Generating Human-readable Algorithms for the Travelling Salesman Problem using Hyper-Heuristics

Patricia Ryser-Welch
University of York
York Road
York
Patricia.Ryser-Welch@York.ac.uk

Julian F. Miller
University of York
York Road
York
Julian.Miller@York.ac.uk

Shahriar Asta
Notthingham University
Wollaton Road
Nottingham
sba@cs.nott.ac.uk

ABSTRACT

Hyper-heuristics search the space of heuristics and metaheuristics, so that it can generate high-quality algorithms. It is a growing area of interest in the research community. Algorithms have been constructed iteratively using "templates of operations" based on well-known heuristic and metaheuristic methods (i.e. Iterated Local Search and Memetic algorithms). These hyper-heuristic algorithms choose sequences of problem-specific heuristics that can find good solutions in the problem domain. Such "adaptive algorithms" have solved several well-established combinatorial problems, with a high level of generality. However, the evolved sequences of heuristic operations are often very long and defy human comprehension. In this paper, we focus on evolving a fixed sequence of operators inside the loop of a metaheuristic, using an innovative automatic algorithm creation method. We have extracted and hard-coded these evolved algorithms in new independent solvers for Travelling Salesman Problems.

Categories and Subject Descriptors

I.2.2 [**ARTIFICIAL INTELLIGENCE**]: Evolutionary Algorithm; D.1.2 [**Software**]: Automatic Programming

Keywords

Hyper-heuristics; Cartesian Genetic Programming; Travelling Salesman Problem

1. INTRODUCTION

Designing effective algorithms to solve computational problems is difficult and time-consuming. The standard methodology for designing such algorithms is "top-down". This process breaks down large problems into more understood components and eventually identifies problem-specific operators that algorithms need to use to solve the given problem. Often, restrictive assumptions have to be made about the use of operators within an algorithm. Indeed, investigating every possible combination of such operators is infeasible when the number of operators grows larger. In general, the constraint of having a tractable system of rules limits the human design space to a small subsets of combinations of instructions [28]. The higher degree of freedom allowed in the automated design of algorithms has the capability to discover unusual orders of operations. It is argued here that by employing the simple idea of *"assemble-and-test"* together with an evolutionary algorithm, a much larger collection of algorithms than humans have considered can be searched over. Some of these algorithms can be classified as "human-readable" and coded again within a programming language. Computation replaces human labour so that powerful optimisation techniques determine what works best in a given context. However, some discovered algorithms are likely to challenge human intelligence, as they can be unnecessarily long and difficult to understand.

The Travelling Salesman Problem (TSP) has been widely studied, despite having a deceptively simple goal; it seeks the shortest tour of a number of cities visiting each city only once. A complete weighted graph naturally models this combinatorial optimisation problem. Each vertex of a graph represents a city, each edge a road and the weight the length of the route between two cities.

More formally, let $G = (V, E)$ define a graph, where $V = \{1, 2, ..., n\}$ is a vertex set and E the set of edges. Denote, $C = c_{i,j}$ to represent a *"weight"* matrix associated with E, that models the distance from city i to city j. When a tour is represented as a permutation $(i_1, i_2, ..., i_n)$, the *"cost matrix"* becomes an essential element to calculate its overall distance. The quantity to minimise becomes $c_{i_1,i_2} + c_{i_2,i_3} + ... + c_{i_n,i_1}$ [21, 11, 10].

This paper is concerned with the evolution of metaheuristics, to solve instances of the Travelling Salesman Problem. This hard combinatorial problem is a well-established testbed for new algorithms. The contributions of the work presented in are four-fold.

1. Cartesian Genetic Programming (CGP) is used to generate and evolve metaheuristics; we are focusing on evolving either a Memetic Algorithm or an Iterated Local Search. We hope to not only evolve algorithms that can reach near-optimal TSP solutions, but also TSP solvers that are human-readable and can easily be coded and used again with a programming language.

GECCO'15 Companion, July 11–15, 2015, Madrid, Spain

DOI: http://dx.doi.org/10.1145/2739482.2768459.

2. For this work CGP has been adapted to provide a flow chart; i.e. the order of instructions within another algorithm. Previously CGP has encoded as data-flow diagram, in which data flow through the links of the graphs to the output.

3. The aim is to generate existing and new orders of instructions inside the loop of a metaheuristic to solve instances the Travelling Salesman Problem, to demonstrate the potential of combining a *Cross-domain Hyper-Heuristics* framework with a form of Genetic Programming. We call this *evolutionary cross-domain hyper-hyperheuristics*.

4. The concept of evolving Evolutionary Algorithms with a form of Genetic Programming, is extended to the evolution of Memetic Algorithm and Iterated Local Search. Such evolved Memetic Algorithms will be referred as hybrid MAs and hybrid ILS.

2. CLASSICAL ALGORITHMS FOR TSP

Exhaustive search can solve the Travelling Salesman Problem, but quickly become prohibitive when the number of cities increases. On the other hand heuristic algorithms construct feasible solutions quickly for any TSP instances, by compromising accuracy and precision for speed. The most studied method for solving the TSP are the *Lin-Kernighan heuristics*. In these, k edges are deleted and subsequently re-assembled to construct the sub-paths of a new tour with a lower minimum weight [22, 10]. Traditionally those are often referred as *k-opt heuristics*. When $k = 2$, edges between two pairs of cities are reconnected in a different way to obtain a new shorter tour.

The standard k-opt heuristics have been improved with the *Stem-and-Cycle* method, so that different permutations can be generated. In this method, an Hamiltonian cycle is formed by a sub-tour, when the edges are re-arranged. All the nodes of the graph are now represented in a tree-shape, with a "stem" and a root that attaches the cycle to the trunk. Then an alternative path is produced by attempting to connect the tip of the stem to the each node of the cyclic sub-path, this is continued until a lowest weight is found[5, 37].

These heuristics have lead to the development of the *Iterated Lin-Kernighan* algorithm [15, 14]. This Iterated Local Search specialises in solving instances of the TSP problems; it "jumps" from a local optima to nearby one. A perturbation alters the permutation of a tour t, to escape the local optima; these changes need to be big enough though. Readers who wish to read further about Iterated Local Search in general will find [23, 18] very informative.

In this paper, the *best2-Opt-Local-Search()* and *2-Opt-Local-Search()* implements a Stem-and-Cycle method, while the *3-OptLocalSearch()* has an additional level of sophistication. Finally, the *Simple Inversion mutation* applies a standard 2-opt heuristic (see Table 1).

3. POPULATION-BASED ALGORITHMS

Inspired from natural evolution, Evolutionary Algorithms maintain a population of solutions over a number of generations; these individuals compete for resources and the better solutions are more likely to survive. A population of TSP solutions is first initialised and the quality of each permutation is determined using a fitness function. The selected parents reproduce by applying some genetic operators; these could be a crossover and/or a mutation operator. The crossover operator should increase the average quality of the population; some of the genes are exchanged between individuals to create one or two offspring. Mutation should prevent a local optima by adding some perturbations to the candidates. Then the offspring are evaluated before being considered for survival to the next generation using a "replacement operator" [12].

The crossover and mutation operators discussed below have been specialised for the TSP problem. This is so that invalid tours are prevented. Quite recently, an innovative technique known as a stem-and-cycle ejection chain method was applied in a new crossover operator, with promising results [16].

- *Order Based Crossover (OX)* chooses a subtour in one parent and imposes the relative order of the cities of the other parent [3].
- *Partially-Mapped Crossover (PMX)* copies an arbitrary chosen subtour from the first parent into the second parent, before applying minimal changes to construct a valid tour [6, 7].
- *Voting Recombination Crossover (VR)* uses a randomized Boolean voting mechanism to decide from which parents each city is copied from [26].
- *Subtour-Exchange Crossover (SEC)* preserves randomly selected subtours from both parents to construct one new offspring [17].
- *Insertion Mutation (IM)* moves a randomly chosen city in a tour to randomly selected place [4].
- *Exchange Mutation (EM)* swaps two randomly selected cities[1].
- *Scramble Mutation (SM)* rearranges a random subtour of cities [3]. Hyflex applies this mutation operator on a subtour and on the whole tour.
- *Simple Inversion Mutation (SIM)* implements a 2-opt heuristics.

A Memetic Algorithm couples an evolutionary algorithms with Local Search, to mimic cultural evolution. First, the genetic operators generate the genetic code of a TSP solution, but it remains unchanged during its life. Then, the Local Search mechanism performs individual learning by refining the quality of the TSP solutions. These metaheuristics have solved well-established instances of the TSP with cities up to 1000 cities, however, their performance can decrease with the larger tours [31, 19, 9].

4. HYPER-HEURISTICS

The techniques described in the previous section are the product of human ingenuity. We argue that it is desirable to automate the design of algorithms. A wider of range of possible algorithms can be generated automatically and new TSP solvers could be discovered, in a reasonable amount of time and without the restrictions imposed by the human mind. Hyper-heuristics is one of the techniques that could contribute to this aim.

This extension of metaheuristics searches the space of heuristics and metaheuristics, so that it can generate high-quality algorithms for a problem.

For example, in Figure 5 this approach has two distinct searches: the search space of all possible Memetic Algorithms (feasible and infeasible) and the search space of solutions to the TSP problem[33, 36]. The achievements, current challenges and suggestions for future research of this branch of artificial intelligence are well-documented in [39, 38, 2, 35].

Cross-Domain Hyper-Heuristics guides the constructions of algorithms, using a general-purpose *"template of types of instructions"*. Instead of referring to a specific TSP primitives, the template randomly chooses a TSP operator from a specific subset. These subsets can include mutation, crossover or even a Local Search operator. The results of the CHeSCs 2011 competition represent the state-of-the art in the automatic selection of algorithms for optimisation problems. It was made possible by the use of the HyFlex framework [32][1], which includes several test problems together with their specific heuristics. For the TSP, this framework includes 10 benchmarks and 13 low-level heuristics for the Travelling Salesman Problem. The operators discussed in Sections 2 and 3 have been grouped in four subsets (crossover, mutation, local search and ruin-recreate); all of them only produce one TSP candidate-solution and evaluate the length of its associated tour.

The advantage of *Cross-Domain hyper-heuristics* lies in letting the programmers develop an automatic algorithm creation method, without any extensive knowledge of the problems to solve. For example, Adaptive Hyper-Heuristics (AdaptiveHH) can efficiently solve combinatorial optimisation problems; it applies a randomly-selected operator to a problem-domain solution and either accepts or rejects the new solution. This hill-climbing process is repeated until a good solution has been found. One of these templates has been loosely modelled on evolution. At each iteration, a new algorithm-solution is initialised; a randomly selected crossover, mutation or ruin-and-recreate heuristics are inserted between two randomly chosen Local Search heuristic [25, 30, 20].

In most cases, the generated algorithms are not algorithms that programmers would be familiar with. They are continuous sequences of problem-specific operations (each of them represented by a number), without any normal algorithms structure (i.e. iterations). They are generally not human-readable and very long.

Algorithm-Portfolio Hyper-Heuristics configure existing computer programs by adapting them to a given context. The performance of the algorithms stored in a portfolio is tested in a given context. Their performance is then improved by tuning their parameters. This technique improves the quality of programs, without requiring costly human activity. However, these techniques rarely generate a metaheuristic from evolution and the outcomes tend to be merely tuned parameters, instead of a full algorithm [8, 13, 29].

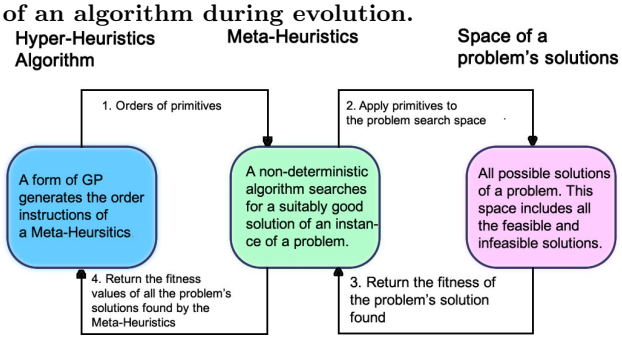
[1]Details of the challenge and the results can be found at http://www.asap. cs.nott.ac.uk/external/chesc2011/ and http://www.hyflex.org/chesc2014/

It is not common yet to view an actual algorithm as the product of hyper-heuristics, but rare exceptions can be found [34, 24]. These published algorithms have been created with very little human intervention; a stochastic search improves a randomly-generated sequence of heuristics using a stochastic evolutionary process.

5. THE PROPOSED METHOD

We are proposing to use evolution to automatically design a high-quality solver. At the end of this process, a generated algorithm can then be extracted, and if desired, coded with a programming language. Subsequently, the algorithm solutions are analysed in an independent process, to determine the quality of the order of the operations. Not only the algorithm fitness obtained from the hyper evaluation (see below) is taken into account, but also the way the algorithm is constructed (i.e. "the grammar"). This independent human process prevents applying algorithms that are known to be non-functional (i.e. algorithms with only replacement operators).

Figure 1: Process used to calculate the fitness value of an algorithm during evolution.



Chosen problem domain: Solutions of the Travelling Salesman Problem are searched for using a hybrid Memetic Algorithm; this is shown in the pink component in Figure 1. A TSP-candidate solution is obtained with a TSP-specific fitness function; the length of a tour. We will be using all the TSP problem instances and some of the heuristics offered by Hyflex, a cross-domain Hyper-Heuristics framework. These benchmarks include problems with various numbers of cities ranging from 299 to 13509. The names of these instances are: PR299, PR439, RAT575, U724, RAT783, PCB1173, D1291,U2152, USA13509, and D18512. Hyflex parameters were set to to 0.89 for the depth of the local search, increasing the number of maximum number of iterations to 40 for any local search. The mutation was set to 0.8, to increase of the effect of the Simple Inversion and Scramble mutation.

The instructions can call heuristics that alter a temporary population t; these heuristics are labelled no. 0-12 in table 1 have been introduced in Section 2. These TSP operators can be crossover, mutation and Local Search operators. Each of these genetic and cultural operators evaluate the new TSP solution they have created.

Another type of instructions manages a population p over time; they can replace older individuals of the population p by the offspring in t (heuristic no. 13 and 14 table 1). The population p can also be restarted (heuristic no. 15 in table 1). Only sequences of heuristics with at least one replacement operator are considered as valid.

- *Generate Initialise Population* randomly generate a population of TSP solutions.
- *Select Parents* identify the best problem-solutions of the population p; these individual solutions are selected for reproduction and initialise the temporary population t. A minimum of two problem-solutions must be selected. When the size of both population p and t is the same, they both become identical.
- *Replace Random* repeatedly randomly select an individual in the population p and replaces it with a solution from the temporary population t until all t have replaced a member of p.
- *Replace least-fit* selects the longest tour of the population p and replaces it with a equal or better TSP solutions from the temporary solution t. This process is repeated for each member of solution t.
- *Restart population* initialises again the whole population, if no solutions have improved enough after a certain number of iterations (i.e. the current minimum needs to improved by at least 2% every 200 iterations).

Table 1: Heuristics provided by Hyflex that will be used as the primitive function set in the CGP graph for generating two types of evolving metaheuristics. The operations highlighted in bold forms the CGP function set for the*Evolving Iterated Local Search*. The entire list of operators in the table define the CGP function set for the *Evolving Memetic Algorithms*.

CGP node function	TSP Hyflex heuristics
0	**InsertionMutation()**
1	**ExchangeMutation()**
2	**ScrambleWholeTourMutation()**
3	**ScrambleSubtourMutation()**
4	**SimpleInversionMutation()**
6	**2-OptLocalSearch()**
7	**Best2-OptLocalSearch()**
8	**3-OptLocalSearch()**
9	OrderBasedCrossover()
10	PartiallyMapCrossover()
11	VotingRecombinationCrossover()
12	SubtourExchangeCrossover()
13	**ReplaceLeastFit()**
14	**ReplaceRandom()**
15	RestartPopulation()

Metaheuristics: Our hybrid metaheuristic (the green component in Figure 1) applies the template described in algorithm 1. The body of the loop of a population-based metaheuristic is generated by the evolutionary Hyper-Heuristic algorithm (CGP) (see the instructions in bold blue text in Algorithm 1). This template prevents having an invalid algorithm.

Hyper-Heuristics algorithms : Cartesian Genetic Programming (CGP) automatically generates the sequence of instructions; the natural features of the directed acyclic graphs are applied to produce a flow chart. Each node represents a TSP operator and the edges connects the nodes together(see example in Figure 2). The evolution assembles and tests part of the algorithm (in blue in Figure 1 and the bold blue text in Algorithm 1) to construct an ordered sequences of TSP operations.

Algorithm 1 : The template of a population-based metaheuristic, with its core being evolved (in blue and bold text) by an evolved Hyper-Heuristic algorithm (the green circle in Figure 1.

```
p_0 ← GenerateInitialSolution()
p ← Apply a Local Search(p_0)
while Not optimum and EvalCount < MaxEvals do
  t ← SelectParents(p)
  NumEvals = 0
  while Not end of evolved sequence of operations do
    Apply current operation to t or p
    if current operation is applied on t then
      NumEvals = NumEvals + 1
    end if
  end while
  EvalCount = EvalCount + NumEvals
end while
```

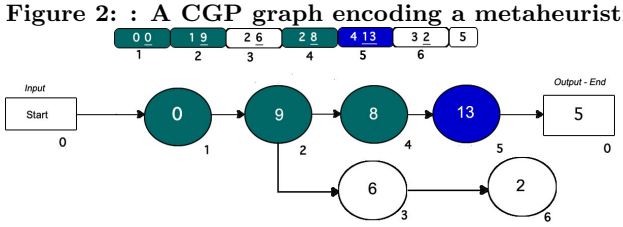
Algorithm 2 : Memetic Algorithm generated by CGP. The larger TSP operators set described in Table 1 was used.

```
p_0 ← GenerateInitialSolution()
p ← 3-Opt Local Search(p_0)
while InstanceMinima has not been found or
number of evaluations left > 0 do
  t ← SelectParents(p)
  t ← InsertionMutation()
  t ← OrderBasedCrossover()
  t ← 3-OptLocalSearch()
  p ← ReplaceLeastFit(t)
end while
```

We use a one-dimensional CGP geometry for this as recommended in [27]. The genotype is a collection of integers that encodes a directed acyclic graph. The links in the CGP graph define a variable length ordering of instructions. An example showing how a CGP graph encodes an algorithm is shown in Figure 2 and discussed in Section6. As usual with CGP when a genotype is decoded, it can happen that some nodes are not referenced. This means that the size of the algorithms (number of nodes/operations) can be anything from zero to the maximum number nodes defined in the CGP genotype. Nodes take their inputs in a feed-forward manner from either the output of nodes or from an input. To search the algorithm space, we use a $1 + 1$ Evolutionary Strategy (see Algorithm 3). The reasons why such a simple evolutionary strategy works well is primarily due to the presence of non-coding genes and the 1+1 strategy can only move forwards, improving the quality of the algorithms. In our case, this means that the applying simple mutations can explore a wide distribution of evolved metaheuristics. This allows continual exploration of the algorithm space even if the algorithm fitness (performance measure) is fixed [27].

For example, the CGP graph in Figure 2 encodes the algorithm 2. The first node encodes InsertationMutation heuristic, the second node the OrderBasedCrossover(), the third node the 3-OptLocalSearch() and the last node ReplaceLeastFit(). All these heuristics are discussed in the next subsection.

Figure 2: : A CGP graph encoding a metaheuristic



Algorithm 3 The $(1 + 1)$ evolutionary strategy

Randomly generate individual i
Select the fittest individual, which is promoted as the parent (algorithm)
while solution is not found **or** the generation limit is not reached **do**
  Mutate the parent to generate offspring
  if offspring fitness >= the parent **then**
    offspring replaces parent
  end if
end while

Hyper evaluations: The fitness of an algorithm is obtained by applying the method in Figure 1. Our hybrid metaheuristics finds solutions to the Travelling Salesman Problem. There is intrinsic relationship between the fitness of the TSP solutions (i.e. the length of a tour) and the fitness of the algorithm candidates. Its quality measurement should be a fair reflection of its ability to solve the TSP. The hybrid metaheuristic applies the encoded algorithm a number of times, to calculate the average length of the tours found. The function also penalises sequences of heuristics without any replacement operators by setting the length of the tour to a very large value. This is to decrease the likelihood that such encoded algorithms survive to the next generation.

6. EXPERIMENTAL RESULTS

Algorithm solutions: Algorithms [6, 2, 5] show the best hybrid metaheuristics evolved by CGP with 100 unary-nodes, with $\mu = 1$,$\lambda = 1$, a maximum number of allowed iterations set to 1200 (1202 hyper evaluations), and finally a mutation rate of 5%. Our chosen learning instances were PR299, PR439, and U724. The computer cluster N8 HPC[2] hosted our evolutionary cross-domain hyper-heuristics to generate the algorithms and test their performance. Early experiments have discovered again the most effective TSP operators provided by Hyflex includes the Lin-Kerninghan heuristics (i.e. the three k-opt Local Search and the simple inversion mutation operators) and then the exchange mutation operators. The two Iterated Local Search algorithms apply these powerful TSP heuristics. Also we have written a Memetic Algorithm (see alg 4), so that we can compare the performance of a human-written algorithm against one generated automatically.

[2]N8 HPC provided and funded by the N8 consortium and EPSRC (Grant No.EP/K000225/1), The Centre is coordinated by the Universities of Leeds and Manchester.

Algorithm 4 : Humanly-written Memetic Algorithm.

$p_0 \leftarrow$ GenerateInitialSolution()
$p \leftarrow$ Apply a Local Search(p_0)
while Not optimum and EvalCount < MaxEvals **do**
  $t \leftarrow$ SelectParents(p)
  $t \leftarrow$ **OrderBasedCrossover()**
  $t \leftarrow$ **3-Opt Local Search()**
  $t \leftarrow$ **ExchangeMutation()**
  $t \leftarrow$ **3-Opt Local Search()**
  $p \leftarrow$ **ReplaceLeastFit**(t)
  $p \leftarrow$ **RestartPopulation**(p)
end while

Algorithm 5 : A rediscovered Iterated Local Search. This algorithm was human-written and also generated using both sets TSP operations provided by Hyflex (see Table 1).

$p_0 \leftarrow$ GenerateInitialSolution()
$p \leftarrow$ Apply a Local Search(p_0)
while Not optimum and EvalCount < MaxEvals **do**
  $t \leftarrow$ SelectParents(p)
  $t \leftarrow$ **ExchangeMutation()**
  $t \leftarrow$ **3-Opt Local Search()**
  $p \leftarrow$ **ReplaceLeastFit**(t)
end while

Algorithm 6 : A Iterated Local Search generated by CGP, using all the TSP operations in bold in Table 1.

$p_0 \leftarrow$ GenerateInitialSolution()
$p \leftarrow$ Apply a Local Search(p_0)
while Not optimum and EvalCount < MaxEvals **do**
  $t \leftarrow$ SelectParents(p)
  $t \leftarrow$ **ExchangeMutation()**
  $t \leftarrow$ **Best2-OptLocalSearch()**
  $t \leftarrow$ **ExchangeMutation()**
  $t \leftarrow$ **3-OptLocalSearch()**
  $p \leftarrow$ **ReplaceLeastFit**(t)
end while

The sequence of instructions of algorithms [6, 2, 5] were translated from their CGP graphs to be hard-coded in three unique TSP solvers; another TSP solver was created for algorithm 4. These solvers were programmed with the programming language Java and used again all the primitives and the benchmarks D1291, U2152, USA13509, and D18512 of our chosen problem domain.

Performance of the algorithms solutions: The relative error was obtained using the formulae: $\frac{tour\ length - known\ optimum}{known\ optimum}$. Figures [5-8] compare statistically the solutions found by the four aforementioned algorithms. Overall, the algorithms no 5 and no 2 have found the best solutions. Set side by side the biggest effects of the two generated algorithms no 2 and no 5 occurred the longest instances (i.e. U2152, USA13509 and D18512); the Mann-Whitney statistical tests consistently rejected the null hypothesis H_0 when $p = 0.05$. For the instance D1291, the Mann-Whitney statistical test rejected the null hypothesis H_0, when $p = 0.05$, only for the Algorithm no 5. Otherwise, the null hypothesis was accepted. Figures 3 and 4 shows the descent towards a minima during the search, when the four new solvers are applied.

Figure 3: Comparison of the fitness solutions during an MA search with a maximum 3000 evaluations were applied to the problem D18512.

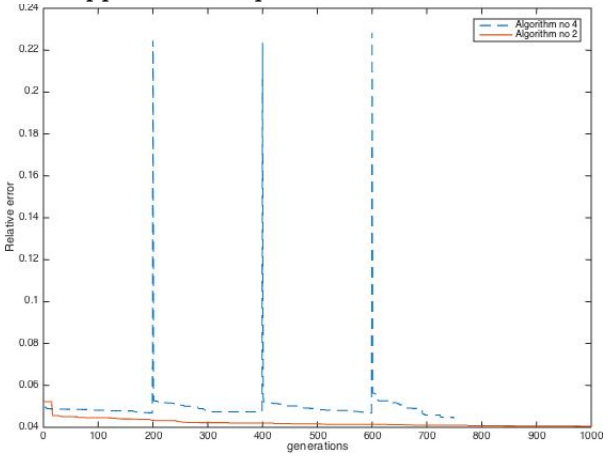


Figure 4: Comparison of the fitness solutions during an ILS search with a maximum 3000 evaluations applied problem D18512.

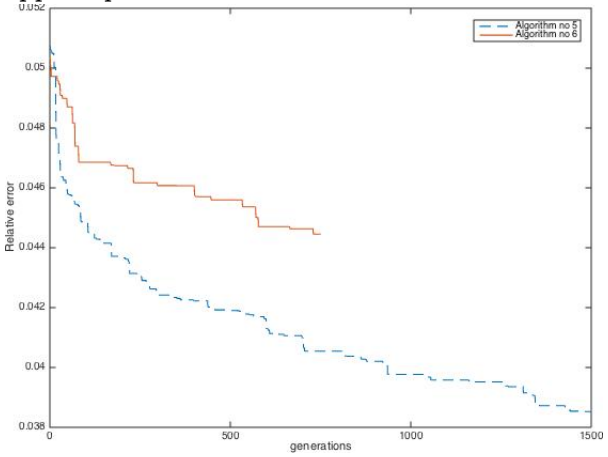


Figure 5: Statistical comparison of solutions of the TSP problem D1291 (1291 cities) found over 20 runs.

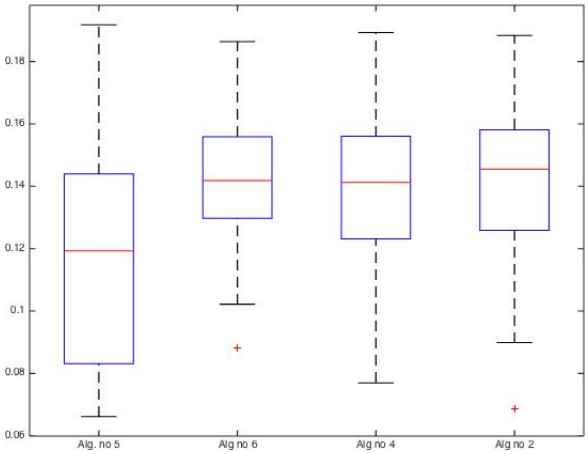


Figure 6: Statistical comparison of solutions of the TSP problem U2152 (2152 cities) found over 20 runs.

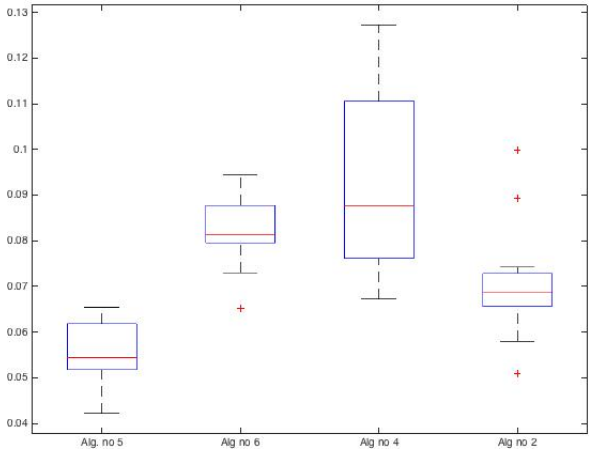


Figure 7: Statistical comparison of solutions of the TSP problem USA13509 (13509 cities) found over 20 runs.

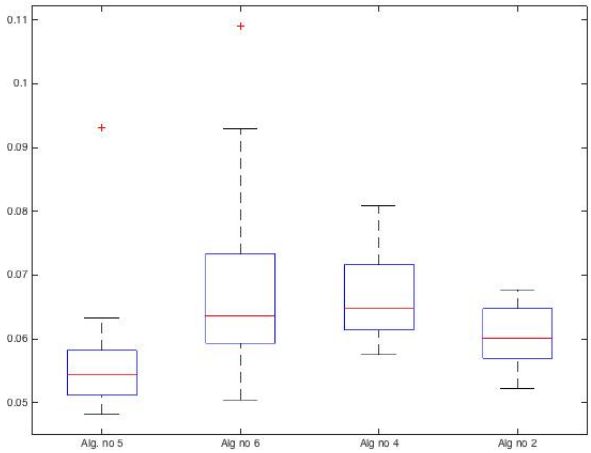
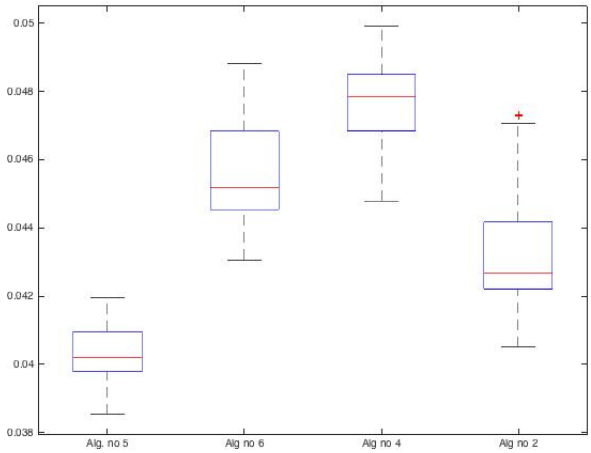


Figure 8: Statistical comparison of solutions of the TSP problem D18512 (18512 cities) found over 20 runs.



7. DISCUSSIONS AND CONCLUSIONS

We have presented a new approach to the evolution of metaheuristics; our variants have created four new TSP solvers and were applied to benchmark instances of the Travelling Salesman Problem. We show that not only can the method can produce human-readable, but also rediscover effective algorithms and generate new ones. Our sequence of operations were hard-coded in four new TSP solvers using Hyflex. However, since they are small and fairly simple they could be programmed outside Hyflex using other programming platforms and tested against a wider range of TSP instances. Interesting new variants of metaheuristics have been generated by our method. The results of our experiments are promising. Close solutions to the actual known optima have been found for the TSP benchmarks. We would have preferred to have established new global optima though. However, this could be just a matter of more evaluations. Nonetheless, we believe evolutionary cross-domain hyper-heuristics needs to be applied to other problems, such the personal scheduling and the vehicle routing problems to generate new hybrid metaheuristics. Then the full potential of this technique can be fully evaluated. These new hybrids of metaheuristics could have more freedom if the template used to safeguard the validity of the hybrid metaheuristic could be relaxed. Perhaps encoding iteration in a form of GP, could generate more complex human-readable algorithms, that have yet to be thought by humans.

In the future we intend to investigate less restrictive patterns of instructions, in order to learn new possible sequences of operations that humans have not yet though of, but which still remain readable and understandable. Perhaps these new algorithms could lower the known minima of these instances. It would also be interesting to use an encoding scheme that is expressive enough to encode more challenging programming structure (i.e. iterations). It has recently been shown that CGP can easily encode repeated loops with good results [40]. With such approaches the level of complexity of the algorithm could increase without losing its "human readability".

8. ACKNOWLEDGMENTS

Thanks to Ender Ozcan and Gabriela Ochoa for kindly answering to all our requests. This work made use of the facilities of N8 HPC provided and funded by the N8 consortium and EPSRC (Grant No.EP/K000225/1). The Centre is co-ordinated by the Universities of Leeds and Manchester.

9. REFERENCES

[1] Wolfgang Banzhaf. The "molecular" traveling salesman. *Biological Cybernetics*, 64(1):7–14, 1990.

[2] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[3] Lawrence Davis et al. *Handbook of genetic algorithms*, volume 115. Van Nostrand Reinhold New York, 1991.

[4] David B Fogel. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 60(2):139–144, 1988.

[5] Fred Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1):223–253, 1996.

[6] David E Goldberg and Robert Lingle. Alleles, loci, and the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, volume 154. Lawrence Erlbaum, Hillsdale, NJ, 1985.

[7] John J Grefenstette. Incorporating problem specific knowledge into genetic algorithms. *Genetic algorithms and simulated annealing*, 4:42–60, 1987.

[8] Aldy Gunawan, Hoong Chuin Lau, and Mustafa Mısır. Designing a portfolio of parameter configurations for online algorithm selection. 2015.

[9] Gregory Gutin and Daniel Karapetyan. A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, 9(1):47–60, 2010.

[10] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[11] Karla L Hoffman, Manfred Padberg, and Giovanni Rinaldi. Traveling salesman problem. In *Encyclopedia of Operations Research and Management Science*, pages 1573–1578. Springer, 2013.

[12] John H Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[13] Holger H Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012.

[14] David S Johnson. Local optimization and the traveling salesman problem. In *Automata, languages and programming*, pages 446–461. Springer, 1990.

[15] David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1:215–310, 1997.

[16] E Kasturi and S Lakshmi Narayanan. A novel approach to hybrid genetic algorithms to solve symmetric tsp. *International Journal*, 2(2), 2014.

[17] K Katayama, H Sakamoto, and H Narihisa. The efficiency of hybrid mutation genetic algorithm for the travelling salesman problem. *Mathematical and Computer Modelling*, 31(10):197–203, 2000.

[18] Oliver Kramer. Iterated local search. In *A Brief Introduction to Continuous Evolutionary Optimization*, pages 45–54. Springer, 2014.

[19] Natalio Krasnogor, Jim Smith, et al. A memetic algorithm with self-adaptive local search: Tsp as a case study. In *GECCO*, pages 987–994, 2000.

[20] Jiří Kubalík. Hyper-heuristic based on iterated local search driven by evolutionary algorithm. In *Evolutionary Computation in Combinatorial Optimization*, pages 148–159. Springer, 2012.

[21] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[22] S Lin and BW Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[23] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and

applications. In *Handbook of Metaheuristics*, pages 363–397. Springer, 2010.

[24] Nuno Lourenco, Francisco Pereira, Ernesto Costa, Gisele L Pappa, and John Woodward. Evolving evolutionary algorithms. In *GECCO 2012 2nd Workshop on Evolutionary Computation*, pages 51–58. ACM, 2012.

[25] Richard J Marshall, Mark Johnston, and Mengjie Zhang. Hyper-heuristic operator selection and acceptance criteria. In *Evolutionary Computation in Combinatorial Optimization*, pages 99–113. Springer, 2015.

[26] H Miihlenbein and J Kindermann. The dynamics of evolution and learning-towards genetic neural networks. *Connectionism in perspective*, pages 173–197, 1989.

[27] J. F. Miller, editor. *Cartesian Genetic Programming*. Natural Computing Series. Springer, 2011.

[28] Julian F Miller, Dominic Job, and Vesselin K Vassilev. Principles in the evolutionary design of digital circuits—part i. *Genetic programming and evolvable machines*, 1(1-2):7–35, 2000.

[29] Mustafa Mısır, Stephanus Daniel Handoko, and Hoong Chuin Lau. Oscar: Online selection of algorithm portfolios with case study on memetic algorithms.

[30] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. A new hyper-heuristic as a general problem solver: an implementation in hyflex. *Journal of Scheduling*, 16(3):291–311, 2013.

[31] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[32] Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J Parkes, Sanja Petrovic, et al. Hyflex: A benchmark framework for cross-domain heuristic search. In *Evolutionary Computation in Combinatorial Optimization*, pages 136–147. Springer, 2012.

[33] Gabriela Ochoa, Rong Qu, and Edmund K Burke. Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 341–348. ACM, 2009.

[34] Mihai Oltean and Crina Groşan. Evolving evolutionary algorithms using multi expression programming. In *Advances in Artificial Life*, pages 651–658. Springer, 2003.

[35] Gisele L Pappa, Gabriela Ochoa, Matthew R Hyde, Alex A Freitas, John Woodward, and Jerry Swan. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 15(1):3–35, 2014.

[36] Riccardo Poli and Mario Graff. There is a free lunch for hyper-heuristics, genetic programming and computer scientists. In *Genetic Programming*, pages 195–207. Springer, 2009.

[37] César Rego, Dorabela Gamboa, Fred Glover, and Colin Osterman. Traveling salesman problem heuristics: leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3):427–441, 2011.

[38] Peter Ross. Hyper-heuristics. In *Search Methodologies*, pages 611–638. Springer, 2014.

[39] Patricia Ryser-Welch and Julian F Miller. A review of hyper-heuristic frameworks.

[40] Andrew James Turner and Julian Francis Miller. Recurrent cartesian genetic programming. In Thomas Bartz-Beielstein, Jürgen Branke, Bogdan Filipič, and Jim Smith, editors, *Parallel Problem Solving from Nature – PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 476–486. Springer International Publishing, 2014.