Subheuristic Search and Scalability in a Hyperheuristic

Robert E Keller, Riccardo Poli
Dept. of Computing and Electronic Systems, University of Essex, Colchester, United Kingdom
robert.e.keller@gmx.net, rpoli@essex.ac.uk

ABSTRACT

Our previous work has introduced a hyperheuristic (HH) approach based on Genetic Programming (GP). There, GP employs user-given languages where domain-specific local heuristics are used as primitives for producing specialised metaheuristics (MH). Here, we show that the GP-HH works well with simple generic languages over subheuristic primitives, dealing with increases of problem size and reduction of resources. The system produces effective and efficient MHs that deliver best results known in a chosen test domain. We also demonstrate that user-given, modest domain information allows the HH to produce an improvement over a previous best result from the literature.

Categories and Subject Descriptors Artificial Intelligence [**Problem Solving, Control Methods, and Search**]: Heuristic methods **General Terms** Algorithms Experimentation Languages

1. INTRODUCTION

A *hyperheuristic* is a heuristic that builds MHs that solve a given problem. In [2] we proposed a GP-HH that allows its user to define different domain-specific target languages in which the HH expresses its evolved MHs, making it a more generic solver. Here, we suggest that the GP-HH may further improve its search behaviour, by using elementary components of local heuristics, so that it evolves its MHs by also employing these basic, *subheuristic* primitives, and by using user-provided knowledge.

The GP-HH accepts the definition of a language in which it expresses MHs for D, an arbitrary domain of problems. To give such a description, one may represent search methods for D, e.g., low-level heuristics or proven MHs, as components of a grammar, G, that produces its language, $\mathsf{L}(G)$. In this manner, $\in \mathsf{L}(G)$ defines a MH for D. Then, any form of grammar-based GP over $\mathsf{L}(G)$ is a HH for D. An individual MH is represented as $g \in \mathsf{L}(\mathrm{G})$. T shall designate the set of terminals of G. $\mathsf{L}(\mathrm{G}) \subset \mathbf{T}^*$, the set of all strings over T. We call a terminal $t \in T$ a *primitive*. Primitives may represent manually created MHs, local heuristics, or *subheuristics*, i.e., parts of local heuristics.

Starting a run of the GP-HH, *initialisation* produces random sequences from T^* that have the same length. Until some termination criterion is met, selection, reproduction, and mutation of a MH take place in an iterative fashion. If a sequence $\notin \mathsf{L}(\mathrm{G})$, *EDITING* [2] will turn it into a string from $\mathsf{L}(G)$. A MH, M, holds an individual repetition value, , that determines how often an iterative primitive of M is repeated at most. The GP-HH co-evolves the population and its values. To that end, the used flavour of *tournament selection* favours higher fitness, as usual, as well as lower values.

```
metaheuristic       ::= NATURAL
                      | NATURAL search
search              ::= heuristic
                      | heuristic search
heuristic ::= 2-CHANGE
            | loop IF_2-CHANGE
            | loop IF_3-CHANGE
loop      ::= REPEAT_UNTIL_IMPROVEMENT
            | /* empty */
```

Figure 1: **Description of language *Complete*.**

```
heuristic ::= 2-CHANGE
            | loop IF_2-CHANGE
            | loop IF_3-CHANGE
            | SWAP_NODE
```

Figure 2: **Language *Swap*. Other rules as given in Figure 1.**

2. RESULTS

The set of travelling salesperson problems (TSP) is an appropriate domain for experiments. With n nodes of a problem given, one describes a cycle (tour) as a permutation of nodes, $p = (v_0, ..., v_{n-1})$, over $\{0, ..., n-1\}$. We call permutation $(0, 1, ..., n-1)$ the *natural* cycle of the problem. The primitive `NATURAL` creates this cycle. For a tour, the low-level heuristic `2-CHANGE`, when given two edges $(a,b),(c,d) : a \neq d, b \neq c$, replaces them with $(a,c),(b,d)$. Therefore, when a produced MH M is about to call `2-CHANGE`, it randomly selects two appropriate edges as arguments for `2-CHANGE`. Another primitive, `IF_2-CHANGE`, executes `2-CHANGE` only if this will improve the tour under construction. In analogy, we also supply a heuristic that we call `IF_3-CHANGE`. Eventually, we give the primitive `REPEAT_UNTIL_IMPROVEMENT` p that executes the primitive p until this leads to a better result or until p has been executed $_M$ times. A grammar, *Complete*, using primitives described above, is shown in Figure 1.

For a permutation (tour), the simplest subheuristic randomly selects a node and swaps it with one of its direct neighbours, here, its right one. We call this operator `SWAP_NODE`. Figure 2 shows the rule resulting from the addition of `SWAP_NODE`. We call the associated language *Swap*.

We consider problem `eil51` from TSPLIB. Its dimension is $n = 51$ nodes, and its best high-precision solution known has a length of 428.871765 as discovered by a MH evolved by the GP-HH [2]. We summarise experimental parameters in the caption of Table 1. During the execution of a MH, we count each of its calls to a primitive and call the sum . We give performance results in Table 1. Search effectiveness (col. "mean best") and reliability ("S.D.") of the GP-HH under language *Swap* are slightly worse than under *Complete*. We explain this by the random nature of `SWAP_NODE`. However, top-

Table 1: **Effectiveness over *Complete* and *Swap*, 100 runs each, max. iteration value $_{max} = 1{,}000$. Parameters: Popul. size 100, Genotype size 500, Offspring 100,000, Mut. prob. 0.5. P.%: Mean best or natural length in % of best low-precision result known from literature [1], $= 428.87$. Best: shortest length over all runs.**

eil51	Mean best	S.D.	Best	P.%
Nat. length	1,313.74	n.a.	n.a.	206.26
Complete	428.9	0.17	**428.872**	0.006
Swap	429.98	1.29	**428.872**	0.26

Table 2: **Effectiveness over *Swap*. Parameters: Popul. size 100; Offspring 1,000,000; Mut. prob. 0.5; $_{max}$ 2,000. P.%: Mean best or natural length N_{76} in % of best high-precision results known from literature: $_{76} = 544.36908$ [2]; `eil101`: $_{101} = 640.975$ [1]. *r* runs. g: genotype size**

eil	Mean	S.D.	Best	P.%	g	*r*
N_{76}	1,974.71	n.a.	n.a.	262.75	n.a.	n.a.
76	545.38	1.001	$_{76}$	0.19	900	100
101	645.12	1.2	641.697	0.65	1,800	32

Table 3: **Efficiency regarding Table 2. and : means over and values.**

Swap	P.%		S.D.		S.D.
eil76	0.19	1,371.3	491.3	43,879.3	20,570.4
eil101	0.65	1,973	0	107,429	0

Table 4: **Effectiveness and efficiency over *Swap*. Parameters: Table 2, genotype size 800. P.%: Mean best in % of $_{76}$. and : means over and values of those metaheuristics that find best result known.**

eil76	Mean best	S.D.	Best	P.%	Runs
Swap	545.83	1.09		0.27	68

P.%		S.D.		S.D.
0.27	1,354.7	485.99	39,146	18,052.2

quality MHs ("Best") are still easily found, and effectiveness is well within $1/3\%$ of the best result known.

Next, we consider `eil76`, a 76-node problem, and `eil101`. Parameter values and results are given in Table 2. For `eil76`, with genotype size $g = 900$, one sees that the best solution known and other, very good solutions are found with high reliability, so that the average best is within $1/5\%$ of the best solution known. For `eil101`, we double the genotype size, while the search-space size increases by about factor 10^{48}. Still, the best solution is around 0.1% of $_{101}$, the best high-precision result known, and the mean best is within $2/3\%$ of the best solution known. We conclude that, with a comparatively modest increase of resources, effectiveness scales up favourably and stays very high. Table 3 gives results as to efficiency: the use of non-greedy `SWAP_NODE` does not come with unfavourable scaling of the search effort. For `eil101`, compared to `eil76`, the best evolved MH compensates for the much larger space with an effort (" ") that is but 2.4 times higher.

The GP-HH can maintain its good performance with fewer resources, as follows. We shorten the maximal genotype size from 900 to 800. Table 2 gives the other parameter values. Table 4 shows results. One sees from a comparison with Table 2 that overall effectiveness and peak effectiveness practically have not changed. Also, the constant effectiveness does not come with a loss in efficiency. This follows from a comparison of Table 4 and Table 3 that displays the efficiency under genotype size 900.

To gain another subheuristic, we break up `IF_2-CHANGE`. Its conditional component is generic regarding domains and some problem-specific primitive, `p`: if `p` improves a solution, execute `p`. We call this primitive `IF_improve` and use it for describing a language, *Ifi* (see Figure 3). We perform experiments over this language, applying the parameter values from Table 2, using genotype size 900. Table 5 gives results. For `eil76`, one sees that the best solution known and other, very good solutions are found, so that the average best is within $1/2\%$ of the best solution known. Thus, language *Ifi*, allowing for more flexibility in balancing greedy vs randomised search, still supports effectiveness of the metaheuristic search. For `eil101`, the mean best is within 3% of the best solution known. This is remarkable because no GP-HH parameter was changed, while the search space is significantly larger. For `eil101`$_d$, i.e., `eil101` approached with doubled genotype size ($g = 1{,}800$), the mean best is well within $4/3\%$ of the best solution known. The best solution is around 0.05% of the best solution known. So, here, the GP-HH, given comparatively few additional resources, produces MHs that deal very effectively with a much larger problem.

```
heuristic ::= 2-CHANGE
            | ifi SWAP_NODE
            | loop IF_2-CHANGE
            | loop IF_3-CHANGE
ifi       ::= IF_improve
            | /* empty */
```

Figure 3: **Language *Ifi*. Other rules as given in Figure 1.**

Table 5: **Effectiveness over *Ifi* and *NN*. Parameters: Table 2. P.%: Mean best over all runs in % of best high-precision result known for `eil76`: $_{76} = 544.36908$ [2]; `eil101`: $_{101} = 640.975$, `d198`: $_{198} = 15{,}876.38$ [1]. g: genotype size.**

Problems	Mean	S.D.	Best	P.%	g	r
76	546.85	1.14	**$_{76}$**	0.46	900	100
101	659.68	3.70	651.250	2.9	900	100
101_d	648.33	2.16	641.302	1.15	1,800	100
101_dNN	640.95	0.28	640.212	-0.004	1,800	24
$d198_d$NN	15,910.2	12.66	15,883.72	0.21	1,800	64

Next, we provide a *randomised nearest-neighbour* flavour `R` that the HH calls instead of `NATURAL`: randomly add a node n as first node of a permutation p under construction, add a non-added node m that is nearest to n, repeat adding step for m, and so forth, until completion of p. We call the corresponding language *NN* and the according experiment `eil101`$_{d^{\mathrm{NN}}}$ (Table 5). Overall effectiveness and reliability strongly improve compared to `eil101`$_d$ (language *Ifi*). In particular, one of the best evolved MHs, $\mu_{\ NN}$, improves over $_{101}$, delivered by a hand-crafted specialised algorithm, by about 0.12%. We give this new benchmark value with high precision: $_{101NN} = 640.211609$.

For `d198` (TSPLIB), a much larger problem, with unchanged GP-HH parameter values, effectiveness is also very good. In particular, the best found solution is within 0.05% of the best result known. Thus, the provided modest domain knowledge has supported efficiency. Furthermore, on average, a MH is produced and executed in a practical amount of time.

Eventually, when the GP-HH produces competitive, real-valued solutions on a TSP problem, it may easily keep hitting the provably global, integer optimum, as we have observed. Naturally, this happens because different real solutions may represent the same integer solution.

The authors acknowledge support from EPSRC (EP/C523377/1, EP/C523385/1) and thank the anonymous reviewers.

3. REFERENCES

[1] G. Jayalakshmi and S. Sathiamoorthy, et. al. An hybrid genetic algorithm. *International Journal of Computational Engineering Science*, 2001.

[2] R. E. Keller and R. Poli. Linear genetic programming of parsimonious metaheuristics. In D. Srinivasan et al., eds., *2007 IEEE CEC*, 2007.