

Self-adaptive Hyperheuristic and Greedy Search

Robert E. Keller

Riccardo Poli

Abstract—In previous work, we have introduced an effective and resource-efficient hyperheuristic that uses Genetic Programming as its search heuristic on the space of heuristics. Here, we show that the hyperheuristic performs better than purely greedy and even only mostly greedy flavours of hill climbing. We also introduce a generic principle that allows the hyperheuristic to automatically find good parameter values for its effective and efficient search.

I. INTRODUCTION

A metaheuristic (MH) is a heuristic that approaches a given problem by employing heuristics. The term *hyperheuristic* [24], see [25], [7] for its origin, refers to a heuristic that, given a problem, searches for a heuristic that solves the problem. In this paper, we present our hyperheuristic that uses Genetic Programming (GP) to evolve MHs from given heuristics.

Over the past few years, hyperheuristics (HH) have increasingly attracted research interest. For example, [6] suggests a method of building low-level heuristics for personnel scheduling, [4] proposes tabu search on the space of heuristics, [10] describes a timetabling application of a hyperheuristic, and [9] suggests simulated annealing as the learning strategy used by a hyperheuristic. [21] employs Genetic Programming (GP) [17], [2], [18], [23] for evolving evolutionary algorithms that are applied to problems of discrete optimisation. For the bin-packing problem, [5] introduces a hyperheuristic that is driven by GP.

While the approaches presented in these papers use fixed, problem-specific languages, we have recently proposed a linear GP hyperheuristic (GP-HH) [15], [16], [14] which allows the use of different target languages in which to express an evolved metaheuristic, making this HH a more generic solver. For the TSP domain, the hyperheuristic routinely produces metaheuristics that yield high-quality solutions (in some cases equivalent to the best-known ones), despite the simplicity of the low-level heuristics given to the hyperheuristic as building material.

An advantage of the approach is that domain knowledge becomes a free resource for the GP hyperheuristic that does not have to rediscover the provided component heuristics. Moreover, by designing a target language appropriately, one can direct evolutionary search towards promising types of metaheuristics. After all this is what is needed: because there is no fixed HH that works efficiently and well on all domains [27], it is important to develop optimisation methods

that are more flexible in their application to different practical domains.

As seen in our previous work, the demands on the user of the GP-HH are very modest in terms of sophistication of heuristics to be supplied to the hyperheuristic. As demonstrated in [14], the HH is also easy on its computing resources.

However, while our GP hyperheuristic is simpler than most hand-made domain-specific approaches, it still is more complex than other basic, yet often effective search methods, such as, for example, hill climbing.

Thus, we must address the question whether the effort of the hyperheuristic's development and use is justified. To that end, on the one hand, we must investigate whether there actually are performance and flexibility gains of the GP-HH over hill climbing. On the other hand, we must see to a reduction of required user interaction with the hyperheuristic. In particular, while the GP-HH itself puts only mild demands on the user in terms of the number of its parameters to be set, the evolved metaheuristics may also need parametrisation. For instance, in our previous work, we introduced a particularly success-critical parameter which controls execution flow in an evolved metaheuristic. For this reason, we must introduce an extension of the hyperheuristic that effectively relieves the user from determining an appropriate value for this parameter.

The paper is organised as follows. In Section II, we introduce the hyperheuristic in detail. In Section III, we describe the types of problem used in experiments with the hyperheuristic. In Section IV, we describe the language that we then use for the experiments described in Section V. In Section VI, we give a summary and conclusions, while in Section VII we describe interesting avenues for future work.

II. A LINEAR-GP HYPERHEURISTIC

Our GP hyperheuristic accepts the definition of the language of metaheuristics that may be desirable for D , an arbitrary, fixed domain of problems. Then, in principle, after changing this description appropriately, one can apply the HH to a different domain. To give such a definition, one may represent some of D 's low-level heuristics or well-known metaheuristics as components of the language that one describes by a grammar, G . In this manner, $\sigma \in L(G)$ defines a metaheuristic for D . Then, any form of grammar-based GP (e.g., [22], [26], [20], [13], [28], [11]), evolving programs from $L(G)$, is a hyperheuristic for D . Here, we describe our HH approach that makes use of linear GP [3].

A metaheuristic is represented as a genotype $g \in L(G)$ with a domain-specific grammar G . T shall designate the

Algorithm 1 GP-based HYPERHEURISTIC.

```
1: given: grammar  $G$ , population size  $p$ , length  $l$ 
2: repeat
3:   produce next random primitive-sequence  $\sigma : |\sigma| = l$ 
4:   EDITING( $\sigma, G$ )  $\rightarrow g$  genotype
5: until  $p$  genotypes created
6: while time available do
7:   Selection: 2-tournament
8:   Reproduction: Copy winner  $g$  into loser's place  $\rightarrow g'$ 
9:   Exploration: with a given probability
       Mutate copy  $g' \rightarrow \delta$ 
       EDITING( $\delta, G$ )  $\rightarrow g''$  genotype
10: end while
```

set of terminals of G . $L(G) \subset T^*$, the set of all strings over T . We call a terminal $t \in T$ a *primitive* to avoid confusion regarding “terminal” as used in the field of GP. Primitives may represent manually created metaheuristics, low-level heuristics, or parts of them. The execution of a metaheuristic, g , with $g = i_0 i_1 \dots i_n$, $i_j \in T$, means the execution of the i_j s one by one. This execution constructs a complete structure, s , that is a candidate solution to the given problem. More specifically, s is obtained from an initial, complete structure, $i_0()$: $s = i_n(\dots(i_1(i_0()))\dots) = g()$. All i_j with $j \neq 0$ accept a complete structure as input. All i_j deliver a complete structure as output. In particular, i_0 , in some straightforward fashion, delivers the initial structure $i_0()$. It follows that g 's *fitness* depends on the quality of s .

At the beginning of a run of the GP hyperheuristic (see Algorithm 1), *initialisation* produces p random primitive-sequences from T^* , where p is the population size. All such sequences are of the same length, l , e.g., $l=500$.

Mutation of a genotype $g \in L(G)$ randomly selects a locus, j , of g , and replaces the primitive at that locus, i_j , with a random primitive, $t \in T, t \neq i_j$.

Naturally, both initialisation and mutation may result in a primitive-sequence, $\sigma = i_0 \dots i_j \dots i_k \in T^*$, that is not a valid genotype, i.e., $\sigma \notin L(G) \subset T^*$. In this case, the sequence is passed to an operator called *EDITING* that is described in detail in [14]. It represents a simple way of turning a string from T^* into a string from $L(G)$ without introducing subsequences, which would add information that would not have been evolved. Note that, although the hyperheuristic initialises a population using sequences of a fixed length, l , the application of *EDITING*, as a side effect, leads to a population containing genotypes of variable lengths not longer than l . This variation in genotype size is, of course, desirable, as, in principle, it allows the evolution of parsimonious heuristics. We actually observed this effect and described it in [16]. Naturally, this also contributes to saving run time since a shorter genotype often executes faster.

III. PROBLEM DOMAIN

To compare greedy and hyperheuristic search, we select the set of travelling salesperson problems (TSP) [19] that

is NP-hard and, therefore, serves as a good representative of problem domains that are targets of hyperheuristics. In its simplest form, a TSP involves finding a minimum-cost *Hamiltonian cycle*, also known as “*tour*”, in a given, complete, weighted graph. Let the n nodes of such a graph be numbered from 0 to $n - 1$. Then, one describes a tour involving edges $(v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_0)$ as a permutation $p = (v_0, \dots, v_{n-1})$ over $\{0, \dots, n - 1\}$. We call permutation $(0, 1, \dots, n - 1)$ the *natural cycle* of the graph. The weight of an edge (i, j) represents the cost of travelling between i and j . Here, we shall interpret this cost as the distance between i and j . Thus, the shorter a tour, the higher its quality.

IV. LANGUAGE

We describe a domain-specific language that will support experimenting. To that end, we require a few simple routines, including basic heuristics, that are represented as primitives of the terminal set of the grammar. The primitive *NATURAL* designates the method that creates the natural cycle for a problem. The low-level heuristic *2-CHANGE* turns a tour H into a slightly different tour: when given two of H 's edges, $(a, b), (c, d) : a \neq d, b \neq c$, then *2-CHANGE* replaces them with $(a, c), (b, d)$. Therefore, when the hyperheuristic is about to call a *2-CHANGE* primitive, it randomly selects two appropriate edges, $(a, b), (c, d)$, as arguments for *2-CHANGE*.

Another primitive, *IF_2-CHANGE*, executes *2-CHANGE* only if this will shorten the tour under construction (note that *IF_2-CHANGE* is not identical to the more complex heuristic “2-opt” [8]).

Another low-level heuristic is known as a *3-change*: delete three mutually disjoint edges from a given tour, and reconnect the obtained three paths so that a different tour results. Given this method, we define a heuristic, called *IF_3-CHANGE*, that randomly selects edges as arguments for *3-change*; then, if *3-change* betters the cycle for the given arguments, *IF_3-CHANGE* actually executes *3-change*.

So far, only sequential and conditional execution of user-provided heuristics are available to evolved metaheuristics. A loop element is required for a set of necessary control structures. So, we introduce the primitive *REPEAT_UNTIL_IMPROVEMENT* p that executes its argument, a primitive p , until this has led to a better result or until p has been executed ι times, where ι is an important parameter that the user must define. We build a grammar over all of our primitives (see Figure 1).

These provided primitives form just a small set, and they represent low-level heuristics, i.e., heuristics that execute few, often merely one elementary operation, such as a single step in the underlying search space. In practice, only having to supply a poor set of primitives is particularly interesting, because a useful hyperheuristic must be able to produce good MHs for domains about which the user has no or little knowledge. Here however, we are dealing with a well-known domain, i.e., TSP problems. This is done so that the

```

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

```

Fig. 1. Grammar

performance of the GP-HH over the given, simple primitive set can be properly assessed by comparing the results, delivered by the evolved MHs, with known good results. Therefore, on the one hand, enriching this primitive set is of no interest here. On the other hand, in practice, i.e., when solving a given problem is of interest *per se*, one either already has a good solver at hand, so that there is no need for bringing in a hyperheuristic, or, if one would want to use our GP-HH, one would put high-level as well as low-level heuristics into its primitive set. Therefore, for the interest of the reader, we mention that, naturally, for long-standing challenges such as combinatorial or discrete optimisation, there is a legion of seasoned heuristics available. These heuristics come from many fields, e.g., tabu search, dynamic programming, or flavours of brute-force search. For TSP problems, for instance, among their most effective stand-alone solvers, there are metaheuristics that employ branch-and-bound/cut methods.

Note that literature often discusses a problem in an integer interpretation, i.e., for two cities, one uses their integer distance gained from rounding their real distance. In particular for small problems given in this interpretation, these MHs can quickly determine global optima. However, as tours of different real length can have the same integer length, there is the challenge of finding or improving over a best-known real-length tour. Below, we shall only deal with the real interpretation of a TSP problem.

V. EXPERIMENTS

For all experiments, mutation probability (cf. Algorithm 1, step 9) shall be 0.5, since this value turned out to provide good performance during a few initial tests of the hyperheuristic. Naturally, we cannot rule out that, given a certain problem and HH run, another value, or indeed different values for different stages of the run, would provide even better performance.

We consider problem `eil51` from [1]. Its dimension is $n = 51$ nodes. One of its best known real-valued solutions has a length of 428.871 [12], with natural length of approximately 1,313.47. Unfortunately, it is unclear whether the solution value is precise or has resulted from truncation or rounding. `eil51`'s best known real-valued, high-precision solution has a length of 428.871765 (not truncated), as

TABLE I
BASIC PARAMETERS

Popul. size	Genotype size	Offspring	Mut. prob.	ι
100	500	100,000	0.5	70

TABLE II
PARAMETERS OF HILL CLIMBER FOR `EIL51`.

Operation	Start	Iterations
2-CHANGE	natural cycle	6×10^{10}

discovered by a metaheuristic evolved by our hyperheuristic [16].

Mostly, for the discussion, all values delivered by the GP hyperheuristic, and their means, shall be rounded off to the nearest hundredth.

We shall only be dealing with real-valued tour lengths, which emphasises differences between very good vs. excellent results, also during tournament selection.

For a symmetrical TSP instance, the number of semantically different tours equals $(n-1)!/2$. However, the evolved metaheuristics operate on permutations of n nodes, so that they deal with a search space of a size of $n!$ permutations. Thus, $n = 51$ gives about 1.6×10^{66} search points and 1.52×10^{64} different tours.

Previous work [16] indicates that random search, the simplest form of search, cannot compete with the hyperheuristic. We are therefore interested to see how flavours of greedy search, still simpler than the hyperheuristic approach, fare against the latter. We implement a hill climber (HC) that randomly selects appropriate arguments for a local heuristic whose execution may improve the climber's best-so-far solution. Some flavours of this HC will only accept a newly found tour if it is better than the best so far, another variant will be less greedy.

A. Comparison between hyperheuristic and hill climbing

Initially, we merely provide the climber with a 2-CHANGE as its sole variation operator in order to see whether this can already reach or exceed the performance of the hyperheuristic approach. However, before we can do a comparison, we must identify a fair number of iterations to be done by the HC. To this end, we present the parameters that lead the GP hyperheuristic on `eil51` to producing metaheuristics that found the best known solution. One of these parameters is the grammar in Figure 1.

We summarise all numerical parameters in Table I. Based on these parameters, we can estimate how often, in the most favourable conditions, an evolved metaheuristic can call a local heuristic.

To be sure not to put the HC at a disadvantage, we make generous assumptions, so that this estimate will be greater than the largest actual number of calls. We can then use the estimate to determine an appropriate number of iterations that we allow the hill climber to make. The calculation is as follows.

The given, potential genotype size is 500. The actual genotype size is often smaller due to editing [16]. Also, not every component of an evolved metaheuristic is the loop construct of the underlying language. Furthermore, each loop call together with the primitive it iterates takes up two components of the metaheuristic (cf. Figure 1). For a safe estimate, we assume that it would be just one component. Thus, since loops iterate an instruction $\iota = 70$ times at most, we conclude that an evolved metaheuristic would call a local heuristic $500 \times 70 = 35,000$ times.

Naturally, in the GP hyperheuristic, through repeated mutation and selection, the MH population improves incrementally by learning from the success and failures of all individuals sampled during an HH run. We therefore must consider the number of all calls of a local heuristic made during a single run. As 100,100 (i.e., members of initial population plus later offspring) individuals are produced during the run, we estimate that it involves $100,100 \times 35,000 = 3.5035 \times 10^9$ calls. We round off this number and shall use 4×10^9 as our estimate.

The climber shall also start out with the natural cycle of the underlying problem, as does an evolved metaheuristic. Initially, we are interested in how well the climber is doing on `eil51` compared to the best known solution, when supplied with one of the most basic domain-specific heuristics, i.e., 2-CHANGE, as its only heuristic, as discussed above. As for the number of iterations of a single HC run, with regard to the simplicity of 2-CHANGE, we want to give the HC more than the 4×10^9 steps mentioned above. So, we decided that an HC run should perform 15 times more steps, i.e., 6×10^{10} operations. We summarise the basic HC parameters in Table II.

Table III gives the results produced by the HC over 100 independent runs. The hill climber comes up with a best result over all runs that is significantly worse than the best known solution. In order to see whether this depends on the number of iterations granted to the climber, we double this number. Thus, we get the parameters shown in Table IV. With these parameters and the same seed values for the random-number generator, the climber produces exactly the results from Table III which indicates that the climber gets regularly stuck in strong local optima. Repeating the experiment with a different set of seed values, where both sets are disjoint, did not help, as shown in Table V. We see that the climber’s respective values of mean best, standard deviation, and minimal tour length are close together.

We therefore conclude again, with higher confidence, that the climber is clearly more susceptible to premature convergence than the GP hyperheuristic, when provided with 2-CHANGE.

Thus, a simple form of purely greedy search with a basic local heuristic is unsatisfying. Naturally, the HH also uses other local heuristics. We therefore, next, provide the hill climber with the most powerful local heuristic used by the hyperheuristic, i.e., 3-change. In principle, now the HC is on par with the HH and could prove at least equally effective,

TABLE III
PERFORMANCE OF HILL CLIMBER OVER 100 RUNS. BASIC PARAMETERS AS GIVEN IN TABLE II. **P.%**: MEAN BEST OR NATURAL LENGTH IN TERMS OF % OF BEST KNOWN RESULT $\alpha = 428.871765$ AS FOUND BY OUR GP HYPERHEURISTIC. **Min**: BEST, I.E., SHORTEST LENGTH OVER ALL RUNS.

<code>eil51</code>	Mean best	S.D.	Min	P.%
Nat. length	1,313.47	n.a.	n.a.	206.26
Hill climber	459.361	9.8428	439.806825	7.10917

TABLE IV
ALTERNATIVE SET OF PARAMETERS FOR THE HILL CLIMBER ON `eil51` (INCREASED ITERATION NUMBER)

Operation	Start	Iterations
2-CHANGE	natural cycle	1.2×10^{11}

perhaps at the expense of more calls to local heuristics. A summary of the parameters is given in Table VI, while Table VII gives the results. One can see that, despite the high iteration number of the climber, its performance (“Best”) and reliability (“Mean best”, “S.D.”) are clearly worse than those of the hyperheuristic.

So, even with 3-change, the climber is clearly more susceptible to premature convergence than our GP hyperheuristic.

We are interested in whether or not this is also true for a larger search space. We therefore consider `eil76` [1], a 76-node problem with a size of about 1.9×10^{111} search points. We make the same argument as above regarding the number of iterations made by the hyperheuristic. The population size and potential genotype size stay at 100 and 500, respectively, but ι equals 2,000, and the number of offspring equals 10^6 .

So, an evolved metaheuristic can call a local heuristic about $500 \times 2,000 = 1,000,000$ times at most.

However, the reality is that, over all runs, after editing, the number of calls of `REPEAT_UNTIL_IMPROVEMENT` by a metaheuristic, as measured on `eil51`, was ca. 91, i.e., clearly less than one fifth of the maximal genotype size. The other primitives, i.e., local heuristics, scored call frequencies

TABLE V
PERFORMANCE OF HILL CLIMBER OVER 100 INDEPENDENT RUNS. SEED VALUES OF RANDOM-NUMBER GENERATOR PAIRWISE DIFFERENT FROM VALUES UNDERLYING TABLE III. BASIC PARAMETERS AS GIVEN IN TABLE IV. **P.%**: MEAN BEST OR NATURAL LENGTH IN TERMS OF % OF BEST KNOWN RESULT $\alpha = 428.871765$.

<code>eil51</code>	Mean best	S.D.	Min	P.%
Nat. length	1,313.47	n.a.	n.a.	206.26
Hill climber	460.99	11.49	438.95	7.49

TABLE VI
PARAMETERS OF ENHANCED HILL CLIMBER FOR `EIL51`.

Operation	Start	Iterations
3-CHANGE	natural cycle	6×10^{10}

TABLE VII

PERFORMANCE OF HILL CLIMBER OVER RUNS WITH 3-CHANGE AS SOLE LOCAL HEURISTIC. BASIC PARAMETERS AS GIVEN IN TABLE II. **P.%**: MEAN BEST OR NATURAL LENGTH IN TERMS OF % OF BEST KNOWN RESULT $\alpha = 428.871765$. **Best**: SHORTEST LENGTH OVER ALL RUNS. PERFORMANCE OF GP HYPERHEURISTIC OVER GIVEN LANGUAGE WITH PARAMETERS FROM TABLE I.

eil51	Mean best	S.D.	Best
Nat. length	1,313.47	n.a.	n.a.
Hill climber	453.39	7.54	439.27
GP-HH	436.61	3.07	428.87
eil51	P.%	iterations/run	runs
Nat. length	206.26	n.a.	n.a.
Hill climber	5.72	6×10^{10}	100
GP-HH	1.80	$\ll 4 \times 10^9$	100

TABLE VIII

PERFORMANCE OF HILL CLIMBER OVER RUNS WITH 2-CHANGE AS SOLE LOCAL HEURISTIC. BASIC PARAMETERS AS GIVEN IN TABLE IV. **P.%**: MEAN BEST OR NATURAL LENGTH IN TERMS OF % OF BEST KNOWN RESULT $\alpha = 544.36908$ AS FOUND BY OUR GP HYPERHEURISTIC. **Best**: SHORTEST LENGTH OVER ALL RUNS. **Last**: MEAN OVER THE LAST IMPROVEMENT NUMBER

eil76	Mean best	S.D.	Best	P.%
Nat. length	1,974.71	n.a.	n.a.	262.75
Hill climber	594.19	12.59	561.60	9.15
GP-HH	548.99	1.67	544.37	0.85
eil76	iterations/run	runs	Last	S.D.
Nat. length	n.a.	n.a.	n.a.	n.a.
Hill climber	1.2×10^{11}	90	13164.6	4287.24
GP-HH	$< 1.12 \times 10^{11}$	100	n.a.	n.a.

of 104, 97, 15, and 1 (NATURAL) [16]. Also, only in the late stages of a run of a metaheuristic (MH), a looping primitive might use up all of its ι iterations, since it terminates as soon as a tour is improved. From measurements, we know, as a rule of thumb, that $21 \times \iota$ is a good estimate for the number of *all* calls of primitives during execution of a MH. The frequencies and the rule echo effects from our use of the high mutation rate that relentlessly reinserts non-looping primitives into every individual of the population. We use the frequencies and the rule for our estimate here, i.e., regarding eil76.

Thus, a MH would call a local heuristic about $91 \times 2,000 + 104 + 97 + 15 + 1 = 182,217$ times; or, as a much more realistic estimate, this would be $21 \times 2,000 = 42,000$ times.

In summary, a more realistic yet still safe, upper bound for the number of calls of a local heuristic is $(182,217 + 42,000)/2 \approx 112,000$. As 1,000,100 individuals are produced during a run, we estimate that $1,000,100 \times 1.12 \times 10^5 = 1.120112 \times 10^{11}$ calls are made by a run of the GP hyperheuristic. So, for simplicity, we shall give the same number of iterations as before to the hill climber, i.e., 1.2×10^{11} (Table IV). To see if and when premature convergence sets in, we shall also compute, over all runs, the average running number of that iteration that found the last improvement of a run.

Table VIII gives the results for eil76. Again, the per-

TABLE IX

PERFORMANCE OF HILL CLIMBER OVER RUNS WITH 3-CHANGE AS SOLE LOCAL HEURISTIC. BASIC PARAMETERS AS GIVEN IN TABLE IV. **P.%**: MEAN BEST OR NATURAL LENGTH IN TERMS OF % OF BEST KNOWN RESULT $\alpha = 544.36908$ AS FOUND BY OUR GP HYPERHEURISTIC. **Best**: SHORTEST LENGTH OVER ALL RUNS.

eil76	Mean best	S.D.	Best	P.%
Nat. length	1,974.71	n.a.	n.a.	262.75
Hill climber	586.32	9.91	561.60	7.71
GP-HH	548.99	1.67	544.37	0.85
eil76	iterations/run	runs	Last	S.D.
Nat. length	n.a.	n.a.	n.a.	n.a.
Hill climber	1.2×10^{11}	76	177983	114343
GP-HH	$< 1.12 \times 10^{11}$	100	n.a.	n.a.

TABLE X

PERFORMANCE OF HILL CLIMBER OVER RUNS WITH 2-CHANGE AND 3-CHANGE AS SOLE LOCAL HEURISTICS. OTHER PARAMETERS AS GIVEN IN TABLE IV. **P.%**: MEAN BEST LENGTH IN TERMS OF % OF BEST KNOWN RESULT FROM LITERATURE, $\alpha = 544.36908$. **Best**: SHORTEST LENGTH OVER ALL RUNS.

eil76	Mean best	S.D.	Best	P.%
Hill climber	570.38	7.56	551.27	4.78
GP-HH	548.99	1.67	544.37	0.85
eil76	iterations/run	runs	Last	S.D.
Hill climber	1.2×10^{11}	100	232780	200610
GP-HH	$< 1.12 \times 10^{11}$	100	n.a.	n.a.

formance of the HC is unsatisfying despite the high number of iterations. The last improvement number clearly shows that the climber has not been interrupted while there was still some probability of improvement. We explain this bad performance partially by the simplicity of 2-CHANGE, and partially by the situation that the number of local optima rises exponentially with space size.

Since the GP hyperheuristic can use the potentially more powerful 3-CHANGE operator as a component of an evolved metaheuristic, we repeat the experiment with identical parameters, but the hill climber uses 3-CHANGE only, instead of 2-CHANGE only. The results are shown in Table IX. The mean best and standard deviation are somewhat better than in the previous experiment (see Table VIII), endorsing the well-known rule of thumb that 3-CHANGE can often introduce better improvement than 2-CHANGE. However, the “Best” values are identical, suggesting that strong local optima of the underlying problem trap purely greedy search. Also, again, the very low last-improvement number underscores that the climber has not been interrupted while it was making progress.

Since the GP hyperheuristic may use the 3-CHANGE and the 2-CHANGE heuristic as components of the same evolved metaheuristic, next, we rerun the experiment with the same parameter values, but the climber randomly uses either a 3-CHANGE or a 2-CHANGE as its next step, with equal probability. Table X gives the results. The climber shows clearly better effectiveness in terms of reliability when compared to the 2-CHANGE-only case. However, even the

TABLE XI

PERFORMANCE OF HILL CLIMBER OVER RUNS WITH 2-CHANGE AND 3-CHANGE AND NON-GREEDY 2-CHANGE AS LOCAL HEURISTICS. OTHER PARAMETERS AS GIVEN IN TABLE IV. **P.%**: MEAN BEST OR NATURAL LENGTH IN TERMS OF % OF BEST KNOWN RESULT FROM LITERATURE, $\alpha = 544.36908$. **Best**: SHORTEST LENGTH OVER ALL RUNS.

e_{il76}	Mean best	S.D.	Best	P.%
Nat. length	1,974.71	n.a.	n.a.	262.75
Hill climber	1,362.89	15.31	1327.84	150.36
GP-HH	548.99	1.67	544.37	0.85
e_{il76}	iterations/run	runs	Last	S.D.
Nat. length	n.a.	n.a.	n.a.	n.a.
Hill climber	1.2×10^{11}	20	5.48×10^{10}	2.82×10^{10}
GP-HH	$< 1.12 \times 10^{11}$	100	n.a.	n.a.

2/3-CHANGE-case is still very unsatisfying when compared to the hyperheuristic. Even the high standard deviation of the last-improvement number does not give hope that the climber would manage an improvement beyond its granted number of iterations.

So far, we conclude that purely greedy hill climbing does not compete well with the hyperheuristic. We therefore, next, allow the climber to use non-greedy 2-CHANGE on top of greedy 2- and 3-CHANGE used so far. In this manner, we give the climber all local, domain-specific heuristics that are available to the GP hyperheuristic. We assign equal probability, i.e., one third, to the use of each of these local heuristics as next climbing step. Table XI gives results.

As one sees, adding the non-greedy heuristic leads to abysmal performance of the hill climber in all respects, even when compared only to its earlier instances. This is because it cannot learn to apply the non-greedy 2-CHANGE only sparingly, as does the GP hyperheuristic [16].

The crucial information is in the last-improvement number. It still is smaller by one order of magnitude than the maximal iteration number of the climber, so that, again, the climber was not hampered by an insufficient number of iterations. However, the drastic rise of this number demonstrates the high significance of using non-greedy 2-CHANGE carefully.

B. Self-adaptive hyperheuristic

In the previous section we have seen that the hill climber is outperformed by the hyperheuristic. It appears that the higher complexity of the GP hyperheuristic is justified by clearly better search performance.

However, the hyperheuristic leaves its user with the considerable burden of manually adapting parameters. Notably, he or she must determine a favourable value of the critical parameter ι . This parameter is of particular interest because it controls the iteration of some given local heuristic, and such iteration is a typical heuristic itself, applicable to all problem domains. We therefore ask whether we can significantly improve the hyperheuristic by having it find appropriate ι values, in parallel to locating good metaheuristics.

We suggest a modification of the hyperheuristic. Each metaheuristic (GP individual) will have a personal ι which

TABLE XII

BASIC PARAMETER VALUES FOR MODIFIED HYPERHEURISTIC.

Pop.size	Geno.size	Offspring	Mut. prob.	Max. iteration no.
100	500	100,000	0.5	1,000

TABLE XIII

PERFORMANCE OF GP HYPERHEURISTIC OVER 100 RUNS, WITH INDIVIDUAL ι VALUE FOR A METAHEURISTIC. PARAMETERS AS GIVEN IN TABLE XII. **P.%**: MEAN BEST OR NATURAL LENGTH IN TERMS OF % OF BEST KNOWN REAL-VALUED RESULT FROM LITERATURE, $\alpha = 428.87$. **Best**: SHORTEST LENGTH OVER ALL RUNS. ι and γ : MEANS OVER ι AND γ VALUES OF THOSE METAHEURISTICS THAT FIND BEST KNOWN RESULT. LINES “GP-HHx” GIVE DATA GAINED UNDER CO-SELECTION FOR ι .

e_{il51}	Mean best	S.D.	Best	P.%
Nat. length	1,313.74	n.a.	n.a.	206.260
GP-HH	428.88	0.04	428.87	0.002
GP-HHx	428.90	0.17	428.87	0.006
	P.%	ι	S.D.	min
Nat. length	206.260	n.a.	n.a.	n.a.
GP-HH	0.002	626.4	263.87	29
GP-HHx	0.006	610.8	263.47	10
	P.%	γ	S.D.	min
Nat. length	206.260	n.a.	n.a.	n.a.
GP-HH	0.002	13065	6893	1629
GP-HHx	0.006	12965	6797	838

is used in all calls to REPEAT_UNTIL_IMPROVEMENT in that metaheuristic. During initialisation of the HH, the ι of each created metaheuristic M is set to a random value in the interval $[2, m]$, where m is a user-given, maximal iteration value. As part of M 's genotype, its ι may be subject to mutation. If the hyperheuristic has chosen an M for mutation, it will add a value to M 's ι . This value comes from the range $[-m, m]$. Should the resulting ι' be outside of $[2, m]$, the HH attempts another mutation of ι , and so forth, until $\iota' \in [2, m]$. This simple routine allows for large and small mutations of an individual ι .

We supply the modified HH with basic parameter values as given in Table I, except ι that is now dynamically evolved. We set $m = 1,000$. Table XII summarises the parameter values.

We are interested in understanding the potential of the introduced principle regarding effectiveness and efficiency of the search. To this end, during the execution of an evolved metaheuristic, we count each of its calls of a local search heuristic, i.e., NATURAL, 2-CHANGE, IF_2-CHANGE, and IF_3-CHANGE, and call the sum γ . Table XIII gives the results. Here, we focus on the lines labelled “GPHH”.

The best known result is found with high reliability (see columns “Mean best”, “S.D.”, “Best”, and “P.%”).

Column ι gives the mean over the individual ι values of those metaheuristics that find the best known result; ditto for γ .

The mean ι value is clearly lower than the user-given, maximal iteration number, which, together with the associated standard deviation, shows that the best evolved metaheuristics usually do not exhaust the available number of iterations.

This can be explained as an effect of selection: overly greedy use of iteration falls prey to premature convergence. So, such MHs do not perform as well as less greedy metaheuristics. Also, dividing mean γ by mean ι gives a value of about 21, the factor used above for estimating the number of iterations done by the HH. Eventually, the minima of ι and γ demonstrate the great potential of using individual iteration values: efficiency is increased without any reduction in effectiveness.

However, the mentioned potential, so far, is exploited only through a weak emergent effect, somewhat similar to drift. This may already suffice to yield acceptable efficiency.

Otherwise, we suggest a small alteration of the 2-tournament selection used by the hyperheuristic, in order to slightly boost this effect. We still draw two candidates from the population, uniformly at random. Then we check if the fitness of the second candidate is within an interval of radius w , centred on the fitness of the first candidate. If the fitness of the second candidate is outside the interval, selection occurs in the usual manner, with the better individual winning the tournament. Otherwise, the candidate with the lower personal ι wins; if the individuals have the same ι , the first candidate wins.

This modification gives a small selective pressure in favour of a low ι . The parameter w controls this pressure. We suggest to set w to around 1% of a critical fitness value that, here, is the best-known result. For simplicity, we round up and, for an experiment with otherwise unchanged parameter values, we therefore set $w = 5$. Table XIII gives the results for our new form of tournament selection. As one can see, the suggested slight co-selection for ι does not jeopardise effectiveness and reliability (see first four columns). However, it mildly but noticeably lowers the average individual ι of the top evolved metaheuristics, which also lowers the average γ value. This again means saving time in the hyperheuristic search, due to the multiplicative effect over many metaheuristics and their use of loops.

C. Reusability

In this work we have used simple, randomised primitives for building metaheuristics, e.g., 2-CHANGE and 3-CHANGE randomly select edges to be swapped. This approach means that the same produced MH may perform differently on different calls. Naturally, the user is interested in timely, good results, possibly even only in an acceptable solution to one particular problem from a given domain. If hyperheuristic search leads to such a solution fast enough, one can even use this search type for finding such a solution only once. So, here, if one is satisfied with a solution (tour) found in time by an evolved MH, one may never look at this disposable MH again. That is why we are interested in an efficient HH.

However, we also want to ensure evolution produces somewhat general MHs. To this end, the HH always reevaluates each candidate of a selective tournament, implying the reevaluation of a candidate reselected for another tournament. We

are interested in understanding whether, after completion of a hyperheuristic run, the best produced MH, under repeated evaluation, would still produce good solutions reasonably frequently in terms of real time. To that end, we chose a top MH with its evolved, personal $\iota' = 608$ that is about equal to the mentioned average $\iota = 610.8$. Reassuringly, we find that the chosen MH, when continuously being restarted after its termination, produces the best-known result every 57.04 sec on average, suggesting that at least the best evolved MHs are general and reusable. In future research we will further investigate this matter.

VI. SUMMARY AND CONCLUSIONS

Our undemanding GP hyperheuristic [14] is competitive with and far simpler than hand-made, sophisticated, domain-specific approaches [16]. However, it is more complex than basic (yet often effective) search methods, such as instances of purely greedy search. We, therefore, were interested in whether or not better search behaviour comes from the hyperheuristic. As shown by the outcome of our experiments, the effort of the hyperheuristic's development and use is indeed justified by its resulting better performance. For the chosen benchmark problems we have shown that the hyperheuristic works clearly better than greedy and even a non-greedy flavour of hill climbing, and the divide appears to widen for larger problem instances.

The comparison with the more flexible, non-greedy hill climber is of particular interest, as it shows that merely mixing exploring and exploiting low-level heuristics in a simple randomised or regular pattern does not help with hard problems. Indeed, the GP hyperheuristic is able to produce significantly more intricate patterns of calls of such heuristics.

We conclude that, at least where the difference between medium vs very good quality of solutions and search efficiency are important, one should consider using GP-based hyperheuristics.

Also, we introduced a self-adaptive version of our hyperheuristic that automatically finds parameter values for effective and efficient search. To that end, we have proposed integrating a parameter in a genotype, so that the parameter value becomes a part of the hyperheuristic search. We demonstrated this principle for the adaptation of the value of a particularly critical parameter, ι , that influences the flow of execution in a produced MH.

In summary, on the one hand, there actually are performance and flexibility gains of the GP-HH over the chosen, simpler search method. On the other hand, the effort of using the more complex method, i.e., our GP-HH, can be reduced by allowing evolution itself to set MH parameter values, thus relieving the user from this task.

VII. FURTHER WORK

It is of interest to break up heuristics, given as primitives to the hyperheuristic, into their components and represent those

as primitives as well. In this way, the hyperheuristic might be able to invent more effective or efficient metaheuristics, because it would be entirely free from constraints of human design that come in the shape of the given heuristics.

While we have demonstrated that our GP hyperheuristic works well in the TSP problem domain, we have not used knowledge that is specific to this set of problems, neither for the design of the hyperheuristic nor for the crafting of the primitives. Specifically, merely basic heuristics have been used as primitives. Therefore, it should be possible to get good performance from our hyperheuristic not only for the domain chosen here but also for other important domains. We must test this hypothesis in future research.

Also, as discussed, to adapt the individual ι value of a metaheuristic, M , the GP hyperheuristic adds a value to M 's ι . This value comes from the range $r = [-m, m]$, where m denotes the maximal number of consecutive iterations of a certain heuristic. On the one hand, if the user of the hyperheuristic decides on setting m to a small value, the adapting of ι may be slow. On the other hand, if m is a large value, a fine tuning of ι , which is desirable in the vicinity of a good ι value, is difficult for the GP-HH. To improve this situation in a simple yet effective manner, we could make r dynamic, with its diameter depending on, e.g., the performance of M relative to the average performance of the population. Especially, a very well behaving M may only require a tiny change of its ι value, while a badly performing M may disappoint merely because of a ι value that is too big or too small, so that a big change of this value is required.

ACKNOWLEDGEMENTS

The authors acknowledge financial support from EPSRC (grants EP/C523377/1 and EP/C523385/1) and thank the anonymous reviewers for helpful comments.

REFERENCES

- [1] <http://www.iwr.uni-heidelberg.de/groups/comopt/software/tsplib95/tsp/>.
- [2] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, Jan. 1998.
- [3] M. Brameier and W. Banzhaf. *Linear Genetic Programming*. Number 1 in Genetic and Evolutionary Computation. Springer, 2006.
- [4] E. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, Dec 2003.
- [5] E. K. Burke, M. R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In T. P. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervos, L. D. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *LNCS*, pages 860–869, Reykjavik, Iceland, 9-13 Sept. 2006. Springer-Verlag.
- [6] K. Chakhlevitch and P. Cowling. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In G. R. Raidl and J. Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2005*, volume 3448 of *LNCS*, pages 23–33, Lausanne, Switzerland, 30 March–1 April 2005. Springer Verlag.
- [7] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Lecture Notes in Computer Science*, number 2079, pages 176–190, 2001.
- [8] G. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6:791–812, Nov–Dec 1958.
- [9] K. Dowsland, E. Soubeiga, and E. Burke. A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research*, 179:759–774, 2007.
- [10] A. Gaw, P. Rattadilok, and R. Kwan. Distributed choice function hyper-heuristics for timetabling and scheduling. In *Proceedings of the 2004 International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004)*, Pittsburgh USA, pages 495–497, 2004.
- [11] C. Z. Janikow. Constrained genetic programming. In T. S. Hussain, editor, *Advanced Grammar Techniques Within Genetic Programming and Evolutionary Computation*, pages 80–82, Orlando, Florida, USA, 13 July 1999.
- [12] G. Jayalakshmi, S. Sathiamoorthy, and R. Rajaram. An hybrid genetic algorithm — a new approach to solve traveling salesman problem. *International Journal of Computational Engineering Science*, 2(2):339–355, 2001.
- [13] R. E. Keller and W. Banzhaf. The evolution of genetic code on a hard problem. In L. Spector, W. B. Langdon, A. Wu, H.-M. Voigt, and M. Gen, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 50–56, San Francisco, CA, 7–11 July 2001. Morgan Kaufmann, San Francisco, CA.
- [14] R. E. Keller and R. Poli. Cost-benefit investigation of a genetic-programming hyperheuristic. In *Proceedings of the 8th International Conference on Artificial Evolution (EA'07)*, Tours, France, 29-31 Oct. 2007. Evolution Artificielle, Springer.
- [15] R. E. Keller and R. Poli. Linear genetic programming of metaheuristics. In *GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, London, 7-11 July 2007. ACM Press.
- [16] R. E. Keller and R. Poli. Linear genetic programming of parsimonious metaheuristics. In D. Srinivasan and L. Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 4508–4515, Singapore, 25-28 Sept. 2007. IEEE Computational Intelligence Society, IEEE Press.
- [17] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [18] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.
- [19] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, editors. *The Travelling Salesman Problem*. Wiley, Chichester, 1985.
- [20] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.
- [21] M. Oltean. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, Fall 2005.
- [22] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, volume 4 of *Genetic programming*. Kluwer Academic Publishers, 2003.
- [23] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published by <http://lulu.com> and freely available at <http://www.gp-field-guide.org.uk>, 2008. (With contributions by J. R. Koza).
- [24] P. Ross. Hyperheuristics. In E. Burke and G. Kendall, editors, *Search Methodologies*, pages 529–556. Springer-Verlag, Berlin, Heidelberg, New York, 2005.
- [25] E. Soubeiga. *Development and application of hyper-heuristics to personnel scheduling*. PhD thesis, Computer Science, University of Nottingham, 2003.
- [26] P. A. Whigham. Search bias, language bias, and genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 230–237, Stanford University, CA, USA, 28–31 July 1996. MIT Press.
- [27] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.
- [28] M. L. Wong and K. S. Leung. Applying logic grammars to induce sub-functions in genetic programming. In *1995 IEEE Conference on Evolutionary Computation*, volume 2, pages 737–740, Perth, Australia, 29 Nov. - 1 Dec. 1995. IEEE Press.