Linear Genetic Programming of Parsimonious Metaheuristics

R. E. Keller R. Poli

Abstract— **We use a form of grammar-based linear Genetic Programming (GP) as a hyperheuristic, i.e., a search heuristic on the space of heuristics. This technique is guided by domain-specific languages that one designs taking inspiration from elementary components of specialised heuristics and metaheuristics for a domain. We demonstrate this approach for traveling-salesperson problems for which we test different languages, including one containing a looping construct. Experimentation with benchmark instances from the TSPLIB shows that the GP hyperheuristic routinely and rapidly produces parsimonious metaheuristics that find tours whose lengths are highly competitive with the best real-valued lengths from literature.**

I. Introduction

A heuristic is a method that, given a problem, often finds a good solution within acceptable time, while it cannot be shown that a found solution cannot be bad, or that the heuristic will always operate reasonably quickly. A metaheuristic is a heuristic that approaches a problem by employing heuristics. The term *hyperheuristic* [21], see [22] for its origin, refers to a heuristic that explores the space of metaheuristics for a given problem.

For example, [6] suggests a method of building low-level heuristics for personnel scheduling, where useless heuristics are filtered in order to obtain practical heuristics that make up a solver. [4] proposes tabu search [10] on the space of heuristics for an instance of scheduling. [9] describes a timetabling application of a hyperheuristic with particular emphasis on its implementation in terms of parallel computing. [8] suggests simulated annealing as learning strategy for a hyperheuristic that approaches a highly practical instance of a p-median problem. [19] employs Genetic Programming (GP) [2], [14], [15], 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. Despite not employing loops in the evolved heuristics, this system successfully reproduces a human-designed bin-packing method. The approaches used in the mentioned papers did not make use of grammars to guide the search, as is, instead, the case with our GP hyperheuristic.

Hyperheuristics are increasingly attracting interest as one hopes that they will lead to optimisation methods that are more flexible in their application to different practical domains. However, a fixed HH that efficiently operates for all domains cannot be designed [24].

In view of the above, we are interested in the objective of a domain-independent approach that a decision maker can specialise for use on a given problem domain, that imposes limited demands on the user, and that yet produces effective metaheuristics. In particular, it would be ideal if the user was only required to provide simple heuristics with which a hyperheuristic could evolve effective metaheuristics within reasonable time.

R. E. Keller and R. Poli are with the Department of Computer Science, University of Essex, UK. E-mail: {rkeller,rpoli}@essex.ac.uk

Algorithm 1 GP-based HYPERHEURISTIC.

```
given: grammar G, population size p, length l
repeat
    produce next random primitive-sequence σ : |σ| = l
    EDITING(σ,G) → g genotype
until p genotypes created
while time available do
    Selection: 2-tournament
    Reproduction: Copy winner g into loser's place → g'
    Exploration: with a given probability
        Mutate copy g' → δ
        EDITING(δ,G)→ g'' genotype
end while
```

As an approach toward this goal, we envision a hyperheuristic that accepts the description of structures of desirable metaheuristics for D, an arbitrary, fixed domain of problems. Then, in principle, after changing this description appropriately, one can apply the hyperheuristic to a different domain. To the end of giving such a description, we propose to represent some of D's low-level heuristics and well-known metaheuristics as components of sentences of a language that we describe by a grammar G. In this manner, $\sigma \in L(G)$ may define a metaheuristic for D. Then, any form of grammar-based GP (e.g., [20] [23] [18] [13] [25] [11]) that produces programs in $L(G)$ is a hyperheuristic for D. In this paper, we implement this approach as a flavour of linear GP [3].

The advantage of the described principle is that domain knowledge becomes a free resource for the GP hyperheuristic that does not have to rediscover the provided component heuristics. Moreover, by crafting an appropriate bias into G, one can direct evolutionary search towards types of metaheuristics that seem promising.

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 design grammars that we then compare in experiments described in Section V. In Section VI, we draw some conclusions while in Section VII we describe interesting avenues for future work.

II. A linear-GP hyperheuristic

A metaheuristic is represented as a genotype $g \in L(G)$ for a domain of interest, with grammar $G = (N, T, P, S)$ where N is a set of non-terminals, T is a set of terminals, P is a set of production rules, and $S \in N$ is a start symbol.

$L(G) \subset \mathbf{T}^*$, the set of all strings over T. We call a terminal $t \in T$ a *primitive*. Primitives may represent manually created metaheuristics, low-level heuristics, or parts of them.

The execution of a metaheuristic, g, with $g = i_0 i_1 ... i_n$ and $i_j \in T$, implies the sequential execution of the i_j. In this manner, they construct a complete structure, s, that is a candidate solution to the given problem. More specifically, s is obtained from an initial, complete structure, $i_0()$:

$$s = i_n(...(i_1(\ i_0()\ ))...).$$

g's primitives, with the exception of i_0, assume a complete structure as input. All of g's primitives deliver a complete structure as output. In particular, i_0, in some straightforward fashion, delivers an initial, complete structure.

We define g's fitness, or quality, $\mathbf{q}(g)$, as the value of an objective function, $\mathbf{o}$, on its argument, s, i.e., $q(g) = o(s)$.

At the beginning of a run of the GP hyperheuristic, given population size p, an *initialisation* routine produces p random primitive-sequences from T^*. All such sequences are of the same length, l. *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, $\sigma = i_0..i_j..i_k \in T^*$, that is not a valid genotype, i.e., $\sigma \notin L(G) \subset T^*$. If this is the case, the sequence is passed to an operator, *EDITING*, that attempts to turn σ into an element from $L(G)$. To that end, the operator starts reading σ from left to right. If EDITING reads a primitive, p, that represents a syntax error in its current locus, EDITING replaces it with the *no-operation primitive*, $\mathbf{n}$. These steps are repeated until the last primitive has been processed. Then, either the current version of σ is in $L(G)$, and EDITING ends, or still $\sigma \notin L(G)$. In the latter case, EDITING keeps repeating the above steps on σ, but this time processing it from right to left. The result is either a $\sigma \in L(G)$ or a σ that consists of n-instances only. In this latter, unlikely case, EDITING then assigns the lowest available fitness value to σ. This way σ will most likely disappear from the population during selection. The selection scheme used by the GP hyperheuristic is *tournament selection*. We summarise the hyperheuristic in Algorithm 1.

Note that, although we initialise the population using sequences of a fixed length, l, the application of the EDITING operator effectively leads to a population containing genotypes of variable lengths, which also allows the evolution of parsimonious heuristics.

III. Problem domain

To study the GP hyperheuristic, we select an NP-hard domain from discrete optimisation: the set of traveling salesperson problems (TSP) [16]. In its simplest form, a TSP involves finding a minimum-cost Hamiltonian cycle ("tour") in a given, 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), ..., (v_{n-1}, v_0) as a permutation $p = (v_0, ..., v_{n-1})$ over $\{0, ..., n-1\}$. We call permutation $(0, 1, ..., n-1)$ the *natural* Hamiltonian cycle of the graph ("natural cycle"). In the following sections, for empirical studies, we will use 600 symmetrical, Euclidean, random-generated problems of ten nodes. The weight of an edge (i, j) is the distance between i and j. To demonstrate the practical applicability of the GP hyperheuristic, we will also use TSP problems of more realistic sizes.

```
metaheuristic        ::= NATURAL
                       | NATURAL search

search               ::= heuristic
                       | heuristic search

heuristic            ::= NATURAL
                       | 2-CHANGE
```

Fig. 1. Grammar N2O

IV. Grammars

A. Grammar N2O

We start by providing the GP HH with two low-level heuristics only: NATURAL that creates the natural cycle for a problem, and 2-CHANGE which executes a minimal change of a tour H into a different tour. That is, given two of H's edges $(a, b), (c, d) : a \neq d,\ b \neq c$, 2-CHANGE replaces them with $(a, c), (b, d)$. 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.

The primitives NATURAL and 2-CHANGE are terminal symbols for a grammar, which we call $N2O$, that governs creating and mutating the metaheuristics by the GP hyperheuristic. $N2O$ is given in pseudo Backus-Naur form (BNF) [7] in Figure 1. We shall employ the top two rules of $N2O$, metaheuristic and search, in several grammars to be discussed later where these rules shall be represented by **Preamble**. Note that there is no implicit beneficial bias in $L(N2O)$, the language generated by $N2O$. For instance, the sequence NATURAL 2-CHANGE 2-CHANGE NATURAL, where the final execution of NATURAL destroys whatever improvement may have been made by the repeated execution of 2-CHANGE, is in $L(N2O)$. Also, we have not provided a *non-destructive* local heuristic, for instance, one that only executes 2-CHANGE if this results in a shorter cycle. So, the system has no domain-specific, built-in intelligence, and has no way of evolving it over the given grammar. Can the GP hyperheuristic still create an effective metaheuristic, specific to the given instance of a problem? We will see this in Section V-A.

B. Grammars If and NoNatural

Next, we extend grammar $N2O$, giving a grammar we call *If*, shown in Figure 2.

IF_2-CHANGE is a conditional 2-CHANGE, which executes 2-CHANGE only if the execution will shorten the tour under construction. As every greedy operator, IF_2-CHANGE is a

```
Preamble

heuristic   ::= NATURAL
              | 2-CHANGE
              | IF_2-CHANGE /* added */
```

Fig. 2. Grammar If

```
Preamble

heuristic           ::= 2-CHANGE
                      | IF_2-CHANGE
```

Fig. 3. Grammar NoNatural

boon and a curse, but its introduction is safe here since there is a randomising counterweight in the form of `2-CHANGE`.

Hoping to define an even more apt language, we do away with `NATURAL`, and we define grammar *NoNatural* shown in Figure 3.

C. Grammar ThreeChange

We extend the target language by a further low-level heuristic that is known as a *3-change*: delete three mutually disjoint edges from a given cycle, and reconnect the obtained three paths so that a different cycle results. Given this method, we define the heuristic `IF_3-CHANGE`: 1) randomly select edges as arguments for `3-change` 2) if `3-change` will better the cycle, execute `3-change`.

Since `IF_3-CHANGE` introduces a greedy bias, it may or may not be helpful to provide a counter-bias, for instance by allowing for an occasional worsening of a cycle. We decide on this step, and to this end we define the heuristic `IF_NO_IMPROVEMENT`:
if none of the latest $1,000$ individuals produced has found a better best-so-far tour, execute a `2-change`.

We add `IF_3-CHANGE` and `IF_NO_IMPROVEMENT` to the previous grammar and call the new one *ThreeChange*. It is shown in Figure 4.

D. Grammar NoNoImprove

A small language is desirable, as it reduces the search-space size for the hyperheuristic. To understand whether `IF_NO_IMPROVEMENT` does or does not improve the effectiveness of the GP hyperheuristic, we remove it from grammar *ThreeChange*. We call the resulting grammar and its language *NoNoImprove*. The grammar is shown in Figure 5.

```
Preamble

heuristic   ::= 2-CHANGE
              | IF_2-CHANGE
              | IF_3-CHANGE
              | IF_NO_IMPROVEMENT
```

Fig. 4. Grammar ThreeChange

```
Preamble

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

Fig. 5. Grammar NoNoImprove

```
Preamble

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

loop      ::= REPEAT_UNTIL_IMPROVEMENT
            | /* empty */
```

Fig. 6. Grammar DoTillImprove

E. Grammar DoTillImprove

So far, only sequential and conditional execution of user-provided heuristics is available to evolved metaheuristics. A loop element is required to complete the set of essential control structures. To that end, we introduce the primitive

`REPEAT_UNTIL_IMPROVEMENT` p: execute primitive p until it has lead to a shorter cycle or until it has been executed ι times for user-given ι.

An example for the use of the primitive `REPEAT_UNTIL_IMPROVEMENT` in a grammar, *DoTillImprove*, is shown in Figure 6. Note that this grammar allows for evolving a metaheuristic that represents one of a few simple search types, such as memory-less iterated local search [17], greedy search, or random search, without enforcing the evolution of only one such type. This is because the grammar contains primitives that represent elements of such search types, but it does not describe primitive-sequences of exclusively one such type.

V. Experiments

Initially, we are interested in whether or not, on average, for small problems from the chosen domain, an evolved metaheuristic does at least better than blind search, the minimal approach to any problem. This is to see whether the hyperheuristic manages to acquire any domain information at all and represent it in the structure of the evolved heuristics. Later, we shall go for bigger problems. There, a comparison with the second-simplest approach, a pure hill climber, is not ambitious, as it does poorly on larger TSPs. Indeed, literature ignores it. Instead, we shall compare evolved solvers to a state-of-the-art hybrid Genetic Algorithm.

Here, we compare the search performed by evolved metaheuristics to a form of blind search that operates on the solution space of a problem to be given to the hyperheuristic. For this purpose, blind search creates $n > 0$ random permutations, with n to be given. That is, each of the n search steps creates a random cycle, i.e., a candidate solution.

TABLE I

BASIC PARAMETERS

Popul. size	Genotype size	Offspring	Mut. prob.
10	10	2,500	0.3

For a meaningful comparison in terms of efficiency, one must ensure that the processes to be compared perform approximately equal numbers of their respective elementary operations. Thus, for the present purpose, we must set $n = 10$ for blind search, as this number equals the genotype size which equals the number of simplest search steps that an evolved metaheuristic performs during its evaluation. In this manner, the best of the n cycles found by a run of blind search can be compared with the single cycle constructed by n calls of heuristics as orchestrated by a metaheuristic.

To compare an achieved tour length l to a given length L, we define $\delta(l, L) = 1 - l/L$. Thus, for l, m with $\delta(l, L) > \delta(m, L)$, l is a better result than m. Also, $\delta(l, L) < 0$ implies that l is worse than L, a δ value of zero indicates stasis, and positive values signal an improvement.

A. Grammar $N2O$

For the small TSP problems, we will employ a restricted version of the GP hyperheuristic where all produced primitive-sequences shall be genotypes of the same size. For the larger problems, we shall drop these restrictions.

The genotype size shall be $\mathsf{l} = 10$. The initial genotypes shall all be identical: $g =$ `NATURAL`...`NATURAL`, $|g| = \mathsf{l}$.

Thus, there is no random search during initialisation, so that any beneficial effect observed later comes from evolutionary search alone.

For the present setup of the hyperheuristic, its random choice of an element from a set shall be uniform.

We number the loci of genotypes, beginning with zero. In the context of $L(N2O)$, the described setup implies that we exclude position zero from mutation, as only `NATURAL` is legal there. We set mutation probability to 0.3 (cf. Algorithm 1, step 9).

The GP HH shall measure time in terms of the number of individuals produced after creation of $\mathsf{p} = 10$ initial individuals. We set this offspring number to 2,500 (cf. Algorithm 1, step 6). Table I reports the used parameters.

On a single, random-generated ten-node problem, P, we observe tour lengths found by evolved metaheuristics that are, on average over 1,000 independent runs, better by 0.3 [δ] than P's natural length, opposed to 0.2 for random search.

Interestingly, the hyperheuristic produces such metaheuristics despite the fact that the underlying primitives do not allow for direct use of problem-specific information. For example, an informed decision on the choice of arguments to a `2-CHANGE` execution is not an option. To establish whether this phenomenon is independent from P, we perform 1,000 independent runs of the hyperheuristic for each of 600 randomly created, ten-node problems, collecting the natural lengths of these instances and the lengths of the shortest tours found by the runs. Table II shows the results.

TABLE II

HYPERHEURISTIC, 600 10-NODE RANDOM PROBLEMS, 1,000 RUNS. IMPROVEMENT δ RELATIVE TO MEAN NATURAL LENGTH. "N.A.": NOT APPLICABLE

Length	Mean	SD	δ
Natural	$3,827$	539	n.a.
Best	$2,654$	265	0.31

TABLE III

GP HYPERHEURISTIC V IDEAL RANDOM SEARCH ON $L(N2O)$. HITS OF OPTIMAL GENOTYPE.

Approach	**Hits**
GP hyperheuristic	14,171
Random	4,902

We notice a mean improvement of about 0.31, so that the good performance of the GP hyperheuristic, observed for P, appears to be rather general.

Let us shift the focus from the problem space to the metaheuristic space. We therefore consider, for P, both the GP hyperheuristic and pure random search in the space of metaheuristics, i.e., $L(N2O)$. As the given genotype size equals ten, the genotype that starts with `NATURAL`, followed by the longest possible unbroken chain of nine `2-CHANGE` calls, offers most opportunities for improving a tour. On P, the 1,000 runs of the hyperheuristic locate this genotype 14,171 times, while the expected value for 1,000 runs of 2,510 random samples each is merely

$1,000 \cdot 2,510/2^9 \approx 4,902$.

Table III collects the values.

B. Grammars *If and* NoNatural

We change some of the basic parameters. The genotype size, which equals the number of times an evolved metaheuristic calls a provided heuristic, shall be twenty. 5,000 offspring shall be produced per run. Table IV reports the used parameters.

For a comparison of the resulting behaviours of the hyperheuristic when it is fed with the different grammars, we consider forty symmetrical, Euclidean, random-generated problems of ten nodes. On each problem, the HH performs 500 independent runs. Regarding the shortest tour found during a run, we give means and standard deviations (SD) over all runs over all problems, rounded to the nearest integer. Table V summarises the results of the experiments. Its first line gives the values over the natural lengths of the random problems.

The improvements in the mean-best values show that one can improve performance of the GP hyperheuristic by

TABLE IV

BASIC PARAMETERS

Popul. size	Genotype size	Offspring	Mut. prob.
10	20	5,000	0.3

TABLE V

PERFORMANCE OF HYPERHEURISTIC OVER DIFFERENT LANGUAGES, ON 40 TEN-NODE RANDOM PROBLEMS, 500 RUNS. IMPROVEMENTS δ RELATIVE TO NATURAL LENGTH.

40 problems	**Mean best**	SD	δ
Natural length	3,774	508	n.a.
N2O	2,600	277	0.31
If	2,530	251	0.33
NoNatural	2,423	204	0.36

TABLE VI

BASIC PARAMETERS

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

increasing expressiveness of the target language L, or by changing a bias in L, as we have done over the series of grammars discussed in Section IV.

Language *NoNatural*, the one of the three giving the best results, is still very simple in terms of the structural and functional sophistication of its sentences when one compares them to procedures used by state-of-the-art TSP solvers. We therefore ask whether or not, even with this quite rudimentary language at hand, the GP hyperheuristic can still be of use given a real-world problem.

So, we consider problem `eil51` from [1]. Its dimension is $n = 51$ nodes, its best known solution has a length of 428.87 [12] with natural length of approximately 1,313.47. All values delivered by the GP hyperheuristic shall be rounded off to the nearest hundredth. We shall only be dealing with real-valued tour lengths, emphasising differences between very good vs. excellent results. Literature often mentions integer lengths that result from measuring a distance between two nodes by rounding it to the nearest integer, which implies that tours with different real-valued lengths can have the same integer length.

For a symmetrical TSP instance, the number of tours that are different in human terms equals $(n-1)!/2$. The evolved metaheuristics operate on permutations of n nodes, so that the search-space size is $n!$. So, while the previous problem dimension, $n = 10$, yields a size of $n! \approx 3.6 \times 10^6$, $n = 51$ gives about 1.6×10^{66} search points and 1.52×10^{64} different tours. To accommodate for the larger search space, we set the basic parameters as shown in Table VI.

Table VII gives the results obtained for `eil51`. Column "Best" gives the length of the shortest cycle found over all runs. On average, even the simple language *NoNatural* guides the GP hyperheuristic to an improvement of about 0.3 relative to the original tour.[1]

[1]Naturally, longer runs may produce even better results. For example, in one run with 10^6 produced offspring, the shortest tour found had a length of about 749. However, we did not perform enough such runs to be able to report reliable results here.

TABLE VII

PERFORMANCE FOR eil51 OVER *NoNatural*, 100 RUNS. IMPROVEMENT δ OF MEAN BEST RELATIVE TO NATURAL LENGTH.

eil51	**Mean best**	SD	Best	δ
Natural length	1,313.47	n.a.	n.a.	n.a.
NoNatural	922.00	26.96	853.73	0.3

TABLE VIII

PERFORMANCE FOR eil51 OVER *ThreeChange*, 30 RUNS. IMPROVEMENT δ OF MEAN BEST RELATIVE TO NATURAL LENGTH.

eil51	**Mean best**	SD	Best	δ
Natural length	1,313.47	n.a.	n.a.	n.a.
ThreeChange	874.96	26.55	810.73	0.33

C. *Grammars* ThreeChange *and* NoNoImprove

Table VIII gives the results obtained with grammar *ThreeChange*. We notice another improvement in the mean-best value. This means that the combination or the individual effects of the added primitives, `IF_3-CHANGE` and `IF_NO_IMPROVEMENT`, increase the search power of the hyperheuristic.

Table IX gives the results obtained using grammar *NoNoImprove*. This grammar lacks the primitive `IF_NO_IMPROVEMENT`. Interestingly, we see that removing this primitive yields even better search performance.

D. *Grammar DoTillImprove*

Successful optimisation algorithms all use the iteration of effective heuristics. It is interesting then to see if the looping construct available in grammar *DoTillImprove* (which is otherwise identical to the previous grammar, *NoNoImprove*) further improves the performance of the GP hyperheuristic.

Table X gives the results of our experiments with grammar *DoTillImprove*. One sees that search performance, in terms of the mean-best value, clearly improves, even when a small number of loop iterations (ι) is used. For comparison, the bottom row of the table gives the best known result for `eil51`, taken from [12].[2]

We have provided the GP hyperheuristic with very trivial, low-level heuristics only and a simple grammar. It is therefore most remarkable that, for $\iota = 70$, one run of the hyperheuristic produces a metaheuristic that locates a tour whose rounded length equals the best known result. We call such a product a *best metaheuristic*. Unfortunately, [12] does

[2][12] presents a hybrid Genetic Algorithm, only applicable to Euclidean TSPs, that uses a specialised crossover together with several non-trivial, handcrafted heuristics.

TABLE IX

PERFORMANCE FOR eil51 OVER *NoNoImprove*, 30 RUNS. IMPROVEMENT δ OF MEAN BEST RELATIVE TO NATURAL LENGTH.

eil51	**Mean best**	SD	Best	δ
Natural length	1,313.47	n.a.	n.a.	n.a.
NoNoImprove	798.32	15.98	763.30	0.39

TABLE X

PERFORMANCE FOR eil51 OVER *DoTillImprove* (*DTI*) FOR DIFFERENT VALUES OF ι, ι ITERATIONS AT MOST, 100 RUNS PER VALUE. **P**: MEAN BEST OR NATURAL LENGTH IN TERMS OF % OF BEST KNOWN RESULT. ALL REAL VALUES ROUNDED OFF TO NEAREST HUNDREDTH EXCEPT WHERE HIGHER PRECISION REQUIRED FOR DISTINCTION.

eil51	**Mean best**	S.D.	Best	ι	P.%
Nat. length	1,313.47	n.a.	n.a.	n.a.	206.26
DTI	528.89	8.98	508.75	10	23.32
”	477.82	8.11	454.12	20	11.41
”	458.61	5.79	444.21	30	6.93
”	439.42	2.90	431.95	60	2.46
”	436.61	3.07	**428.87**	70	1.8
”	429.28	0.56	*428.87*	240	0.1
”	428.88	0.03	*428.87*	480	0.002
”	**428.87**	**0.00**	*428.87*	**800**	**0.0**
”	*428.87*	*0.00*	*428.87*	$\mathbf{10^4}$	*0.0*
Hybrid GA	n.a.	n.a.	***428.87***	*best known*	n.a.

TABLE XI

FOR eil51, GRAMMAR *DoTillImprove*, $\iota = 800$, ι ITERATIONS AT MOST, 100 RUNS: AVERAGE PRIMITIVE FREQUENCY ϕ. FIXED MAX. GENOTYPE SIZE $\mathsf{l} = 500$.

ϕ.	S.D.	*Primitive*
190.37	11.9	n “no operation”
0.999	0.00008	NATURAL
90.77	2.87664	REPEAT_UNTIL_IMPR.
104.27	7.5	IF_3−CHANGE
99.61	7.89451	IF_2−CHANGE
14.98	3.6872	2−CHANGE
309.63	$\mathsf{l}-\phi(\mathtt{n})$	*mean effective MH size*

not mention the precision of its result. We report that our full-precision value equals 428.871765.

Actually, for $\iota = 800$, the hyperheuristic routinely produces best metaheuristics, i.e., each of its 100 runs produces at least one such MH, as evidenced by standard deviation zero. Also, with much less effort, given $\iota = 480$, almost every run produces at least one best MH.

For $\iota = 800$, a run lasts 10.1 minutes on average on a single core of an Intel Xeon 3.2 GHz dual core machine without additional load, but including I/O time for storing about 1 GB of data about states of the hyperheuristic. Even so, as 100,100 (initial individuals plus offspring) $/10.1/60 \approx 165$, this many metaheuristics are produced each second on average, including their execution that locates tours as solutions to the given problem.

We define the *primitive-frequency* $\phi(p)$ of a primitive p regarding a considered string, s, as the number of p's occurrences in s.

We are interested in the average primitive-frequencies over all runs with $\iota = 800$, i.e., we ask how many instances of a given primitive occur, on average, in a produced genotype. Table XI summarises the results. While each run yields at least one best metaheuristic, the evolved metaheuristics do not, on average, exhaust the available maximal genotype length, as evidenced by $\phi(\mathtt{n})$. This means that the hyperheuristic works against bloat, a typical nuisance with many GP flavours, i.e., undesirable growth of evolved structures. Hence, the mean effective size of a produced metaheuristic is considerably smaller than l, the fixed maximal size of a MH. This can be seen in the bottom row of the table.

$\phi(\mathtt{NATURAL})$ does not exactly equal 1 due to rare, evolved strings that only consist of $\mathtt{n}$-instances and that never got selected for competition with a metaheuristic.

$\mathtt{IF\_3-CHANGE}$ has the highest ϕ value among the non-$\mathtt{n}$ primitives. This is of interest because human MH designers prefer the use of $\mathtt{3-CHANGE}$ over $\mathtt{2-CHANGE}$: while the former runs longer, the latter often does not shorten a tour as much. $\phi(\mathtt{IF\_2-CHANGE})$ is the second-highest frequency, so that $\mathtt{IF\_3-CHANGE}$ and $\mathtt{IF\_2-CHANGE}$, the two non-destructive, tour-changing primitives, are the most frequent effective elements of an evolved MH.

$\phi(\mathtt{REPEAT\_UNTIL\_IMPROVEMENT})$, the third-highest frequency, still equals about 29% of the mean effective MH size. This shows that the hyperheuristic makes strong use of this loop-based primitive, approximating a k-opt heuristic that is widely used in hand-crafted TSP solvers. This heuristic executes a k-change until no further improvement occurs, while REPEAT_UNTIL_IMPROVEMENT, together with its primitive to be repeated, runs until a first improvement happens or ι is exhausted. Eventually, $\phi(\mathtt{2-CHANGE})$ equals about 4.8% only. This shows that the hyperheuristic mostly does not use this tour-changing primitive that represents the only destructive operator among all given heuristics.

Next, we consider eil76 [1], a 76-node problem with a size of about 1.9×10^{111} search points. We use the same basic parameters as given in Table VI, and, again, grammar *DoTillImprove*.

Results are summarised in Table XII. Despite the vastly larger search space, already $\iota = 800$, the value that results in zero standard deviation over eil51, yields a promising result as given in column “Best”, i.e., $\beta = 576.7$. Relative to 1,974.71, the length of the natural cycle that a metaheuristic starts changing into its output cycle, β is already in the vicinity of the best known result from literature [12], i.e., $\alpha = 544.37$.

We therefore keep increasing ι over some orders of magnitude, up to 5×10^6, which results in a new $\beta = 557.9$ that is within $\beta/\alpha - 1 \approx 2.49\%$ of α.

TABLE XII

PERFORMANCE FOR eil76, DETAILS AS GIVEN IN CAPTION OF TABLE X.

eil76	**Mean best**	S.D.	Best	ι	**P**.%
Nat. length	1,974.71	n.a.	n.a.	n.a.	262.75
DTI	600.09	12.37	*576.60*	*800*	10.24
”	592.93	12.10	564.48	2×10^3	8.92
”	589.63	11.98	562.27	5×10^3	8.31
”	586.29	12.81	559.78	1.5×10^4	7.7
”	586.39	12.55	561.79	5×10^4	7.72
”	586.10	12.47	559.09	5×10^5	7.67
”	586.48	12.81	557.90	5×10^6	7.74
Hybrid GA	n.a.	n.a.	***544.37***	*best known*	n.a.

TABLE XIII
BASIC PARAMETERS

Popul. size	Genotype size	**Offspring**	Mut. prob.
100	500	1×10^6	0.5

TABLE XIV
PERFORMANCE OF METAHEURISTICS EVOLVED OVER LANGUAGE DTI, ON PROBLEM eil76. 100 RUNS OF GP HYPERHEURISTIC FOR GIVEN ι VALUE. BEST EVOLVED METAHEURISTICS AT LEAST MATCH EFFECTIVENESS OF HAND-CRAFTED HYBRID GA. OTHER DETAILS AS GIVEN IN CAPTION OF TABLE X.

eil76	Mean best	S.D.	Best	ι	P.%
Nat. length	1,974.71	n.a.	n.a.	n.a.	262.75
DTI	548.99	1.67	**544.37**	2×10^3	0.85
Hybrid GA	n.a.	n.a.	***544.37***	***best known***	n.a.

The stagnation of P.% over ι in magnitude orders 4, 5, 6, the corresponding mean best values, and their standard deviations suggest increasing the number of offspring to be evolved. We therefore set this basic parameter to 1×10^6, keeping all other parameter values. Judging by table XII, already $\iota = 2 \times 10^3$ is a promising value. Table XIII summarises the values of the basic parameters.

Table XIV shows results. The mean best over all runs is well within one percent of α, the best known solution. Our best evolved metaheuristics yield tour lengths that are actually shorter than $\alpha = 544.37$. Unfortunately again, [12] does not specify whether α is a rounded value. As tour lengths can be close to each other – the hyperheuristic finds, for instance, tour lengths 547.*737*183 and 547.*740*601 – full-precision values can be critical to comparing effectiveness of TSP solvers. Thus, we report that our GP hyperheuristic has found an overall best tour length of $\alpha_{HH} = \mathbf{544.36908}$.

This result and also the best GP HH result over problem eil51 easily beat best results from [19] that compares its evolved evolutionary algorithms (EA) against a standard GA, while our evolved metaheuristics must compete with the sophisticated, specialised GA from [12] that has produced α. In particular, the evolved EAs start with initial tours gained from the Nearest-Neighbour heuristic, while our evolved metaheuristics begin with the natural tour as given by the problem statement.

To our knowledge, our best results on both benchmark problems are the best over all GP work done on TSP.

For $\iota = 2,000$, on average, the hyperheuristic produces 97 metaheuristics per second on eil76. On eil51, the previous problem, we obtained a value of 165. Thus, an increase in problem size by factor 1.5 has increased effort by factor 1.7.

For eil76, we are interested in whether or not, at a reasonable expense of more run time, the hyperheuristic can find more very good results. Thus, we run the HH again, changing ι from 2,000 to 15,000. Table XV shows results. The mean best's P value drops down to 0.26%, less than a third of its previous value. On average, the GP hyperheuristic produces 56 metaheuristics per second, which is still about 60% of the previous rate. While we consider only 30 runs this time, the very low standard deviation indicates reliable search behaviour. In view of these values, the more than sevenfold increase of ι has paid off.

TABLE XV
PERFORMANCE OVER 30 RUNS OF GP HYPERHEURISTIC FOR GIVEN ι VALUE. OTHER DETAILS AS GIVEN IN CAPTION OF TABLE XIV.

eil76	Mean best	S.D.	Best	ι	P.%
Nat. length	1,974.71	n.a.	n.a.	n.a.	262.75
DTI	545.77	0.97	**544.37**	1.5×10^4	0.26
Hybrid GA	n.a.	n.a.	***544.37***	***best known***	n.a.

VI. SUMMARY AND CONCLUSIONS

We have introduced a domain-independent, linear GP hyperheuristic that produces metaheuristics from provided heuristics. The metaheuristics are expressions from a user-given language.

We demonstrated this approach for the domain of traveling-salesperson problems. To this end we provided the hyperheuristic with simple heuristics from this domain and with a progression of simple grammars, the best performing of which allows for sequences of heuristics and loops with a fixed maximum number of iterations of a single heuristic.

Despite the simplicity of this grammar, for the considered, realistic benchmark TSPs, the GP hyperheuristic evolves metaheuristics that find tours with lengths that are highly competitive with the best known lengths from literature. These lengths are often yielded by specialised, man-made solvers that use sophisticated, hand-crafted heuristics and that are initialised with tours that are provided by some smart, manually produced initialisation heuristic. None of this is available to our GP hyperheuristic.

In addition, by setting the iteration number and the number of produced metaheuristics appropriately, the hyperheuristic has been able to routinely produce metaheuristics capable of matching best known results, with modest run times.

The approach produces parsimonious metaheuristics whose sizes are, on average, considerably smaller than the maximally available size. Out of the provided primitives, the GP hyperheuristic strongly favours the non-destructive primitives and the loop-based primitive. These proved critical to giving competitive effectiveness to the produced metaheuristics.

However, the hyperheuristic did not ignore the destructive primitive but discovered a good ratio of the critical primitives. In particular, it did not wander off into producing very greedy metaheuristics only, as, for instance, a greedy search algorithm on the space of metaheuristics would do.

As the principles of the GP hyperheuristic are domain-independent, one may hope it will work for other domains when one supplies it with domain-specific heuristics and languages. The user of this approach can influence the search performance of the hyperheuristic at an abstract, problem-oriented level by modifying the used language, i.e., by adding

or removing heuristics, and by describing optional patterns for the order of execution of the given heuristics.

To that end, for an arbitrary domain, one may manually develop sophisticated heuristics or, saving expensive manpower, merely provide trivial heuristics, or simply extract smart methods from existing solvers, and represent all of them as primitives. These are the terminals of a grammar that generates the language used by the hyperheuristic. The latter will then perform the tedious task of searching for a promising metaheuristic, i.e., an effective combination of the provided heuristics.

Also, one may allow a generous maximal size of the metaheuristics that are to be evolved, since the presented approach displays an implicit tendency toward keeping them parsimonious. In particular, the EDITING operator implies a variance of the effective genotype size, i.e., the number of effective primitives in a genotype. This variance is beneficial as it is a necessary condition for the emergence of metaheuristics that are both parsimonious and good.

EDITING requires an only small computational effort, since the elimination of an erroneous primitive i_j from a metaheuristic does not force i) the calculation of the set of primitives that are proper in locus j, and ii) subsequent selection of one of them with which to replace i_j.

Thus, only given point mutation, EDITING, and a fixed maximal size of genotypes, the hyperheuristic may benefit from parsimonious genotypes and varying genotype sizes, without risking bloat.

VII. Further work

We intend to test the presented GP hyperheuristic on further real-world problems from several domains, comparing the produced metaheuristics to domain-specific algorithms.

Additionally, one may identify a domain for which few or no well-established solvers exist, such as fully autonomous real-time control of agents in dynamic, hazardous environments.

Also, we shall break up low-level heuristics into their components and represent them as primitives. In this way, in principle, the hyperheuristic would be able to produce even more novel and powerful metaheuristics. Furthermore, the hyperheuristic could, on the fly, represent an evolved metaheuristic as a further primitive, enriching its language for expressing metaheuristics.

Acknowledgements

The authors acknowledge financial support from EPSRC (grants EP/C523377/1 and EP/C523385/1) and comments from anonymous reviewers that helped improve this paper.

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-Verlag, 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] Control Data Corporation. *Control Data Corporation: Algol-60, Version 5, Reference Manual, Appendix D*, 1979.

[8] 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*. To appear.

[9] 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.

[10] F. Glover and M. Laguna. *Tabu Search*. Springer, 1997.

[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] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[15] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[16] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, editors. *The Travelling Salesman Problem*. Wiley, Chichester, 1985.

[17] H. R. Lourenco, O. C. Martin, and T. Stutzle. Iterated local search. *ISORMS*, 57:321–353, 2002.

[18] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[19] M. Oltean. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, Fall 2005.

[20] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, volume 4 of *Genetic programming*. Kluwer Academic Publishers, 2003.

[21] P. Ross. Hyperheuristics. In E. Burke and G. Kendall, editors, *Search Methodologies*, pages 529–556. Springer-Verlag, Berlin, Heidelberg, New York, 2005.

[22] E. Soubeiga. *Development and application of hyper-heuristics to personnel scheduling*. PhD thesis, Computer Science, University of Nottingham, 2003.

[23] 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.

[24] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.

[25] 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.