P-GLS-II: An Enhanced Version of the Population-based Guided Local Search

Nasser Tairan
University of Essex
School of Computer Science and Electronic Engineering
Wivenhoe Park, Colchester, CO4 3SQ, U.K.
nmtair@essex.ac.uk

Qingfu Zhang
University of Essex
School of Computer Science and Electronic Engineering
Wivenhoe Park, Colchester, CO4 3SQ, U.K.
qzhang@essex.ac.uk

ABSTRACT

We have recently proposed a Population-based Guided Local Search (P-GLS) framework for solving difficult combinatorial optimization problems. In P-GLS, several agents of guided local search (GLS) procedures are run in a parallel way. These agents exchange information acquired from their previous search to make their further search more rational. We suggested based on the well-known proximate optimality principle (POP) that the shared features between the current agents' local optimal solutions are more likely to be part of the best solution to the problem; therefore these features should not be penalized. However, sometimes some of these common features may not exhibit in a global optimal solution. In this paper, a new framework is proposed to improve the performance as well as overcome the limitations in P-GLS. It applies two new different penalization strategies that increase favouring common features based on their occurrences in the agents' local optimal solutions during the search. The performance of the new algorithm, examined on the Traveling Salesman Problem (TSP), is investigated and evaluated in terms of solution quality and the speed. The experimental results demonstrate that the new algorithm outperforms the parallel GLS algorithm without collaboration and other state-of-the-art algorithms.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search Heuristics

General Terms

Algorithms

1. INTRODUCTION

Over the past few decades, metaheuristics have proven to be successful for dealing with difficult optimization problems. These techniques can obtain high quality solutions within reasonable computational time for many hard problems.

Population-based algorithms, including evolutionary algorithms have shown a good performance in solving many optimization problems [8, 6, 5]. One of the reasons behind their success is the cooperation among their individuals (agents) during the search.

Single-point based iterated metaheuristics, such as Simulated Annealing [12], Tabu Search [7] and Guided Local Search [16], have been well developed. We believe that using cooperative agents equipped with advance single point meta-heuristic is a natural way for designing efficient optimization methods. Some effort has been made along this line [10, 3, 9, 4, 2, 1], but it is still in its very infancy.

Guided Local Search (GLS) is a simple single-point metaheuristic for optimization and has many successful applications [16]. GLS has been applied to TSP and compared with other well known or TSP-specific metaheuristic techniques such as Tabu search, Simulated Annealing, Genetic Algorithm and Iterated Lin-Kernighan. It showed a superior performance in comparison with other methods [15]. Very recently, we have proposed a cooperative algorithm based on GLS, called Population-based Guided Local Search (P-GLS) [13]. P-GLS runs several GLS procedures (agents) in a parallel way. The agents exchange their information periodically during the search to make the search effective. In this paper, we propose a new version of P-GLS, called P-GLS-II to overcome the limitations as well as enhance the performance in P-GLS. It adopts two new different penalty strategies for implementing cooperation among different agents. It has been tested on the traveling salesman problem (TSP) test instances.

This paper is organized as follows. Section 2 explains GLS and how it can be applied to TSP. Section 3 reviews the previous algorithm, population based guided local search (P-GLS). Section 4 describes, in details, the new algorithm P-GLS-II. The experimental results and the discussion are presented in sections 5. Section 6 concludes the work.

2. GUIDED LOCAL SEARCH

GLS is a general meta-heuristic framework for solving optimization problems. It has a simple penalty-based approach for helping the local search (LS) to escape from local optima [16]. The basic idea is to use a new guided function to guide the LS procedure to explore the solution space. This new function is a result of augmenting the given objective function by adding penalties to selected features of solutions.

GECCO'11, July 12–16, 2011, Dublin, Ireland.

Therefore, before applying GLS, one needs to define features for the given problem. These features are used through the search to distinguish between the solutions. Every feature in a solution is assigned a penalty or a weight to indicate its attractiveness.

In the GLS framework, the new guiding function is defined as follows:

$$h(s) = cost(s) + \lambda \times \sum p_k \times I_k(s) \quad (1)$$

where s is a candidate solution, $cost(s)$ is the main objective function. λ is a control parameter, k ranges over all the features , and p_k is the penalty for $feature_k$. $I_k(s)$ is 1 if s has $feature_k$, otherwise it is zero.

The utility of $feature_k$ is defined as follows:

$$u_k = I_k(s^*) \times \frac{c_k}{1 + p_k} \quad (2)$$

where c_k is the cost of $feature_k$. When the LS algorithm traps in a local optimum s^*, for every $feature_k$ exhibited in s^* with the greatest utility value, GLS increase its penalty:

$$p_k = p_k + 1 \quad (3)$$

Then the LS algorithm is restarted from s^* with the new guided function. In such a way, GLS can iterate its LS algorithm as many times as one likes.

It is clear from (2) that the penalty of p_k is more likely to be increased if $feature_k$ has a higher cost. Moreover, p_k has few chances to be increased if $feature_k$ has been penalized for many times. The idea behind it is that a feature might be part of a best solution if it appears in many local optima.

2.1 GLS for the Traveling Salesman Problem

The Traveling salesman problem (TSP) is one of well-known NP-complete combinatorial optimization problems. Given a set of cities and a distance between any two cities, the goal of the TSP is to find the shortest tour for visiting all cities once and returning to the starting city. There are many variations of the Traveling Salesman Problem. In this paper we consider the classic symmetric TSP, in which the distance from city i to city j is equal to the distance from city j to i. Therefore, a tour is undirected.

In order to apply the GLS to TSP the following components are used in our work:

2.1.1 Local Search:

Fast local search (FLS) under the 2-opt neighbourhood [16] is used as the LS algorithm in our method. The idea is to divide the neighbourhood of the current solution into several small sub-neighbourhoods with activation bits associated with everyone. In the case of n cities, the 2-opt neighbourhood is divided into n sub-neighbourhoods. At the beginning of the search, all the sub-neighbourhoods will be activated by assigning 1 to their activation bits. Then, those sub-neighbourhoods that do not cause any improvement will be deactivated by flipping their activation bits to 0. These inactive sub-neighbourhoods will not be considered in the later search [15]. More information about FLS can be found in [15].

2.1.2 Features and their Costs:

A feature in the TSP can be defined as a link between two cities [15]. Therefore, there are $n \times (n-1)$ features in the TSP. A candidate solution (i.e tour) can exhibit n features. The cost c_k for every $feature_k$ is its length. The penalty p_k for each $feature_k$ is assigned 0 at the beginning of the search.

3. POPULATION-BASED GUIDED LOCAL SEARCH (P-GLS)

P-GLS is a population-based guided local search method [13] which applies the proximate optimality principle (POP) via a simple cooperation mechanism.

POP is a well-known principle in meta-heuristics which assumes good solutions should have similar structures [7]. Based on this principle, we suggest in P-GLS that the shared features which appear in all agents' local optimal solutions should be considered good features and not be penalized during the next search stage.

In P-GLS, several GLS procedures (agents) are run in a parallel way. After a number of LS calls, these agents exchange information obtained during previous search to speed up their search and to move to promising search regions.

P-GLS with I GLS agents works as follows:

Step 0 Initialization: For $i = 1$ to I,

1. Agent i randomly selects a solution as its current solution.
2. Agent i sets its penalty to $feature_k$, $p_k^i = 0$

Step 1 GLS: For $i = 1$ to I, each agent i runs GLS, starting from its current solution with its feature penalty values for a number of iterations.

Step 2 Stopping Condition: If a present stopping condition is met, stop and output the best solution obtained by these agents.

Step 3 Information Collection: The manager agent collects the current optimal solutions of all the agents and finds all the common features in these solutions.

Step 4 Behaviour Adjustment: For $i = 1$ to I, agent i resets its utility of $feature_k$ to zero if k is a common feature found by the manager in Step 3. Then go to Step 1.

P-GLS had shown a good performance in terms of the quality of solution and speedy convergence compared to the standard GLS and parallel independent GLS algorithms. The obtained results show that the cooperation between GLS agents can speed the search and make the further search more concentrated. It also helps to achieve a high rate of convergence to promising parts of the search space with less effort.

Although P-GLS has achieved good results, it suffers from some drawbacks that need to be investigated. These disadvantages are listed as follows:

1. Preserving the common features by not penalizing them sometimes makes the search traps in a local optimum of low quality. This is because some common features may not exhibit in a global optimal solution.
2. Tuning the points for exchanging information is a problem dependant and need to be carefully adjusted to avoid premature convergence or, being trapped in local optima.

3. The agents may acquire good information at early stages of the search that helps the search to early converge to the global solutions before reaching the predetermined exchanging information points.

4. P-GLS-II

4.1 General Framework

In this section, we turn our discussion to how P-GLS algorithm is improved and extended to overcome the above limitations. So, we propose a substantial improvement of P-GLS which applies new different penalization strategies that work by increasing the favor of the common features based on their occurrences in the agents local optimal solutions during the search. For distinguishing, we called this new enhanced version P-GLS-II. The general idea is to do a wide search at the beginning of the search and then directs the further search toward promising directions. A new utility function for achieving that is devised and incorporated in P-GLS-II agents as follows:

$$u_k = I_k(s^*) \times \frac{c_k}{(1+p_k)} \times w_k \quad (4)$$

Where, w_k is a parameter $\in [0,1)$ and used to indicate the amount of influence of $feature_k$ on its utility u_k. w_k can be statically adjusted in advance before the beginning of the search or dynamically tuned during the search. It is used to control the speed of the convergence rate to the final solution. More precisely, w_k determines how fast P-GLS-II converges. An extreme case when $w_k = 0$; in this case the common features will not be penalized. On the other hand, when $w_k = 1$ the common features will be considered as normal features and P-GLS-II will behave like the parallel independent GLS. In general, the bottleneck in designing an efficient metaheuretsic algorithm is to balance between the speedy convergence and the diversity. In most cases, one of these two factors is sacrificed. In P-GLS-II, w_k plays a very important role in trying to balance these two factors during the search. More information about adjusting w_k in the next sub-sections.

In P-GLS-II, All the GLS agents run a local search; when all the agents reach their local optima. Then, each agent uses information, the locations of local optima, from other agents to adjust its search behaviour. Apart from the running GLS agents in P-GLS-II, a manager agent is used for collecting and analysing all the local optimal solutions obtained by the agents at the points of information exchange. In this paper, the manager only finds the features which appear in all these optimal solutions and then passes these common features on to all the GLS agents. Then, each GLS agent will evaluate the utilities for these features using the new utility function (4) during the next search stage. The utilities values for un-common features will be normally calculated using the original utility function (2).

The proposed algorithm with I GLS agents works as follows:

Step 0 Initialization: For $i = 1$ to I,

1. Agent i randomly selects a solution as its current solution.

2. Agent i set its penalty to feature k, $p_k^i = 0$

Step 1 LS: For $i = 1$ to I, each agent i run LS, starting from its current solution until it trapped in a local optimum solution.

Step 2 Information Collection: The manager agent collects the current optimal solutions of all the agents and finds all the common features in these solutions.

Step 3 GLS: For $i = 1$ to I, agent i does:

Step 3.1 Behaviour Adjustment: Calculating the utility for feature k, u_k^i using the new utility function (4).

Step 3.2 Penalizing: Finding the maximum feature's utility u_k and penalize it

Step 4 Stopping Condition: If a present stopping condition is met, stop and output the best solution obtained by these agents. Otherwise go to Step 1

Due to the behaviour adjustment through calculating the features utilities based on w_k parameter in Step 3.1, these GLS agents will search in more promising regions in the solution space.

4.2 Variants of P-GLS-II

As already mentioned, the value of w_k in (4) can be adjusted in two ways: statically before the search or dynamically during the search. The main idea that should be taken into consideration while adjusting w_k is to care about the diversity as well as the speedy convergence. In this paper two approaches have been suggested and used as following.

4.2.1 Static

In this approach, the influence of the weight, w_k, for feature k on its utility u_k is statically increased during the search if feature k is a common. For distinguishing we call P-GLS-II based on this approach P-GLS-II-Constant; simply *c*P-GLS-II. In *c*P-GLS-II, w_k is assigned a constant value before the search procedure starts and will not be changed through the search.

To find a suitable constant value that tries to equitably balance between the speedy convergence and the diversity, we test different values of $[0,1)$ for w_k on TSP instances of different size from TSPlib [11]. The reason behind selecting these instances is to ensure choosing the right parameter that represents the whole problem with its classes. As mentioned before, when $w_k = 1$ this represents parallel independent GLS (*ind*P-GLS). In these experiments we include 1 in the weight domain to compare *c*P-GLS-II with different values of w_k to *ind*P-GLS. In *c*P-GLS-II and *ind*P-GLS we run 2 agents of GLS procedures. The running length allowed for each algorithm is expressed in term of number of LS calls ($maxLS$). So, each algorithm stops when it finds the best known solutions or has called LS as many as $maxLS$ times. The best well known solutions and the maximum allowed LS calls for each agent on each instance are given in table 1. We try to choose different values for w_k that represent the weight domain; these values are given in table 2. 30 runs are carried out on each instance.

Table 2 shows the results for *c*P-GLS-II based on different values of w. We record the average of the best costs for the best tours obtained during the 30 runs on each instance. The best and second best obtained performance for each tested instance are identified in table 2 using bold and italic font

Table 1: Parameter setting of *c*P-GLS-II. Best Solution: the cost (distance) of the shortest tour. maxLS: number of local search (LS) calls allowed on each instance for every agent.

Instance Name	Best Solution	maxLS
St70	675	2000
rd400	15281	100000
u1432	152970	200000

Table 2: Experimental results for *c*P-GLS-II to indicate the best and second best value for w parameter on the tested instances. w: the weight value. AVG: the average of the costs of the best tours obtained during 30 runs.

w	Instance Name		
	st70 AVG	rd400 AVG	u1432 AVG
0	677	15305	153479
0.2	676.03	15281.43	153234.70
0.4	**675**	15281.43	153050.43
0.6	675.67	*15281.10*	*152998.57*
0.8	*675.27*	**15281**	**152994.63**
1	675.57	15281.9	153045

respectively. It can be seen from the results that the best value for w_k that generally achieved a good performance on all tested instances was when $w_k = 0.8$. Thus, this value of w_k is used for *c*P-GLS-II in all of the following experiments.

Table 3: Experimental results for *a*P-GLS-II to indicate the best and second best value for δ parameter on the tested instances. AVG: the average of the costs of the best tours obtained during 30 runs.

δ	Instance Name		
	st70 AVG	rd400 AVG	u1432 AVG
0.2	675.60	15285.60	153193.67
0.4	675.57	15281.53	153118.33
0.6	675.47	15281.67	153074.86
0.8	*675.43*	*15281.2*	*153026.73*
1	**675.4**	**15281.13**	**153020**

4.2.2 Dynamic

In this approach the weight, w_k, or importance of the common features are dynamically increased during the search. The idea is to consider the common features at the early stages of the search as normal features. Then, these features' importance is exponentially increased during the search if they are still common. This approach will help to avoid favouring features that appear as common at the beginning of the search and then disappear later. To distinguish this variant of the previous one, we will call this approach P-GLS-II-Annealing-like (*a*P-GLS-II).

The value of w_k in (4) for *a*P-GLS-II is adjusted using the following equation:

$$w_k = \frac{1}{2^{CurLS \times \beta}}, \tag{5}$$

and

$$\beta = \frac{1}{\delta \times maxLS} \tag{6}$$

Where, $CurLS$ is the current LS call, and $CurLS \geq 1$. δ is a tuning parameter and $\delta \in (0, 1]$.

β is used to control the speed of the convergence rate to the final solution. It is clear from (6) that its value is affected by δ parameter. At large values of δ, the influence of information exchanged with other agents is low, and thus agents are more likely to be independent. As the value of δ decreases, the influence of feature's commonality on its utility increases, and the higher the speedy convergence of *a*P-GLS-II. This strategy makes *a*P-GLS-II exponentially increases the favour of common features based on when they occur in the agents' local optimal solutions during the search. The aim is that this approach would help making the search to be wide at the beginning and the further search more rational, and therefore δ should be carefully set to avoid speedy convergence towards local optima of low quality.

It is clear from (5) that the value of w_k in (4) is decreased by a small amount during the search. At the beginning of the search when $CurLS$ is equal 0 (i.e., no LS call is called), w_k value will be equal to 1. Therefore, $feature_k$ will be considered as normal feature if it is a common feature and its probability to be penalized will be based on its cost c_k. With the increase of LS calls during the search, w_k value is decreased based on the $CurLS$ parameter. As a consequence, the probability of $feature_k$ to be penalized is also exponentially decreased if it is still a common feature.

In order to choose the best value for δ, we test different values of δ among its domain $\in (0, 1]$. We used the same TSP instances and parameters setting used in table 1. Also, 2 agents were run in *a*P-GLS-II and 30 runs were carried on each instance. Table 3 shows the results for *a*P-GLS-II with different values of δ. As in table 2 we record the average of the best costs for the best tours obtained during the 30 runs for *a*P-GLS-II on each instance. The best and second best value for δ parameter on each tested instance are identified using bold and italic font respectively. It clearly obvious from the results that the performance of *a*P-GLS-II increases with increasing the value of δ parameter. Thus, the best performance was achieved on all tested instances was when $\delta = 1$. This indicates that it is better for the different agents to exchange valuable features information which can be obtained at later stage of the search. This value of δ is used for *a*P-GLS-II in all of the following experiments.

5. RESULTS AND DISCUSSION

Several experiments have been conducted in order to examine the performance of P-GLS-II algorithms as well as to show how the incorporated collaborative mechanisms improve the performance and speed up the search against *ind*P-GLS and other state-of-the art algorithms. In our experiments, we used the Traveling Salesman Problem (TSP) as a well defined standard combinatorial optimization problem. Also, many benchmarks for TSP are available through TSPLIB [11]. The programs of P-GLS-II algorithms and *ind*P-GLS have been written in java and compiled using JetBrains IntelliJ IDEA 8.1.4 for windows. All computations were carried out on Intel Core 2 (2.13 GHz) with 3 of RAM. The implementation of P-GLS-II and *ind*P-GLS is based on running several sequential search agents of GLS procedures

Table 4: Average percentage above optimal for 30 runs of *ind*P-GLS-II, *c*P-GLS-II and *a*P-GLS with four hundred thousand local search calls performed.

Instance	*ind*P-GLS		*c*P-GLS-II		*a*P-GLS-II	
Name	avg.	std. dev.	avg.	std. dev	avg.	std. dev
pr1002	0.1752	0.0678	0.0051	0.0101	**0.0034**	0.0119
u1060	0.1682	0.0430	**0.0245**	0.0152	0.0274	0.0210
vm1084	0.0825	0.0290	0.0207	0.0127	**0.0199**	0.0200
pcb1173	0.1582	0.0528	**0.0189**	0.0186	0.0204	0.0292
d1291	0.1741	0.0492	0.1213	0.0924	**0.0952**	0.0654
rl1304	0.1149	0.1185	**0.0315**	0.0720	0.0404	0.0702
rl1323	0.1619	0.0635	**0.0781**	0.0600	0.0837	0.0659
fl1400	1.0598	0.2080	**0.9900**	0.1881	1.0389	0.3820
u1432	0.0513	0.0246	**0.0081**	0.0178	0.0196	0.0237
vm1748	0.2122	0.0489	**0.0684**	0.0374	0.0818	0.0460
rl1889	0.3393	0.1222	0.1605	0.0767	**0.1160**	0.0509
pr2392	0.4764	0.0859	**0.2234**	0.1112	0.2404	0.1198

and only simulates concurrency. To simplify the discussion of the simulation results, the performance of algorithms is measured by calculating the mean of the percentage excess above the best solution. *Excess* in a single run is defined as follows:

$$\frac{\text{Solution cost-Best known solution cost}}{\text{Best known solution cost}} \times 100 \qquad (7)$$

Three sets of experiments are carried out. The first set of experiments was carried out to compare the P-GLS-II with its two variants, *c*P-GLS-II and *a*P-GLS-II, to *ind*P-GLS. In the second set of experiments, a set of comparison studies was made to two cooperative algorithms designed based on two state-of-the-art algorithms. The third set of experiments was conducted to test the effect of increasing the number of agents on the performance of *c*P-GLS-II and *a*P-GLS-II.

5.1 Experiments 1

The aim of these experiments is to study the effectiveness of P-GLS-II algorithms compared with *ind*P-GLS. The two variants of P-GLS-II, *c*P-GLS-II and *a*P-GLS-II, are compared with *ind*P-GLS. The total number of the allowed calls of LS for each algorithm on each instance ($maxLS$) is set to 400000 which are divided equally between the running agents. Therefore, the algorithm stops when it finds the best well known solution or when the total number of LS called exceeds $maxLS$. The number of agents used in the algorithms is set to 2. 30 independent runs of each algorithm on each instance have been carried out.

The results for experiment are given in Tables 4, 5 and 6. Table 4 shows the *Mean Excess*% of *c*P-GLS-II, *a*P-GLS-II and *ind*P-GLS on the tested instances. The results show that *c*P-GLS-II and *a*P-GLS-II outperform *ind*P-GLS on all instances. *c*P-GLS-II beats the other two algorithms on 8 out of 12 of the tested instances. Also, *a*P-GLS-II was the best on 4 out of 12 of tested instances. In table 6, the one tailed t-test results at the 0.05 significant level are performed on each instance to confirm that the results for *c*P-GLS-II and *a*P-GLS-II are significantly better than *ind*P-GLS-II. Also, table 6 shows that there is no significant difference in the performance between *c*P-GLS-II and *a*P-GLS-II algorithms except one instance which is rl1889. On this instance *a*P-GLS-II is significantly better than *c*P-GLS-II.

To do further investigations about the performance of *c*P-GLS-II and *a*P-GLS-II we record the best and worst obtained solutions during their runs in table 5. The results show that *a*P-GLS-II outperforms *c*P-GLS-II on 5 out of 12 of tested instances in terms of the best solutions obtained during its runs. In terms of the worst obtained solutions, *a*P-GLS-II was worse than *c*P-GLS-II on 7 out of 12 of tested instances.

Table 5: Experimental Results for *c*P-GLS-II and *a*P-GLS-II. Worst: the worst final solution among 30 runs. Best: the best final solution among 30 runs.

Instance	*c*P-GLS-II		*a*P-GLS-II	
Name	Worst	Best	Worst	Best
pr1002	0.0309	0.0000	**0.0537**	0.0000
u1060	0.0518	0.0000	**0.0701**	0.0000
vm1084	0.0476	0.0000	**0.0685**	0.0000
pcb1173	0.0809	0.0018	**0.1195**	0.0018
d1291	**0.2854**	0.0018	0.2008	**0.0000**
rl1304	**0.2708**	0.0000	0.2487	0.0000
rl1323	**0.1958**	0.0100	0.1902	**0.0037**
fl1400	1.4011	0.7304	**2.2557**	**0.6757**
u1432	0.0490	0.0000	**0.0536**	0.0000
vm1748	0.1471	**0.0033**	0.1581	0.0089
rl1889	**0.3102**	0.0256	0.2275	**0.0196**
pr2392	0.4492	0.0751	**0.4568**	**0.0513**

From this experiment we can come up with the following findings:

1. The collaboration among agents does improve the performance in P-GLS-II algorithms.
2. The improvements observed with the two designed algorithms, *c*P-GLS-II and *a*P-GLS-II, are statistically significant better than the standard parallel GLS (*ind*P-GLS).
3. In the comparison between *c*P-GLS-II and *a*P-GLS-II, every algorithm beats the other in some instances. This observation indicates that these two algorithms are different and it is worth to be all considered while solving a problem.

Table 6: Results of t-tests comparing methods of *ind*P-GLS, *c*P-GLS-II and *a*P-GLS-II.

Instance Name	*c*P-GLS-II vs. *ind*P-GLS	*a*P-GLS-II vs. *ind*P-GLS	*a*P-GLS-II vs. *c*P-GLS-II
pr1002	0.0000	0.0000	0.6455
u1060	0.0000	0.0000	0.6093
vm1084	0.0000	0.0000	0.9056
pcb1173	0.0000	0.0000	0.8672
d1291	**0.0523**	0.0003	0.3671
rl1304	0.0157	0.0053	0.7219
rl1323	0.0000	0.0015	0.8034
fl1400	**0.1410**	**0.7687**	0.5446
u1432	0.0000	0.0015	0.0990
vm1748	0.0000	0.0000	0.3232
rl1889	0.0000	0.0000	**0.0393**
pr2392	0.0000	0.0000	0.5688

5.2 Experiments 2

The motivation behind doing this set of experiments is to evaluate the effectiveness of *c*P-GLS-II and *a*P-GLS-II with some other state-of-the-art algorithms. Hence, we compared *c*P-GLS-II and *a*P-GLS-II algorithms to a method which was recently published in [14]. The method is called Pattern Reduction Enhanced Ant Colony Optimization (PREACO). We compared our algorithms to the results in [14]. Eight instances from TSPLIB [11] were used in [14], we just compared our methods with the large instances that above 1000 cities. Similar to [14], 30 runs were carried out on each instances and the various performance measures (solution quality and running time) were averaged.

For convenience, we briefly explained PREACO with same parameters setting and notations as in [14]. PREACO works by running ACO with 25 ants (i.e. $m = 25$) for a number of iterations set to 1000. Then after a predefined number of iterations (ϱ) set to 5, a threshold (ψ) is calculated using this formula: $\varrho m\lambda$, where λ is a tuning parameter and was tested using two constant values 0.9 and 1. PREACO finds the edge among all edges that was traversed by all ants over the last ϱ iterations at least ψ time. Then sets a high probability for this edge to be in the solution [14].

PREACO was firstly compared with *ind*P-GLS and the results are shown in table 7. The results indicates that *ind*P-GLS with less running times outperforms PREACO on all tested instances in terms of the solution quality. This observation confirms the outstanding performance of GLS as state-of-the-art algorithm for TSP.

To ensure a fair comparison with our proposed methods, we adapted the collaboration mechanism used between ACO ants in [14] to work in our framework. Then we compare this new method with our proposed methods. For distinguishing, we denote this new method *pre*P-GLS.

The adaptation of P-GLS-II to work with the collaboration mechanism in [14] is simply applied through setting the value of w_k in the utility function (4) for $feature_k$ to 0, if $feature_k$ is traversed by all the agents (GLS procedures) at least ψ time over the last ϱ iterations. An iteration for P-GLS-II is considered one LS iteration (i.e. complete search of the neighbourhood). Due to the heavy nature of the agents in P-GLS-II framework as every agent is considered an independent solution strategy, we set m in *pre*P-GLS to 5. Similar to PREACO in [14], ϱ is set to 5 in *pre*P-GLS which means the agents exchange information acquired from their previous search after each 5 LS iterations. Finally, λ value is set to be 0.9. Thus, *pre*P-GLS will not consider just features that appear in all agents but also those features that appear in 90% of the agents. In case λ is set to 1 (i.e features should appear in all agents), *pre*P-GLS behaviour will be quite similar to first version of P-GLS framework [13] and the only difference is the time for exchanging information. 30 runs were carried out on each instance and every algorithm stops when it performs two hundred million of 2-opt swaps on each instance.

The results of *ind*P-GLS, *pre*P-GLS, *c*P-GLS-II and *a*P-GLS-II are given in table 8. The best and second best performance obtained by the algorithms are identified using the bold and italic font respectively. The results show that *a*P-GLS-II obtains better results on all tested instances than others. It also shows that the second best performance was achieved by *c*P-GLS-II. For example, apart from d1291 and d1655 instances, *c*P-GLS-II obtains better result than *ind*P-GLS-II and *pre*P-GLS. It also can be noticed from the results that *ind*P-GLS-II was better than *pre*P-GLS-II and *c*P-GLS on d1291 instance while preP-GLS-II beats *ind*P-GLS-II and *c*P-GLS on d655 instance. It can also be noticed from results that the performance of *c*P-GLS-II and *a*P-GLS with running 2 agents as in table 4 using the same instances is better than running 5 agents. This observation encourages studying the affect of increasing the number of agents for proposed algorithms on the performance in the next sub-section. In general, the results confirm the significant enhancement that our collaboration mechanisms have made to the performance of P-GLS-II framework in terms of the solution quality.

Table 7: Experimental Results for *ind*P-GLS and PREACO. AVG: the average percentage above optimal of the best tours obtained during 30 runs. Ts: the average of the time in seconds of the best tours obtained during 30 runs.

Instance Name	*ind*P-GLS		PREACO	
	AVG	Ts	AVG	Ts
pr1002	**2.24**	40.26	5.37	67.36
d1291	**4.56**	34.38	8.85	62.01
u1432	**1.46**	80.28	5.92	165.05
d1655	**5.74**	76.02	8.70	144.43

We also, compared our algorithms with another cooperative algorithm called SAGA [3]. SAGA was designed based on the well known meta-heuristic method SA. SAGA runs parallel agents of SA and after a number of annealing stages uses crossover/mutation operator from Genetic Algorithm as an exchanging information operator between the agents. The idea is to preserve the common components of the solutions at the exchanging information point and concentrate the crossover/mutation operator on un-common components with the hope of the further search to be more focus. The first preliminary results with same parameter settings in [3] show that *ind*P-GLS generally beats SAGA on the tested instances. Again this confirms the outstanding performance of GLS on TSP. In future, it will be interesting to study the performance of P-GLS-II frameworks with applying the exchanging information mechanism used in SAGA.

Table 8: Average percentage above optimal for 30 runs of *ind*P-GLS, *pre*P-GLS, *c*P-GLS-II and *a*P-GLS-II with two hundred million of 2-opt swaps performed.

Instance	*ind*P-GLS		*pre*P-GLS		*c*P-GLS-II		*a*P-GLS-II	
Name	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.
pr1002	1.87	0.20	1.84	0.19	*1.80*	0.16	**1.77**	0.21
d1291	*2.56*	0.76	2.64	0.81	2.86	0.76	**2.53**	0.90
u1432	1.46	0.19	1.57	0.20	*1.44*	0.36	**1.43**	0.34
d1655	5.50	0.58	*5.27*	0.47	5.49	0.66	**5.22**	0.46

5.3 Experiments 3

The aim of this final set of experiments is to study the sensitivity of the performance to the *c*P-GLS-II and *a*P-GLS-II algorithms of increase in the number of agents. Three instances of TSP have been arbitrary chosen from the tested instances, namely pr1002, d1291 and u1432. The domain values of number of agents is 2, 4, 6 and 8 respectively. Similar to the previous experiments, the running lengths allowed for algorithms are expressed in terms of the 2-opt swaps performed on each agent and set to two hundred million 2-opt swaps. Then, every algorithm stops when it find the best solution or when the number of 2-opt allowed exceeds. Table 9 and 10 show the results on pr1002, d1291 and u1432 for*c*P-GLS-II and *a*P-GLS-II respectively. Results are again reported as percent distance above the known optimal solution. In general, the results show that the performance of the both methods degrades with increasing number of agents. This can be attributed to the applied condition of a common feature which is defined as the one that appears in all agents' local optimal solutions. Therefore, the number of common features would shrink as the number of agents grows. This remark encourages further search on the definition of features' commonalty.

6. CONCLUSION

A cooperative framework based on GLS (P-GLS-II) for dealing with difficult optimization problems is proposed in this paper. It has been proposed to overcome the limitations in the first version of P-GLS as well as to improve the search efficiency. In P-GLS-II, several agents run GLS in a parallel way. These agents exchange information obtained during the previous search after every LS call during the search process and then use such information for speeding up their search. We have suggested that the common features of agents local optimal solutions are very important and their importance are gradually increased based on their occurrences in the agents local optimal solutions during the search. Based on this hypothesis, two P-GLS-II based frameworks called *c*P-GLS-II and *a*P-GLS-II are proposed. The main idea in both frameworks is to increase the favour of those features that always shared by the current solutions of all the agents during the search in static or dynamic manners. The performance of *c*P-GLS-II and *a*P-GLS-II was tested on the TSP. The obtained experimental results show the effectiveness of *c*P-GLS-II and *a*P-GLS-II compared to parallel GLS without cooperation and other state-of-the-art algorithms. The results confirm that the new defined collaboration mechanisms do improve P-GLS-II algorithm's performance significantly. Finally, the applications of P-GLS-II to other combinatorial optimization problems have their potential. Also, the definition of features' commonality should be more investigated and studied to consider not only those features that appeal in all agents but also those that appear in some agents.

7. REFERENCES

[1] G. Acampora, M. Gaeta, and V. Loia. Hierarchical optimization of personalized experiences for e-learning systems through evolutionary models. *Neural Computing & Applications*, pages 1–17, 2009.

[2] G. Acampora, V. Loia, and M. Gaeta. Exploring e-learning knowledge through ontological memetic agents. *Computational Intelligence Magazine, IEEE*, 5(2):66–77, 2010.

[3] S. Chen. SAGA: Demonstrating the benefits of commonality-based crossover operators in simulated annealing. Technical report, School of Analytical Studies and Information Technology, York University, CANADA, 2003.

[4] M. Daum and W. Menzel. Parsing natural language using guided local search. In *Proceedings in 15th European Conference on Artificial Intelligence, (ECAI-2002)*, pages 435–439, Lyon, France, 2002.

[5] M. Dorigo and T. StÃijtzle. *Ant Colony Optimization*. MIT Press, 2004.

[6] R. C. Ebehart and Y. Shi. Particle swarm optimization: Developments, applications and resources. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, volume 1, pages 81–86, 2001.

[7] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[8] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.

[9] Y. He, G. Liu, and Y. Qiu. A parallel tabu search algorithm based on partitioning principles for tsps. *IJCSNS Journal of Computer Science and Network Security*, 6(8A), 2006.

[10] D. J. Ram, T. H. Sreenivas, and K. G. Subramaniam. Parallel simulated annealing algorithms. *Journal of Parallel and Distributed Computing*, 37(2):207–212, 1996.

[11] G. Reinelt. A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.

[12] R. A. Rutenbar. Simulated annealing algorithms : An overview. *IEEE Circuits and Devices Magazine*, 15(1):19–26, 1989.

[13] N. Tairan and Q. Zhang. Population-based guided local search: Some preliminary experimental results. In *Proceedings of the IEEE World Congress on Computational Intelligence*, Barcelona-Spain, 2010.

Table 9: Average percentage above optimal for 30 runs of *c*P-GLS-II with two hundred million of 2-opt swaps performed on each agent.

Instance	2		4		6		8	
Name	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.
pr1002	**0.79**	0.21	1.71	0.23	1.87	0.22	2.17	0.34
d1291	**1.80**	0.65	2.56	0.71	3.19	0.68	3.13	0.90
u1432	**0.74**	0.15	1.31	0.17	1.88	0.26	2.50	0.33

Table 10: Average percentage above optimal for 30 runs of *a*P-GLS-II with two hundred million of 2-opt swaps performed on each agent.

Instance	2		4		6		8	
Name	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.	avg.	std. dev.
pr1002	**0.80**	0.17	1.68	0.18	1.87	.22	2.13	0.18
d1291	**1.62**	0.60	2.42	0.70	2.98	0.84	3.21	0.67
u1432	**0.71**	0.16	1.35	0.16	1.75	0.26	2.59	0.26

[14] S. Tseng, C. Tsai, M. Chiang, and C. Yang. A fast ant colony optimization for traveling salesman problem. In *Proceedings of the IEEE World Congress on Computational Intelligence*, Barcelona-Spain, 2010.

[15] C. Voudouris and E. P. Tsang. Guided local search and its application to the travelling salesman problem. *European Journal of Operational Research*, 113:469–499, 1999.

[16] C. Voudouris, E. P. Tsang, and A. Alsheddy. *Handbook of Meta-heuristics*, chapter Guided Local Search, pages 321–362. Springer Verlag, 2010.