A New Cooperative Framework for Parallel Trajectory-Based Metaheuristics

Jialong Shi\textsuperscript{a}, Qingfu Zhang\textsuperscript{a}

\textsuperscript{a}Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong

Abstract

In this paper, we propose a framework for parallel trajectory-based metaheuristics. In our framework, multiple processes search the solution space concurrently and send their best found solutions to each other on a distributed topology. More importantly, instead of restarting from the received solutions or executing path-relinking on them (which are commonly seen in other parallel trajectory-based metaheuristics), each process in our framework continues searching from its current solution, but its search direction has bias toward the received solutions. In such a way, historical information of each process is used efficiently. To illustrate the effectiveness of our framework, we design a parallel variant of Guided Local Search (GLS), a classical trajectory-based metaheuristic. Extensive experiments have been conducted on the Tianhe-2 supercomputer to test the performance of the designed parallel GLS algorithm. Based on the experimental results, we conclude that the proposed parallel framework is effective and comparable to other widely-used parallel frameworks. Our work represents a new direction to design 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 with a reasonable amount of time. There are two

\textit{Email addresses:} jishi2-c@my.cityu.edu.hk (Jialong Shi), qingfu.zhang@cityu.edu.hk (Qingfu Zhang)
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. Most trajectory-based metaheuristics are based on Local Search (LS) and they employ different strategies to escape from local optima in order to explore the solution space. Examples of trajectory-based metaheuristics include Simulated Annealing (SA) [3], Tabu Search (TS) [4] and Guided Local Search (GLS) [5,6]. 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 Algorithms (GA) [7], Ant Colony Optimization (ACO) [8], Artificial Bee Colony (ABC) [9] and Particle Swarm Optimization (PSO) [10] are 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. By now, most parallel metaheuristics are population-based metaheuristics. Not much effort has been done on parallel trajectory-based metaheuristics [11]. In this paper, we propose a parallel framework which can be used to design efficient parallel variants of traditional trajectory-based metaheuristics.

To design a parallel trajectory-based metaheuristic with multiple search processes, three issues must be addressed: (i) What information should be exchanged among different processes? (ii) How the processes exchange the information? (iii) How the processes utilize the received information? On the first issue, our framework selects the best found solutions as the information exchanged among different processes. On the second issue, our framework lets different processes only send solutions to their predefined neighbors in a distributed topology. The main contribution of this paper is reflected on how our framework deals with the third issue, i.e., how the processes utilize the received solutions. In the related works, after receiving a new solution, a process normally restarts from the received solution, or executes path-relinking on its current solution and the received solution. In our framework, the process continues searching from its current solution, but its search direction will have bias toward the received solutions. In such a way, the historical information of each process is used efficiently.

Using the proposed framework, we have designed the Parallel Elite Biased Tabu Search (PEBTS) algorithm for the Unconstrained Binary Quadratic Programming (UBQP) problem and showed that the cooperation among the
PEBTS processes can improve the overall efficiency [12]. In this paper, we use the proposed framework to design a parallel variant of GLS, called Parallel Elite Based GLS (PEBGLS). We also conduct extensive experiments to study the performance and behavior of PEBGLS using the symmetric Traveling Salesman Problem (TSP) as the test problem. We do not aim to obtain the best parallel metaheuristic for the TSP. Our objective is to illustrate the effectiveness of the proposed framework in designing efficient parallel variants of trajectory-based metaheuristics. We hope that our study can provide a new possible direction of designing and studying parallel trajectory-based metaheuristics.

This paper is structured as follows. Section 2 reviews some existing works related to the parallel trajectory-based metaheuristics. In Section 5 we describe the proposed parallel framework. Section 4 introduces the sequential GLS algorithm. Section 5 explains how we design the parallel variant of GLS, i.e. PEBGLS, based on the proposed framework. Section 6 gives experimental study. Section 7 concludes the paper.

2. Related Works

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

- **Low-level parallelism** or acceleration strategy [13, 14]. 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 works to the slaves. The works can be evaluating moves, or evaluating partial solutions. After collecting and integrating the results returned by the slaves, the master proceeds the subsequent procedure.

- **High-level parallelism** or multiple search strategy [15, 16, 17]. 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 same or different. They may run independently and communicate at the end to identify the best overall solution, or they may try to enhance the global search efficiency by exchanging information during
the search. In the latter case, the behavior of the parallel metaheuristic is different from the corresponding sequential one.

Researchers in this field put more attention on the high-level parallelism strategy. A number of high-level parallel trajectory-based metaheuristics have been proposed for various problems. To design a high-level parallel trajectory-based metaheuristic, three issues must be addressed: (i) What is the information exchanged among different processes? (ii) How the processes exchange the information? (iii) How the processes utilize the received information? Based on the three issues, Table 2 summarizes the related works on parallel trajectory-based metaheuristics.

From Table 1 we can see that, the most common type of information exchanged among the parallel processes are solutions, which can be the best solutions found so far by the processes or some elite solutions stored in the memory. A few works employ other information types, e.g. the visiting frequency. From Table 1 we can also see that, in most of the works the communication methods are based on centralized topologies. In these works, either a master process is in charge of collecting and delivering information, or there is a central memory which can be accessed by all processes. In the former case, the computation load of the master process usually is not at the same level as the slave processes. The unbalanced computation load may make the parallel metaheuristic unable to fully exploit the computation power of multiple uniform processors. In the latter case, the communication load could be very big due to the exclusiveness of the write operations. Some other works apply distributed communication methods, in which each process only share information with a limited number of processes. If well designed, the distributed communication method can guarantee a balanced communication load and a reasonable communication load. Our proposed framework employs a distributed communication method.

In Table 1 the information utilizing methods in many works are “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 of its own search 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. But the current solution still is abandoned and the process jumps to the newly generated solution. In our parallel framework, we proposed a new method to utilize the received solutions. The process
| Related Work | Algorithm | Problem | Information Type | Information Exchanging Method | Information Utilizing Method |
|--------------|-----------|---------|------------------|-----------------------------|-----------------------------|
| Garcia-Lopez et al. 2002 [18] | Parallel VNS | P-median problem | Best found solutions | Centralized | Restart |
| Bortfeldt et al. 2003 [19] | Parallel TS | Container loading problem | Best found solutions | Distributed | Restart |
| Attanasio et al. 2004 [20] | Parallel TS | Dynamic multi-vehicle dual-vehicle problem | Best found solutions & Visiting frequencies | Distributed | Restart & Refer to the frequencies |
| Bano et al. 2004 [21] | Parallel SA, TS | Graph partitioning | Best found solutions | Distributed | Restart |
| Crama et al. 2004 [22] | Parallel VNS | P-median problem | Best found solutions | Centralized | Restart |
| Blasewicz et al. 2004 [23] | Parallel TS | 2-dimensional cutting | Best found solutions | Centralized | Restart |
| Le Bouthillier and Crainic 2005 [24] | Parallel TS, EA, Post-optimization | VRPTW | Elite solutions | Centralized | Path-relinking |
| Le Bouthillier, et al. 2005 [25] | Parallel TS, EA, Post-optimization & Pattern identification | VRPTW | Elite solutions & Solution attributes | Centralized | Restart & Fix or prohibit the attributes |
| Talbi and Bachelet 2006 [26] | COSEARCH (GA, QGA, Operator & TS) | QAP | Elite solutions & Global frequencies | Centralized | Restart & Refer to the frequencies |
| Fischer and Mezz 2005 [27] | Parallel Channel Lin-Kernighan | TSF | Best found solutions | Distributed | Restart |
| Lukasik et al. 2007 [28] | Parallel SA | Graph coloring problem | Best found solutions | Centralized | Restart |
| Rubner and Rosseti 2007 [29] | Parallel GRASP | 2-path network design problem | Elite solutions | Centralized | Path-relinking |
| Argoto et al. 2007 [30] | Parallel GRASP-ILS | Minimum spanning tournament problem | Elite solutions | Centralized | Restart |
| Aydin and Sevki 2008 [31] | Parallel VNS | Job shop scheduling | Best found solutions | Distributed | Restart |
| Polacek et al. 2008 [32] | Parallel VNS | MDVRPTW | Best found solutions | Centralized | Restart |
| dos Santos et al. 2009 [33] | Parallel GRASP, GA & Q-learning | TSF | Best found solutions | Centralized | Restart & Update Q-values table |
| Lopes et al. 2010 [34] | Parallel SA | DNA fragment assembly, MAXSAT & RO | Best found solutions | Distributed | Combination operation |
| Subramaniam et al. 2010 [35] | ILS-RND | VRSPD | Best parameter values | Centralized | Set the parameters |
| Hing and Chen 2011 [36] | Parallel Branch-and-Bound method & TS | TSF | Best found solutions | Centralized | Restart |
| Cordes and Maischberger 2012 [37] | Parallel ILS-TS | VRP | Best found solutions | Distributed | Restart with a probability |
| Lee et al. 2012 [38] | Harmony search | Task scheduling problem | Elite solutions | Centralized | Restart |
| Lopes and Alba 2015 [39] | Parallel SA | DNA fragment assembly & QAP | Best found solutions | Distributed | Path-relinking |
| Sousa Filho et al. 2016 [40] | GRASP-VNS | Block cluster editing problem | Best found solutions | Centralized | Restart |
3. Proposed Parallel Framework

Due to the stochastic nature of the trajectory-based metaheuristics, the time to find a solution within a target value (e.g., the globally optimal value) is a random variable. Assuming that this random variable is exponentially distributed, then the following proposition exists [40, 28]:

**Proposition.** Let $P_k(t)$ be the probability of not having found a target solution value in time $t$ with $k$ independent parallel search processes. If $P_1(t) = e^{-t/\lambda}$ with $\lambda \in \mathbb{R}^+$ (exponential distribution), then $P_k(t) = e^{-kt/\lambda}$.

This proposition is derived from the property of the exponential distribution. The probability of a single search process finding a solution within a target solution value in time $kt$ is $1 - e^{-kt/\lambda}$, which equals to the probability of $k$ independent parallel search processes finding a solution with the same quality in time $t$. It implies that a linear speedup is possible to be achieved by parallel metaheuristics. Here the cooperation of search processes is not involved, so the speedup of the parallel metaheuristics can be further improved if the processes cooperate with each other. Of course, this is just a rough analysis of the speedup in parallel metaheuristics. But there is no doubt that parallelism can improve the effectiveness and robustness of metaheuristics.

In this paper we proposed a parallel framework to design parallel trajectory-based metaheuristics. In the proposed parallel framework, multiple processes shares their best found solutions with each other by a distributed communication method. More importantly, the proposed parallel framework employs a new method to utilize the information shared among processes. By using the proposed parallel framework, one can easily design effective parallel implementations of trajectory-based metaheuristics. The proposed parallel framework has been used successfully to design an parallel tabu search algorithm called PEBTS [12].

3.1. Communication Method

The proposed parallel framework is based on a distributed topology. There is no master process or central memory. To reduce the communication load, each process only communicates with a limited number of processes, i.e., its neighbors. In this paper we consider 2 distributed topologies: the bidirectional ring topology and the torus topology. Both topologies support...
flexible process number. Fig. 1 illustrates the examples of these parallel topologies. In the bidirectional ring topology (Fig. 1(a)), each process has 2 neighbors. For example, the neighbors of process 1 are process 2 and process 8. In the torus topology (Fig. 1(b)), each process has 4 neighbors. For example, the neighbors of process 1 are process 2, process 5, process 4 and process 13.

The communication load is a very important issue in a parallel meta-heuristic. A rigidly synchronous communication strategy may cause heavy communication load and reduce the efficiency of the algorithm. In our proposed parallel framework, communications are based on message passing and the messages are solutions. For each search process, we denote $s_{hb}$ as the historical best solution found by itself and $S_r$ as the set of solutions received from its neighbors. Each member in $S_r$ corresponds to a neighbor and it always is the newest solution received from that neighbor. After a given period of time, each process checks whether $s_{hb}$ has changed since the previous sending. If so, it will send the new $s_{hb}$ to its neighbors. Meanwhile each process always prepares to receive new solutions and updates $S_r$ by the newly received solutions. Note that, although a process may receive better solutions from its neighbors, it still sends the best solution found by itself to its neighbors. This keeps the diversity of processes. When the target solution value is achieved by a process, this process will broadcast a stop message to all the other processes. We can see that, our proposed parallel framework
does not require the processes to be synchronous. The communication only happens when a sending request occurs. It does not require the sender and the receiver to be in the same iteration. Hence in the proposed parallel framework, different search processes may have passed different iterations when the communication happens. This strategy is helpful to reduce the communication load among processes.

3.2. Information Utilizing Method

The parallel framework proposed in this paper is based on solution exchanging. Recall that, for a search process, \( s_{hb} \) is the historical best solution found by itself and \( S_r \) is the set of solutions received from its neighbors. We call the best solution in the set \( S_r \cup \{s_{hb}\} \) as the elite solution \( s_e \). To utilize the information in \( s_e \), in our framework, the search direction of each process always have a bias toward \( s_e \). Hence we call our information utilizing method as the Elite-Biased method. Fig. 2 illustrates the sketch of the elite-biased method. For comparison, Fig. 2 also shows the independent case and the other 2 commonly used information utilizing methods: the restart method and the path-relinking method. In Fig. 2 process A and process B start from different solutions and performs different trajectories. The circles in Fig. 2 represent the solutions found by A and B in each move (iteration) and the arrows represent the search directions of A and B. In the last move, process A finds a high-quality solution (the red filled circle in Fig. 2) which is better than the current overall best solution. In the independent case (Fig. 2(a)), the trajectory of process B is not influenced by the new best solution found by process A, because there is no communication between A and B. In the restart method (Fig. 2(b)), after process A sends the newfound best solution to process B, process B restarts from the received solution. In the path-relinking method (Fig. 2(c)), process B constructs a new solution using the path-relinking operator based on the received solution and its current solution, then it proceeds the search from the resulting solution. In the elite-biased method (Fig. 2(d)) proposed by this paper, process B continues searching from its current solution, but the search direction is attracted by the newfound best solution of process A. Note here that, Fig. 2 only shows the sketches of the search processes, in fact the solution space of a combinatorial optimization problem is very complicated and hard to be visualized.

Compared to the restart method, the proposed elite-biased method does not give up the search region near to its current solution. Meanwhile its search direction leads it toward the regions near to the received solutions,
Figure 2: Different information utilizing methods
hence the information of the received solutions is also utilized. If process B finds a better solution in the future, process A will be attracted by process B too. The elite-biased method can prevent the case when a deceptive high-quality solution leads all processes to a less-promising region in the solution space. It also can maintain the diversity of processes.

Although the path-relinking method and the elite-biased method both let the current solution be “attracted” by the newfound best solution, they are different. In the path-relinking method, the path-relinking phase and the neighborhood descent phase are separated. The attraction of the newfound best solution only influences the path-relinking phase. In contrast, the elite-biased method integrates the attraction of the newfound best solution to the neighborhood descent search. As a consequence, the overall procedure is simplified, and both the external information (the newfound best solution) and the local information (the neighborhood of the current solution) are employed at the same time.

3.3. Pseudocode

In summary, the procedure of each process in the proposed parallel 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. once in a while), 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. In the main procedure, i.e., the EliteBiasedSearch procedure, the original search procedure of the trajectory-based metaheuristic is combined with the elite-biased method. Through this setting, the process continues searching from its current solution $s$, but the search direction is attracted by the elite solution $s_e$.

This parallel framework can be applied to many trajectory-based metaheuristics. In the following of this paper, we use GLS to illustrate the application of our proposed parallel framework. GLS is a trajectory-based metaheuristic which has been successfully applied to many optimization problems [41, 42, 43, 44]. For the parallel variant of GLS, Tairan and Zhang [45] proposed an enhanced GLS algorithm called P-GLS-II. However Tairan first transforms GLS into a population-based metaheuristics, then run it in a parallel way. To our best knowledge, there is no parallel trajectory-based variant of GLS. Following the proposed parallel framework, we design a parallel variant of GLS, which is called Parallel Elite Biased GLS (PEBGLS).
We use the symmetric TSP as the test problem to investigate the performance of PEBGLS. Our objective is not to obtain the best parallel metaheuristic for the TSP, but to illustrate the effectiveness of our parallel framework by showing the efficiency of PEBGLS.

4. Guided Local Search

GLS is a simple yet 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 $g : S \rightarrow \mathbb{R}$ 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. GLS does not use $g(\cdot)$, but the augmented objective function $h(\cdot)$ as the guide function of LS:

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

where $s$ is a candidate solution, $\lambda$ 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 $i$:

$$I_i(s) = \begin{cases} 
1 & \text{if feature } i \text{ is in } s, \\
0 & \text{otherwise}. 
\end{cases}$$

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

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

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 $h(\cdot)$, 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 $\lambda$, the feature set $M$ and the cost of each feature $\{c_i \mid i \in M\}$.

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.

5. Parallelization

The penalties in GLS are always increased, so the search process of GLS does not converge. If we do not set a stopping criterion for GLS, it will execute forever. Usually the stopping criterion of GLS is a maximum runtime or a target solution cost. The time for GLS to find a solution within a target solution cost is a random variable. The variance of this random variable reflects the risk of not finding the target value in a reasonable time. Here we conduct a pilot experiment to observe the distribution of the time for GLS to find the global optima of TSP instance att532 in TSPLIB [46].
Algorithm 2 Guided Local Search

1: **input:** $g, \lambda, M, c$
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** $s_{hb}$
Figure 3: The time for GLS to find the global optimum of TSP instance att532 nearly follows an exponential distribution

independent runs of GLS are executed from different randomly generated solutions and stop only when the globally optimal cost is reached. The CDF of the recorded runtime is shown in Fig. 3, in which an exponential distribution with mean value 507.36 also is illustrated. We can see that in this case the runtime of GLS approximately follows an exponential distribution. According to the proposition in Section 5.1, it is possible to achieve a linear speedup by running multiple GLS processes in parallel. In Figure 3, the standard deviation of the runtime of GLS is 430.45, which is relatively large. Hence running multiple independent GLS processes in parallel can reduce the risk of not finding the target value in a reasonable time. In this paper we will show that, by applying our proposed parallel framework, the efficiency of GLS can be further improved.

5.1. Changing the search direction of GLS

In the proposed parallel 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 direction of the process is attracted by $s_e$. Hence to apply the proposed parallel framework to GLS, one must design a method to change the search direction of GLS.
GLS is a penalization based metaheuristic. It 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 features that belong to \( s_e \), and increase more penalties on the features not belong to \( s_e \). As a result, the search process will be orientated to the search regions near to \( s_e \). Meanwhile \( s_e \) is continuously updated by the newfound better solutions of the search process and its neighbors, which can prevent the search process from stalling. To achieve this aim, we modify the formula of \( util_i \), i.e. formula (3). The new formula is:

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

where \( w > 1 \) is a pre-defined parameter. In (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 on \( s_e \) will become relatively small. Due to the descending nature of the LS, the search direction of GLS will be navigated to the region near to \( s_e \).

5.2. PEBGLS

The parallel variant of GLS we proposed is called PEBGLS, which applies the proposed parallel elite-biased framework. To our best knowledge, PEBGLS is the first multi-start parallel variant of GLS. Here we stipulate that every \( U \) iterations, each PEBGLS process selects the best one of the set \( S_r \cup \{ s_{hb} \} \) as \( s_e \), and sends its \( s_{hb} \) to all neighbors if \( s_{hb} \) has changed since the previous sending. \( U \) is a pre-defined parameter of PEBGLS. The detailed procedure of each PEBGLS process is shown in Algorithm 3. The inputs are: objective function \( g \), the feature set \( M \), the cost of each feature \( \{ c_i | i \in M \} \) and the user-defined parameters \( \{ \lambda, w, U \} \).

6. Experimental Studies

In the experimental studies, we test the performance of PEBGLS on the TSP. Our purpose is not to design a state-of-the-art TSP algorithm, but to use TSP to illustrate that our proposed parallel framework can truly speedup the trajectory-based metaheuristics and find even better solutions through the cooperation among different search processes.
Algorithm 3 Parallel Elite Biased Guided Local Search

1: input: $g, M, c, \lambda, w, U$
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:     $p_i \leftarrow 0$
8: end for
9: while !StoppingCriterion do
10:     $S_r \leftarrow$ TryToReceive()
11:     if $j \% U == 0$ then
12:         $s_e \leftarrow$ SelectBestSolution($S_r \cup s_{hb}$)
13:         if $s_{hb}$ has updated since the previous sending then
14:             SendToNeighbors($s_{hb}$)
15:         end if
16:     end if
17:     $h \leftarrow g + \lambda \sum p_i I_i$
18:     $\{s_{j+1}, s_{hb}\} \leftarrow$ LocalSearch($s_j, s_{hb}, h$)
19:     for $i = 1 \rightarrow |M|$ do
20:         if feature $i$ is in $s_e$ then
21:             $util_i \leftarrow I_i(s_{k+1}) \cdot c_i/(1 + p_i)$
22:         else
23:             $util_i \leftarrow I_i(s_{k+1}) \cdot w \cdot c_i/(1 + p_i)$
24:         end if
25:     end for
26:     for each $i$ such that $util_i$ is maximum do
27:         $p_i \leftarrow p_i + 1$
28:     end for
29:     $j \leftarrow j + 1$
30: end while
31: return $s_{hb}$
6.1. 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:

\[
g(s) = \sum_{e \in s} c_e.
\]

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 on 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 \( I_i(s) = 0 \). The mainstream LS heuristics for TSP are 2-Opt heuristic \([47]\), 3-Opt heuristic \([48]\) and Lin-Kernighan (LK) heuristic \([49]\). All of them are based on edge exchanging. At each step, 2-Opt heuristic (Fig. 4(a)) replaces 2 edges of the current solution by another 2 edges to get a better solution. For 3-Opt heuristic (Fig. 4(b)), the number of exchanged edges is 3. Compared to 2-Opt and 3-Opt, LK heuristic (Fig. 4(c)) is more sophisticated. For LK heuristic, the number of exchanged edges is variable.
6.2. Elite biased strategy in sequential GLS

We state that our elite-biased strategy is also useful in sequential algorithm. We support our statement by studying a modified sequential GLS algorithm: Elite Biased GLS (EBGLS). EBGLS can be seen as a special PEBGLS with only one search process. In the single search process of EBGLS, $s_e$ is only updated by $s_{hb}$ periodically. To verify whether EBGLS can enhance the performance of the original GLS, we conduct a performance comparison experiment. In this experiment, 10 symmetric TSP instances are chosen from TSPLIB. The globally optimal cost of those instances are known, hence we can calculate the excess of a given solution. The excess is defined by:

$$\text{excess} = \frac{\text{solution cost} - \text{globally optimal cost}}{\text{globally optimal cost}} \times 100\%.$$  \hspace{1cm} (6)

On each test instance, 100 runs of GLS and 100 runs of EBGLS start from different randomly generated solutions. A max runtime is set for each run. For the instance with fewer than 10000 cities, the max runtime is 500s CPU time, and the max runtime is 1500s CPU time for the instance with more than 10000 cities. If the globally optimal cost is achieved, algorithms will terminate immediately. Both EBGLS and GLS are implemented in GNU C++ with O2 optimization. The computing platform is two 6-core 2.00GHz Intel Xeon E5-2620 CPUs under CentOS 6.4.

For GLS, 2-Opt heuristic is selected as the LS method. We also use FLS strategy and Bentley’s improvement [50] to enhance the efficiency of 2-Opt heuristic. Based on [51], the coefficient $\lambda$ is calculated by:

$$\lambda = 0.3 \cdot \frac{g(s_{*,1})}{N},$$ \hspace{1cm} (7)

where $s_{*,1}$ is the first local optimum visited by GLS, $N$ is the number of cities of the TSP instance.

For EBGLS, all of the experiment settings regarding to GLS are the same as above. As to $w$, our pilot experiments show that EB-GLS is not very sensitive to $w$. So here we set $w = 2$. $U$ (the updating cycle of $s_e$) is set to be 100. This means that $s_e$ is updated by $s_{hb}$ every 100 iterations.

We use 4 metrics to measure the performances of GLS and EBGLS. The first metric is the success run number, which is the number of runs that successfully achieved the globally optimal cost. The second metric is the
average runtime, which is the average of the 100 runs’ real runtimes. The third metric is the average excess, which is the average of the excesses of the 100 runs’ historical best solutions. The fourth metric is the median excess, which is the median of the excesses of the 100 runs’ historical best solutions. We also perform Mann-Whitney U-test for the excess data of the 2 algorithms. Table 2 shows the comparison results, in which bold font means the better metric value. The last column of Table 2 shows the P-values of the Mann-Whitney U-test, in which the null hypothesis is that the excess data of GLS and EBGLS are sampled from distributions with equal medians.

From Table 2 we can see that EBGLS shows better performance on the first 4 metrics. Besides, if we set the significance level to be 5%, then the Mann-Whitney U-test will reject the null hypothesis on all instances. Considering that EBGLS gets lower median excess on all instances, the Mann-Whitney U-test concludes that EBGLS performs better than GLS on all of the instances. The results of this experiment shows that, the proposed elite-biased method can improve the efficiency of GLS even if there is only one search process. In the following experiment studies, we will show that the proposed elite-biased framework can further enhance the performance of GLS when there are multiple search processes.

6.3. Speedup

The main purpose of applying parallel metaheuristics is to accelerate the algorithm, hence time is an important metric to measure the performance of parallel metaheuristics. Speedup is a widely used metric to compare the runtime of parallel algorithms against the runtime of sequential algorithms. Note that the runtime measured in parallel algorithms is wall-clock time.

There are different definitions of speedup, here we first measure the speedup of PEBGLS against EBGLS, which is denoted by $S^{(1)}_K$. In Section 6.2 we know that EBGLS is a special case of PEBGLS which only has one process. We denote $T_{1,1}$ as the runtime of EBGLS because it executes 1 process on 1 processor. Analogously, we denote $T_{K,K}$ as the runtime of $K$-process PEBGLS on $K$ processors. Then $S^{(1)}_K$ is calculated by:

$$S^{(1)}_K = \frac{E[T_{1,1}]}{E[T_{K,K}]}.$$  

where $E[\cdot]$ is the expectation function.

The second speedup $S^{(2)}_K$ we measure is the orthodox speedup. This metric is the ratio between the runtime of the parallel algorithm on
Table 2: EBGLS compared with GLS on TSPLIB instances

| Instance | Maximum Runtime (s) | Algorithm | Success Run Number | Average Runtime (s) | Average Excess (%) | Median Excess (%) | P-value of Mann-Whitney U-test on Excess |
|----------|---------------------|-----------|--------------------|---------------------|--------------------|------------------|-----------------------------------------|
| u724     | 500                 | GLS       | 18/100             | 457.28              | 0.0108             | 0.0095           | 5.07e-28                                |
|          | EBGLS               |           | 97/100             | 104.36              | 0.0001             | 0.0000           |                                          |
| d1655    | 500                 | GLS       | 0/100              | 500.00              | 0.7730             | 0.2873           | 4.98e-04                                |
|          | EBGLS               |           | 4/100              | 485.15              | 0.7399             | 0.2583           |                                          |
| pcb3038  | 500                 | GLS       | 0/100              | 500.00              | 0.1683             | 0.1674           | 2.25e-33                                |
|          | EBGLS               |           | 0/100              | 500.00              | 0.0558             | 0.0516           |                                          |
| fnl4461  | 500                 | GLS       | 0/100              | 500.00              | 0.2302             | 0.2292           | 2.59e-34                                |
|          | EBGLS               |           | 0/100              | 485.15              | 0.0998             | 0.0983           |                                          |
| rl5915   | 500                 | GLS       | 0/100              | 500.00              | 0.5233             | 0.4971           | 7.34e-10                                |
|          | EBGLS               |           | 0/100              | 500.00              | 0.3797             | 0.3429           |                                          |
| pla7397  | 500                 | GLS       | 0/100              | 500.00              | 0.6349             | 0.4208           | 7.55e-07                                |
|          | EBGLS               |           | 0/100              | 500.00              | 0.3666             | 0.3485           |                                          |
| rl11849  | 1500                | GLS       | 0/100              | 1500.00             | 0.6716             | 0.6760           | 1.07e-03                                |
|          | EBGLS               |           | 0/100              | 1500.00             | 0.6091             | 0.6130           |                                          |
| usa13509 | 1500                | GLS       | 0/100              | 1500.00             | 0.7306             | 0.7339           | 4.94e-27                                |
|          | EBGLS               |           | 0/100              | 1500.00             | 0.5222             | 0.4986           |                                          |
| d15112   | 1500                | GLS       | 0/100              | 1500.00             | 0.7159             | 0.7127           | 2.18e-22                                |
|          | EBGLS               |           | 0/100              | 1500.00             | 0.5865             | 0.5902           |                                          |
| d18512   | 1500                | GLS       | 0/100              | 1500.00             | 0.8462             | 0.8389           | 1.51e-01                                |
|          | EBGLS               |           | 0/100              | 1500.00             | 0.8295             | 0.8154           |                                          |

one processor and the runtime of the same parallel algorithm on multiple processors. We denote $T_{K,1}$ as the runtime of $K$-process PEBGLS on 1 processor. Then $S^{(2)}_K$ is calculated by:

$$S^{(2)}_K = \frac{E[T_{K,1}]}{E[T_{K,K}]}.$$  \hspace{1cm} (9)

For the speedup metrics, if $S_K < K$, we call it a sublinear speedup; if $S_K = K$, we call it a linear speedup; if $S_K > K$, we call it a superlinear speedup. Linear speedup and superlinear speedup are very desirable because they mean that parallelization does not cost extra overhead. The other widely used metrics is efficiency $e_K$, it equals to $S_K/K$. Obviously $e_K \geq 1$ is desirable.

An experiment is conducted on the Tianhe-2 supercomputer to measure the speedups $S^{(1)}_K$ and $S^{(2)}_K$ of PEBGLS. The test instance is the symmetric TSP instance pr1002. 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. For the
PEBGLS processes, we set $w = 2$ and $U = 100$. The other settings are the same as the experiment settings in Section 6.2. Our PEBGLS program is implemented in GNU C++ with O2 optimization. The OpenMPI library is used as the message passing tool. To calculate $E[T_{1,1}]$, 1000 runs of EBGLS is executed on 1 core. The runs starts from randomly generated solutions and terminate only when the globally optimal cost of pr1002 is achieved. The wall-clock time of each run is recorded. To calculate $E[T_{K,1}]$, 1000 runs of $K$-process PEBGLS is executed on one core. All processes simultaneously share this single core. To calculate $E[T_{K,K}]$, 1000 runs of $K$-process PEBGLS is executed on $K$ cores. Each process occupies a core. For PEBGLS with bidirectional ring topology (PEBGLS-br), $K$ separately takes the values $\{8, 16, 24, 32\}$. For PEBGLS with torus topology (PEBGLS-t), $K$ separately takes the values $\{9, 16, 24, 32\}$.

The resulting $S_K^{(1)}$ values and the corresponding efficiency values are shown in Fig. 5. From Fig. 5 we can see that, both PEBGLS-t and PEBGLS-br achieves superlinear speedup when the process number is relatively small. When the process number becomes large, the growth of the speedup value slows down in both variants of PEBGLS. It is because
that, no matter how powerful a search process is, it still needs some time to reach the target solution cost. When the process number becomes larger, the overall runtime is more closer to its minimum limitation and the communication among processes is increasing, so the effect of involving new processes becomes smaller. This phenomenon also appears in other related works\[53, 54, 34, 55, 39, 56\]. In Fig. 5 PEBGLS-t achieves higher $S_K^{(1)}$ value compared to PEBGLS-br, which means that having 4 communication neighbors is better than having 2 communication neighbors in this case. Note here that $S_K^{(1)}$ is the speedup of PEBGLS against EBGLS, and EBGLS is shown to be better than GLS in Section\[6.2\]. So in theory the speedup of PEBGLS against GLS will be higher than the values in Fig. 5.

Fig. 6 shows the second speedup metric: $S_K^{(2)}$ and the corresponding efficiency values. In Fig. 6 we can see that, although there is no superlinear speedup, both variants of PEBGLS show relatively large speedup values. This reflects the effectiveness of running them on multiple processors. Different from Fig. 5 in Fig. 6 the $S_K^{(2)}$ values of PEBGLS-br and that of PEBGLS-t are incomparable, because in the definition of $S_K^{(2)}$ an algorithm is compared to itself. So there is no common comparison object between PEBGLS-br and PEBGLS-t.

### 6.4. The effectiveness of cooperation

In the proposed parallel framework, a new cooperative method called elite-biased method is used. To verify whether this cooperative method can improve the overall solution quality of parallel GLS, we conduct the following experiment. In this experiment PEBGLS-br and PEBGLS-t are compared with Parallel Independent EBGLS processes (P-I-EBGLS). In P-I-EBGLS, multiple EBGLS processes are executed simultaneously, but they do not share information with each other and $s_e$ is only updated by $s_{hb}$ every $U$ iterations. P-I-EBGLS has one advantage: it does not have a communication load. Comparing the performance of cooperative PEBGLS with that of P-I-EBGLS can help us investigate whether involving the communication load of cooperation is worthy in the proposed parallel framework.

The platform of this experiment is Tianhe-2 supercomputer. We select TSP instances rl5915, pla7397 and rl11849 as the test instances. The maximum runtime on different instances are different: \{rl5915: 119s, pla7397: 237s, rl11849: 371s\}. For each algorithm, the process number $K = 48$ and $U$ separately takes the values of \{1, 500, 1000\}. For each TSP instance and each $U$ value, we execute each algorithm 50 runs. The other experiment settings
are the same as the settings in Section 6.2. Fig. 7 shows the boxplot of the best excesses got by each algorithm.

From Fig. 7 we can see that, PEBGLS-br and PEBGLS-t both get lower excess values than P-I-EBGLS in most cases. Then we conduct the Mann-Whitney U-test on the best results got by the algorithms, the resulting P-values are shown in Table 3. Based on Table 3 we can state that, with a 5% significance level, the excess data of PEBGLS-br and that of PEBGLS-t are better than that of P-I-EBGLS. We also can state that PEBGLS-t performs better than PEBGLS-br.

To test whether PEBGLS-t performs better than P-I-EBGLS on other TSP instances, we conduct another experiment in which the test instances are pr1002, pr2392, fnl4461, usa13509, brd14051 and d15112. We set $U = 1$ to exclude the influence of $U$. The other settings are the same as the settings in the previous experiment. The comparison results are shown in Table 4. From the comparison results in Table 4 we can conclude that PEBGLS-t outperforms P-I-EBGLS on 5 of the 6 test instances. On the instance usa13509, the Mann-Whitney U-test conclude that PEBGLS-t performs worse than P-I-EBGLS. But note that in this experiment $U = 1$. From
Figure 7: The best excesses got by P-I-EBGLS, PEBGLS-br and PEBGLS-t, process number $K = 48$
Table 3: The results of the Mann-Whitney U-test on the best excess data got by P-I-EBGLS, PEBGLS-br and PEBGLS-t

| Instance | Algorithm | Lowest Excess (%) | Median Excess (%) | P-value of Mann-Whitney U-test on Excess vs. P-I-EBGLS | P-value of Mann-Whitney U-test on Excess vs. PEBGLS-br | P-value of Mann-Whitney U-test on Excess vs. PEBGLS-t |
|----------|-----------|-------------------|-------------------|-------------------------------------------------------|-------------------------------------------------------|-------------------------------------------------------|
| rl5915   | P-I-EBGLS | 0.1202            | -                 | 7.9185e-16                                            | 3.9606e-06                                           | -                                                     |
|          | PEBGLS-br | 0.0577            | 7.9185e-16        | -                                                     | -                                                     | 3.2096e-06                                           |
|          | PEBGLS-t  | 0.0326            | 3.9606e-17        | 3.2096e-06                                           | -                                                     | -                                                     |
| pla7397  | P-I-EBGLS | 0.1003            | -                 | 1.7491e-16                                            | 1.6330e-17                                           | -                                                     |
|          | PEBGLS-br | 0.0606            | 1.7491e-16        | -                                                     | -                                                     | 6.3765e-05                                           |
|          | PEBGLS-t  | 0.0433            | 1.6330e-17        | 6.3765e-05                                           | -                                                     | -                                                     |
| rl11849  | P-I-EBGLS | 0.4366            | -                 | 9.8014e-15                                            | 6.8828e-15                                           | -                                                     |
|          | PEBGLS-br | 0.3603            | 9.8014e-15        | -                                                     | -                                                     | 7.3017e-04                                           |
|          | PEBGLS-t  | 0.2991            | 6.8828e-15        | 7.3017e-04                                           | -                                                     | -                                                     |

Table 4: PEBGLS-t compared with P-I-EBGLS on TSPLIB instances, process number $K = 48$

| Instance | Maximum Runtime (s) | Algorithm Success Number | Run | Average Runtime (s) | Average Excess (%) | Median Excess (%) | P-value of Mann-Whitney U-test on Excess vs. P-I-EBGLS |
|----------|---------------------|--------------------------|-----|---------------------|--------------------|-------------------|--------------------------------------------------------|
| pr1002   | 21                  | P-I-EBGLS 50/50          | 3.67| 0.0000              | 0.0000             | -                 | -                                                      |
|          |                     | PEBGLS-t 50/50           | 1.08| 0.0000              | 0.0000             | -                 | -                                                      |
| pr2392   | 48                  | P-I-EBGLS 2/50           | 48.00| 0.0043              | 0.0038             | 4.6400e-19        | -                                                      |
|          |                     | PEBGLS-t 50/50           | 18.63| 0.0000              | 0.0000             | -                 | -                                                      |
| fnl4461  | 90                  | P-I-EBGLS 0/50           | 90.00| 0.0764              | 0.0772             | 2.4422e-17        | -                                                      |
|          |                     | PEBGLS-t 0/50            | 90.00| 0.0278              | 0.0277             | -                 | -                                                      |
| usa13509 | 271                 | P-I-EBGLS 0/50           | 271.00| 0.0753              | 0.6802             | 4.9444e-02        | -                                                      |
|          |                     | PEBGLS-t 0/50            | 271.00| 0.6917              | 0.6996             | -                 | -                                                      |
| brd14051 | 282                 | P-I-EBGLS 0/50           | 282.00| 1.9815              | 1.9742             | 7.0661e-18        | -                                                      |
|          |                     | PEBGLS-t 0/50            | 282.00| 1.5126              | 1.5248             | -                 | -                                                      |
| d15112   | 303                 | P-I-EBGLS 0/50           | 303.00| 0.9079              | 0.9097             | 1.8763e-03        | -                                                      |
|          |                     | PEBGLS-t 0/50            | 303.00| 0.8936              | 0.8919             | -                 | -                                                      |

Fig. 7 we know that PEBGLS performs better with relatively large $U$ value. meanwhile the performance of P-I-EBGLS is not sensitive to $U$. So we set $U = 1000$ and execute another 50 runs of PEBGLS-t on usa13509, and we get a 0.0053 median excess, which is better than P-I-EBGLS’s median excess in Table 4 (Mann-Whitney U test P-value = 1.6691e-13).

Based on the above experimental studies we can state that, although PEBGLS-t has communication load, it out-performs P-I-EBGLS. This conclusion reflects the effectiveness of the proposed cooperative method. It has been shown that our parallel framework can further improve the performance of trajectory-based GLS compared to simply running multiple processes in parallel.
Figure 8: Average communication times per 10,000 iterations

6.5. Influence of communication frequency

Every $U$ iterations, a PEBGLS process will send $s_{hb}$ to all neighbors if $s_{hb}$ has changed since the previous sending. Hence when the parameter $U$ increases, the communication frequency will decrease. But the communication frequency is also influenced by the updating frequency of $s_{hb}$, which is uncontrollable. From Fig. 7 we can see that when $U$ increases to 1000, the performance of PEBGLS-br and PEBGLS-t does not become worse. Instead, in most cases their performance becomes better. In the experiment of Fig. 7, we also count the actual communication times among processes. Fig. 8 shows the average communication times per 10,000 iterations. From Fig. 8 we can see that, the communication frequency is relatively low even when $U = 1$. When $U = 500$ and 1000, the communication frequency significantly decreases. But from Fig. 7 we know that the performance does not decreases when the communication frequency decreases. Hence we state that PEBGLS can work with a relatively low communication frequency (e.g. setting $U$ to be hundreds of iterations).

6.6. Internal behavior of PEBGLS-t

The previous experiments show that PEBGLS-t performs better than P-I-EBGLS. In this section we study the cause of the superiority by tracking the internal behavior of PEBGLS-t and P-I-EBGLS. The experimental platform is a computer cluster formed by 8 computers, each computer contains an
Intel Core2 Duo E8500 CPU. Hence this cluster contains 16 cores in total. The operating system of each computer is ubuntu 15.04. Computers are connected by LAN. The test instance is att532.

In our experiment, we first randomly generate $16 \times 1000$ different initial solutions. Then 1000 runs of PEBGLS-t with 16 processes and 1000 runs of P-I-EBGLS with 16 processes are started from the generated solution set. Hence for each PEBGLS-t run, there is a P-I-EBGLS run starts from the same initial solutions. All the runs end when the globally optimal cost of att532 is reached. In the first 1000 iteration, the PEBGLS-t processes do not communicate with each other. The other experiment settings are the same as the settings in Section 6.2. In this experiment, the entire history of each process is recorded.

Fig. 9 shows how the average best excess changes with the iteration number. We can see that, in the first 1000 iterations, the behavior of the PEBGLS-t processes is the same as that of P-I-EBGLS processes, because in the first 1000 iterations we let the PEBGLS-t processes do not communicate with each other. After the PEBGLS-t processes start communicate with each other, the average best excess of PEBGLS-t becomes smaller than that of P-I-EBGLS. It means that the communication among processes can truly improve the solution quality.
GLS is based on penalization. At each iteration, each edge of att532 has a penalty value. In our experiment, we find that all the runs of PEBGLS-t and P-I-EBGLS end in 2 different globally optimal solutions, and the first globally optimal solution only has 2 different edges to the second one. Each globally optimal solution has 532 edges, so there are totally 534 different edges in the 2 globally optimal solutions. Obviously it is undesirable to increase the penalties on the 534 edges that belong to the 2 global optima. For each search process, we define a metric called ratio of undesirable penalties, which is the ratio between the total penalty on those 534 edges over the total penalty on all the edges of att532. Then we calculate the average ratio of undesirable penalties among all processes in all runs. Obviously, everything being equal, a lower ratio value means that the edges in the global optima are less likely to be penalized. Fig. 10 shows how the average ratio of undesirable penalties changes with the iteration number. In Fig. 10, after the PEBGLS-t processes start cooperate with each other, their average ratio value becomes smaller than that of the P-I-EBGLS processes. This means that, sharing elite solutions among processes reduce the probability of penalizing the edges in the global optima. Hence the search processes in PEBGLS-t become more targeted and the probability of finding the global optima is increased. According to the “big valley” structure [57, 58] of the
symmetric TSP, high-quality solutions are more likely to have more common edge with the global optima. Hence by reducing the penalties on the edges of the global optima, the PEBGLS-t processes have more chance to find high-quality solutions so that their efficiency is improved. However, in Fig. 10 we also can see that, the difference between the 2 curves become smaller when the iteration number increases. It is because GLS only increases penalties on the local optima it finds. As illustrated in Fig. 9 the PEBGLS-t processes find 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 optima found by P-I-EBGLS. So the difficulty of PEBGLS-t not penalizing the edges of global optima increases. Hence the difference between the ratio value of PEBGLS-t and the ratio value of P-I-EBGLS becomes smaller as time goes 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. If there exists a moment, at which the current overall best solution is the one found by process $A$, we call process $A$ the best-contributor. In our experiment, each run of PEBGLS-t/P-I-EBGLS has 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. Fig. 11 shows the number of runs which have the same best-contributor number. From Fig. 11 we can see that, PEBGLS-t has a higher best-contributor number than P-I-EBGLS. In fact, the average best-contributor number in PEBGLS-t is 10.56, and in P-I-EBGLS it is 7.74.

In addition, we define the leading time of a process as the sum of the time period in which it is the finder of the current overall best solution of all processes. We define the leading ratio of a process in a run as its leading time divided by the total time of the run. Obviously, we do not want a process to have a too large leading ratio in a run, because this means the performance of other processes is relatively poor. In each run of PEBGLS-t/P-I-EBGLS, we sort the processes from the largest leading ratio to the smallest leading ratio, and denote the sorted processes as \{1st contributor, 2nd contributor, \ldots\}. Then we calculate the average leading ratio on each position among all the runs. Fig. 12 shows the resulting average leading ratio. In Fig. 12 we can see that the leading ratio distribution of PEBGLS-t is more uniform than that of P-I-EBGLS. Compared to P-I-EBGLS, in PEBGLS-t the first 2 contributors have lower leading ratios and the rest contributors have higher
Figure 11: The number of runs that have the same best-contributor number

(a) P-I-EBGLS

(b) PEBGLS-t
leading ratios. So in PEBGLS-t the global search progress is not always leaded by few powerful processes. The other processes also have a relatively high opportunity to lead the global search progress. The results in Fig. 11 and Fig. 12 show that the processes of PEBGLS-t have more chance to find a new globally best solutions compared to the processes of P-I-EBGLS. In other words, the processes of PEBGLS-t are more active.

6.7. Comparison with the restart based parallel framework

Recall in Section 2 that a lot of parallel trajectory-based metaheuristics use a restart based framework, in which each process restarts from the received elite solutions. To compare the restart based parallel framework with the proposed parallel framework, we conduct an experiment in which 3 different parallel GLS algorithms are executed on the Tianhe-2 supercomputer. The first algorithm is PEBGLS-t which applies the proposed parallel framework. We design the second parallel algorithm by applying the restart based parallel framework to GLS. It is called Parallel Restart GLS with torus topology (P-R-GLS-t). P-R-GLS-t follows a torus neighborhood topology and executes multiple GLS processes simultaneously. In each process of P-R-GLS-t, there is no elite solution $s_e$ which navigates the search direction and the penalizing utility $util$ is calculated by the original formula.
Every $U$ iterations, each P-R-GLS-t process exchanges the historical best solution $s_{hb}$ with its neighbors and restarts from the best solution of the set $S_r \cup \{s_{hb}\}$. The third parallel algorithm we design combines the proposed framework and the restart based framework, which is called Parallel Restart EBGLS with torus topology (P-R-EBGLS-t). In P-R-EBGLS-t, a torus neighborhood topology is constructed and multiple EBGLS processes are executed simultaneously. 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 of $S_r \cup \{s_{hb}\}$ as the elite solution $s_e$. It uses the new formula (4) to calculate $util$, so that the search direction can be attracted by $s_e$.

The experiment settings are the same as the settings in Section 6.4, in which the run number of each algorithm is 50 and the process number of each run is 48. For PEBGLS-t, $U \in \{1, 500, 1000\}$. For P-R-GLS-t and P-R-EBGLS-t, $U \in \{500, 1000\}$ because when $U = 1$ the search processes will restart too frequently. Fig. 13 shows the best excesses got by each algorithm. From Fig. 13 we can see that, PEBGLS-t gets the smallest excess data on rl5915 and pla7397. On rl11849, P-R-GLS-t and P-R-EBGLS-t both get smaller excess data compared to PEBGLS-t. To further compare the results, we conduct the Mann-Whitney U-test on the best result of each algorithm on each instance, as shown in Table 5. Based on Table 5, the Mann-Whitney U-test concludes that PEBGLS-t out-performs the other 2 algorithms on rl5915 and pla7397 with a 5% significance level. The Mann-Whitney U-test also conclude that the other 2 restart based algorithms both perform better than PEBGLS-t on rl11849. Based on these conclusions, we state that in some cases our proposed parallel framework can achieve better performance than the restart based framework. Our parallel framework is an effective framework which is comparable to some widely-used parallel frameworks.

7. Conclusion

Parallel metaheuristics can exploit the potential computation power of multi-processor system. This paper proposes a new framework to design the parallel variants of trajectory-based metaheuristics. The proposed parallel framework applies a distributed topology and an asynchronous communication strategy. More importantly, the proposed parallel framework employs a new cooperative method called elite-biased method, which is different from the widely-used cooperative methods including the restart
Figure 13: The best excesses got by PEBGLS-t, P-R-GLS-t and P-R-EBGLS-t, process number $K = 48$
Table 5: The results of the Mann-Whitney U-test on the best excess data got by PEBGLS-t, P-R-GLS-t and P-R-EBGLS-t

| Instance | Algorithm | Lowest Excess (%) | Median Excess (%) vs. PEBGLS-t | Median Excess (%) vs. P-R-GLS-t | Median Excess (%) vs. P-R-EBGLS-t |
|----------|-----------|-------------------|-------------------------------|-------------------------------|-------------------------------|
| rl5915   | PEBGLS-t  | 0.0326            | -                             | 4.1992e-13                   | 7.9029e-13                   |
|          | P-R-GLS-t | 0.1565            | 4.1992e-13                   | -                             | 3.6830e-01                   |
|          | P-R-EBGLS-t | 0.1392          | 7.9029e-13                   | -                             | 3.6830e-01                   |
| pla7397  | PEBGLS-t  | 0.0433            | -                             | 5.2389e-15                   | 9.9193e-16                   |
|          | P-R-GLS-t | 0.1432            | 5.2389e-15                   | -                             | 5.1031e-01                   |
|          | P-R-EBGLS-t | 0.1543          | 9.9193e-16                   | -                             | 5.1031e-01                   |
| rl11849  | PEBGLS-t  | 0.2991            | -                             | 4.3636e-03                   | 2.2583e-04                   |
|          | P-R-GLS-t | 0.2556            | 4.3636e-03                   | -                             | 3.1252e-01                   |
|          | P-R-EBGLS-t | 0.2570          | 2.2583e-04                   | -                             | 3.1252e-01                   |

Based on the results in Table 5, we can observe the following:

- The elite-biased method and the path-relinking method. In the proposed 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 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 proposed parallel framework has been used successfully to design a parallel tabu search algorithm called PEBTS [12]. In this paper, we design a parallel variant of GLS using the proposed parallel framework, called PEBGLS. We conduct systematic experiments on the Tianhe-2 supercomputer to test the performance of PEBGLS on symmetric TSP. By analyzing the experimental results, we conclude that:

- The elite-biased method can improve the performance of sequential GLS.
- PEBGLS can achieve a relatively high speedup compared to its sequential version. In some cases the speedup is superlinear.
- In PEBGLS, the elite-biased cooperative method improves the overall solution quality compared to the parallel case in which the processes do not cooperate with each other.
- PEBGLS can work with a relatively low communication frequency.
- The search processes in PEBGLS are more targeted and active compared to the independent processes. The number of overall useful processes is also higher.
In some cases, PEBGLS out-performs the parallel GLS algorithms which use the restart based cooperative method.

According to the above results, we state that our proposed parallel framework is an effective framework just like the other widely-used frameworks in literature. Our work provides a new possible way to design the parallel trajectory-based metaheuristics.

Acknowledgments. The work described in this paper was supported by a grant from ANR/RCC Joint Research Scheme sponsored by the Research Grants Council of the Hong Kong Special Administrative Region, China and France National Research Agency (Project No. A-CityU101/16).

References

[1] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys (CSUR) 35 (3) (2003) 268–308.

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

[3] E. Aarts, J. Korst, W. Michiels, Simulated annealing, in: Search methodologies, Springer, 2005, pp. 187–210.

[4] F. Glover, M. Laguna, Tabu search, Springer, 1999.

[5] C. Voudouris, Guided local search for combinatorial optimisation problems., Ph.D. thesis, University of Essex (1997).

[6] C. Voudouris, E. P. Tsang, Guided local search, Springer, 2003.

[7] R. L. Haupt, S. E. Haupt, Practical genetic algorithms, John Wiley & Sons, 2004.

[8] M. Dorigo, M. Birattari, Ant colony optimization, in: Encyclopedia of machine learning, Springer, 2010, pp. 36–39.

[9] D. Karaboga, C. Ozturk, A novel clustering approach: Artificial bee colony (abc) algorithm, Applied soft computing 11 (1) (2011) 652–657.
[10] R. Poli, J. Kennedy, T. Blackwell, Particle swarm optimization, Swarm intelligence 1 (1) (2007) 33–57.

[11] E. Alba, G. Luque, S. Nesmachnow, Parallel metaheuristics: recent advances and new trends, International Transactions in Operational Research 20 (1) (2013) 1–48.

[12] J. Shi, Q. Zhang, B. Derbel, A. Liewooghe, A parallel tabu search for the unconstrained binary quadratic programming problem, in: Evolutionary Computation (CEC), 2017 IEEE Congress on, IEEE, 2017.

[13] B.-L. Garcia, J.-Y. Potvin, J.-M. Rousseau, A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints, Computers & Operations Research 21 (9) (1994) 1025–1033.

[14] S. C. Porto, C. C. Ribeiro, Parallel tabu search message-passing synchronous strategies for task scheduling under precedence constraints, Journal of heuristics 1 (2) (1996) 207–223.

[15] 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.

[16] 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.

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

[18] 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.

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

[20] 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.
[21] R. Banos, C. Gil, J. Ortega, F. G. Montoya, A parallel multilevel metaheuristic for graph partitioning, Journal of Heuristics 10 (3) (2004) 315–336.

[22] 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.

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

[24] 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.

[25] 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. 16b–16b.

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

[27] 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.

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

[29] 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.

[30] 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.
[31] 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.

[32] 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.

[33] 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.

[34] A. Subramanian, L. M. d. A. Drummond, C. Bentes, L. S. Ochi, R. Farias, A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery, Computers & Operations Research 37 (11) (2010) 1899–1911.

[35] 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.

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

[37] 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.

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

[39] 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.

[40] M. G. A. Verhoeven, E. H. Aarts, Parallel local search, Journal of Heuristics 1 (1) (1995) 43–65.

[41] R. Bai, G. Kendall, R. Qu, J. A. Atkin, Tabu assisted guided local search approaches for freight service network design, Information Sciences 189 (2012) 266–281.

[42] A. Alhindi, Q. Zhang, Moea/d with guided local search: Some preliminary experimental results, in: Computer Science and Electronic Engineering Conference (CEEC), 2013 5th, IEEE, 2013, pp. 109–114.

[43] A. R. Shaghaghi, T. Glover, M. Kampouridis, E. Tsang, Guided local search for optimal gpon/fttp network design, in: Computer Networks & Communications (NetCom), Springer, 2013, pp. 255–263.

[44] S. Cramer, M. Kampouridis, Optimising the deployment of fibre optics using guided local search, in: Evolutionary Computation (CEC), 2015 IEEE Congress on, IEEE, 2015, pp. 799–806.

[45] 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.

[46] G. Reinelt, Tsplib-a traveling salesman problem library, ORSA journal on computing 3 (4) (1991) 376–384.

[47] G. A. Croes, A method for solving traveling-salesman problems, Operations research 6 (6) (1958) 791–812.

[48] S. Lin, Computer solutions of the traveling salesman problem, Bell System Technical Journal, The 44 (10) (1965) 2245–2269.

[49] S. Lin, B. W. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Operations research 21 (2) (1973) 498–516.

[50] J. J. Bentley, Fast algorithms for geometric traveling salesman problems, ORSA Journal on computing 4 (4) (1992) 387–411.
[51] 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.

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

[53] H. Sanvicente-Sánchez, J. Frausto-Solís, Mpsa: A methodology to parallelize simulated annealing and its application to the traveling salesman problem, in: Mexican International Conference on Artificial Intelligence, Springer, 2002, pp. 89–97.

[54] W. Bożejko, M. Uchorński, M. Wodecki, Parallel hybrid metaheuristics for the flexible job shop problem, Computers & Industrial Engineering 59 (2) (2010) 323–333.

[55] K. Rocki, R. Suda, Large scale parallel iterated local search algorithm for solving traveling salesman problem, in: Proceedings of the 2012 Symposium on High Performance Computing, Society for Computer Simulation International, 2012, p. 8.

[56] R. Bazylevych, B. Kuz, R. Kutelmakh, R. Dupas, B. Prasad, Y. Haxhimusa, L. Bazylevych, A parallel ring method for solving a large-scale traveling salesman problem, International Journal of Information Technology and Computer Science (IJITCS) 8 (5) (2016) 1.

[57] K. D. Boese, A. B. Kahng, S. Muddu, A new adaptive multi-start technique for combinatorial global optimizations, Operations Research Letters 16 (2) (1994) 101–113.

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