Off-line Parameter Tuning for Guided Local Search Using Genetic Programming

Abdullah Alsheddy
Computer & Information Sciences College
Imam Muhammad Ibn Saud Islamic University
Riyadh, Saudi Arabia
Email: alsheddy@ccis.imamu.edu.sa

Michael Kampouridis
School of Computer Science and Electronic Engineering
University of Essex
Colchester CO4 3SQ, UK
Email: mkampo@essex.ac.uk

***Abstract*—Guided Local Search (GLS), which is a simple meta-heuristic with many successful applications, has lambda as the only parameter to tune. There has been no attempt to automatically tune this parameter, resulting in a parameterless GLS. Such a result is a very practical objective to facilitate the use of meta-heuristics for end-users (e.g. practitioners and researchers). In this paper, we propose a novel parameter tuning approach by using Genetic Programming (GP). GP is employed to evolve an optimal formula that GLS can use to dynamically compute lambda as a function of instance-dependent characteristics. Computational experiments on the traveling salesman problem demonstrate the feasibility and effectiveness of this approach, producing parameterless formulae with which the performance of GLS is competitive (if not better) than the standard GLS.**

I. Introduction

Guided Local Search (GLS) is a relatively new meta-heuristic method proposed by Voudouris and Tsang in 1997 [1] to solve combinatorial optimization problems. A main feature of GLS is that it is a flexible and rather simple to implement meta-heuristic technique, with few parameters to tune. Yet, it is a successful method showing a state-of-the-art performance on several applications to problems with various structures and objectives from scheduling and routing to assignment problems and constraint optimization [2], [3].

GLS has only one parameter (i.e. lambda) that requires a human-in-the-loop tuning approach. There has been no direct attempt to automatically tune the lambda parameter. Such an effort may lead to a parameterless GLS. Having a successful parameterless meta-heuristic is a challenge that attracts many researchers. This is due to the importance of producing a black-box algorithm to the end-user, which will encourage practitioners and researchers to use meta-heuristics to solve their problems. It is worth mentioning here the work of Mills *et al.* [4] on Extended GLS, where *aspiration criterion* from Tabu Search and *randomness* from Simulated Annealing are embedded in the GLS framework, aimed to minimize the sensitivity of GLS performance to the setting of lambda. Promising results on the quadratic assignment problem were reported in [4].

Parameter setting, which is about assigning appropriate values to the parameters of meta-heuristics, is a common challenging task in the design of meta-heuristics, particularly for those with many tuning parameters such as Evolutionary Algorithms [5]. As classified in [6], parameter setting techniques are divided in parameter tuning and parameter control techniques. Parameter tuning refers to approaches that attempt to find good values for the parameters before the run of the algorithm, and they remain fixed during the run. Parameter control refers to alternative approaches that start an algorithm run with initial parameter values that are automatically changed during the run.

A common parameter tuning technique is done in an "off-fline" manner [5], [7]. The idea is to find the optimal parameter setting for a class of problems using a sweep or optimization procedure on a selected "test suite" of sample problems, and then employ the resulting parameter setting for all instances of the same problem.

In this paper, we will present an off-line parameter tuning approach for GLS. The optimal setting of lambda will be learnt by GP which will evolve a parameter-less, instance-dependent expression that can be used by GLS to compute the value of lambda. To our knowledge, this is the first application of GP to parameter tuning for meta-heuristics.

The organization of this paper is as follows. Section II describes the GLS. Section III presents the proposed approach. Details of computational experiments are given in section IV. Finally, we conclude with section V.

II. Guided Local Search

GLS applies a penalty-based approach that can be superimposed on a local search algorithm to guide it to escape local optima. The objective function is augmented with penalties which are increased when the local search algorithm settles in a local optimum. Penalties are associated to solution features which distinguish between solutions with different characteristics. GLS associates a cost (c) and a penalty (p) to each feature. The costs can often be defined by taking the terms and their coefficients from the objective function. For example, in the Travelling Salesman Problem (TSP) [8], the objective is to find a route that visits a set of cities in a short total travelling distances. the defined feature can be *"whether the candidate tour travels immediately from city X to city Y"*. The set of features then includes all possible links between any two cities. The cost of each feature is the distance of the associated link. The penalty of a feature is initialized to 0 and

will be incremented every time LS settles on a local optimum and the feature is nominated to be penalized.

When features and costs are defined, GLS defines a function h that will be used by local search (replacing the original function g):

$$h(s) = g(s) + \lambda \times \sum_{i \in F} (p_i \times I_i(s)) \quad (1)$$

where s is a candidate solution, g an objective function that maps every candidate solution s to a numerical value, λ is a parameter to the GLS algorithm, i ranges over the features in F, p_i is the penalty for feature i (all p_i's are initialized to 0) and I_i is an indication of whether s exhibits feature i:

$$I_i(s) = 1 \text{ if } s \text{ exhibits feature } i; \text{ 0 otherwise.} \quad (2)$$

The basic procedure of GLS can be described as follows: starting from an initial solution, a local search algorithm is applied until it reaches a local optimum. GLS augments the cost function by adding penalties to selected features. Then, the local search restarts from the same local optimum using the updated objective function. The utility of penalizing feature i ($util_i$) under a local optimum s^*, is defined as follows:

$$util_i(s^*) = I_i(s^*) \times \frac{c_i}{1 + p_i} \quad (3)$$

The lambda parameter (λ) is the only tuning parameter to GLS. It has been observed that, for several problems, lambda can be calculated as a function of a local optimum and the average number of features present [4], [9]. In these problems, lambda is dynamically computed after the first local optimum and before penalties are applied to features for the first time. Providing an α parameter (called *lambda coefficient*), which is relatively instance independent, lambda is calculated by the following formula:

$$\lambda = \alpha * g(s^*)/|F_{s^*}| \quad (4)$$

where s^* is a local optimum and $|F_{s^*}|$ the number of features present in s^*. Tuning alpha can result in lambda values, which work for many instances of a problem class.

This observation encourages the development of a scheme that automatically tunes lambda, which would lead to a parameterless GLS. The idea is to replace Equation 4 by another equation that does not contain any tuning coefficient. We suggest doing this with the use of Genetic Programming (GP). The reason for this is because of GP's transparency as a white-box technique, which allows us to evolve a set of formulae that GLS can then use to calculate lambda. For instance, given a set of mathematical functions and a set of GLS-related terminals, GP could evolve a tree which would look like $\sqrt{|F_{s^*}|}$, where s^* denotes again a local optimum and $|F_{s^*}|$ the number of features present in s^*. However, the strength of the GP approach is that the only parameter of GLS (lambda) is now computed dynamically, which as a result leads to a parameter-less GLS algorithm.

III. The Proposed Approach

As suggested in [4], [9], the parameter of GLS (lambda) can be computed dynamically as a function of some instance-dependent characteristics, namely the objective value of a local optimum and the number of features exhibited in such a local optimum. However, the suggested formula (Equation 4) requires tuning another parameter (i.e. lambda coefficient) which is less instance dependent. Instead of this human observation, we propose the use of GP, as a machine learning technique, to learn the parameterless formula that can be used to dynamically compute lambda for a given optimization problem.

The proposed GP will produce and evolve a set of trees. Each tree represents a formula that expresses lambda as a function of a predefined set of instance-dependent characteristics. These predefined characteristics are part of the GP terminal set. These terminals are: the mean cost (Mean), standard deviation (StDev), maximum (Max), minimum (Min) of features presented in a local optimum, and the number of features present in such a solution ($|F|$). In addition, 10 numerical values, from 0.1 to 1 (with 0.1 increment from each other), are also part of the terminal set. The reason for this is because we wanted the trees to be able to express solutions as part of the predefined characteristics. For instance, while a lambda value equal to $\sqrt{|F_{s^*}|}$ might be effective, it might be even more effective if lambda is equal to $\frac{\sqrt{|F_{s^*}|}}{2}$. The above numerical values thus give us this potential.

Moreover, the terminals are manipulated through the following mathematical functions, which are part of the GP function set: ADD, SUB (Subtract), MUL (Multiply), DIV (Divide), EXP (Exponential), LOG (Logarithm), SQRT (Square Root) and MOD (Modular). Figure 1 presents a sample tree, which could be the result of a GP run. As we can observe, the evaluated tree in this example is $\sqrt{(Mean + StDev) \times 0.5}$. The value of lambda changes each time the tree is evaluated, depending on the respective problem instance's Mean and Standard Deviation's values.

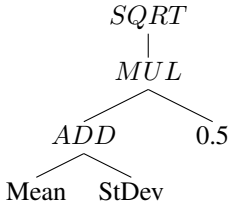

Fig. 1. Sample tree generated by the GP.

The fitness evaluation of each formula (i.e. a GP individual) is obtained by testing a GLS algorithm that replaces Equation 4 with this formula, on a small set of instances from the target problem. This instance set is referred to as the learning set.

It should also be noted that the GP acts as a framework that wraps around the GLS. This means that the GP first creates a population of trees, where each tree is a lambda expression. Each tree is then evaluated based on its fitness function, as described above. Then, GP operators take place and breed

TABLE I
GP PARAMETERS

GP Parameters	Value
Function Set	ADD, SUB, MUL, DIV, EXP, LOG, SQRT, MOD
Terminal Set	Mean, StDev, Max, Min, $\|F\|$, 0.1-1
Max Depth	2/3
Generations	20
Population size	30/50
Tournament size	3
Crossover probability	0.85/0.60
Mutation probability	0.15/0.40

new offspring. At the end of the evolutionary process (after a pre-specified number of generations), the best tree will be then used by GLS to solve further instances from the target problem.

IV. COMPUTATIONAL EXPERIMENTS

In order to examine the feasibility and effectiveness of the proposed approach we have conducted a set of experiments on the well-known TSP problem [8]. The TSP is chosen as a case study for several reasons, including its combinatorial nature and the success of GLS in solving this problem.

We will adopt the GLS proposed in [9] for solving the TSP. It applies the 2-Opt as a local search heuristic combined with Fast Local Search as a speed up technique. In [9], GLS was applied to a set of instances obtained from the TSPLib. The size of these instances ranges from 70 to 1002 cities.

A. The Learning Phase

As already mentioned, the first phase of our experiments involves running the proposed GP algorithm to obtain a candidate formula (i.e. lambda expression). After experimenting with different GP parameters, we found that the results were sensitive to the following parameters: population size, tree maximum depth, and mutation rate (mutation rate = 1-crossover rate). Thus, we decided to run more experiments with different settings of these parameters. Starting with the population size, we experimented with two different values, 30 and 50 individuals. We found that smaller population size could not always lead to high-quality results, because it was hard for the GP to be effective with such a small tree sample; on the other hand, populations with more than 50 individuals were extremely computational costly, due to the fact that GLS has to be applied every time an individual is evaluated. In addition, we experiment with maximum tree depth size of 2 and 3. The reason for not using bigger depth is because we desire the trees to be easily readable by human users. Lastly, we also found that results can be sensitive to the different crossover-mutation rates, and thus in this paper we experiment with two different mutation rates: 15% and 40%. There are thus 8 different settings of the above parameters. Table I summarizes the above settings, along with the other GP parameters.

The fitness evaluation of a GP individual is described by the following procedure:

TABLE II
THE BEST LAMBDA EXPRESSIONS OBTAINED BY THE GP RUNS WITH DIFFERENT SETTINGS

Best Individual	Fitness Value (%)	Given Name
$\sqrt{Mean}$	0.02	GLS-F1
$\sqrt{StDev}$	0.03	GLS-F2
$\log(2 * Mean)\% \log(\exp(Max)$	0.03	GLS-F3
$\log(Mean) + \exp(0.9)$	0.03	GLS-F4

1) The learning set includes four instances, namely kroa200, pr439, pcb442 and rat575. The only criterion used in the selection of the learning set is to choose from different instance classes.
2) For each instance, run GLS while using the candidate formula to calculate lambda instead of Equation 4. Only one GLS run per instance is executed, starting from a fixed initial solution. Each run terminates when the number of local search iterations reaches 100,000 or the optimum solution is found.
3) In each GLS run, the percentage excess above the best known solution (g^*) is calculated as follows:

$$Excess(s) = \frac{g(s) - g^*}{g^*} \times 100 \quad (5)$$

The average of this value over the four instances are computed and returned as the fitness value of the considered formula.

The results of this phase are shown in Table II which gives the best obtained formulae, together with its fitness value, from the different GP parameter settings. For example, a GP algorithm suggests calculating lambda as a function of the mean of the costs of the features presented in the first local optimum, as follows:

$$\lambda = \sqrt{Mean}$$

which is named as GLS-F1.

B. Testing and Evaluation

In this phase, we will examine the effectiveness of the obtained lambda's formulae. This is achieved by comparing a GLS that uses a formula obtained from the GP phase, with the standard GLS with the optimal parameter setting of lambda (that is the use of Equation 4 with $\alpha = 0.1667$) as presented in [9]. In this comparison, 19 well-known TSP instances are used. For each GLS run, the *percentage excess* (Equation 5) is calculated. In these runs, GLS is allowed a maximum number of local search iterations of 500,0000. The reported results are the mean *excess* of 10 executions for each GLS algorithm on each instance. These results are provided in Table III. It also shows how many times each GLS obtained the best-known solution.

The results suggest the following remarks:

- In 14 (out of 19) instances, the standard GLS obtained the best-known solutions within the allowed time in all the 10 runs. Similarly, all the parameter-less GLSs obtained the best-known solutions in all runs.

TABLE III
PERFORMANCE OF THE PARAMETERLESS GLS ALGORITHMS IN COMPARISON WITH THE STANDARD GLS, EXPRESSED IN TERMS OF THE MEAN EXCESS (%). THE VALUES IN THE THE PARENTHESES ARE THE OPTIMAL RUNS OUT OF 10.

Problem	stdGLS	GLS-F1	GLS-F2	GLS-F3	GLS-F4
KroA100	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
KroA150	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
KroA200	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
KroB100	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
KroB150	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
KroB200	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
KroC100	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
KroD100	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
KroE100	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
TSPeil101	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
TSPst70	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
TSPlin318	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
TSPpr299	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
TSPrat195	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
TSPpcb442	0.001 (8)	0.003 (5)	0.001 (7)	0.004 (8)	0.001 (9)
TSPpr1002	0.077 (0)	0.075 (0)	11.867 (0)	0.317 (0)	0.208 (0)
TSPpr439	0.054 (0)	0.067 (0)	11.173 (0)	0.074 (0)	0.052 (0)
TSPrat575	0.030 (0)	0.124 (0)	0.071 (0)	0.046 (1)	0.049 (0)
TSPrat783	0.015(2)	0.084 (0)	0.059 (0)	0.027 (0)	0.040 (0)

- In instance pcb442, the standard GLS obtained the best-known solution in 8 out of 10 runs. The four GLS algorithms with the parameterless lambda's formulae found the best-known solution in 5, 7, 8 and 9 runs. The latter is achieved by GLS-F4.
- In instance rat575, the only algorithm that was capable to find the best known-solution is GLS-F3, however, in only 1 out of 10 runs. With respect to the mean excess value, the standard GLS obtained 0.03 which is slightly better than that of GLS-F3 (0.05).
- In instance rat783, the standard GLS found the best-known solutions in 2 out 10 runs, whereas the proposed GLS algorithms could not find such a solution in any run. This is reflected in the mean excess value obtained by the standard GLS (0.01) which is better than that of other GLS algorithms (0.08, 0.06, 0.03, and 0.04).
- In instances pr439 and pr1002, the standard GLS, as well as the proposed GLS algorithms, could not obtain the best-known solutions in any run within the given time limit. However, in terms of the mean excess value, GLS-F4 obtained better value than the standard GLS on the pr439, and GLS-F1 is better than the standard GLS on the pr1002.

In conclusion, the results reveal that the lambda's formulae evolved by the defined GP produce competitive (if not better) results than the standard formula (Equation 4) that is employed in the standard GLS.

C. Discussions

Learning the parameter-less formula that sets the lambda parameter of GLS, presents a new automatic parameter tuning approach. The originality of this approach is in the search space of the parameter setting decision problem. Instead of using a continuous domain for the tuning parameter (λ or α in the case of GLS), a GP algorithm searches on the space of formulae that comprises a predefined set of functions and instance-deepndent characteristic.

The experimental results have demonstrated the capability of GP to evolve an effective parameter-less formula that can be used in the design of a parameterless GLS for a particular problem. This is achieved while given the GP algorithm a limited set of functions and instance-dependent characteristics as terminals. Moreover, the total search effort required by the GP algorithm is limited, at maximum, to 50 (individuals) $\times$ 20 (generations) $\times$ 4 (test instances) $\times$ 1 (GLS run) = 4000 GLS runs. With these settings, the GP algorithm suggested parameter-less lambda's formulae with which GLS performed competitively to the standard GLS that requires tuning the lambda's coefficient.

Voudouris and Tsang [9] pointed out that α is sensitive to the effectiveness of the underlying local search algorithm, and thus α needs to be tuned for each local search algorithm. This discourages the idea of using another optimization algorithm that searches for the optimal setting of α within the interval $(0, 1]$. However, whether the parameterless formula learnt by GP is general enough to be used with other local search algorithms (e.g. 3-Opt for the TSP) is a question that needs to be investigated. A further question is whether these parameterless formulae can be used even for other problems than those used in the GP's learning set. These questions describe our immediate future work.

V. CONCLUSION

GLS is a successful meta-heuristic with only one parameter to tune, i.e. lambda. This paper presents an automatic parameter tuning of the lambda parameter of GLS. The idea is to use GP, as a machine learning technique, to produce and evolve a parameterless formula that can be used to dynamically calculate lambda based on instance-dependent characteristics. Computational experiments on the TSP have confirmed the

feasibility and effectiveness of this approach. The proposed GP has shown the ability to learn a lambda's formula, based on a training set of 4 instances, that performs competitively to the standard GLS on 19 TSP instances. Further research directions include the following: (1) The generality of the learnt formula to be used with other local search heuristics designed for the same problem, (2) the applicability of generalizing such a parameterless formula to be embedded into a general parameterless GLS algorithm for solving other problems, (3) enhancing the GP function and terminal sets with more functions and (instance-dependent) terminals.

REFERENCES

[1] C. Voudouris, "Guided local search for combinatorial optimisation problems," University of Essex, PhD Thesis, 1997.

[2] C. Voudouris, E. P. K. Tsang, and A. Alsheddy, *Guided Local Search*, ser. Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc., 2010.

[3] ——, "Guided local search," in *Handbook of metaheuristics*, M. Gendreau and J. Potvin, Eds. Springer, 2010, pp. 321–362.

[4] P. Mills, E. P. K. Tsang, and J. Ford, "Applying an extended guided local search to the quadratic assignment problem," *Annals of Operations Research*, vol. 118, no. 1, pp. 121–135, 2003.

[5] K. De Jong, "Parameter setting in eas: a 30 year perspective," *Parameter Setting in Evolutionary Algorithms*, pp. 1–18, 2007.

[6] G. Eiben and M. Schut, "New ways to calibrate evolutionary algorithms," *Advances in Metaheuristics for Hard Optimization*, pp. 153–177, 2008.

[7] D. Goldberg, *The design of innovation: Lessons from and for competent genetic algorithms*. Springer, 2002, vol. 7.

[8] G. Gutin and A. Punnen, *The traveling salesman problem and its variations*. Kluwer Academic Pub, 2002.

[9] C. Voudouris and E. P. K. Tsang, "Guided local search and its application to the traveling salesman problem," *European Journal of Operational Research*, vol. 113, no. 2, pp. 469–499, 1999.