Grammatical Evolution Hyper-heuristic for Combinatorial Optimization problems

Nasser R. Sabar, Masri Ayob, Graham Kendall, *Senior Member, IEEE* and Rong Qu, *Member, IEEE*

***Abstract*— Designing generic problem solvers that perform well across a diverse set of problems is a challenging task. In this work, we propose a hyper-heuristic framework to automatically generate an effective and generic solution method by utilizing grammatical evolution. In the proposed framework, grammatical evolution is used as an online solver builder, which takes several heuristic components (e.g. different acceptance criteria and different neighborhood structures) as inputs and evolves templates of perturbation heuristics. The evolved templates are improvement heuristics which represent a complete search method to solve the problem at hand. To test the generality and the performance of the proposed method, we consider two well-known combinatorial optimization problems; exam timetabling (Carter and ITC 2007 instances) and the capacitated vehicle routing problem (Christofides and Golden instances). We demonstrate that the proposed method is competitive, if not superior, when compared to state of the art hyper-heuristics, as well as bespoke methods for these different problem domains. In order to further improve the performance of the proposed framework we utilize an adaptive memory mechanism which contains a collection of both high quality and diverse solutions and is updated during the problem solving process. Experimental results show that the grammatical evolution hyper-heuristic, with an adaptive memory, performs better than the grammatical evolution hyper-heuristic without a memory. The improved framework also outperforms some bespoke methodologies which have reported best known results for some instances in both problem domains.**

***Index Terms*—Grammatical Evolution, Hyper-heuristics, Timetabling, Vehicle Routing**

I. Introduction

Combinatorial optimization can be defined as the problem of finding the best solution(s) among all those available for a given problem [1]. These problems are encountered in many real world applications such as scheduling, production planning, routing, economic systems and management [1]. Many real world optimization problems are complex and very difficult to solve. This is due to the large, and often heavily constrained, search spaces which make their modeling (let alone solving) a very complex task [2]. Usually, heuristic methods are used to solve these problems, as exact methods often fail to obtain an optimal solution in reasonable times. The main aim of heuristic methods, which provide no guarantee of returning an optimal solution (or even near optimal solution), is to find a reasonably good solution within a realistic amount of time [3, 4]. Meta-heuristic algorithms provide some high level control strategy in order to provide effective navigation of the search space. A vast number of meta-heuristic algorithms, and their hybridizations, have been presented to solve optimization problems. Examples of meta-heuristic algorithms include scatter search, tabu search, genetic algorithms, genetic programming, memetic algorithms, variable neighborhood search, guided local search, GRASP, ant colony optimization, simulated annealing, iterated local search, multi-start methods and parallel strategies [3],[4].

Given a problem, an interesting question that comes to mind is:

> *Which algorithm is the most suitable for the problem at hand and what are the optimal structures and parameter values?*

The most straightforward answer to the above question might be to employ trial-and-error to find the most suitable meta-heuristic from the large variety of those available, and then employ trial-and-error to determine the appropriate structures and parameter values. While these answers seem reasonable, in terms of the computational time involved, it is impractical in many real world applications. Many bespoke meta-heuristic algorithms that have been proposed over the years are manually designed and tuned, focusing on producing good results for specific problem instances. The manually designed algorithms (customized by the user and not changed during problem solving) that have been developed over the years are problem specific, i.e. they are able to obtain high quality results for just a few problem instances, but usually fail on other instances even of the same problem and cannot directly be applied to other optimization problems. Of course, the No Free Lunch Theorem [5] states that a general search method does not exist, but it does not mean that we cannot investigate *more general* search algorithms to explore the limits of such an algorithm [6-8].

Nasser R. Sabar and Masri Ayob are with Data Mining and Optimization Research Group (DMO), Centre for Artificial Intelligent (CAIT), Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia.
email:naserdolayme@yahoo.com, masri@ftsm.ukm.my
Graham Kendall and Rong Qu are with ASAP Research Group, School of Computer Science, The University of Nottingham, Nottingham NG8 1BB, UK. email: Graham.Kendall@nottingham.ac.uk, Rong.Qu@nottingham.ac.uk.
Nasser R. Sabar and Graham Kendall are with The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih, Selangor, Malaysia.
email: Graham.Kendall@nottingham.edu.my.

Numerous attempts have been made to develop automated search methodologies that are able to produce good results across several problem domains and/or instances. Hyper-heuristics [6], meta-learning [9], parameter tuning [10], reactive search [11], adaptive memetic algorithms [12] and multi-method [13], are just some examples. The performance of any search method critically depends on its structures and parameter values [6]. Furthermore, different search methodologies, coupled with different structures and parameter settings may be needed to cope with problem instances or different problem domains [9],[10]. A search may even benefit from adapting as it attempts to solve a given instance. Therefore, the performance of any search method may be enhanced by automatically adjusting their structures or parameter values during the problem solving process. Thus, the ultimate goal of automated heuristic design is to develop search methodologies that are able to adjust their structures or parameter values during the problem solving process and work well, not only across different instances of the same problem, but also across a diverse set of problem domains [6], [9], [10].

Motivated by these aspects, particularly the hyper-heuristic framework [6], in this work, we propose a grammatical evolution hyper-heuristic framework (GE-HH) to generate local search templates during the problem instance solving process, as depicted in Fig 1.

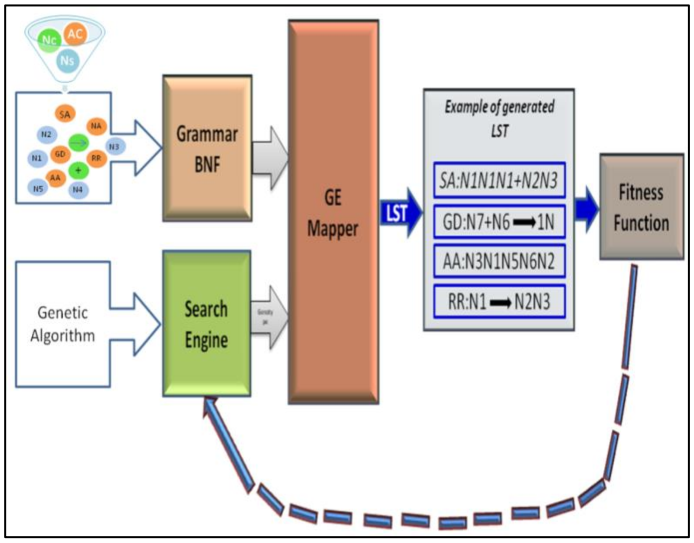

Fig.1.The GE-HH framework

The evolved templates represent a complete local search method which contains the acceptance criteria of the local search algorithm (to determine away of escaping from local optima), the local search structures (neighborhoods), and their combination. The GE-HH operates on the search space of heuristic components, instead of the solution space. In addition, GE-HH also maintains a set of diverse solutions, utilizing an adaptive memory mechanism which updates the solution quality and diversity as the search progresses. We choose grammatical evolution to search the space of heuristic components due to its ability to represent heuristic components and it being able to avoid the problem of code bloat that is often encountered in traditional genetic programming. Our objectives are:

- To design an automatic algorithm that works well across different instances of the same problem and also across two different problem domains.
- To merge the strengths of different search algorithms in one framework.
- To test the generality and consistency of the proposed method on two different problem domains.

The performance and generality of the GE-HH is assessed using two well-known NP-hard combinatorial optimization problems; examination timetabling (Carter [14] and ITC 2007 [15] instances) and the capacitated vehicle routing problem (Christofides [16] and Golden [17] instances). Although both domains have been extensively studied by the research community, the reasons of choosing them are twofold. Firstly, they represent real world applications and the state of the art results, we believe, can still be improved. Currently, a variety of algorithms have achieved very good results for *some* instances. However, most methodologies fail on generality and consistency. Secondly, these two domains have been widely studied in the scientific literature and we would like to evaluate our algorithm across two different domains that other researchers have studied. Although our intention is not to present an algorithm that can beat the state of the art, but rather can work well across different domains, our results demonstrate that GE-HH is able to update the best known results for some instances.

The remainder of the paper is organized as follows: the generic hyper-heuristic framework and its classification are presented in Section II. The grammatical evolution algorithm is presented in Section III, followed by our proposed GE-HH framework in Section IV. The experimental results and result comparisons are presented in Section V and VI, respectively. Finally discussions and concluding remarks are presented in Sections VII and VIII.

II. Hyper-heuristics

Meta-heuristics are generic search methods that can be applied to solve combinatorial optimization problems. However, to find high quality solutions, meta-heuristics often need to be designed and tuned (as do many classes of algorithms, including those in this paper) and they are also often limited to one problem domain or even just a single problem instance. The objective for a solution methodology that is independent of the problem domain, serves as one of the main motivations for designing hyper-heuristic approaches [6],[18].

Recently, significant research attention has been focused on hyper-heuristics. Burke et al. [6] defined hyper-heuristics as

An automated methodology for selecting or generating heuristics to solve hard computational search problems.

One possible hyper-heuristic framework is composed of two levels, known as *high* and *low* level heuristics (see Fig.2).

The *high* level heuristic is problem independent. It has no knowledge of the domain, only the number of heuristics that are available and (non-domain) statistical information that is allowed to pass through the domain barrier. Only the lower part of the framework has access to the objective function, the problem representation and the low level heuristics that have