized	Restart
Dos Santos et al. 2009 [22]	Parallel GRASP, GA & Q-learning	TSP	Best found solutions	Centralized	Restart & Update Q-values table
Luque et al. 2010 [23] 2011 [24]	Parallel SA	DNA fragment assembly, MAXSAT & RND	Best found solutions	Distributed	Combination operation
Subramanian et al. 2010 [25]	ILS-RVND	VRPSPD	Best parameter values	Centralized	Set the parameters
Hung and Chen 2011 [26]	Parallel Branch-and-Bound method & TS	TSP	Best found solutions	Centralized	Restart
Cordeau and Maischberger 2012 [27]	Parallel ILS-TS	VRP	Best found solutions	Distributed	Restart with a probability
Lee et al. 2012 [28]	Harmony search	Task scheduling problem	Elite solutions	Centralized	Restart
Jin et al. 2014 [29]	Parallel TS	Capacitated VRP	Best found solutions	Centralized	Restart
Hemmelmayr 2015 [30]	Parallel Large Neighborhood Search	Periodic Location Routing Problem	Best found solutions	Centralized	Restart
Iturriaga et al. 2015 [31]	Parallel stochastic LS	Heterogeneous Computing Scheduling	Neighboring solutions	Centralized	Evaluate & report
Lahrichi et al. 2015 [32]	Integrative Cooperative Search	MDPVRP	Solutions & partial solutions	Centralized	Restart or integrate
Luque and Alba 2015 [33]	Parallel SA	DNA fragment assembly & QAP	Best found solutions	Distributed	Path-relinking
Tosun 2015 [34]	Parallel GA & TS	QAP	Elite solutions	Centralized	Crossover & restart
Wang et al. 2015 [35]	Parallel SA	VRPSPDTW	Best found solutions	Centralized	Restart
Sousa Filho et al. 2016 [36]	GRASP-VNS	Bicluster editing problem	Best found solutions	Centralized	Restart
Guzman et al 2016 [37]	Parallel TS & SA	TSP	Best found solutions	Centralized	Restart
Quan & Wu 2017 [38]	Parallel ILS	Disjunctively Constrained Knapsack Problem	Elite solutions	Centralized	Restart
Tu et al. 2017 [39]	Parallel ILS	VRP	Subproblems & solutions	Centralized	run ILS on subproblem

4

most of the metaheuristics, while some metaheuristics apply the distributed communication method, in which each process only shares information with a limited number of processes. Compared to the centralized communication method, the distributed communication method is more flexible and can be used in massive parallel processing platforms.

In Table 1, the information utilizing methods in many existing parallel trajectory-based metaheuristics are denoted as "restart". In this method, when a process receives a new solution, it abandons the current solution and restarts from the received one. As a consequence, the information in the current solution is lost. Some works try to overcome this drawback by using path-relinking methods or other combination operators. In their methods, a new solution is generated based on the received solution and the current solution.

3. Parallel Elite Biased Framework

In this section, we propose the Parallel Elite Biased framework (PEB framework) for the designing of parallel trajectory-based metaheuristics with multiple search processes. To design a parallel trajectory-based metaheuristic with multiple search processes, three issues must be addressed:

- What information should be exchanged among different processes? (information type)

- How should information be exchanged? (communication method)

- How should the received information be utilized? (information utilizing method)

For the first issue, in the PEB framework, different processes exchange their best found solutions with each other. For the second issue, the PEB framework follows a distributed topology in which each process only sends solutions to their predefined neighbors. For the third issue, the PEB framework applies a novel information utilizing method. In the related works, when a process receives a new solution, it either restarts from the received solution or executes path-relinking to generate a new solution. In the PEB framework, the process continues searching from its current solution, but its search will have bias toward the received solutions.
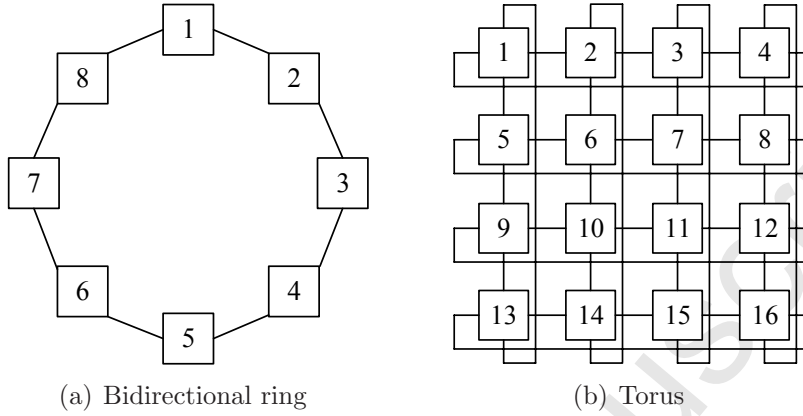
5

(a) Bidirectional ring (b) Torus

Figure 2: Different parallel topologies

3.1. Communication Method

The proposed PEB framework is based on a distributed topology. To reduce the communication load, each process only communicates with a number of neighboring processes. In this paper, we consider two distributed topologies: the bidirectional ring topology and the torus topology. They are two of the most natural distributed topologies and widely used in the area of parallel metaheuristics [40, 41]. Both topologies support flexible process number. Figure 2 illustrates the examples of these parallel topologies. In the bidirectional ring topology, each process has two neighbors. For example, in Figure 2(a), the neighbors of process 1 are process 2 and process 8. In the torus topology, each process has four neighbors. For example, in Figure 2(b) the neighbors of process 1 are process 2, process 5, process 4 and process 13.

A well-designed parallel metaheuristic must control the communication load of the processes. A rigidly synchronous communication strategy may cause heavy communication load and reduce the efficiency of the parallel metaheuristic. The PEB framework follows an asynchronous communication pattern. For each search process in the PEB framework, we denote s_{hb} as the historical best solution found by itself and S_r as the set of solutions received from its neighboring processes. After a given period of time, each process checks whether s_{hb} has changed since the previous sending. If so, it sends the new s_{hb} to its neighbors. Meanwhile each process keeps receiving new solutions from its neighboring processes. Note that, although a process may receive better solutions from its neighbors, it always sends the best solution found by itself to its neighbors. This maintains the diversity of the parallel

6

metaheuristic. This strategy is helpful to reduce the communication load among processes.

3.2. Information Utilizing Method

In the PEB framework, each process maintains an elite solution s_e, which is the best solution in the set $S_r \cup \{s_{hb}\}$. The PEB framework applies a novel method to utilize the information in s_e. Instead of restarting from the s_e, each process continues searching from the current solution and its search procedure has bias toward s_e. In other words, each process is "attracted" by s_e. In such way, each process can utilize the information in s_e without abandoning the information in its current solution. In practise, the way to realize the attraction of s_e is decided by users.

To illustrate the information utilizing method of the PEB framework, we give an example in Figure 3. In Figure 3(a), process A and process B start from different solutions and perform different trajectories. There is no communication between A and B in Figure 3(a). We compare the information utilizing method of the PEB framework with two widely-used methods: the restart method and the path-relinking method. In the restart method, as shown in Figure 3(b), process B receives a solution from process A and restarts from the received solution. The original solution of process B is abandoned. In the path-relinking method (Figure 3(c)), process B generates a new solution using the path-relinking operator based on the received solution and its current solution. Then it proceeds with the search from the resulting solution. The original solution of process B is abandoned too. In the PEB framework (Figure 3(d)), process B continues searching from its original solution, but the search direction of process B is attracted by the solution received from process A. The original solution of process B is not abandoned in the PEB framework. This can maintain the diversity of the processes.

3.3. Pseudocode

The procedure of each process in the PEB framework is shown in Algorithm 1. In Algorithm 1, the *TryToReceive* procedure always prepares to update the set S_r if it receives new solutions from some neighbors. At the pre-defined time points (e.g. a given number of iterations), the *SelectBestSolution* procedure selects the best one of the set $S_r \cup \{s_{hb}\}$ as s_e, and the *SendToNeighbors* procedure sends s_{hb} to all neighbors if s_{hb} has changed since the previous sending. The *EliteBiasedSearch* procedure
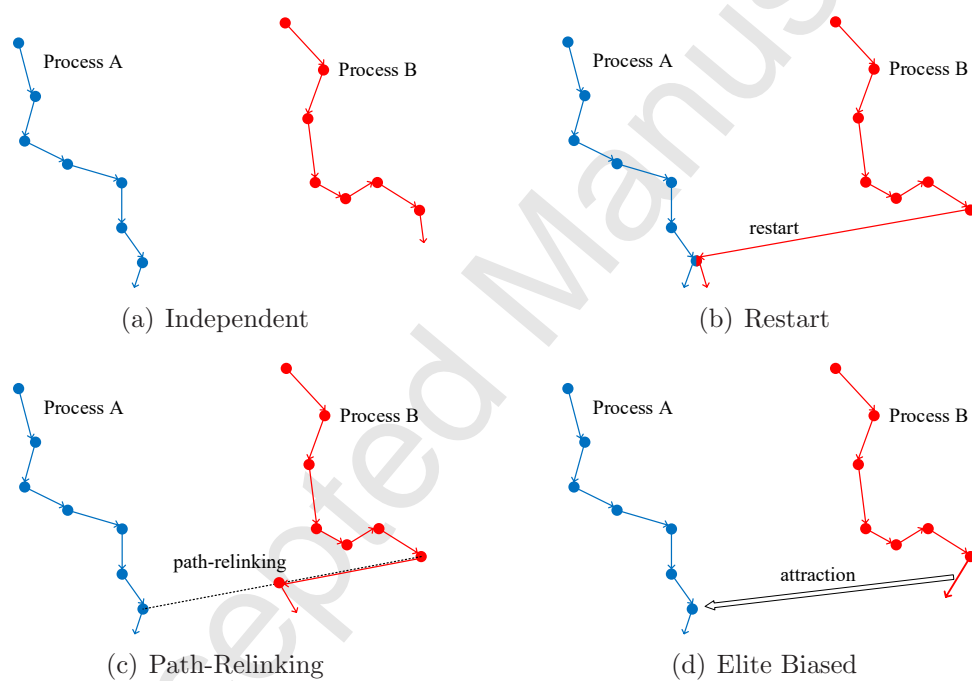
7

(a) Independent

(b) Restart

(c) Path-Relinking

(d) Elite Biased

Figure 3: Different information utilizing methods

8

denotes the combination of the elite biased concept and the original search procedure of a given trajectory-based metaheuristic. The details of the EliteBiasedSearch procedure are decided by users and depend on the mechanism of the metaheuristic users want to parallelize. For example, in PEBTS [5], the EliteBiasedSearch procedure is a TS procedure which is influenced by the recorded elite solution s_e. In this paper, we propose PEBGLS and its EliteBiasedSearch procedure is a GLS procedure which is presented in Section 4.

Algorithm 1 Parallel Elite Biased Framework

 1: **initialize**: s, s_{hb}
 2: **while** !StoppingCriterion **do**
 3:     $S_r \leftarrow$ TryToReceive()
 4:     **if** at the pre-defined time points **then**
 5:         $s_e \leftarrow$ SelectBestSolution($S_r \cup \{s_{hb}\}$)
 6:         **if** s_{hb} has been updated since the previous sending **then**
 7:             SendToNeighbors(s_{hb})
 8:         **end if**
 9:     **end if**
10:     $\{s, s_{hb}\} \leftarrow$ EliteBiasedSearch(s, s_{hb}, s_e)
11: **end while**
12: **return** s_{hb}

---

4. Designing Parallel Guided Local Search

To show the utility of the proposed PEB framework, in this section, we design a parallel variant of Guided Local Search (GLS) following the PEB framework.

4.1. Guided Local Search

GLS is an efficient trajectory-based metaheuristics for combinatorial optimization problems. It iteratively helps a LS procedure to escape from local optima by dynamically adjusting its guide function. We assume that there is a combinatorial optimization problem with solution space S and objective function $g : S \to \mathbb{R}$ to minimize. To apply GLS on this problem, one first needs to define features for candidate solutions in S. Each feature has a fixed cost and a penalty. The cost is related to the objective function $g(\cdot)$. The penalty is set to 0 at the beginning and changes during the search.

9

GLS does not use $g(\cdot)$, but the *augmented objective function* $h(\cdot)$ as the guide function of LS:

$$h(s) = g(s) + \lambda \sum_{i \in M} p_i I_i(s), \tag{1}$$

where s is a candidate solution, λ is a pre-defined parameter that controls the penalizing strength, M is the set of all features in the problem, p_i is the current penalty value of feature i and function $I_i(s)$ is an indicator function of whether solution s has feature i:

$$I_i(s) = \left\{ \begin{array}{ll} 1 & \text{if feature } i \text{ is in } s, \\ 0 & \text{otherwise.} \end{array} \right. \tag{2}$$

In each iteration, GLS executes a LS using $h(\cdot)$ as the guide function. Once the LS stops at a local optimum s_*, GLS adjusts $h(\cdot)$ by increasing the penalties of one or more selected features in s_*. To do so, GLS defines the penalizing utility of each feature i, $util_i$, as

$$util_i(s_*) = I_i(s_*) \cdot \frac{c_i}{1 + p_i}, \tag{3}$$

where c_i is the cost of feature i. GLS selects the features with the highest utility value and increases their penalties by 1. Then a new iteration starts from s_*. In (3), the numerator is the cost of feature, which means that features with higher costs are more likely to be penalized and thus low cost features are exploited. The denominator is the accumulated penalty of feature plus 1, which means that the features that has been rarely penalized before have a good chance to be penalized. In such a way, the search explores new regions of the search space.

The pseudocode of GLS is shown in Algorithm 2. The inputs are the objective function g, the GLS parameter λ, the feature set M and the cost of each feature $\{c_i | i \in M\}$.

In Algorithm 2, the LS procedure is based on $h(\cdot)$, so GLS needs to track the historical best solution s_{hb} with regard to the original objective function g. After each move of LS, GLS checks whether the g value of the new solution is better than that of the recorded best solution, if so, the historical best solution s_{hb} will be updated.

4.2. Parallel Elite Biased Guided Local Search

Tairan and Zhang [42] proposed a parallel GLS algorithm called P-GLS-II. However, they first transformed GLS into a population-based metaheuristic,

10

Algorithm 2 Guided Local Search

1: **input:** g, λ, M, c
2: $j \leftarrow 0$
3: $s_0 \leftarrow$ random or heuristically generated solution.
4: $s_{hb} \leftarrow s_0$
5: **for** $i = 1 \rightarrow |M|$ **do**
6:    $p_i \leftarrow 0$
7: **end for**
8: **while** !StoppingCriterion **do**
9:    $h \leftarrow g + \lambda \sum p_i I_i$
10:    $\{s_{j+1}, s_{hb}\} \leftarrow \text{LocalSearch}(s_j, s_{hb}, h)$
11:    **for** $i = 1 \rightarrow |M|$ **do**
12:       $util_i \leftarrow I_i(s_{j+1}) \cdot c_i/(1 + p_i)$
13:    **end for**
14:    **for** each i such that $util_i$ is maximum **do**
15:       $p_i \leftarrow p_i + 1$
16:    **end for**
17:    $j \leftarrow j + 1$
18: **end while**
19: **return** s_{hb}

11

then ran it in a parallel way. To our best knowledge, there is no parallel trajectory-based variant of GLS. Following the proposed PEB framework, we designed a parallel variant of GLS, which is called Parallel Elite Biased GLS (PEBGLS).

4.2.1. The Attraction of s_e

In the PEB framework, for each process, an elite solution s_e is selected from the set formed by the received solutions and the current historical best solution. Then the search process is attracted by s_e. GLS executes LS based on the function $h(\cdot)$ which is augmented by the penalties. Hence the descending nature of the LS will guide GLS to the solutions with less penalties. The proposed parallel variant of GLS aims to reduce the number of penalties imposed on the features that belong to s_e and increase more penalties on the features not belonging to s_e. As a result, the search process will be orientated to the search regions near to s_e. To achieve this aim, we modified the formula of $util$, i.e. Equation (3). The new formula is:

$$util_i(s_*) = \begin{cases} I_i(s_*) \cdot c_i/(1 + p_i), & \text{if feature } i \text{ is in } s_e; \\ I_i(s_*) \cdot w \cdot c_i/(1 + p_i), & \text{otherwise,} \end{cases} \tag{4}$$

where $w > 1$ is a predefined parameter. In Equation (4), if a feature is not in s_e, its penalizing utility will be multiplied by an extra coefficient w. Since $w > 1$, features not in s_e will have relatively large $util$ values, so they are more likely to be penalized compared to the features in s_e. Hence the penalties imposed on s_e will become relatively small. Due to the descending nature of LS, the search direction of GLS will be attracted by s_e.

4.2.2. The Procedure of PEBGLS

The procedure of each PEBGLS process is shown in Algorithm 3. In PEBGLS, there is a predefined parameter $U \in \mathbb{N}^+$. Every U iterations, the PEBGLS process updates s_e to the best one of the set $S_r \cup \{s_{hb}\}$ and sends its s_{hb} to all neighbors if s_{hb} has changed since the previous sending. Here U can be used to control the communication load. The inputs of PEBGLS are: objective function g, the feature set M, the cost of each feature $\{c_i | i \in M\}$ and the user-defined parameters $\{\lambda, w, U\}$.

This section shows how we apply the PEB framework to GLS, including how to realize the attraction of s_e in GLS. An example of applying the PEB framework to Tabu Search can be found in [5]. When applying the PEB framework to other kinds of trajectory-based metaheuristics, users need to

12

Algorithm 3 Parallel Elite Biased Guided Local Search

1: **input:**g, M, c, λ, w, U
2: $j \leftarrow 0$
3: $s_0 \leftarrow$ random or heuristically generated solution
4: $s_{hb} \leftarrow s_0$
5: $S_r \leftarrow \emptyset$
6: **for** $i = 1 \rightarrow |M|$ **do**
7: $\quad p_i \leftarrow 0$
8: **end for**
9: **while** !StoppingCriterion **do**
10: $\quad S_r \leftarrow$ TryToReceive()
11: $\quad$ **if** $j\%U == 0$ **then**
12: $\quad\quad s_e \leftarrow$ SelectBestSolution($S_r \cup \{s_{hb}\}$)
13: $\quad\quad$ **if** s_{hb} has updated since the previous sending **then**
14: $\quad\quad\quad$ SendToNeighbors(s_{hb})
15: $\quad\quad$ **end if**
16: $\quad$ **end if**
17: $\quad h \leftarrow g + \lambda \sum p_i I_i$
18: $\quad \{s_{j+1}, s_{hb}\} \leftarrow$ LocalSearch(s_j, s_{hb}, h)
19: $\quad$ **for** $i = 1 \rightarrow |M|$ **do**
20: $\quad\quad$ **if** feature i is in s_e **then**
21: $\quad\quad\quad util_i \leftarrow I_i(s_{k+1}) \cdot c_i/(1 + p_i)$
22: $\quad\quad$ **else**
23: $\quad\quad\quad util_i \leftarrow I_i(s_{k+1}) \cdot w \cdot c_i/(1 + p_i)$
24: $\quad\quad$ **end if**
25: $\quad$ **end for**
26: $\quad$ **for** each i such that $util_i$ is maximum **do**
27: $\quad\quad p_i \leftarrow p_i + 1$
28: $\quad$ **end for**
29: $\quad j \leftarrow j + 1$
30: **end while**
31: **return** s_{hb}

13

design unique attraction strategies. Our suggestion is to give priorities to the candidate solutions that are more similar/closer to s_e in each move step of a trajectory-based metaheuristic.

5. Experimental Studies

In the experimental studies, we tested the performance of PEBGLS on the Traveling Salesman Problem. The TSP is one of the most widely-used benchmarks in the area of combinatorial optimization and it is also one of the most well-known applications of GLS [3].

5.1. Applying Guided Local Search to the Traveling Salesman Problem

In the TSP, $G = (V, E)$ is a fully connected graph where V is its node set and E the edge set, $c_e > 0$ is the cost of edge $e \in E$. A solution tour s in G is a cycle passing through every node in V exactly once and its cost is defined as:

$$g(s) = \sum_{e \in s} c_e. \tag{5}$$

Here $g(\cdot)$ is the objective function of the TSP and the goal of the TSP is to find a tour with the smallest g value. This paper considers the symmetric TSP, where the cost from node A to node B is the same as that from B to A. We denote the set of all the feasible tours in G as S, which is the solution space of the TSP.

To apply GLS to the TSP, we define that the features are edges in G (i.e. feature set $M =$ edge set E) and the features' costs are the costs of the corresponding edges. If a solution tour s contains the edge e_i (i.e. the feature i), then $I_i(s) = 1$, otherwise $I_i(s) = 0$. In this paper, we apply 2-Opt move in GLS, because according to [3] GLS performs better with 2-Opt, especially when it is combined with the Fast Local Search (FLS) strategy [43].

5.2. Speedup

The main purpose of applying parallel metaheuristics is to accelerate the sequential metaheuristics; hence *speedup* is an important metric to measure the performance of parallel metaheuristics. In this section, we measured the speedup of the proposed PEBGLS with different process number on different TSP instances, to study the accelerating ability and scalability of PEBGLS.

The speedup metric compares the runtime of parallel algorithms against the runtime of sequential algorithms. We denote T_1 as the runtime of a

14

PEBGLS with only one process and T_K as the runtime of a K-process PEBGLS. Note that the runtime measured in parallel algorithms is wall-clock time. Then the speedup $\mathcal{S}_K$ is calculated by:

$$\mathcal{S}_K = \frac{E[T_1]}{E[T_K]}. \tag{6}$$

where $E[\cdot]$ is the expectation function. If $\mathcal{S}_K < K$, we call it a *sublinear* speedup; if $\mathcal{S}_K = K$, we call it a *linear* speedup; if $\mathcal{S}_K > K$, we call it a *superlinear* speedup. The other widely used metric is *efficiency* e_K, which equals $\mathcal{S}_K/K$. Obviously $e_K \geq 1$ is desirable.

The test instances were att532, pr1002 and rl1304 from the TSPLIB [44]. In TSPLIB, the number in the name of an instance is the node number n of this instance. For PEBGLS, the torus topology (PEBGLS-t) and the bidirectional ring topology (PEBGLS-br) were tested. The experiment was conducted on the Tianhe-2 supercomputer. Tianhe-2 is one of the world's top-ranked supercomputers. It is equipped with 17,920 computer nodes, each comprising two Intel Xeon E5-2692 12C (2.200 GHz) processors. So each node has 24 cores and the system supports elastic parallel computing across nodes. We also used the FLS strategy and Bentley's improvement [43] to enhance the efficiency of the 2-Opt heuristic in PEBGLS. Based on [3], the coefficient λ is calculated by:

$$\lambda = 0.3 \cdot \frac{g(\text{first local optimum})}{n}, \tag{7}$$

where $g(\text{first local optimum})$ is the function value of the first local optimum visited by GLS and n is the number of cities of the TSP instance. Our pilot experiments showed that PEBGLS is not very sensitive to w. So here we set $w = 2$. U was set to be 100. Our PEBGLS program was implemented in GNU C++ with O2 optimizing compilation. The OpenMPI library was used as the message passing tool. To calculate $E[T_1]$, 100 runs of single-process PEBGLS were executed. The runs started from randomly generated solutions. Because the global optimally costs of the test instances are known, all runs terminated only when the globally optimal cost was achieved. The wall-clock time of each run was recorded. To calculate $E[T_K]$, 100 runs of K-process PEBGLS were executed on K cores. Each process occupied a core. For PEBGLS-br, K separately took the values $\{8, 16, 24, 32, 40, 48\}$. For PEBGLS-t, K separately took the values $\{9, 16, 24, 32, 40, 48\}$ and the

15

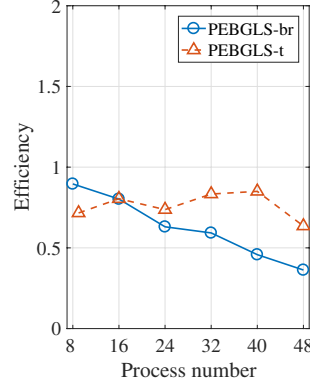Table 2: Speedup and efficiency of PEBGLS-t and PEBGLS-br

		$K=1$	$K=8$	$K=16$	$K=24$	$K=32$	$K=40$	$K=48$
				PEBGLS-br				
att532	$\bar{T}_K$	4.9130s	0.6854s	0.3827s	0.3247s	0.2591s	0.2682s	0.2821s
	$\mathcal{S}_K$	-	7.1681	12.8377	15.1309	18.9618	18.3184	17.4158
	e_K	-	0.8960	0.8024	0.6305	0.5926	0.4580	0.3628
pr1002	$\bar{T}_K$	22.5781s	2.5219s	1.9939s	1.8329s	1.6991s	1.6943s	1.6857s
	$\mathcal{S}_K$	-	8.9528	11.3236	12.3182	13.2883	13.3259	13.3939
	e_K	-	1.1191	0.7077	0.5133	0.4153	0.3331	0.2790
rl1304	$\bar{T}_K$	19.7827s	2.7610s	2.1300s	2.0523s	1.8586s	1.8647s	1.7307s
	$\mathcal{S}_K$	-	7.1650	9.2877	9.6393	10.6439	10.6091	11.4305
	e_K	-	0.8956	0.5805	0.4016	0.3326	0.2652	0.2381
		$K=1$	$K=8$	$K=16$	$K=24$	$K=32$	$K=40$	$K=48$
				PEBGLS-t				
att532	$\bar{T}_K$	4.9130s	0.7623s	0.3818s	0.2778s	0.1842s	0.1445s	0.1611s
	$\mathcal{S}_K$	-	6.4450	12.8680	17.6854	26.6721	34.0000	30.4966
	e_K	-	0.7161	0.8042	0.7369	0.8335	0.8500	0.6353
pr1002	$\bar{T}_K$	22.5781s	1.9496s	1.3762s	1.1720s	0.9076s	0.8648s	0.7761s
	$\mathcal{S}_K$	-	11.5809	16.4061	19.2646	24.8767	26.1079	29.0917
	e_K	-	1.2868	1.0254	0.8027	0.7774	0.6527	0.6061
rl1304	$\bar{T}_K$	19.7827s	2.1997s	1.9255s	1.6398s	1.5846s	1.4923s	1.4689s
	$\mathcal{S}_K$	-	8.9934	10.2741	12.0641	12.4843	13.2565	13.4677
	e_K	-	0.9993	0.6421	0.5027	0.3901	0.3314	0.2806

shape of the torus topology were $\{(3\times3),(4\times4),(4\times6),(4\times8),(5\times8),(6\times8)\}$ respectively. The resulting speedup $\mathcal{S}_K$ values and efficiency e_K values are shown in Table 2 and Figure 4. Table 2 also lists the average runtime $\bar{T}_K$ for each K value.

From Table 2 and Figure 4 we can see that, overall PEBGLS-t achieved higher speedup values than PEBGLS-br, which means the torus topology is better than the bidirectional ring topology on these instances. However, on the largest instance rl1304, the speedup difference between PEBGLS-br and PEBGLS-t was not significant. As the process number K increased, the efficiency values of PEBGLS-br and PEBGLS-t decreased, except for the PEBGLS-t running on att532. On att532, PEBGLS-t attained the highest efficiency value when $K = 40$. When K was relatively large, the efficiency values of PEBGLS-br and PEBGLS-t decreased when the problem size increased. For example, when $K = 40$, PEBGLS-br attained the highest efficiency value on att532 and the lowest on rl1304, so did PEBGLS-t. On
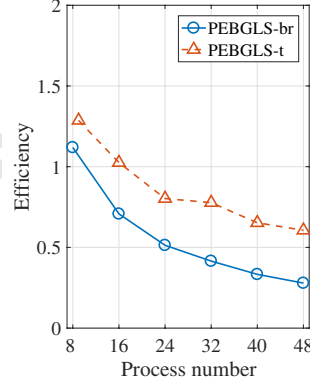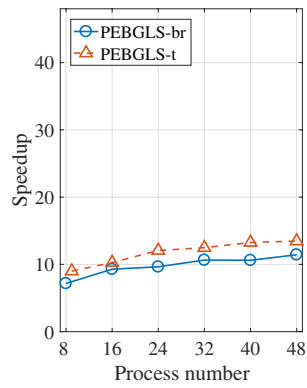
16
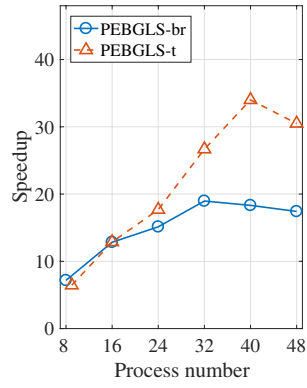
(a) $\mathcal{S}_K$ on att532
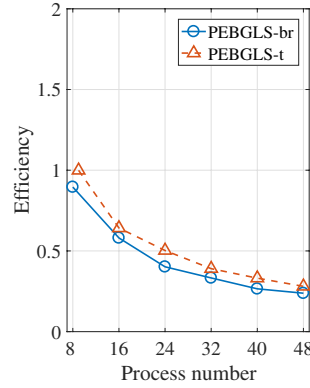
(b) e_K on att532
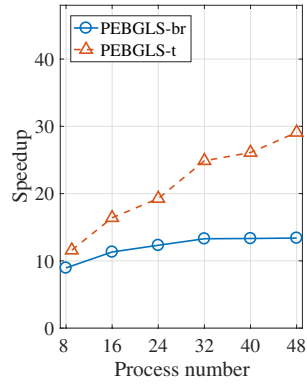
(c) $\mathcal{S}_K$ on pr1002

(d) e_K on pr1002

(e) $\mathcal{S}_K$ on rl1304

(f) e_K on rl1304

Figure 4: The speedup $\mathcal{S}_K$ and efficiency e_K of PEBGLS-t and PEBGLS-br

17

the other hand, when K was relatively small, the efficiency value seems to be unrelated to the problem size. For example, when $K = 9$, on pr1002 the efficient value attained by PEBGLS-t was 1.2868 (superlinear), meanwhile on att532 the value was 0.7161 (sublinear).

5.3. Influence of Communication Frequency

The performance of a parallel metaheuristic is influenced by the communication load among processes. In PEBGLS, every U iterations, each process sends s_{hb} to all neighbors if s_{hb} has changed since the previous sending. Hence the communication frequency of PEBGLS has a negative relation to the parameter U. In this section, we conducted an experiment to investigate the influence of communication frequency in PEBGLS by setting different U values.

The platform of this experiment was the Tianhe-2 supercomputer. We selected pr1002, pr2392, fnl4461, rl5915, pla7397 and rl11849 from the TSPLIB as the test instances. For PEBGLS, the torus topology (PEBGLS-t) and the bidirectional ring topology (PEBGLS-br) were tested. The maximum runtime for different instances were different: {pr1002:21s, pr2392:48s, fnl4461:90s, rl5915:119s, pla7397:237s, rl11849:371s}. For each TSP instance and each U value, we executed 20 runs for each algorithm. In each run, $K = 48$ processes started from different random solutions and the torus topology shape in PEBGLS-t was 6×8. If an algorithm run attains the globally optimal cost before the maximum runtime, it will stop immediately. U separately took the values of $\{1, 500, 1000, 2000\}$. The other experimental settings were the same as the settings in Section 5.2. The performance metrics are *excess* and *runtime*, in which the excess is defined by:

$$\text{excess} = \frac{\text{solution cost} - \text{globally optimal cost}}{\text{globally optimal cost}} \times 100\%. \tag{8}$$

Table 3 shows the average excess and average runtime attained by each algorithm, in which the best metric values are in bold. We can see that, on pr1002 and pr2392, most runs of the algorithms attained zero excess, which means that the globally optimal cost was reached before the maximum runtime. Hence in Figure 5(a) and Figure 5(b) we present the boxplot of the real runtime of each algorithm on these two instances. On the rest four instances, no algorithm reached the globally optimal cost, hence in Figure 5(c), Figure 5(d), Figure 5(e) and Figure 5(f) we present the boxplot of the best excess attained by each algorithm.
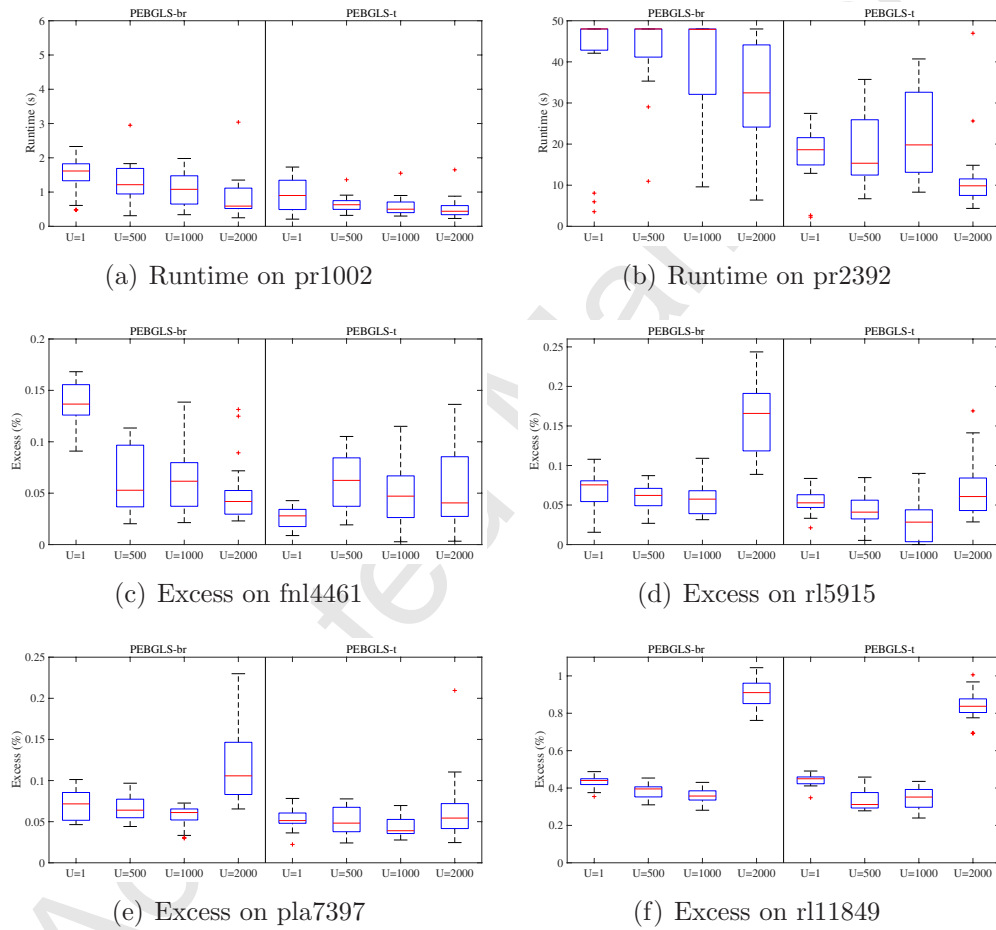
18

(a) Runtime on pr1002

(b) Runtime on pr2392

(c) Excess on fnl4461

(d) Excess on rl5915

(e) Excess on pla7397

(f) Excess on rl11849

Figure 5: Excesses and runtime of PEBGLS-br and PEBGLS-t, process number $K = 48$

19

Table 3: Performance of PEBGLS-br and PEBGLS-t with different U values, process number $K = 48$

	PEBGLS-br				PEBGLS-t			
	$U=1$	$U=500$	$U=1000$	$U=2000$	$U=1$	$U=500$	$U=1000$	$U=2000$
Instance	Average Excess (%)							
pr1002	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
pr2392	0.0004	0.0001	0.0001	0.0001	**0.0000**	**0.0000**	**0.0000**	**0.0000**
fnl4461	0.1368	0.0637	0.0656	0.0506	**0.0266**	0.0613	0.0475	0.0543
rl5915	0.0694	0.0591	0.0578	0.1595	0.0544	0.0432	**0.0306**	0.0693
pla7397	0.0720	0.0658	0.0567	0.1175	0.0534	0.0507	**0.0445**	0.0653
rl11849	0.4311	0.3853	0.3587	0.9066	0.4411	**0.3366**	0.3477	0.8391
Instance	Average Runtime (s)							
pr1002	1.51	1.33	1.07	0.85	0.90	0.65	0.59	**0.54**
pr2392	40.63	42.86	39.29	31.57	17.31	18.59	22.24	**11.87**
fnl4461	90.00	90.00	90.00	90.00	90.00	90.00	90.00	90.00
rl5915	119.00	119.00	119.00	119.00	119.00	119.00	119.00	119.00
pla7397	237.00	237.00	237.00	237.00	237.00	237.00	237.00	237.00
rl11849	371.00	371.00	371.00	371.00	371.00	371.00	371.00	371.00

From Table 3 and Figure 5 we can see that, in most cases, PEBGLS-t attained lower excess values or lower runtime than PEBGLS-br, which means that the torus topology is better than the bidirectional ring topology. This conclusion is the same to the conclusion in Section 5.2. We also can see that, there was a trade-off between the algorithm performance and the communication frequency (parameter U). When the communication frequency was very high (i.e. U was very small), PEBGLS did not perform very well. For example, on rl5915 (Figure 5(d)) the average excess attained by PEBGLS-t when $U = 1$ was worse than when $U = 500$ and 1000. This is because when the communication frequency was very high, each PEBGLS process spent a lot of additional time to communicate with other processes. On the other hand, a relatively low communication frequency also reduced the algorithm performance. For example, on rl11849 (Figure 5(f)) the average excess attained by PEBGLS-t deteriorated when $U = 2000$. An interesting phenomenon is that, on pr1002 and pr2392, the best U value for PEBGLS-t was 2000, which was larger than the best U values on other instances. That is because pr1002 and pr2392 are smaller than other instances, hence in one second, PEBGLS-t executed more iterations on pr1002 and pr2392 than on other instances.

20

5.4. Internal Behavior of PEBGLS-t

The previous experiments show that for PEBGLS the torus topology is better than the bidirectional ring topology, hence in the following experiments we only apply the torus topology to PEBGLS. The pervious experiments also show that the collaboration among different PEBGLS processes improves the overall solution quality. In this section we investigate how the collaboration benefits each process. To answer that question, we recorded and studied the internal behavior of PEBGLS-t during the search. For comparison, we also recorded the behavior of Independent PEBGLS processes (P-I-EBGLS). The only different between PEBGLS-t and P-I-EBGLS is that in P-I-EBGLS the processes do not communicate with each other and s_e is only updated by s_{hb} every U iterations.

The experimental platform was Tianhe2 supercomputer. The test instances were gr431, att532 and rat575 from the TSPLIB. On each instance, we first randomly generated $16 \times 1,000$ different initial solutions. Then 1,000 runs of PEBGLS-t with 16 processes and 1,000 runs of P-I-EBGLS with 16 processes were started from the generated solution set. The torus topology shape in PEBGLS-t was 4×4. Hence for each PEBGLS-t run, there was a P-I-EBGLS run starting from the same initial solutions. All the runs ended when the globally optimal cost was reached. In the first 1,000 iterations, the PEBGLS-t processes did not communicate with each other, i.e. in the first 1,000 iterations the PEBGLS-t processes did not cooperate with each other. The other experimental settings were the same as the settings in Section 5.2. In this experiment, the entire history of each process was recorded.

Figure 6 shows how the average excess changed over time on the three instances. We can see that in the first 1,000 iterations, the average excess attained by PEBGLS-t was the same as that attained by P-I-EBGLS. This is because in the first 1,000 iterations the PEBGLS-t processes did not communicate with each other. After the 1,000th iteration, the PEBGLS-t processes started to communicate with each other. Then the average excess attained by PEBGLS-t became lower than that attained by P-I-EBGLS. This means that the cooperation approach in the PEB framework can truly improve the overall solution quality.

After the experiment, we analyzed all the returned solutions of the runs of PEBGLS-t and P-I-EBGLS. After eliminating the duplicated solutions, there were only two unique solutions left. This means that all the runs of PEBGLS-t and P-I-EBGLS ended in two different globally optimal solutions. By comparing the edges in these two globally optimal solutions, we found that

21
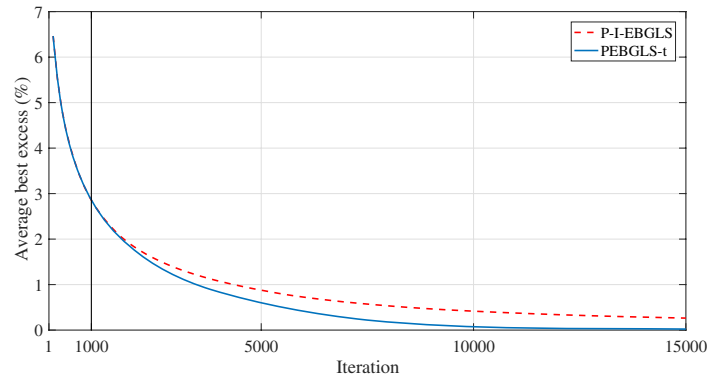
(a) gr431
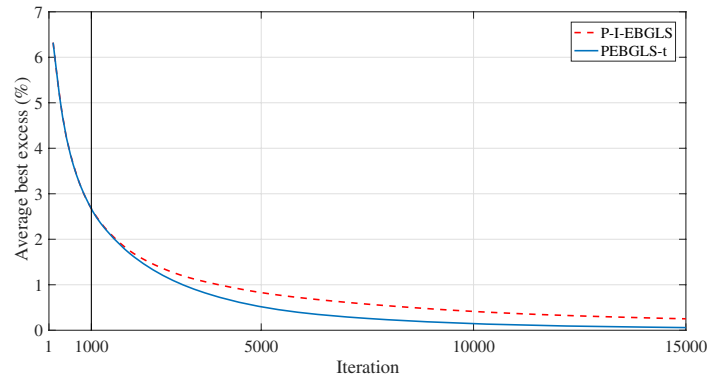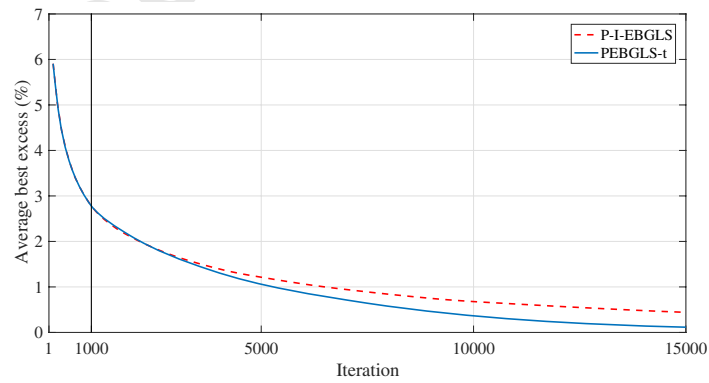
(b) att532

(c) rat575

Figure 6: The average best excess versus the time

22

the first one only had two different edges to the second one. In other words, there were totally 534 unique edges in the two globally optimal solutions of att532. We denotes these 534 edges as the *optimal edges* of att532. Using the same method, we found the numbers of optimal edges in gr431 and rat575 were 433 and 577 respectively. Since the optimal edges are the edges belongs to the globally optimal solutions, it is undesirable for PEBGLS-t to penalize the optimal edges.

For each search process, we define a metric called *ratio of undesirable penalties*, denoted by r, which is the ratio between the total penalty imposed on the optimal edges over the total penalty imposed on all the edges in a TSP instance. We use E' to denote the set of optimal edges. Then r is defined as:

$$r = \frac{\sum\limits_{i \in E'} p_i}{\sum\limits_{i \in E} p_i}. \tag{9}$$

where p_i is the penalty on edge i, E is the set of all edges of the TSP instance. Then we calculated the average ratio of undesirable penalties $\bar{r}$ among all processes in all runs. Obviously, everything being equal, a lower ratio value means a more effective penalizing mechanism of PEBGLS-t. Figure 7 shows how the average ratio of undesirable penalties changed with the time on these three instances.

In Figure 7, after the PEBGLS-t processes started to cooperate with each other, their average ratio value became smaller than that of the P-I-EBGLS processes. This means that, sharing elite solutions among processes reduced the probability of penalizing the edges in the globally optimal solutions. Hence the search processes in PEBGLS-t became more targeted and the probability of finding the global optima was increased. According to the "big valley" structure [45] of the symmetric TSP, high-quality solutions are more likely to have more common edges with the global optima. Hence by reducing the penalties imposed on the edges of the global optima, the PEBGLS-t processes had more chance to find high-quality solutions. However, in Figure 7 we can also see that the difference between the two curves became smaller at the final stage of the search. This is because GLS only increases the penalties imposed on the local optima it finds. As illustrated in Figure 6, the PEBGLS-t processes found better local optima compared to the P-I-EBGLS processes. According to the big valley structure, the local optima found by PEBGLS-t have more common edges with the global optima than the local

23
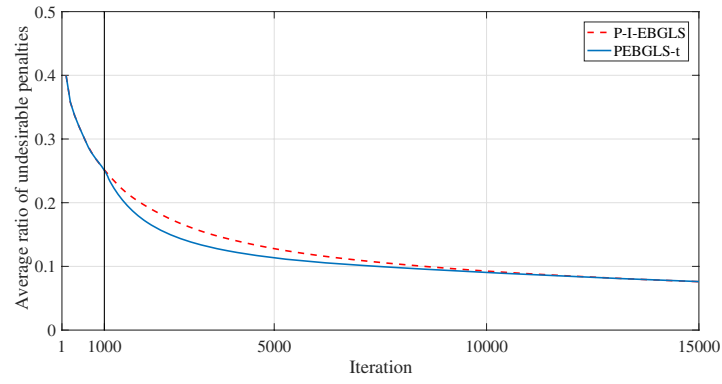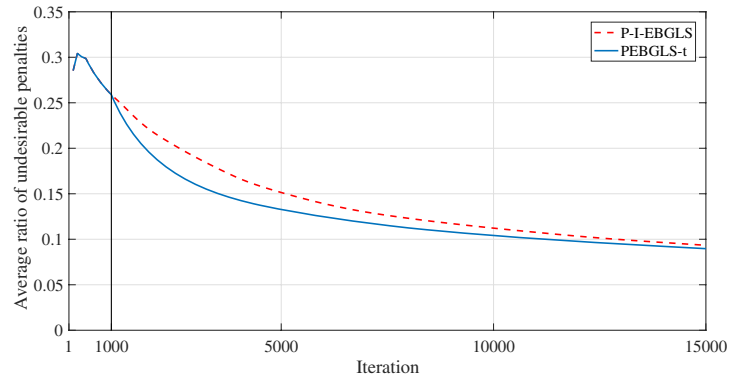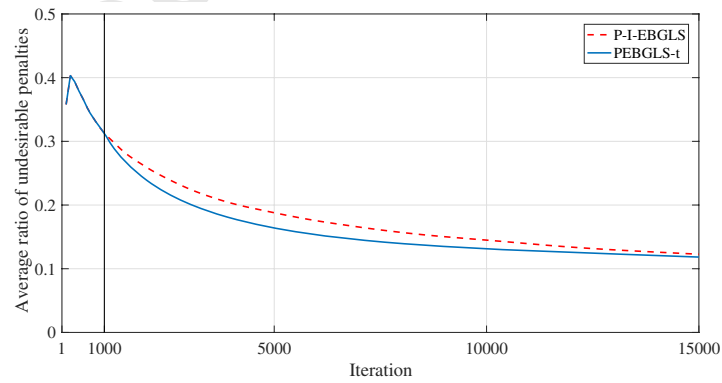
(a) gr431

(b) att532

(c) rat575

Figure 7: The average ratio of undesirable penalties versus the time

24

Table 4: Average number of best-contributors, for PEBGLS-t and P-I-EBGLS

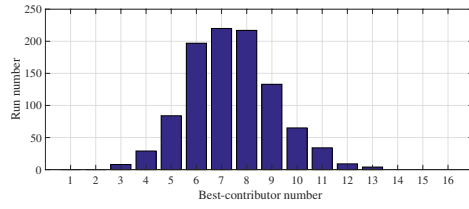|	P-I-EBGLS	PEBGLS-t
gr431	7.40	9.30
att532	7.27	9.45
rat575	8.48	11.28

optima found by P-I-EBGLS. So the difficulty of PEBGLS-t not penalizing the edges of global optima increased. Hence the difference between the ratio value of PEBGLS-t and the ratio value of P-I-EBGLS became smaller as time went by.

In a parallel trajectory-based metaheuristic, different processes search different regions of the solution space. If a process searches in a less-promising region, this process will contribute little to the global search. We call a process the *best-contributor* if it ever found a solution that was better than the overall best solution among all processes in its search history, i.e., this process once updated the overall best solution. In our experiment, each run of PEBGLS-t/P-I-EBGLS had a certain number of best-contributors. Obviously the number of best-contributors reflects the number of "useful" search processes and the overall "activeness" of the search processes. Table 4 shows the average best-contributor number attained by P-I-EBGLS and PEBGLS-t. Figure 8 shows the distribution of the best-contributor number on the 1,000 runs of P-I-EBGLS and PEBGLS-t. From Table 4 and Figure 8 we can see that, PEBGLS-t had a higher best-contributor number than P-I-EBGLS, which means that the cooperation method in PEBGLS-t increased the overall activity of the processes.

Based on the above experimental studies we can state that the new cooperation method in PEBGLS-t is effective. This means that the PEB framework can further improve the performance of GLS compared to simply running multiple processes in parallel.

5.5. Comparison with Other Parallel Metaheuristics

To test whether the proposed PEBGLS is a competitive TSP meta-heuristic, in this section we compared the proposed PEBGLS-t with four parallel metaheuristics for the TSP. The first two are two parallel variants of GLS following the restart based framework. The last two are the parallel variants of two widely-used TSP metaheuristics.
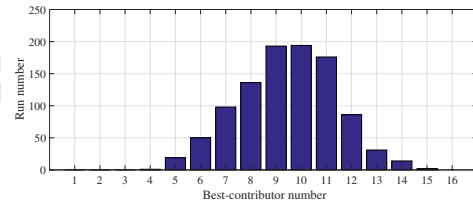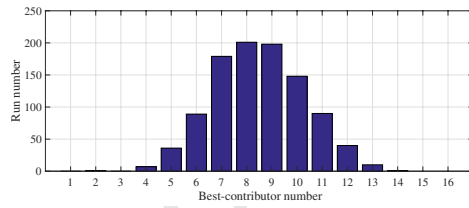
25
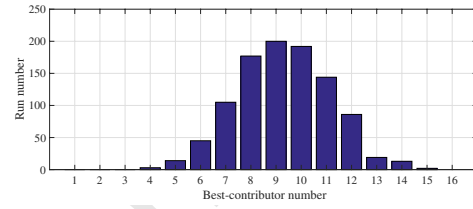
(a) P-I-EBGLS on gr431   (b) PEBGLS-t on gr431
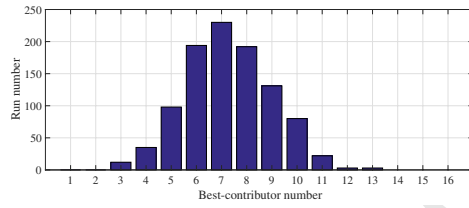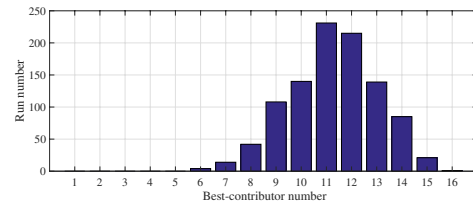
(c) P-I-EBGLS on att532   (d) PEBGLS-t on att532

(e) P-I-EBGLS on rat575   (f) PEBGLS-t on rat575

Figure 8: The distribution of the best-contributor number attained by the 1,000 runs of P-I-EBGLS and PEBGLS-t

26

The first comparison algorithm is called Parallel Restart GLS with torus topology (P-R-GLS-t). P-R-GLS-t follows a torus neighborhood topology and executes K GLS processes simultaneously. Every U iterations, each P-R-GLS-t process exchanges its historical best solution s_{hb} with its neighbors. After that, the P-R-GLS-t process abandons its current solution and restarts from the best solution of the set $S_r \cup \{s_{hb}\}$, where S_r is the set of received solutions. The second comparison algorithm combines the PEB framework with the restart based framework, which is called Parallel Restart Elite Biased GLS with torus topology (P-R-EBGLS-t). Every U iterations, each P-R-EBGLS-t process exchanges s_{hb} with its neighbors and restarts from the best solution of the set $S_r \cup \{s_{hb}\}$. Meanwhile, each P-R-EBGLS-t process selects the second best solution from the set $S_r \cup \{s_{hb}\}$ as the elite solution s_e. Similar to PEBGLS-t, P-R-EBGLS-t uses the new formula (4) to calculate $util$, so that the search direction is attracted by s_e.

The third comparison algorithm is a parallel variant of Ant Colony Optimization (P-ACO). We used the ACOTSP software package available at http://www.aco-metaheuristic.org/aco-code/. In P-ACO, K independent ACO processes are executed simultaneously. P-ACO stops when the maximum runtime is achieved or one of the processes finds the globally optimal solution. The forth comparison algorithm is a parallel variant of Iterated Lin-Kernighan algorithm (P-ILK). In our experiment, the ILK implementation came from the Concorde software package available at http://www.math.uwaterloo.ca/tsp/concorde/. The parallelization method of P-ILK was the same to that of P-ACO: K independent ILK processes are executed simultaneously and stop when the maximum runtime is reached or the globally optimal solution is found.

In our experiment, the test TSPLIB instances and the corresponding maximum runtime were {rd400:8s, att532:11s, gr666:14s, u724:15s, pr1002:21s, d1291:26s, u1432:29s, u1817:37s, pr2392:48s, fnl4461:90s}. On each instance, the run number of each algorithm was 20 and the process number K was 9. The torus topology shape was 3×3. Here we selected a relatively small K value because we intend to show that our algorithm can perform well in multi-core personal computers or small computer clusters. For PEBGLS-t, P-R-GLS-t and P-R-EBGLS-t, we set $U = 1000$ and the other settings were the same to the PEBGLS settings in Section 5.2. In P-ACO, different ACO processes had different random seeds. For each P-ACO process, the parameter settings were based on [46] and shown in Table 5. In P-ILK, different ILK processes started from different randomly generated solutions.

27

Table 5: Parameter settings of each process in P-ACO

Parameters	Description	Values
m_{ACO}	Number of ants	25
α	Influence of pheromone trails	1
β	Influence of heuristic information	2
ρ	Pheromone trail evaporation	0.2
LS	Local search	3-Opt
MMAS	MAX-MIN ant system	Apply

Each ILK process followed the default settings of the Concorde software.

Table 6 shows the comparison results of the five algorithms, in which the best metric values are in bold. From Table 6 we can see that, among all the five parallel metaheuristics, PEBGLS-t performed the best on most instances. For example, on the instances u724, PEBGLS-t achieved a zero average excess value while the other algorithms did not, which means that all the 20 runs of PEBGLS-t found the globally optimal solution. By comparing PEBGLS-t with P-R-GLS-t and P-R-EBGLS-t we can see that, overall the performance of PEBGLS-t was better than that of P-R-EBGLS-t and the performance of P-R-EBGLS-t was better than that of P-R-GLS-t. This means that compared to the restart based framework, the proposed PEB framework can further improve the performance of parallel GLS on these test instances. In summary, the experimental results show the effectiveness of the proposed PEB framework.

6. Conclusion

Parallel metaheuristics can exploit the potential computation power of multi-processor systems. This paper proposes the Parallel Elite Biased framework (PEB framework) to design the parallel variants of trajectory-based metaheuristics. The PEB framework applies a distributed topology and an asynchronous communication strategy. More importantly, the PEB framework employs a new cooperative method, which is different from the widely-used cooperative methods including the restart-based method and the path-relinking method. In the PEB framework, multiple search processes start from different initial solutions. After a predefined period of time, each process communicates with its neighbors to update the set formed by the current historical best solutions found by itself and its neighbors. Then the

28

Table 6: Performance of the five comparison parallel metaheuristics, process number $K = 9$

	PEBGLS-t	P-R-GLS-t	P-R-EBGLS-t	P-ACO	P-ILK
Instance	Average Excess (%)				
rd400	**0.0000**	0.0023	0.0007	**0.0000**	**0.0000**
att532	**0.0000**	0.0031	0.0031	0.0038	**0.0000**
gr666	**0.0003**	0.0173	0.0111	0.0157	0.0022
u724	**0.0000**	0.0562	0.0452	0.0094	0.0012
pr1002	**0.0000**	**0.0000**	**0.0000**	0.0096	**0.0000**
d1291	0.0073	0.0644	0.0775	**0.0021**	0.0120
u1432	**0.0000**	0.2236	0.0918	0.2104	**0.0000**
u1817	**0.0301**	0.2677	0.2753	0.1004	0.0877
pr2392	**0.0011**	0.2376	0.2197	0.1881	0.0020
fnl4461	0.0922	0.3818	0.1873	1.5910	**0.0401**
Instance	Average Runtime (s)				
rd400	**0.10**	2.89	0.95	1.27	0.13
att532	**0.24**	1.36	0.87	6.37	0.57
gr666	**5.25**	9.50	8.93	12.52	5.74
u724	**1.29**	15.00	10.87	9.44	6.76
pr1002	1.62	0.30	1.61	16.52	**0.98**
d1291	**5.33**	14.16	16.36	13.76	6.07
u1432	**3.81**	29.00	27.61	29.00	4.81
u1817	**23.60**	37.00	37.00	37.00	35.61
pr2392	39.22	48.00	48.00	48.00	**19.90**
fnl4461	90.00	90.00	90.00	90.00	90.00

29

process selects the best solution in the set as the elite solution s_e and its search direction will be attracted by s_e.

The PEB framework has successfully been used to design a parallel Guided Local Search (GLS) metaheuristic called Parallel Elite Biased GLS (PEBGLS) for the Traveling Salesman Problem (TSP). We conducted systematic experiments on the Tianhe-2 supercomputer to test the performance of PEBGLS on the TSP. By analyzing the experimental results, we conclude that PEBGLS is a competitive TSP metaheuristic. Hence the proposed PEB framework is useful in designing efficient parallel trajectory-based metaheuristics. Our work provides a new possible way to design parallel trajectory-based metaheuristics.

[1] S. Nesmachnow, An overview of metaheuristics: accurate and efficient methods for optimisation, International Journal of Metaheuristics 3 (4) (2014) 320–347.

[2] G. Jaradat, M. Ayob, I. Almarashdeh, The effect of elite pool in hybrid population-based meta-heuristics for solving combinatorial optimization problems, Applied Soft Computing 44 (2016) 45–56.

[3] C. Voudouris, E. Tsang, Guided local search and its application to the traveling salesman problem, European Journal of Operational Research 113 (2) (1999) 469–499.

[4] D. Karaboga, C. Ozturk, A novel clustering approach: artificial bee colony (ABC) algorithm, Applied Soft Computing 11 (1) (2011) 652–657.

[5] J. Shi, Q. Zhang, B. Derbel, A. Liefooghe, A parallel tabu search for the unconstrained binary quadratic programming problem, in: IEEE Congress on Evolutionary Computation (CEC2017), IEEE, 2017, pp. 557-564.

30

[6] F. Garcia-Lopez, B. Melian-Batista, J. A. Moreno-Perez, J. M. Moreno-Vega, The parallel variable neighborhood search for the p-median problem, Journal of Heuristics 8 (3) (2002) 375–388.

[7] A. Bortfeldt, H. Gehring, D. Mack, A parallel tabu search algorithm for solving the container loading problem, Parallel Computing 29 (5) (2003) 641–662.

[8] A. Attanasio, J.-F. Cordeau, G. Ghiani, G. Laporte, Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem, Parallel Computing 30 (3) (2004) 377–387.

[9] R. Banos, C. Gil, J. Ortega, F. G. Montoya, A parallel multilevel metaheuristic for graph partitioning, Journal of Heuristics 10 (3) (2004) 315–336.

[10] T. G. Crainic, M. Gendreau, P. Hansen, N. Mladenovic, Cooperative parallel variable neighborhood search for the p-median, Journal of Heuristics 10 (3) (2004) 293–314.

[11] J. Blazewicz, A. Moret-Salvador, R. Walkowiak, Parallel tabu search approaches for two-dimensional cutting, Parallel Processing Letters 14 (01) (2004) 23–32.

[12] A. Le Bouthillier, T. G. Crainic, A cooperative parallel meta-heuristic for the vehicle routing problem with time windows, Computers & Operations Research 32 (7) (2005) 1685–1708.

[13] A. Le Bouthillier, T. G. Crainic, P. Kropf, A guided cooperative search for the vehicle routing problem with time windows, IEEE Intelligent Systems 20 (4) (2005) 36–42.

[14] E.-G. Talbi, V. Bachelet, Cosearch: A parallel cooperative meta-heuristic, Journal of Mathematical Modelling and Algorithms 5 (1) (2006) 5–22.

[15] T. Fischer, P. Merz, A distributed chained Lin-Kernighan algorithm for TSP problems, in: 19th IEEE International Parallel and Distributed Processing Symposium, IEEE, 2005, pp. 1–10.

31

[16] T. Fischer, P. Merz, Embedding a chained Lin-Kernighan algorithm into a distributed algorithm, in: Metaheuristics, Springer, 2007, pp. 277–295.

[17] S. Lukasik, Z. Kokosinski, G. Swieton, Parallel simulated annealing algorithm for graph coloring problem, in: International Conference on Parallel Processing and Applied Mathematics, Springer, 2007, pp. 229–238.

[18] C. C. Ribeiro, I. Rosseti, Efficient parallel cooperative implementations of GRASP heuristics, Parallel Computing 33 (1) (2007) 21–35.

[19] A. P. Araujo, C. Boeres, V. E. Rebello, C. C. Ribeiro, S. Urrutia, Exploring grid implementations of parallel cooperative metaheuristics, in: Metaheuristics, Springer, 2007, pp. 297–322.

[20] M. E. Aydin, M. Sevkli, Sequential and parallel variable neighborhood search algorithms for job shop scheduling, in: Metaheuristics for Scheduling in Industrial and Manufacturing Applications, Springer, 2008, pp. 125–144.

[21] M. Polacek, S. Benkner, K. F. Doerner, R. F. Hartl, A cooperative and adaptive variable neighborhood search for the multi depot vehicle routing problem with time windows, BuR-Business Research 1 (2) (2008) 207–218.

[22] J. P. Q. dos Santos, F. C. de Lima, R. M. Magalhaes, J. D. de Melo, A. D. D. Neto, A parallel hybrid implementation using genetic algorithm, GRASP and reinforcement learning, in: 2009 International Joint Conference on Neural Networks, IEEE, 2009, pp. 2798–2803.

[23] G. Luque, F. Luna, E. Alba, A new parallel cooperative model for trajectory based metaheuristics, in: Distributed Computing and Artificial Intelligence, Springer, 2010, pp. 559–567.

[24] G. Luque, F. Luna, E. Alba, S. Nesmachnow, Exploring the accuracy of a parallel cooperative model for trajectory-based metaheuristics, in: International Conference on Computer Aided Systems Theory, Springer, 2011, pp. 319–326.

[25] A. Subramanian, L. M. d. A. Drummond, C. Bentes, L. S. Ochi, R. Farias, A parallel heuristic for the vehicle routing problem with

32

simultaneous pickup and delivery, Computers & Operations Research 37 (11) (2010) 1899–1911.

[26] Y.-F. Hung, W.-C. Chen, A heterogeneous cooperative parallel search of branch-and-bound method and tabu search algorithm, Journal of Global Optimization 51 (1) (2011) 133–148.

[27] J.-F. Cordeau, M. Maischberger, A parallel iterated tabu search heuristic for vehicle routing problems, Computers & Operations Research 39 (9) (2012) 2033–2050.

[28] Y. C. Lee, J. Taheri, A. Y. Zomaya, A parallel metaheuristic framework based on harmony search for scheduling in distributed computing systems, International Journal of Foundations of Computer Science 23 (02) (2012) 445–464.

[29] J. Jin, T. G. Crainic, A. Løkketangen, A cooperative parallel metaheuristic for the capacitated vehicle routing problem, Computers & Operations Research 44 (2014) 33–41.

[30] V. C. Hemmelmayr, Sequential and parallel large neighborhood search algorithms for the periodic location routing problem, European Journal of Operational Research 243 (1) (2015) 52–60.

[31] S. Iturriaga, S. Nesmachnow, F. Luna, E. Alba, A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing systems, The Journal of Supercomputing 71 (2) (2015) 648–672.

[32] N. Lahrichi, T. G. Crainic, M. Gendreau, W. Rei, G. C. Crişan, T. Vidal, An integrative cooperative search framework for multi-decision-attribute combinatorial optimization: Application to the MDPVRP, European Journal of Operational Research 246 (2) (2015) 400–412.

[33] G. Luque, E. Alba, Parallel hybrid trajectory based metaheuristics for real-world problems, in: International Conference on Intelligent Networking and Collaborative Systems (INCOS2015), IEEE, 2015, pp. 184–191.

33

[34] U. Tosun, On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem, Engineering Applications of Artificial Intelligence 39 (2015) 267–278.

[35] C. Wang, D. Mu, F. Zhao, J. W. Sutherland, A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows, Computers & Industrial Engineering 83 (2015) 111–122.

[36] G. F. Sousa Filho, T. L. B. Junior, L. d. A. F. Cabral, L. S. Ochi, F. Protti, A parallel hybrid metaheuristic for bicluster editing, International Transactions in Operational Research 23 (3) (2016) 409–431.

[37] L. Guzman, E. N. Ruiz, C. Ardila, D. Jabba, W. Nieto, A novel framework for the parallel solution of combinatorial problems implementing tabu search and simulated annealing algorithms, in: 6th International Conference on Computers Communications and Control (ICCCC2016), IEEE, 2016, pp. 259–263.

[38] Z. Quan, L. Wu, Design and evaluation of a parallel neighbor algorithm for the disjunctively constrained knapsack problem, Concurrency and Computation: Practice and Experience 29 (20), 2017.

[39] W. Tu, Q. Li, Q. Li, J. Zhu, B. Zhou, B. Chen, A spatial parallel heuristic approach for solving very large-scale vehicle routing problems, Transactions in GIS, 2017, pp. 1130–1147.

[40] E. Alba, Parallel metaheuristics: a new class of algorithms, Vol. 47, John Wiley & Sons, 2005.

[41] T. G. Crainic, M. Toulouse, Parallel meta-heuristics, in: Handbook of metaheuristics, Springer, 2010, pp. 497–541.

[42] N. Tairan, Q. Zhang, P-GLS-II: an enhanced version of the population-based guided local search, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, ACM, 2011, pp. 537–544.

[43] J. J. Bentley, Fast algorithms for geometric traveling salesman problems, ORSA Journal on Computing 4 (4) (1992) 387–411.

34

[44] G. Reinelt, TSPLIB-a traveling salesman problem library, ORSA Journal on Computing 3 (4) (1991) 376–384.

[45] K. D. Boese, Cost versus distance in the traveling salesman problem, UCLA Computer Science Department, 1995.

[46] S. M. Oliveira, M. S. Hussin, T. Stützle, A. Roli, M. Dorigo, A detailed analysis of the population-based ant colony optimization algorithm for the TSP and the QAP, in: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, ACM, 2011, pp. 13–14.

35