Swarm and Evolutionary Computation 20 (2015) 48–57

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Automatically configuring ACO using multilevel ParamILS to solve transportation planning problems with underlying weighted networks

Pengpeng Lin [a,*], Jun Zhang [a], Marco A. Contreras [b]

[a] Department of Computer Science, University of Kentucky, Lexington, KY 40506-0633, USA
[b] Department of Forestry, University of Kentucky, Lexington, KY 40546-0073, USA

ARTICLE INFO

Article history:
Received 12 October 2013
Received in revised form
30 October 2014
Accepted 30 October 2014
Available online 11 November 2014

Keywords:
ACO Metaheuristics
Graph coarsening
Multilevel technique
Automatic configuration
Online tuning

ABSTRACT

Configuring parameter settings for ant colony optimisation (ACO) based algorithms is a challenging and time consuming task, because it usually requires evaluating a large number of parameter combinations to find the most appropriate setting. In this study, a multilevel ParamILS (MParamILS) technique, that combines a graph coarsening method and the ParamILS framework, has been developed for configuring ACO algorithms to solve transportation planning problems with underlying weighted networks. The essential idea is to first use the graph coarsening method to recursively produce a set of increasingly coarser level problems from the original problem, and then apply ParamILS sequentially to the coarser level problems to select high-quality settings from a parameter combination domain. From the coarsest level to the finest (original) level problem, the parameter domain is refined by removing the low-quality settings identified by ParamILS. The size of the combination domain continues to decrease, resulting in fewer number of parameter combinations evaluated at finer level problems, hence the computing time is reduced. The performance of MParamILS was compared with ParamILS. Experimental results showed that MParamILS matches ParamILS in solution quality with significant reduction in computing time for all test cases.

1. Introduction

The ACO framework, inspired by the foraging behaviour of ant colonies, is one of the most successful swarm intelligence techniques in solving combinatorial optimisation problems [8,9]. It was introduced in the 90s by Dorigo et al. [15,14] and was first designed to solve the travelling salesman problem (TSP) [35,6]. Thereafter, the ACO framework has been applied extensively to a variety of real world applications including dynamic routing problems in mobile ad hoc networks [17,13,31,41], stochastic optimisation problems [3,22,11], and multi-objective optimisation problems [34,29]. These studies have shown that ACO-based algorithms can obtain high-quality solutions in a reasonable amount of time.

Similar to other evolutionary algorithms such as genetic algorithm and particle swarm optimisation, the performance of ACO-based algorithms is highly dependent on the values of their parameters. Without proper parameter settings, ACO-based algorithms can either converge very slowly or stagnate in local optimal solutions [16,32,21]. However, setting parameters for ACO algorithms is a difficult task because users have to find a balance between diversification and intensification. [18,14]. On one hand, when choosing parameter values that emphasise diversification, the final solution quality is often better, but more computing time is required. On the other hand, when choosing parameter values that emphasise intensification, ACO algorithms converge quickly but often to sub-optimal solutions.

Traditionally, ACO parameters were configured by manually changing one parameter at the time while keeping other parameters constant [21], which was similar to the Coordinate Search. Such a parameter tuning method was time consuming and prone to human errors [37]. In addition, the performance of the Coordinate Search is highly affected by the initial search point and step size. Without a proper initial setting, the Coordinate Search may converge very slowly, and often obtain local optimal solutions [39].

Automatic parameter configuration techniques have been developed to improve the performance of the manual parameter configurations [33]. Broadly, the automatic configuration techniques are categorised into “online”, which modifies an algorithm's parameter values while solving a problem instance, and “offline”, which adjusts parameter values before the target algorithm is actually deployed [32,19,10]. For example, Khichane et al. proposed two online frameworks (named GPL and DPL) that automatically adapted parameters for ACO algorithms at runtime [27]. The idea behind the frameworks resembled the pheromone update mechanism of ACO and the experiments showed positive results. However, discrete sets of parameters must be known a priori, and they were assumed to contain good values which should

* Corresponding author.
E-mail addresses: M.Lin@uky.edu (P. Lin), jzhang@cs.uky.edu (J. Zhang), marco.contreras@uky.edu (M.A. Contreras).

http://dx.doi.org/10.1016/j.swevo.2014.10.006
2210-6502/

allow ACO to find good results. This requirement is often difficult to meet for parameters with large value ranges. Birattari et al. [4,2,5] presented the iterated F-Race (I/F-Race) method for offline algorithm configurations. This method consisted of sampling parameter values from a probability distribution, selecting best parameter settings according to results of F-Race, and updating the probability distribution to bias the sampling towards good parameter values. Although the method was used to automatically tune parameters, I/F-Race required a well defined probability distribution that is usually difficult to determine if a limited set of instances are available [4], and the adoption of full factorial decision was impractical and computationally prohibitive for a large number of parameter configurations [2]. Manuel et al. [30] proposed two offline parameter variation strategies called "delta" and "switch", which were used for changing parameter values in the I/F-Race. In the delta strategy, parameter values first increased by a certain amount at each algorithm iteration, and then decreased when the values exceed a maximum allowable range. In the switch strategy, parameter values were either randomly selected from a value range or kept constant at each algorithm iteration. The experimental results showed that the performance of the automatic configuration method was able to match that obtained by the parameter variation strategies designed by a human expert.

To develop a more robust and generic parameter configuration technique that can configure ACO-based algorithms to solve large-scale transportation planning problems, we present the design, implementation and testing of a multilevel online parameter configuration framework (MParamILS), which combines a graph coarsening technique and the ParamILS framework. Compared with the aforementioned methods, MParamILS is more robust because it does not rely on a well defined probability distribution or require predetermined good parameter value sets. Moreover, it can be potentially generalised to other problem domains and optimisation algorithms. Then, instead of using ParamILS, other parameter configuration methods can be used to solve a different kind of problems (such as those presented in [3,22,11]).

The fundamental idea of the proposed multilevel framework is to configure ACO algorithms to solve the original problem, which might be computationally expensive, using a set of increasingly coarser level problems of which the computational cost is cheaper. The graph coarsening technique is recursively applied to the original problem, from which the general structure and certain edge properties are inherited by the resulted coarser level problems. Hence, coarser level problems can be used to evaluate the quality of parameter combinations for the original problem. To test the performance, we used the MAX-MIN ant system (MMAS) [36] in the experiments. MParamILS was compared with ParamILS in terms of solution quality and computing time using five test cases.

2. Background

2.1. Algorithm configuration problem

The algorithm configuration problem is comprised of a set of input data, a given target algorithm A, and a parameter domain $\Theta = \Theta_1 \times \cdots \times \Theta_k$, where Θ_i represents a parameter value set. Let $A(\vec{\theta})$ be an instantiation of the target algorithm with a parameter combination $\vec{\theta} \in \Theta$, $O_{A(\vec{\theta})}$ the objective function that measures the observed cost for running $A(\vec{\theta})$, and $\mathbb{E}(O_{A(\vec{\theta})})$ a statistical measurement such as the expectation, median, or norm. The objective of the algorithm configuration problem is to find the parameter combinations $\Theta^* \subseteq \Theta$ such that the target algorithm achieves the best possible performance on any input data that minimises $\mathbb{E}(O_{A(\vec{\theta})})$, $\forall \vec{\theta} \in \Theta^*$. Mathematically, the objective of the algorithm configuration problem is described as

$$\Theta^* = \{\vec{\theta} : \arg\min \mathbb{E}(O_{A(\vec{\theta})}), \vec{\theta} \in \Theta\} \quad (1)$$

Accordingly, algorithms that solve the algorithm configuration problem are called configurators, and algorithms to be configured are called target algorithms.

2.2. ParamILS framework

ParamILS framework is a state of the art procedure to solve the algorithm configuration problem [25]. To configure parameters, ParamILS makes a sequence of algorithm runs for each parameter combination. It, then, selects the best one based on the statistical information calculated from obtained objective values.

In order to properly compare between two parameter combinations, sufficient number of algorithm runs are required for calculating the necessary statistical information. Due to the stochastic nature of ACO algorithms, a high-quality or the optimal solution may be obtained by a low-quality parameter setting purely by chance. Therefore, the expected objective function value is used to assess qualities of parameter combinations.

As the expected values are calculated with the number of algorithm runs, two parameter combinations are comparable only when they have been evaluated the same number of times. Following, we formally state in Definition 1 that a parameter combination $\vec{\theta}_1$ dominates $\vec{\theta}_2$ if the expected objective value of $\vec{\theta}_1$ is better than that of $\vec{\theta}_2$ for the same number of algorithm runs:

Definition 1 (*Domination*). $\vec{\theta}_1$ *dominates* $\vec{\theta}_2$ if $N(\vec{\theta}_1) \geq N(\vec{\theta}_2)$ and $\mathbb{E}(\vec{\theta}_1) < \mathbb{E}(\vec{\theta}_2)$, where $N(\vec{\theta}_1) = \text{length}(R_{\vec{\theta}_1})$.

In this work, Definition 1 is used to modify the original ParamILS framework for configuring ACO algorithms.

3. Methods

3.1. Multilevel parameter configuration framework

The proposed multilevel framework (Fig. 1) consists of a target algorithm for parameter configuration, a problem that is coarsened into a set of increasingly coarser level problems, a parameter domain that contains possible parameter Combinations, and a configurator that runs the target algorithm to evaluate the qualities of parameter combinations in the domain.

In this study, the ACO-based algorithms developed for transportation problems with underlying weighted networks are the target algorithms, and the ParamILS is used as the configurator. The qualities of parameter combinations in the parameter domain are evaluated by running the target algorithm. Based on the obtained objective function values, low-quality parameter combinations can be identified and removed from the domain. This evaluation process is applied first to the coarsest level problem, and then applied subsequently to the next coarser level problem, until to the finest level (original) problem. Because of the smaller sizes of coarser level problems, evaluating parameter combinations on a coarser level problem is expected to finish quicker than that on the original problem. The size of the domain decreases as the low-quality parameter combinations are constantly being eliminated. As a result, the required computing time is reduced due to fewer number of parameter combinations having to be evaluated at finer level problems. After the parameter evaluation process on the coarser level problems, the remaining parameter

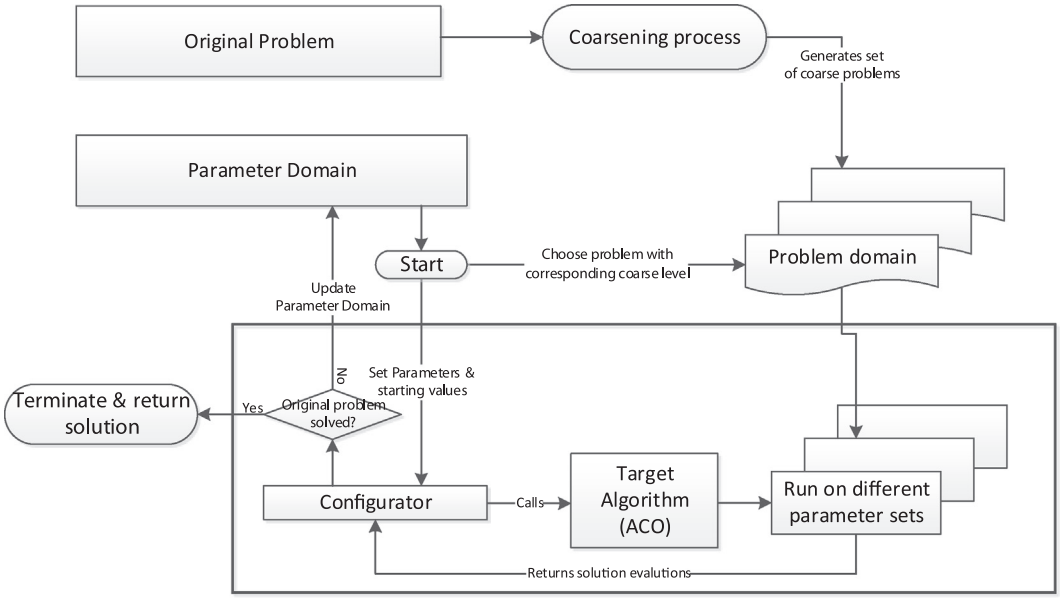

Fig. 1. Multilevel parameter configuration framework.

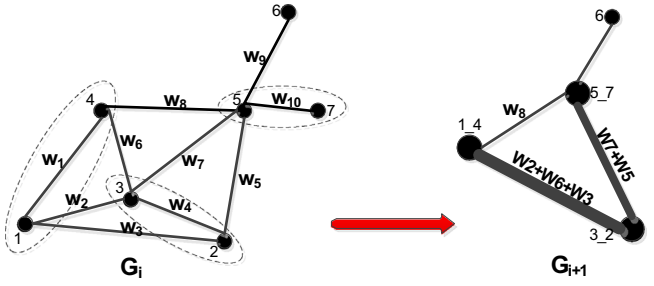

Fig. 2. Edge contraction process. A finer level graph G_i is coarsened by collapsing the matching edges (encompassed with dashed ovals) to construct a coarser level graph G_{i+1}. The incident nodes on the matching edges (Nodes 1, 2, 3, 4, 5, 7 in G_i) are aggregated into coarser level nodes (Nodes 1_4, 3_2, 5_7 in G_{i+1}). The weights on the jointed finer level edges ($E_{5,3}$ and $E_{5,2}$, $E_{3,1}$ and $E_{3,4}$, $E_{1,3}$ and $E_{1,2}$) are summed to form coarser level edge weights (W_2+W_6, $W_2+W_6+W_3$).

combinations in the domain at the finest level are considered high-quality and used to solve the original problem.

3.2. Generating coarser level problems

For combinatorial optimisation problems modelled with weighted networks, graph coarsening techniques can be naturally used to construct coarser level problems. This is because the underlying networks can be easily represented with graphs. More importantly, given a graph $G_0=(V_0,E_0)$, graph coarsening techniques can construct a sequence of graphs $G_1,\ldots,G_L$, where $G_i=(V_i,E_i)$ is a coarse approximation of G_{i-1} such that a solution of a given problem for G_i can be "efficiently" extended to that for G_{i-1} and vice versa [38,40]. This functionality implies that the obtained coarser level problems may be used to evaluate qualities of parameter settings for the finer level problems as a result of the similar structures and edge weight distributions. To produce a coarser level graph, the coarsening technique implemented in this work selects and collapses the edges of a finer level graph (Algorithms 1 and 2). Thus, the next level coarser graph G_{i+1} is constructed from G_i through a process of edge contractions and node aggregations.

The edges selected for contractions are the matching edges [26,28] of which no two edges are incident on the same node (such as the edges encompassed with dashed ovals in G_i in Fig. 2). Hence, collapsing the matching edges results in each node in the finer level graph being mapped to an unique node in the resulted coarser level graph. There are many existing techniques for finding a maximum matching in a graph [26,20,7,42]. We implement a matching algorithm (Algorithm 1) modified from a randomised algorithm proposed in [26], which the complexity is proportional to the number of edges $O(|E|)$ [23]. A maximum matching is found by visiting every node in a finer graph. When an unmatched node u is visited, the algorithm checks whether it has an adjacent node that has not been matched. If such a node v_i exists, edge(u,v_i) is included in the matching and nodes u and v_i are marked as matched. Otherwise, the node u remains unmatched and the algorithm continues to visit the next unvisited node. If u is adjacent to more than one unmatched node, the algorithm is designed to include the incident edge with the largest weight into the matching. This design is particularly suitable for minimisation problems, because edges with large weights are unlikely to be part of good solutions. In contrast, for the maximisation problems, edges with small weights should be considered for the contraction.

After a matching is found from a finer level graph, the selected edges are collapsed to generate a coarser level graph (Algorithm 2). The incident nodes of matching edges are aggregated to form coarser level nodes (an example is shown in Fig. 2). Those nodes that are not incident on the matching edges are directly copied to the coarser graph.

Algorithm 1. Finding matching edges algorithm.

```
Q:=all nodes except source and destination nodes;
QE = ϕ;

begin
  while Q has unvisited nodes do
    visit an unvisited node u in Q
    if u has been matched then
      continue to next while iteration
    end
    for adjcant unmatched nodes v ofu do
      find v_i ∈ v|weight(u, v_i) is maximum
    end
    make vertices u and v_i as "matched"
    add edge(u, v_i) to QE
  end
  return (QE)
end
```

Algorithm 2. Graph coarsening algorithm.

```
QE:=matching edges
begin
  for edge(u, v) ∈ QE do
    u_v ⇐ aggregate u and v
    if u, v are both adjacent to a node k then
      weight(u_v, k):=weight(u, k)+weight(v, k)
    end
  end
end
```

While matching edges are collapsed together in the coarsening process, edges that both connect to the aggregated nodes and incident to the same node (such as $E_{5,3}$ and $E_{5,2}$ in G_i in Fig. 2) are jointed together to form coarser level edges. Correspondingly, the weights of two jointed edges are combined to give a new weight to the resulted coarser level edge (i.e., in Fig. 2, the weight W_7+W_5 obtained by combining weights on $E_{5,3}$ and $E_{5,2}$). This approach is adopted from [26], in which adjacent nodes to a node u_1 in a graph is defined as

$$Adj(u_1) = (\{Map[x] | x \in Adj(v_1)\} \cup \{Map[x] | x \in Adj(v_2)\}) - \{u_1\}$$

where v_1 and v_2 are the finer level nodes aggregated to obtain the coarser level node u_1, and $Map[x]$ maps a finer level node to a coarser level node such that $Map[v_1] = Map[v_2] = u_1$. The combined weight of an edge (u_1, u_2) is given by

$$w(u_1, u_2) = \sum_x \{w(v_1, x) | Map[x] = u_1\} + \sum_x \{w(v_2, x) | Map[x] = u_2\}.$$

3.3. Modifying ParamILS framework

Algorithm 3. Objective($\vec{\theta}$, N, bound). $R[i]$ denotes ith ACO run.

```
Input (Parameter Set θ⃗, number of runs: N, bound)
Output (the expected objective value E(θ⃗))
begin
  runtime ← number of algorithm runs for θ⃗
  sum ← sum of objective values in R_θ⃗[1], ..., R_θ⃗[runtime]
  if runtime ≥ N then
    return (sum/runtime)
  end
  foreach i = runtime to N do
    O_i ← objective from a newly executed run of A(θ⃗)
    sum ← sum + O_i
    runtime ← runtime + 1
    R_θ⃗[i] ← O_i
    //terminate early for bad parameter set
    if (sum/N) > bound then
      return(WorstPossibleObjective)
    end
  end
  return(sum/runtime)
end
```

A modified ParamILS framework (Algorithms 3–5) is implemented according to Definition 1 to configure ACO, where procedure *Objective* (Algorithm 3) evaluates a parameter combination and returns the expected objective value. For the evaluated parameter combinations, a global cache is used to archive their evaluation information, which contains the number of times the parameter combination has been evaluated (*runtime*), the best objective value obtained, and the sum of all the obtained objective values (*sum*). *Objective* procedure takes an integer value (N), which is the number algorithm runs required for the evaluation, and a threshold *bound*, which is used to stop the evaluation process for low-quality parameter combinations. Before evaluating a parameter combination $\vec{\theta}$, its *runtime* is extracted from the global cache to compare with N. If $\vec{\theta}$ has already been evaluated N times (i.e., *runtime* $\geq N$), it will not be evaluated again and *Objective* simply returns the expected objective value calculated based on the evaluation information. Otherwise, $\vec{\theta}$ is evaluated until the number of evaluations equals N or the parameter combination is considered as low-quality. This is determined when the expected objective value exceeds the *bound*. Then, *Objective* stops evaluating the parameter combination further and returns a worst possible objective value (i.e., a big number for minimisation problems) as an indication of bad quality. Note that a proper *bound* value can save computing time by avoiding evaluating low-quality parameter combinations. An improper *bound* value (such as too small or too large), however, can either cause a good parameter combination being falsely labelled as low-quality or a bad parameter combination not being identified. In this study, based on the preliminary experimental results, *bound* is set to be twice as big as the best expected objective value found. However, we emphasise that this setting is by no means optimal or universal. For a different set of testing problems, a new *bound* setting may be more appropriate to achieve good results.

Algorithm 4. Better($\vec{\theta}_1$, $\vec{\theta}_2$).

```
Input(Parameter Set θ⃗_1, Parameter Set θ⃗_2)
Output(true if θ⃗_1 dominates θ⃗_2, false otherwise)
begin
  if θ⃗_1 = = θ⃗_2 then
    return(true)
  end
  if N(θ⃗_1) ≤ N(θ⃗_2) then
    N ← N(θ⃗_1)
  else
    N ← N(θ⃗_2)
  end
  while true do
    N ← N + 1
    Objective(θ⃗_1, N, bound)
    Objective(θ⃗_2, N, bound)
    if Dominate(θ⃗_1, θ⃗_2) then
      return true
    end
    if Dominate(θ⃗_2, θ⃗_1) then
      return(false)
    end
  end
end
```

Procedure: Dominate($\vec{\theta}_1$, $\vec{\theta}_2$)

begin
  if $N(\vec{\theta}_1) < N(\vec{\theta}_2)$ **then**
    `return(`**false**`)`
  else
    `return(`
    `Objective(`$\vec{\theta}_1, N(\vec{\theta}_2), bound$`)` $\leq$
    `Objective(`$\vec{\theta}_2, N(\vec{\theta}_2), bound$`))`
  end
end

Procedure *better* (Algorithm 4) compares two parameter combinations $\vec{\theta}_1$, $\vec{\theta}_2$. It returns true if $\vec{\theta}_1$ dominates $\vec{\theta}_2$ and false otherwise. The *Objective* procedure and an auxiliary procedure *Dominate* are used to make the comparison. To determine which parameter combination is superior, $\vec{\theta}_1$ and $\vec{\theta}_2$ have to be evaluated for the same number of times. If both parameter combinations produce the same expected objective value,*better* will continue the evaluation process until one of them dominates the other.

Algorithm 5. ParamILS(Θ, $\vec{\theta}_{start}$, r, s); *Nbh*($\vec{\theta}$): returns a neighbouring parameter combination;$R_{\vec{\theta}}$: denotes for the evaluation information of $\vec{\theta}$;*getbest*($R_{\vec{\theta}}$): returns the best objective value obtained;*IterativeLocalImprovement*: iteratively compares the neighbouring parameter combination with the current best one.

`Output(`*Best parameter setting, parameter combination set*`)`
begin
  $\vec{\theta}_{best} = \vec{\theta}_{start}$
  foreach $i = 1$ **to** r **do**
    $\vec{\theta} \leftarrow$ random $\vec{\theta} \in \Theta$
    if `better(`$\vec{\theta}$, $\vec{\theta}_{best}$`)` **then** $\vec{\theta}_{best} \leftarrow \vec{\theta}$
  end
  $\vec{\theta}_{best} \leftarrow$ `IterativeLocalImprovement(`$\vec{\theta}_{best}$`)`
  while *NotTerminationCriterion* **do**
    $\vec{\theta} \leftarrow \vec{\theta}_{best}$
    `// parameter local perturbation`
    foreach $j = 1$ **to** s **do**
      $\vec{\theta} \leftarrow$ random $\vec{\theta}' \in$ `Nbh(`$\vec{\theta}$`)`
    end
    $\vec{\theta} \leftarrow$ `IterativeLocalImprovement(`$\vec{\theta}$`)`
    `//  parameter acceptance`
    if `better (`$\vec{\theta}$, $\vec{\theta}_{best}$`)` **then** $\vec{\theta}_{best} \leftarrow \vec{\theta}$
  end
  $O_{\vec{\theta}_{best}} \leftarrow$ `getbest(`$R_{\vec{\theta}_{best}}$`)`
  $E_{\vec{\theta}_{best}} \leftarrow E(R_{\vec{\theta}_{best}})$ `// expected objective value for` $\vec{\theta}_{best}$
  forall the *evaluated* $\vec{\theta} \in \Theta$ **do**
    if `getbest(`$R_{\vec{\theta}}$`)` $\leq O_{\vec{\theta}_{best}}$ ***and*** $E(R_{\vec{\theta}}) < BF \times E_{\vec{\theta}_{best}}$ **then**
      add $\vec{\theta}$ to $\Theta_{selected}$
    end
  end
  `return(`$\Theta_{selected}$, $\vec{\theta}_{best}$`)`
end

Procedure: `IterativeLocalImprovement(`$\vec{\theta}$`)`
begin
  while $\vec{\theta}' \neq \vec{\theta}$ **do**
    $\vec{\theta}' \leftarrow \vec{\theta}$
    foreach $\vec{\theta}'' \in$ `Nbh(`$\vec{\theta}'$`)` *in randomized order* **do**
      if `better(`$\vec{\theta}''$, $\vec{\theta}'$`)` **then** $\vec{\theta} \leftarrow \vec{\theta}''$
      break
    end
  end
  `return(`$\vec{\theta}$`)`
endProcedure: `Nbh(`$\vec{\theta}$`)`
begin
  visit each $\vec{\theta}' \in \vec{\theta}$ in randomized order
  if $\vec{\theta}'$ *differs from* $\vec{\theta}$ *in one parameter* **then** `return(`$\vec{\theta}'$`)`
end

The main procedure of ParamILS (Algorithm 5) uses *better*, and other two auxiliary procedures (*Nbh* and *IterativeLocalImprovement*) to select high-quality parameter combinations. *Nbh* takes a parameter combination and randomly selects one of its neighbouring parameter combinations that differ in one parameter value. *IterativeLocalImprovement* uses *Nbh* to compare the current best parameter combination $\vec{\theta}_{best}$ with the neighbouring parameter combinations. If a neighbouring parameter combination is found to be better, it replaces $\vec{\theta}_{best}$ and becomes the new best parameter combination. *IterativeLocalImprovement* continually refines the best parameter combination until a dominating one, with no better neighbours can be found, is obtained.

ParamILS takes two parameters: r, which is the number of attempts to find a better parameter combination than $\vec{\theta}_{start}$ at the beginning, and s, which is number of parameter neighbouring searches (*IterativeLoca-*

Table 1
Tested methods, parameter settings, and running environment.

Method	Description	Parameter settings	Implementation	OS
Exhaustive	Target ACO runs 10 times for every parameter setting to obtain the best solution.	$\alpha \in [0, 0.05, 0.1, \ldots, 0.95, 1]$, $\beta \in [0, 0.05, 0.1, \ldots, 0.95, 1]$, $\rho \in [0, 0.1, 0.2, \ldots, 0.9, 1]$, $\lvert\Theta\rvert = 4000$, *Max iteration* = 10,000, Max pheromone = 0.01, Min pheromone = 0.00001.	Divide Θ into 10 partitions and use 10 processors to run GuidedACO with each parameter combination partition simultaneously.	C++, Linux.
ParamILS	ParamILS configures parameter according to solution quality.	Same ACO settings as Exhaustive, Number of iterations: 100, Local sÂarch $r = 10$, Perturbation $s = 3$, Initial best parameter Setting: $(\alpha = 0.5, \beta = 0.5, \rho = 0.5)$.	Parameter settings in Θ is sequentially configured and ParamILS stops after 100 iterations. Best solution is obtained during parameter configuration.	C++, Linux.
MParamILS		Same ACO settings as Exhaustive, Four level problems: Level 0 (original problem), Level 1 (first coarser), Level 2 (second coarser), Level 3 (coarsest).		C++, Linux

llmprovement). The values of the two parameters affect the overall computing time and final solution quality. In the experiments, r was set to 10, s was set to the number of parameters being configured (α, β, ρ for ACO), and $\vec{\theta}_{start}$ was set to $\alpha = 0.5, \beta = 0.5, \rho = 0.5$. ParamILS iteratively evaluates parameter combinations. Meanwhile, the evaluation information is archived and updated. In the end, a set of good parameter combinations are selected from evaluated parameter combinations according to the evaluation information. A parameter combination is considered good if its best objective value obtained is no worse than the one obtained by $\vec{\theta}_{best}$ and their expected objective value is less than $\mathbb{E}_{\vec{\theta}_{best}}$ times a boundary factor (*BF*):

Definition 2 (*Selecting good parameter combinations*). A parameter combination $\vec{\theta}$ is considered good if $getbest(R_{\vec{\theta}}) \leq getbest(R_{\vec{\theta}_{best}})$ and $\mathbb{E}(R_{\vec{\theta}}) < BF \times \mathbb{E}(R_{\vec{\theta}_{best}})$, where *BF* is an integer scaler and $getbest(R_{\vec{\theta}})$ is a function that returns the best objective value obtained from evaluating $\vec{\theta}$.

Similar to the *bound* setting in Algorithm 4, based on the preliminary experimental results, the boundary factor was set to 2 in the implementation. Time complexity of ParamILS depends on the number of parameter combinations to be evaluated. The convergence of ParamILS is proven in [25], where the probability of finding the optimal parameter configuration $\vec{\theta}^*$ approaches one as number of parameter combination searches goes to infinity.

3.4. Multilevel ParamILS framework

The Multilevel ParamILS framework (Algorithm 6) combines the above described graph coarsening algorithm and ParamILS. The graph coarsening algorithm coarsens a problem G into a set of increasingly coarser level problems $G_0, G_1, \ldots, G_n$. ParamILS is first applied to the coarsest problem (G_n) with a start parameter combination. It selects high-quality parameter combinations from a parameter combination domain Θ and the identified low-quality parameter combinations are discarded. Next, ParamILS is applied to the second coarsest level problem (G_{n-1}) and uses the best parameter combination selected from the coarsest level problem as the start parameter combination. It again selects high-quality parameter combinations, removes low-quality ones from the Θ, and possibly obtains a new best parameter combination. Subsequently, this process continues to the next level coarser problem until ParamILS is applied to the finest level problem G_0 (original problem), resulting in a significant reduction in size of the Θ at the finest level. Consequently, computing time is greatly saved in that (1) identifying and eliminating parameter combinations on coarser level problems is faster than that on the original problem, (2) the target algorithm (ACO) is expected to converge faster using selected high-quality parameter combinations.

Algorithm 6. Multilevel ParamILS framework($\Theta, \vec{\theta}_{start}, D$).

```
Input(Θ, θ⃗_start, ProblemG)
Output(good parameter combinations set, best solution)
begin
    (G_n, ..., G_0 ⇔ D) ← obtain coarse problems from D
    foreach i = n to 0 do
        // update domain Θ, obtain θ⃗_best
        (Θ, θ⃗_best) ← ParamILS(Θ, θ⃗_start G_i)
        θ⃗_start ← θ⃗_best
    end return(Θ, bestsolution)
end
```

4. Implementation and experiment setup

All the algorithms and procedures introduced in this study were implemented using C++ and uploaded to the Lipscomb High Performance Computing Cluster (HPC) supported and maintained by the University of Kentucky Center for Computational Science. All programs were executed on the computing nodes of HPC: Dual Intel E5-2670 of 8 Cores at 2.6 GHz with 64 GB of 1600 MHz RAM and Linux Red Hat OS. To test for performance, MParamILS was compared with an Exhaustive method and the ParamILS (Table 1). The MMAS [36] was used as the target algorithm and the objective was to configure its three main parameters: α that controls relative importance of pheromone information, β that controls relative importance of heuristic information, and ρ which is the pheromone evaporation rate used to prevent ACO from converging to local optimal or reaching premature stagnation. The value ranges of the three parameters were confined to [0,1] with a step of 0.05 for α, β and 0.1 for ρ, which makes a total 4000 parameter combination domain. We implemented the MMAS according to the study of Contreras et al. [12], where an ACO-based algorithm was applied to forest transportation planning problems (FTPPs) with consideration of an environmental impact factor called sediment.

The MMAS was applied to five test cases (Table 2). Case I was a fixed charge transportation problem [24] where fixed and variable costs were involved. Case II and Case III were constrained fixed charge transportation problems where constraints of different strictness levels were imposed. Case IV was a minimisation problem where the objective was to obtain a set of routes from sources to destinations with minimum total weights. Case V was the Hamiltonian cycle problem which is a special case of the travelling salesmen problem (TSP) obtained by setting the edge attributes to one [1].

The test cases were designed on a network (Original Problem in Fig. 3) that consists of 500 road segments and 200 intersection nodes with 25 sources and one destination. The graph coarsening technique was applied to the problem network to produce coarser level problems. Special nodes, such as source and destination nodes, were reserved from the coarsening process and copied directly from a finer level to a coarser level problem. Five coarser level problems (Level 1, 2, 3, 4, 5 in Fig. 3) with increasingly reduced sizes were obtained (Table 3). The problem size was reduced considerably from the original problem to the coarser level problems (about 40% from the original problem to Level 1 problem). However, the size reduction rate gradually decreased as the coarser level increased (i.e., only 15% from Level 4 to Level 5). This can be explained by the fact that a coarser level problem contains fewer number of matching edges than a finer level problem. Therefore, fewer number of nodes and edges are aggregated and collapsed when the coarsening process is applied, causing a smaller size reduction rate.

Because the size reduced from a coarser level to the next coarser level problem can be inadequate, not all five coarser level problems might be needed in MParamILS. To select appropriate coarser level problems, we ran MMAS 100 times on each of the coarser level problem (including original problem) to solve for Case I and compared their average running times (Fig. 4). MMAS was set to stop after 10,000 iterations and the three parameters were set: α=0, β=0, ρ=1, which turned off the functionality of the pheromone information and made MMAS a randomised search algorithm. The average computing time reduction was highest from the original problem to Level 1 problem, and lowest from Level 4 problem to Level 5 problem (Fig. 4). In general, the differences in computing time between coarser level problems became less significant as the problem size differences became smaller. This was especially evident for

Table 2
Test cases, objective functions, and constraints.

Test case	Name	Objective function	Constraints	Description
Case I	Fixed charge problem (FCP)	$\min \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} VC_{i,j}\times E_{i,j}+FC_{i,j}\times E_{i,j}$ where $E_{i,j}=\{0,1\}$		VC: variable cost, FC: fixed cost, Route: multiple sources to one or more destinations.
Case II & Case III	Constrained FCP (CFCP)	$\min \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} VC_{i,j}\times E_{i,j}+FC_{i,j}\times E_{i,j}$ where $E_{i,j}=\{0,1\}$	$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} Sed_{i,j}\times E_{i,j}\leq SedRct$	$Sed_{i,j}$:sediment, SedRct=2000 for Case II, SedRct=1500 for Case III, Route: multiple sources to one or more destinations.
Case IV	Minimisation transportation problem	$\min \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} Sed_{i,j}\times E_{i,j}$ where $E_{i,j}=\{0,1\}$		Route: multiple sources to one or more destinations.
Case V	Hamiltonian cycle problem	$\sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} E_{i,j}$ where $E_{i,j}=\{0,1\}$	$\sum_{\substack{i=0,\\ i\neq j}}^{n} E_{i,j}=1$, $\sum_{\substack{j=0,\\ j\neq i}}^{n} E_{i,j}=1$	Route: single source, every node has to be visited exactly once.

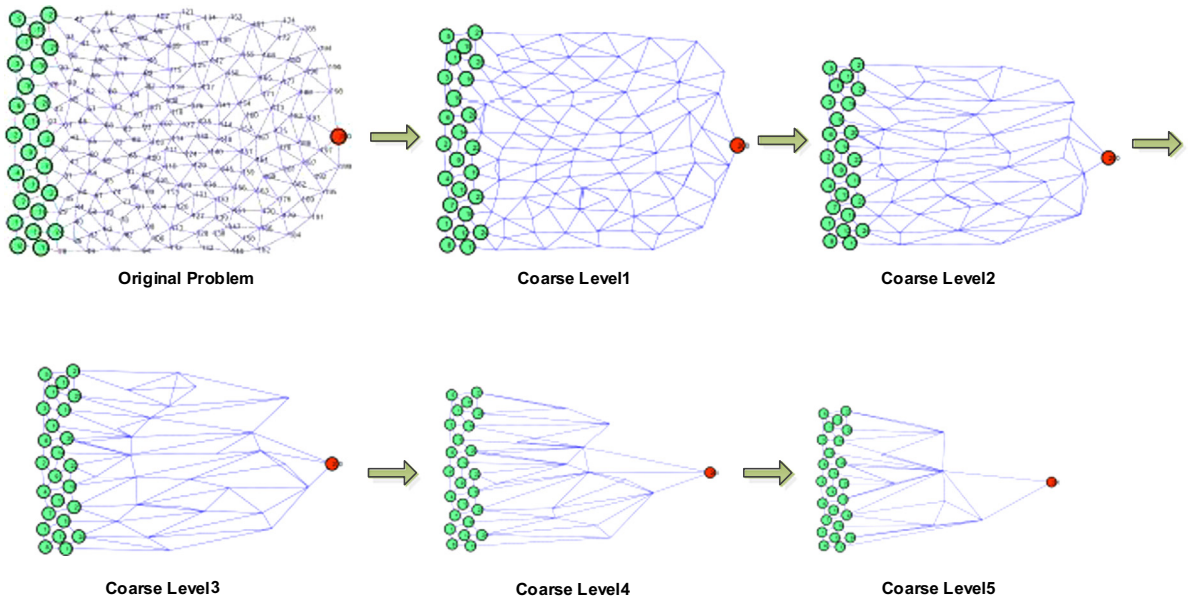

Fig. 3. Original problem and its subsequent coarser level problems.

Table 3
Number of nodes and edges between the original network and coarser level networks. Percentage of nodes and edges reduced from finner level networks to coarser level networks.

	Original	Level 1	Level 2	Level 3	Level 4	Level 5
# of nodes	200	116	74	53	41	35
# of Edges	500	300	190	134	102	86
Nodes reduction rate	N/A	42.00%	36.21%	28.37%	22.64%	14.63%
Edges reduction rate	N/A	40.00%	36.66%	29.47%	23.88%	15.68%

Level 3, 4 and 5 problems, where their computing times only differed in one or two seconds. Due to the small time difference, we only considered Level 1, 2 and 3 problems in our experiments for the MParamILS.

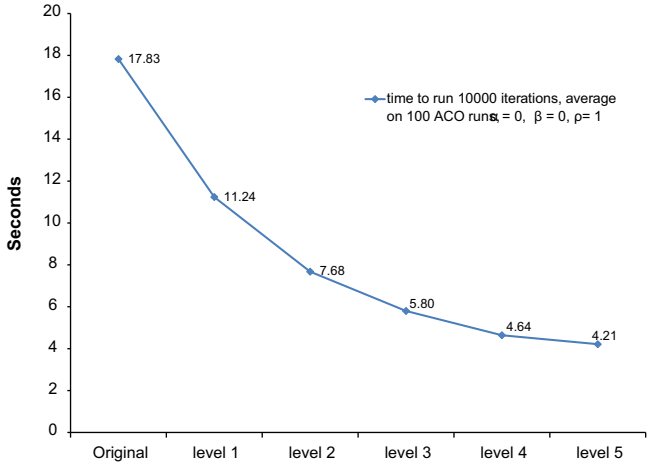

Fig. 4. Average computing time (in seconds) of running MMAS 100 times on the original problem and its coarse level problems for Case I (ACO stops search for solution after 10,000 iterations, $\alpha=0$, $\beta=0$, $\rho=1$).

5. Results and discussion

The Exhaustive method was applied to each test case once and used to compare solution quality. It ran MMAS on each parameter combination 10 times, and the best solutions were selected according to the objective functions. Since sequential exhaustive methods can take a long time, we divided the parameter combination domain into 10 intervals and applied the Exhaustive method to each of them simultaneously. The computing time of the Exhaustive method was calculated by adding the time spent for each interval. ParamILS and MParamILS were run independently 10 times for each test case, and average computing time and

Table 4
Best objective values obtained by the Exhaustive, average objective values obtained by ParamILS and ParamILS$_s$ and the standard deviations for the five test cases.

Test Cases	Exhaustive	ParamILS (AVG)	ParamILS (Std)	ParamILS$_2$ (AVG)	ParamILS$_2$ (Std)	ParamILS$_3$ (AVG)	ParamILS$_3$ (Std)	ParamILS$_4$ (AVG)	ParamILS$_4$ (Std)
Case I	1,496,560	1,496,560	0	1,496,560	0	1,496,560	0	1,496,560	0
Case II	1,585,590	1,585,705	484.79	1,585,840	655.74	1,585,860	644.67	1,590,043	6395.21
Case III	2,048,090	2,053,620	16,405.11	2,052,869	17,955.51	2,058,922	16,088.22	2,055,849	15194.02
Case IV	948.6	948.6	0	948.6	0	948.6	0	948.6	0
Case V	199	199	0	199	0	199	0	199	0

Table 5
Results of statistical significance tests (ANOVA test) on computing time of methods for each test case and the corresponding least-square means of computing time. The difference in computing time is significant if the p value is less than 0.05.

P < 0.05	Exhaustive	ParamILS	ParamILS$_2$	ParamILS$_3$	ParamILS$_4$	*P−Value*
Case I	951,413.80	244,822.90	98,266.35	75,390.81	49,693.95	< 0.0001
Case II	921,843.70	246,613.80	99,363.67	69,789.65	54,150.39	< 0.0001
Case III	866,465.40	277,902.80	104,019.05	73,155.85	56,313.06	< 0.0001
Case IV	849,819.00	19,1257.50	82,751.51	73,309.86	63,709.44	< 0.0001
Case V	1,018,748.20	172,384.10	53,592.25	35,869.23	34,975.68	< 0.0001
Average	921,658.02	226,596.22	87,598.57	65,503.08	51,768.50	
Reduced		75%	61%	25%	21%	

objective function value were used for comparisons in the experimental results. Because MParamILS was set to use different number of coarser level problems, we use MParamILS$_i$ to denote for the different settings, where i indicates the maximum number of coarsening levels used. For example, MParamILS$_4$ indicates that Level 0–3 problems were used. Additionally, we use MParamILSs as a collective name referring to MParamILS with all the tested coarsening level settings as a whole.

The performances were first compared between the Exhaustive method and the two heuristic methods. The purpose was to investigate whether the tested heuristics can produce the same solution qualities. No significant differences among the tested methods in terms of solution quality can be observed (Table 4). Especially for the Case I, IV, and V, where the both ParamILS and MParamILSs obtained the same best objective values as the Exhaustive method with zero standard deviations. This indicates that ParamILS and MParamILSs were able to match the performance of the Exhaustive method, and high-quality solutions were consistently produced by both heuristic methods. In terms of computing time, in average for all test cases, the Exhaustive method spent substantially longer time (about 75% more) compared to ParamILS of which the required computing times for all the test cases were the second longest (Table 5).

Second, comparisons of performances between ParamILS and MParamILSs were conducted to investigate how much MParamILSs can save time compared to ParamILS. According to the statistical results, the computing times were significantly different among the tested methods for all the test cases, and MParamILSs was always faster than ParamILS (Table 5). On average when compared to ParamILS, MParamILS2 reduced more than 60% of the computing time, MParamILS3 and MParamILS4 reduced more than 70% of the computing time. This demonstrates, for all tested problem cases, that MParamILSs required significant shorter time than ParamILS. It can also be observed that MParamILS$_2$ required longer time than MParamILS$_3$, followed by MParamILS$_4$, which indicates that larger time savings were obtained with MParamILS using more coarser level problems.

Next, comparisons in terms of computing time variation were made among MParamILSs with ParamILS as the baseline method, in order to investigate the performance of MParamILS using different number of coarser level problems. Fig. 5 displays five box plots of computing time distributions of 10 runs of the methods on each test case. Results show that ParamILS had largest computing time variations compared to MParamILSs for all the test cases. Moreover, time variations among MParamILSs were similar, and varied for different test cases. For example in Case III, MParamILS$_3$ had larger variation than MParamILS$_4$, whereas an opposite result can be observed in Case IV. This shows that (1) MParamILSs was more stable (less variation) than ParamILS in terms of computing time, and (2) time variations among MParamILSs were small and not related to the number of coarser level problems used.

Finally, we conducted significance statistical analyses on both objective values and computing times considering all test cases (Diffograms in Fig. 6(a)). The vertical and horizontal axes in the Diffograms were marked with the least-square means. Coloured lines indicate if there is a significant difference between two methods. When a coloured line crosses the reference line (the diagonal dashed line), the least-square means associated with the center point (joint point) of the line are not significantly different. The results were consistent between any combination of methods that there were no significant differences in objective value (Fig. 6(a)). This indicates MParamILSs was able to match ParamILS in solution quality for all the test cases. On the other hand, results show that computing times were significantly different among ParamILS and MParamILSs (Fig. 6(b)). This difference was largest between ParamILS and each MParamILS and relatively small among MParamILSs. This corresponds with the results in Table 5 that MParamILSs required substantially less computing time than ParamILS for all the test cases.

6. Conclusion

In this study, a multilevel parameter configuration scheme (MParamILS) has been developed to configure ACO-based algorithms for solving TPPs modelled with underlying weighted networks. It combines a graph coarsening technique and a modified ParamILS procedure. MParamILS coarsens a large problem into smaller problems and uses the ParamILS to select high-quality parameter combinations from the smaller problems. The selected parameter combinations are then used to solve the original problem. As a result, the computing time is significantly reduced by evaluating a larger number of parameter combinations on less complex (smaller) problems and evaluating fewer number of parameter combinations on more complex (larger) problems. MParamILS was applied to five test cases and compared with

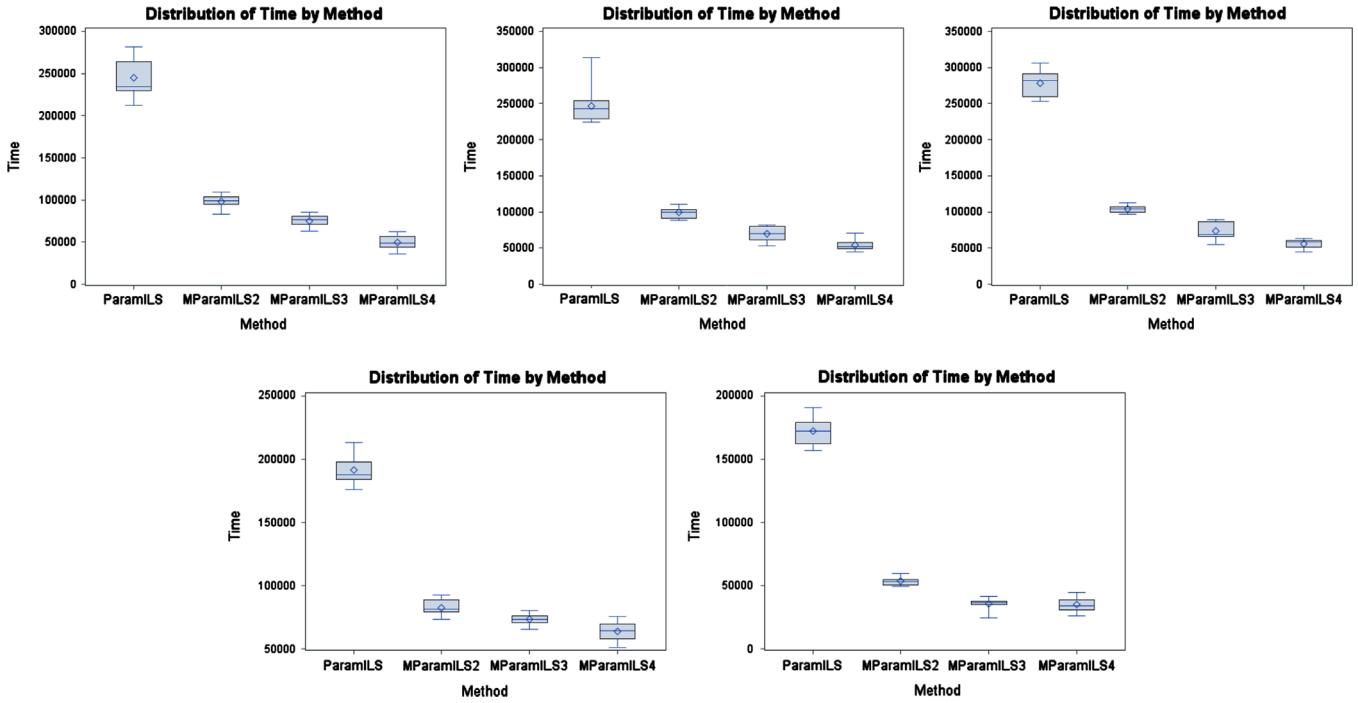

Fig. 5. Distribution of computing times of ParamILS and MParamILSs over 10 runs for each test case. (a) Computing time distribution of Case I, (b) computing time distribution of Case II, (c) computing time distribution of Case III, (d) computing time distribution of Case IV and (e) computing time distribution of Case V.

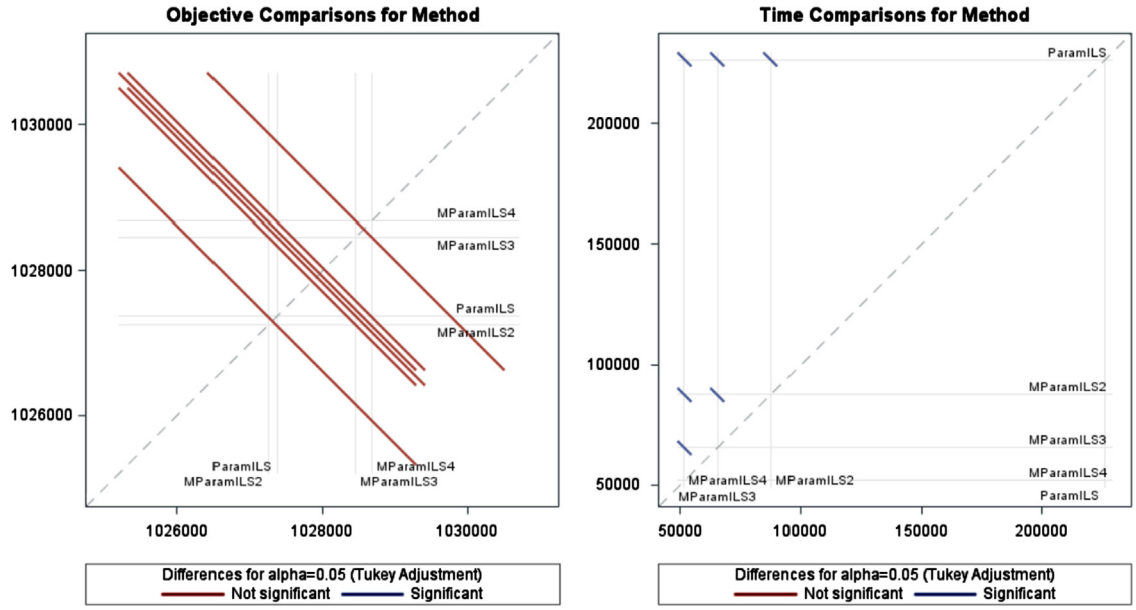

Fig. 6. Diffograms illustrating the objective and time significance comparisons between ParamILS and MParamILSs. (a) Significance comparison on objective value among methods and (b) significance comparison on computing time among methods.

ParamILS and the Exhaustive method in terms of solution quality and computing time. Results show that MParamILS was able to match solution qualities obtained by the other methods and reduce computing times substantially for all test cases.

Moreover, savings in computing time increased with the number of coarser level problems used in MParamILS. Consideration should be given to select an appropriate number of increasingly coarser problems used in MParamILS. A large number of coarser problems might result in low-quality parameter combinations as the structure of the coarsest problem might differ too much from the original problem. On the other hand, a small number of coarser problems can result in MParamILS requiring a longer computing time. In addition, the graph coarsening technique presented in this study is designed to produce coarser level graphs by contracting finer level edges with large attribute weights. This is an important property for solving minimisation problems. However, depending on different applications, weights on contracted edges can be calculated differently. Analogously, parameter configurators other than ParamILM in the proposed multilevel scheme can be considered. Lastly, although we applied MParamILS to configure ACO algorithms, it can be extended to any parameterized algorithms.

References

[1] R.G. Askin, S.H. Cresswell, J.B. Goldberg, A.J. Vakharia, A Hamiltonian path approach to reordering the part-machine matrix for cellular manufacturing, Int. J. Prod. Res. 29 (6) (1991) 1081–1100.

[2] P. Balaprakash, M. Birattari, T. Stützle, Improvement strategies for the f-race algorithm: sampling design and iterative refinement, in: Hybrid Metaheuristics, Springer, 2007, pp. 108–122.

[3] L. Bianchi, L.M. Gambardella, M. Dorigo, An ant colony optimization approach to the probabilistic traveling salesman problem, in: Parallel Problem Solving from Nature PPSN VII, Springer, 2002, pp. 883–892.

[4] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, et al., 2002. A racing algorithm for configuring metaheuristics. in: GECCO, vol. 2. Citeseer, pp. 11–18.

[5] M. Birattari, Z. Yuan, P. Balaprakash, T. Süttzle, F-race and iterated f-race: an overview, in: T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (Eds.), Experimental Methods for the Analysis of Optimization Algorithms, Springer, Berlin Heidelberg, 2010, pp. 311–336.

[6] C. Blum, Ant colony optimization: introduction and recent trends, Phys. Life Rev. 2 (4) (2005) 353–373.

[7] N. Blum, A New Approach to Maximum Matching in General Graphs, Springer, Berlin Heidelberg, 1990.

[8] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence, Oxford, 1999.

[9] E. Bonabeau, M. Dorigo, G. Theraulaz, Inspiration for optimization from social insect behaviour, Nature 406 (6791) (2000) 39–42.

[10] J. Branke, J.A. Elomari, Meta-optimization for parameter tuning with a flexible computing budget, in: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference, ACM, 2012, pp. 1245–1252.

[11] J. Branke, M. Guntsch, New ideas for applying ant colony optimization to the probabilistic TSP, in: Applications of Evolutionary Computing, Springer, 2003, pp. 165–175.

[12] M.A. Contreras, W. Chung, G. Jones, Applying ant colony optimization metaheuristic to solve forest transportation planning problems with side constraints, Can. J. For. Res. 38 (11) (2008) 2896–2910.

[13] G. Di Caro, F. Ducatelle, L.M. Gambardella, Anthocnet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks, Eur. Trans. Telecommun. 16 (5) (2005) 443–455.

[14] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE Comput. Intell. Mag. 1 (4) (2006) 28–39.

[15] M. Dorigo, G. Di Caro, L.M. Gambardella, Ant algorithms for discrete optimization, Artif. Life 5 (2) (1999) 137–172.

[16] H. Duan, G. Ma, S. Liu, Experimental study of the adjustable parameters in basic ant colony optimization algorithm, in: IEEE Congress on Evolutionary Computation, CEC 2007, IEEE, 2007, pp. 149–156.

[17] F. Ducatelle, G. Di Caro, L.M. Gambardella, Using ant agents to combine reactive and proactive strategies for routing in mobile ad-hoc networks, Int. J. Comput. Intell. Appl. 5 (02) (2005) 169–184.

[18] D. Favaretto, E. Moretti, P. Pellegrini, On the explorative behavior of max–min ant system, in: Engineering Stochastic Local Search Algorithms, Designing, Implementing and Analyzing Effective Heuristics, Springer, 2009, pp. 115–119.

[19] G. Francesca, P. Pellegrini, T. Stützle, M. Birattari, Off-line and on-line tuning: a study on operator selection for a memetic algorithm applied to the QAP, in: Evolutionary Computation in Combinatorial Optimization, Springer, 2011, pp. 203–214.

[20] H.N. Gabow, An efficient implementation of Edmonds' algorithm for maximum matching on graphs, J. ACM (JACM) 23 (2) (1976) 221–234.

[21] D. Gaertner, K.L. Clark, On optimal parameters for ant colony optimization algorithms, in: IC-AI. Citeseer, 2005, pp. 83–89.

[22] W.J. Gutjahr, S-aco: an ant-based approach to combinatorial optimization under uncertainty, in: Ant Colony Optimization and Swarm Intelligence, Springer, 2004, pp. 238–249.

[23] B. Hendrickson, R.W. Leland, A multi-level algorithm for partitioning graphs, Super Computing 95 (1995) 28.

[24] W.M. Hirsch, G.B. Dantzig, The fixed charge problem, Naval Res. Logist. Quart. 15 (3) (1968) 413–424.

[25] F. Hutter, H.H. Hoos, K. Leyton-Brown, T. Stützle, Paramils: an automatic algorithm configuration framework, J. Artif. Intell. Res. 36 (1) (2009) 267–306.

[26] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.

[27] M. Khichane, P. Albert, C. Solnon, An aco-based reactive framework for ant colony optimization: first experiments on constraint satisfaction problems, in: Learning and Intelligent Optimization, Springer, 2009, pp. 119–133.

[28] R.J. Lipton, R.E. Tarjan, A separator theorem for planar graphs, SIAM J. Appl. Math. 36 (2) (1979) 177–189.

[29] M. López-Ibánez, T. Stützle, The automatic design of multiobjective ant colony optimization algorithms, IEEE Trans. Evol. Comput. 16 (6) (2012) 861–875.

[30] M. Lopez-Ibanez, T. Süttzle, Automatically improving the anytime behaviour of optimisation algorithms, Eur. J. Oper. Res. 235 (3) (2014) 569–582.

[31] R. Montemanni, L.M. Gambardella, A.E. Rizzoli, A.V. Donati, Ant colony system for a dynamic vehicle routing problem, J. Comb. Optim. 10 (4) (2005) 327–343.

[32] P. Pellegrini, T. Stützle, M. Birattari, Off-line vs. on-line tuning: a study on maxmin ant system for the TSP, in: Swarm Intelligence, Springer, 2010, pp. 239–250.

[33] A. Radulescu, M. López-Ibánez, T. Stützle, Automatically improving the anytime behaviour of multiobjective evolutionary algorithms, in: Evolutionary Multi-Criterion Optimization, Springer, 2013, pp. 825–840.

[34] C. Solnon, K. Ghédira, Ant colony optimization for multi-objective optimization problems, in: 2012 IEEE 24th International Conference on Tools with Artificial Intelligence, vol. 1, IEEE Computer Society, 2007, pp. 450–457.

[35] T. Stützle, M. Dorigo, ACO algorithms for the traveling salesman problem, in: Evolutionary Algorithms in Engineering and Computer Science, 1999, pp. 163–183.

[36] T. Stützle, H.H. Hoos, MAX–MIN ant system, Future Gener. Comput. Syst. 16 (8) (2000) 889–914.

[37] T. Stützle, M. López-Ibánez, P. Pellegrini, M. Maur, M.M. de Oca, M. Birattari, M. Dorigo, Parameter adaptation in ant colony optimization, in: Autonomous Search, Springer, 2012, pp. 191–215.

[38] S.-H. Teng, Coarsening, sampling, and smoothing: elements of the multilevel method, in: Algorithms for Parallel Processing, Springer, 1999, pp. 247–276.

[39] V. Torczon, On the convergence of pattern search algorithms, SIAM J. Optim. 7 (1) (1997) 1–25.

[40] C. Walshaw, Multilevel refinement for combinatorial optimisation problems, Ann. Oper. Res. 131 (1–4) (2004) 325–372.

[41] B. Yu, Z.-Z. Yang, B. Yao, An improved ant colony optimization for vehicle routing problem, Eur. J. Oper. Res. 196 (1) (2009) 171–176.

[42] M. Zaslavskiy, F. Bach, J.-P. Vert, A path following algorithm for the graph matching problem, IEEE Trans. Pattern Anal. Mach. Intell. 31 (12) (2009) 2227–2242.