Exploring the Landscape of the Space of Heuristics for Local Search in SAT

Andrew W. Burnett
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
Email: psxab4@exmail.nottingham.ac.uk

Andrew J. Parkes
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, UK
Email: andrew.parkes@nottingham.ac.uk

***Abstract*—Local search is a powerful technique on many combinatorial optimisation problems. However, the effectiveness of local search methods will often depend strongly on the details of the heuristics used within them. There are many potential heuristics, and so finding good ones is in itself a challenging search problem. A natural method to search for effective heuristics is to represent the heuristic as a small program and then apply evolutionary methods, such as genetic programming. However, the search within the space of heuristics is not well understood, and in particular little is known of the associated search landscapes. In this paper, we consider the domain of propositional satisfiability (SAT), and a generic class of local search methods called 'WalkSAT'. We give a language for generating the heuristics; using this we generated over three million heuristics, in a systematic manner, and evaluated their associated fitness values. We then use this data set as the basis for an initial analysis of the landscape of the space of heuristics. We give evidence that the heuristic landscape exhibits clustering. We also consider local search on the space of heuristics and show that it can perform quite well, and could complement genetic programming methods on that space.**

I. Introduction

Many hard combinatorial optimisation problems can be effectively addressed using local search methods. To provide more effective local search algorithms, heuristics are employed to direct the search. However, designing an effective heuristic, for use within a local search, can be a difficult task. Tree based Genetic Programming (GP) [12], a program synthesis technique, has previously been used to create heuristics in this way [8]. GP is itself a search on the space of semantically valid programs in a given language; hence, using GP, with a language that describes heuristics, becomes a search on the space of heuristics. In this paper, we investigate the nature of the space of heuristics for solving SAT through a particular kind of local search, with the intent to improve search methods, and to ultimately find better heuristics in a more effective manner. For clarity, we emphasise that the search process, and landscapes, that we consider are on the space of heuristics used for solving SAT, and *not* on the original SAT problem.

The structure of this paper is as follows. In Section II, we describe the SAT problem domain, the pertinent local search methods, and discuss previous work in the automated synthesis of heuristics. We also give the language we use to encode the heuristics. In Section III, we give the methods we use to generate extensive data on the space of heuristics, and give an some initial landscape study, as well as describing the tree-edit distance metric [14] that we use. In Section IV, we consider local search methods, on the search space of heuristics, and give the results of a study of the local optima found. Finally, in Section V, we discuss overall conclusions and future work.

II. Problem Domain

The Boolean Satisfiability Problem (SAT) is a standard decision problem; given a propositional formula P, (in clausal normal form, CNF[1]) over a set of propositional variables v_i, determine if there exists an assignment of truth values to the variables such that P evaluates to $True$. Exact SAT algorithms [9] allow us to definitively answer this question on some classes of SAT instances. However, local search algorithms also perform well on some classes of instances; or when searching for a solution that maximally satisfies the set of clauses (MaxSAT), even if the entire set of clauses cannot be satisfied [10].

Algorithm 1 Pseudo-code for local search on SAT

```
Precondition: P: A SAT formula
  function LOCAL-SEARCH(P)
    solution ← INITIALISATION(P)
    if solution satisfies P then
      return solution
    for 0 → maxIterations do
      v_i ← HEURISTIC
      solution ← solution{v_i = ¬v_i}
      if solution satisfies P then
        return solution
    return No-Solution
```

Local search is a (metaheuristic) technique that works by first generating a candidate solution, then iteratively aims to improve the solution by modifying a (small) part of it. In SAT, a candidate solution can be represented as array of boolean variables with each index in the array referring to a

[1]CNF is a conjunction of clauses; a clause is a disjunction of literals; a literal is a variable or its negation.

978-1-5090-4601-0/17/$31.00

variable in the problem. In this context, the heuristic chooses an index, and then "flips" the variable value from $False$ to $True$ or vice-versa. Algorithm 1 shows the pseudo-code for this procedure.

Local search algorithms for SAT have gone through multiple improvements since the introduction of GSAT [16] a hill-climbing SAT heuristic. In particular, work such as that on WalkSAT [15], Novelty [13] and, more recently, ProbSAT [2] and configuration checking [5] have developed new heuristics, and greatly enhanced the performance of local-search based SAT solvers.

Separately, Fukunaga [8] observed that in the (then) better-performing local-search SAT heuristics, there existed several common metrics and programmatic structures used in many of their formulations. For example, GSAT and WalkSAT pick a variable from a broken clause according to their net gain and negative gain respectively. Fukunaga referred to these, and an additional positive gain, as the "gaintype". These are defined as follows:

- *PosGain* The number of clauses that will become $True$ if this variable is flipped
- *NegGain* The number of clauses that will become $False$ if this variable is flipped
- *NetGain* The net change in the number clauses that will become $True$ if this variable is flipped. Positive denotes it increases, negative denotes it decreases

From these, and other observations of similarity between heuristics, Fukunaga defined a common language that could express many of these heuristics through combinations of the functions and terminals. Program synthesis through tree-based GP was then performed within this language to synthesize new local-search SAT heuristics. The results showed that these heuristics were able to perform as well as those then existing in the literature.

Some subsequent research in using program synthesis techniques to create heuristics for solving combinatorial optimisation problems has recently been grouped under the broad label of *generative hyper-heuristic* [3]. This term includes the generation of bespoke heuristics for a specific problem domain; GP is one of the methods commonly used as a 'generative hyper-heuristic'.

In previous work [4], we used a subset of Fukunaga's language to systematically generate all heuristics that contained 15 or fewer symbols; that is, all heuristics up to *depth* 15. The language used can be seen in Tables I and II. This depth was chosen as several instances of the classic WalkSAT algorithm [15] can be found at depth 14, each with varying noise parameters. An example of WalkSAT encoded in the language is shown in Figure 1.

Using systematic generation produces around three million heuristics. We then took each of these heuristics and ran them against a set of 100 satisfiable SAT instances taken from the SATLIB[2] library of SAT problems and scored them according to the following formula:

[2]http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

```
WalkSAT(0.5):
 IfVarCond NegGain == 0
   (GetBestVar NegGain BC0)
   (IfRandLt 0.5
     (GetBestVar NegGain BC0)
     (VarRandom BC0))
```

Fig. 1. The WalkSAT heuristic with a noise parameter of 0.5 written in the language shown in Tables I and II

$$\#\ problems\ solved + \frac{1}{\#Sum\ of\ flips\ to\ solution}$$

We repeated this experiment 5 times and took the average to be that heuristic's fitness. The original WalkSAT variants had a fitness of approximately 20 using this measure. It was found that around 1% of all the heuristics up to a depth of 15 had a score greater than 20, and the maximum score was of 45.0. This gave evidence that better heuristics are potentially available with program sizes that are within the reach of (partial) systematic enumeration. This suggests that search methods other than GP may also be effective, and also that there is the potential to perform extensive studies of the space of heuristics, rather than having to rely on a (sparse) sampling.

III. Initial Landscape Studies

In Figure 2 we show the approximately two million heuristics that contain exactly 15 symbols and their associated fitness. The heuristics are presented in the order that they are systematically generated. The generation algorithm is simple; upon initialisation, a program tree with a single node is generated containing no symbol, but with an identifier representing what the type of the expression will be that is contained this node (in this case, of type `Var`) and put into the list L. The algorithm then continues to do the following iteratively until L is empty. It takes the first program tree p from L and performs an in-order traversal of p, finding the first occurrence of a node with a annotated with a "place-holder" type that needs to be fulfilled. Each symbol in the language that could satisfy this type are generated and inserted into this node. If any additional arguments are required, they are inserted as children to the inserted node as new place-holder nodes and the set of p's with one additional argument is then inserted into L. If the size of p is too large, then no successors are generated and, if p has no place-holder nodes, then the program-tree is returned as a valid program.

In the graph in Figure 2, this means that the closer two heuristics are together on the x axis, the more likely they are to share a similar initial structure. Figure 2 can be expected to give some indication of the landscape structure.

In this form, we can see that there exist regions that have a greater density of better performing heuristics, and so there is some clustering, but also potentially a somewhat rugged landscape. However, with the data presented in this fashion, it does not accurately reflect the close relationship that two

TABLE I
THE SET OF FUNCTIONS IN THE LANGUAGE

Name	Type	Arguments	Description
`VarRandom`	Var	VarSet vs	Choose a variable randomly from vs
`IfRandLt`	Var	Probability p Var $v1$, Var $v2$	With random probability p, choose $v1$ else choose $v2$
`GetBestVar`	Var	GainType g VarSet vs	Return variable with best GainType g from VarSet vs. Break ties randomly
`GetSecondBestVar`	Var	GainType g VarSet vs	Return variable with second best GainType g from VarSet vs. Break ties randomly
`GetOldestVar`	Var	Var v1, Var v2	Pick the oldest variable from $v1$ and $v2$
`IfTabu`	Var	Age a Var $v1$, Var $v2$	If $v1$'s Age is less than a choose $v2$, else choose $v1$
`IfVarCompare`	Var	Gaintype g Comparator c Var $v1$, Var $v2$	If the result of g c $v1$ $v2$ returns $True$ then $v1$ else $v2$. For example, `IfVarCompare PosGain` $\leq$ a b checks if the `PosGain` of a is $\leq b$. If $True$, choose a else b
`IfVarCond`	Var	Gaintype g Comparator c Integer i Var $v1$, Var $v2$	If the test g c i $v1$ returns $True$, then choose $v1$ else choose $v2$ For example, `IfVarCond PosGain` $= 0$ a b checks if the `PosGain` of $a = 0$. If $True$ choose a else b
`IfNotMinAge`	Var	VarSet vs Var $v1$, Var $v2$	If $v1$ does not have minimal age among the VarSet vs then choose $v1$ else $v2$

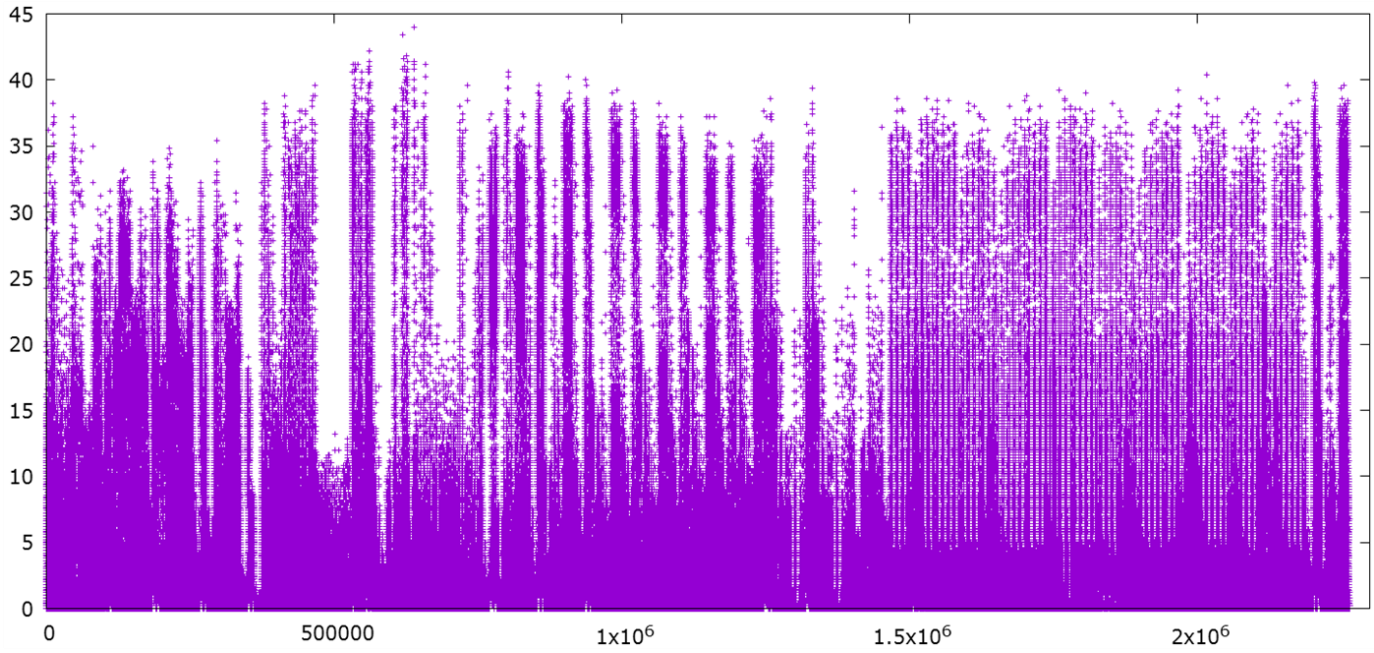

Fig. 2. All heuristics that contain exactly 15 symbols in the order they are generated (x-axis), plotted against their associated fitness (y-axis).

TABLE II
THE SET OF TERMINALS IN THE LANGUAGE

Type	Values
Probability	0.1, 0.3, 0.5, 0.7, 0.9
Integer	-2, -1, 0, 1, 2, 3, 4, 5
Comparator	$<$, $\leq$, $=$
GainType	PosGain, NetGain, NegGain
VarSet	BrokenClause0

heuristics may have when they are nearly identical, except that they initially used a different construct. In essence they could only differ by a single function or terminal, but be very far apart in the sequence (x-axis) in Figure 2.

Accordingly, to investigate the similarity between any two heuristics, we use the *minimum tree-edit distance* [14], a method that has been used previously to compare tree-like program structures in GP [14]. This is defined as the size of the minimum set of the following operations to get from one tree to another:

- *Relabel:* Change the label on a node in the tree to another label
- *Delete:* Remove a node n from the tree and, if n had children, make them the children of n's parent node
- *Insert:* Insert a node as a child in a currently existing node

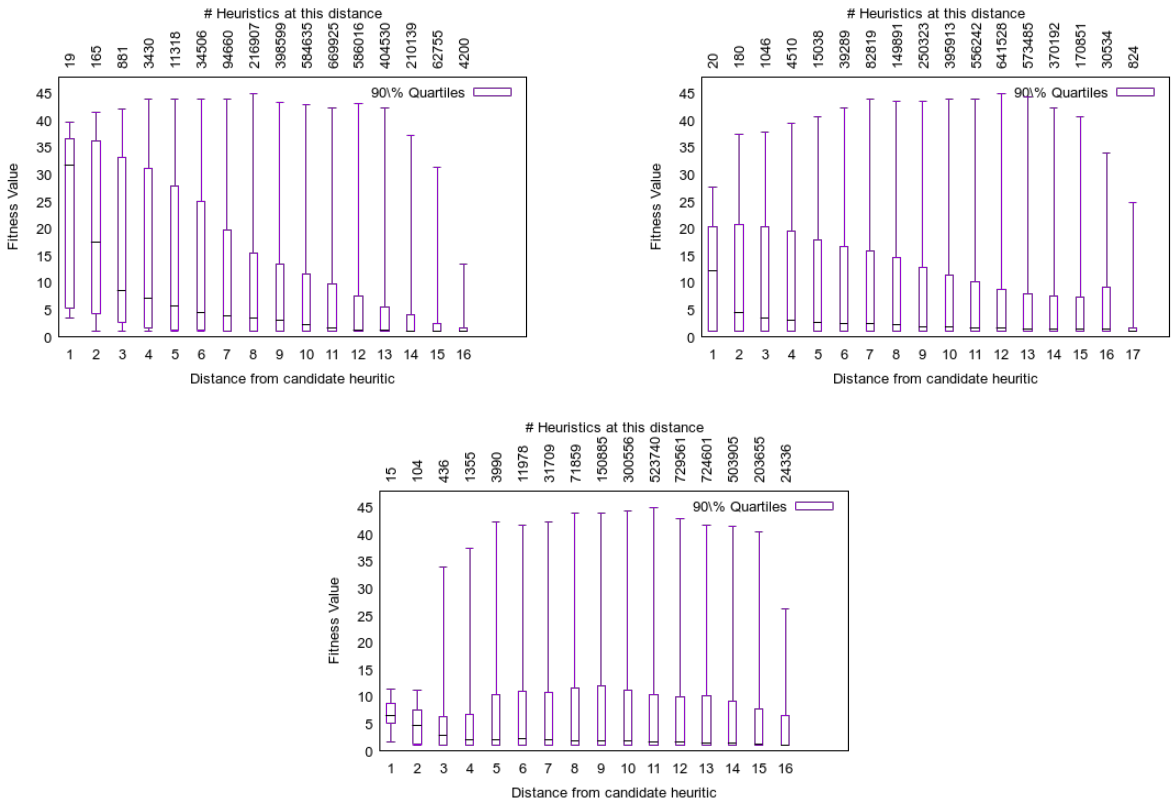

Fig. 3. Box and whisker plots of typical heuristics compared to all other heuristics within the data-base. At each distance, from the candidate we show (using a whisker plot) the median fitness, the 90% quartiles of fitness, and the minimum and maximum fitness values. On the top x-axis is the number of heuristics at that distance. Top-left: A heuristic of good quality with fitness of 30.4. Top-right: A heuristic of WalkSAT-like quality of fitness 21.4. Bottom: A heuristic of poor quality with fitness 7.8.

Using the tree-edit distance metric described above, we can now compare any two heuristics. Using this metric, we firstly looked at a random selection of good and bad performing heuristics, and computed the distance for all other heuristics from these. We then put these heuristics into sets ordered by their distance from the candidate, and gathered statistics on the fitness of the heuristics in each set.

As an example, we show three typical heuristic's results in Figure 3. One clear observation we can make is that for these candidate heuristics, the number of heuristics within an edit distance of 3 is small compared to the whole search space. We can also see that, on the two examples of better performing heuristics, other better quality heuristics exist at small distances in comparison to the worse performing heuristic.

IV. Local Search for Local Optima

The results in the previous section suggest that some heuristic within a distance of 3 of a candidate heuristic may well have a better fitness; and also the number of heuristics within this neighbourhood would not be excessive. Hence, in this section, we describe experiments using direct simple hill-climbing, with restarts, on the space of heuristics; investigating the effectiveness of the search in terms of the fitness of the resulting local optima, and also doing an initial study of their distribution in the search space.

Our experiments are designed as follows; given a set of heuristics, we pick a random starting point and perform local search on it, with the neighbourhood of heuristics consisting of those heuristics within a minimum tree-edit distance of 1, 2 or 3 of the current candidate. From this set we either pick the heuristics randomly (terminating when we have evaluated all neighbours and no better successor has been found that improves on the current candidate) or we pick the best heuristic in the neighbourhood (terminating when a heuristic does not have a better neighbour). We consider three sets of heuristics, consisting of all heuristics up to the depths of 13, 14 and 15, respectively. We repeated these experiments multiple times: $1,000$ times for depth 13; 500 times for depth 14; and 100 times for depth 15. It is worth noting that during these experiments we used the pre-computed fitness so that computation time was greatly reduced, enabling a much better exploration of different local search options.

The results of these experiments are given in Figures 4, 5, 6 and 7; and show the fitnesses of the final solutions arrived at by the local searches. Note that different runs of the local search can finish at the same local optima, and so give duplicate heuristics. Accordingly, in the results we have also provided graphs of the results with the duplicate heuristics removed. We

have only included the results from the best heuristic method of local search for heuristics at depth 13, as the results at the other depths are very similar to the results for the random-first method. (Due to the nature of the fitness evaluation as a floating point number arising from detailed tests on multiple instances, then there is a very little chance that two different heuristics will have the same fitness value. So there are no plateaux or neutral directions in the landscape, and these need not be accounted for in this work; in particular, duplicates in the fitness function do correspond to duplicates in the heuristics discovered.)

We can see that, with duplicates removed, the number of actual heuristics that these searches arrive at is reduced significantly. In Figure 4(right) there are only 80 unique heuristics from the total 1,000 runs performed. At higher heuristic depth this effect is lessened. This suggests that there are many local-optima among solutions, and this method of local-search allows this to be illustrated.

Furthermore, unsurprisingly, we can see that the fitness of the final solutions is generally better when the minimum tree-edit distance is larger. Also, of interest are the areas on these graphs where it appears that there is a clustering of heuristics around a certain fitness - for example at distance 2 on Figure 6(left), with a fitness level of approximately 25. Of these runs, we found that on average the changes made to the current best heuristic for the random method was between 5 and 20. The average number of heuristics considered as candidates at distance 1 was between 30 and 50. For distance 2, between 250 and 400 and for distance 3 between 1800 and 2300.

Finally, we remark that 80 - 90 % of runs generate a heuristic that is of comparable or better fitness than the original WalkSAT achieved on our fitness function.

A. Analysis of the good optimal solutions

In the previous experiments, we noted that in comparison to the number of runs, very few unique heuristics were generated. This can be seen in Figure 4(right). Using the minimum tree-edit distance, we computed the distance matrix between the 80 unique heuristics found for the random-first sampling method, with a tree-edit distance of 3, on the set of heuristics up to depth 13. From these, taking the 20 heuristics with the best fitness, we show the partial distance matrix in Table III. Note that the smallest distance allowed in the matrix is 4, because if the distance were 3 or less then one of the heuristics would be better and so not a local optimum. When considering the whole matrix of the 80 unique heuristics, the largest distance was 15. We can see that in the partial distance matrix between the top 20, the largest distance is just 12 and even that is fairly rare. For the sub-matrix between the top 9 heuristic, the largest distance drops to be only 9. We believe this gives initial indications of a general structure to the space of heuristics – maybe even a broad big-valley structure.

Figure 8 gives the plot of fitnesses of the 80 local optima as a distance from the best heuristic of that set. It gives some evidence of a fitness-distance correlation; as we move further away from the best heuristic then the fitness tends to drop.

V. Conclusions and Future Work

In this paper, we considered the search for heuristics to be used within local search to solve SAT. We consider heuristics in the same style as variations of the WalkSAT algorithm, and gave the language/grammar for such variations. We were then able to systematically generate all heuristics to a given depth (number of terminals in the heuristic), and furthermore were able to evaluate a measure of the fitness of each of these heuristics be (exhaustively) running them on a standard suite of SAT instances. With a choice of depth 15 as the largest size (maximum number of terminals) this resulted in generating a dataset[3] of just over three million heuristics together with their fitness values. This dataset contained heuristics that were at least as good as the original WalkSAT; one of surprises to us was that 'only' three million were needed. Note that such numbers of heuristics are in the realm of 'big data' and so the data set is a strong candidate for analysis using big data techniques. The relatively manageable size of the useful space of heuristics is in strong contrast to the underlying combinatorial problems where the size of the search space is invariably far too large for enumeration.

The explicit complete enumeration of the heuristics, then gave a novel opportunity to be able to study the nature of the associated landscape; and for the notion of distance we used the 'tree-edit distance'. Initial studies based on sampling within the set of heuristics gave evidence of a clustering; better heuristics tend to be more likely to be close to other good heuristics. In fact it seems that within a distance of just 3 there were often better heuristics, and this suggested studying local search on the space. The dataset allowed to do this conveniently and relatively quickly by using it as a cache of the fitness value. Simple hill-climbing tended to reach a local optimum within typically 5-20 moves, and depending upon the distance used, between 50 and 2300 candidate heuristics considered. Running the hill-climbing many times with random initial starting positions then allowed us to collect sets of local optima. With neighbourhoods of distance 3, around 80 - 90 % of hill-climbing runs generated a heuristic that had a better fitness than the original WalkSAT (though not as good as more recent advances in local search SAT heuristics), demonstrating the potential power of such a simple method. We performed an initial analysis of the resulting set of local optima; finding that they were quite widely distributed in the search space, but did show signs of some clustering. Other evidence for the clustering, or exploitable structure, was that many runs of the hill-climbing, from random initial heuristics, ended up at the same final heuristic. To the best of our knowledge, these results provide the first large-scale study of the landscape of the space of (SAT) heuristics represented as tree-based programs. (Though note a landscape study of heuristics, represented as matrices rather than trees, was given in [1].) Hopefully, this study will provide a potential basis for insights into genetic programming and other related methods, such as hyper-heuristics, or heuristics in general.

[3]Which we intend to make openly available in due course.

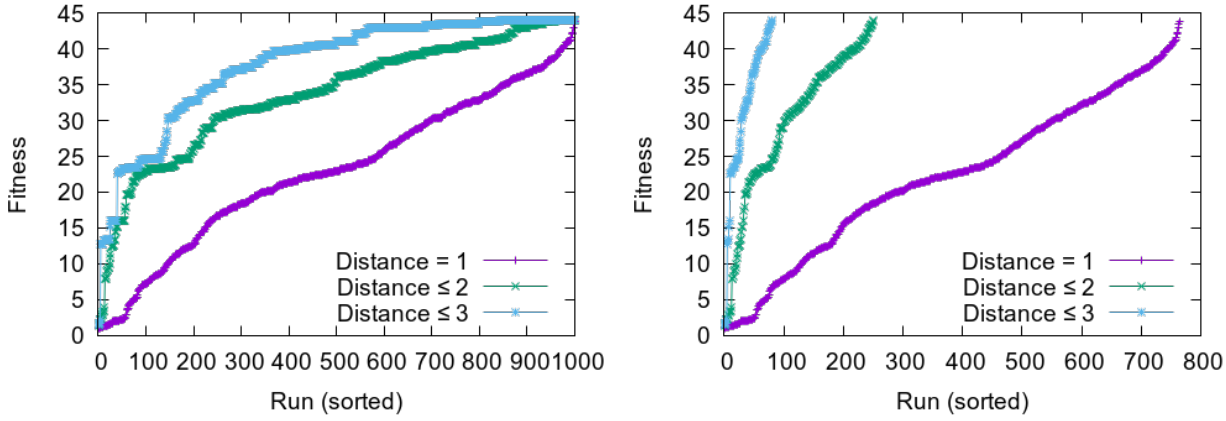

Fig. 4. Graph of the final heuristics returned after 1000 runs of random-greedy local search on all heuristics at depth 13 or below, terminated when no neighbours that are within tree-edit distance of 1,2 or 3 of the candidate solution. Left: all results, Right: duplicates removed

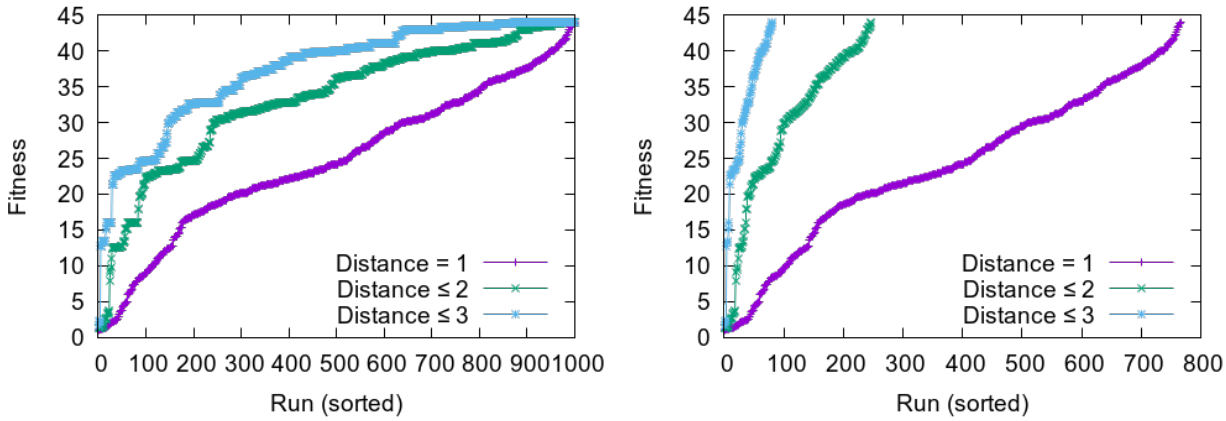

Fig. 5. Graph of the final heuristics returned after 1000 runs of best-greedy local search on all heuristics at depth 13 or below, terminated when no neighbours that are within tree-edit distance of 1,2 or 3 of the candidate solution. Left: all results, Right: duplicates removed

0																		
6	0																	
6	5	0																
6	8	7	0															
5	6	8	4	0														
6	6	4	5	6	0													
6	6	6	5	4	4	0												
7	6	8	8	6	6	9	0											
6	5	6	6	6	5	8	4	0										
6	10	8	6	8	9	8	10	8	0									
7	7	8	5	7	6	4	10	8	8	0								
7	4	7	7	7	6	6	6	4	8	6	0							
7	8	10	10	7	10	10	8	9	11	11	9	0						
9	8	10	9	6	8	10	5	7	12	11	9	5	0					
10	10	12	8	7	10	10	9	8	11	10	9	11	10	0				
9	10	10	7	7	8	8	9	8	9	10	9	11	8	5	0			
10	10	9	8	9	7	9	7	7	11	11	9	10	7	9	7	0		
7	7	6	9	10	8	10	9	6	4	10	7	11	11	13	11	10	0	
8	10	11	7	6	9	9	10	10	10	10	11	9	9	7	5	9	12	0
8	9	8	9	8	8	9	10	10	12	9	8	6	9	10	13	8	11	11	0

TABLE III

EXAMPLE OF A DISTANCE MATRIX. USING THE TOP 20 LOCALLY-OPTIMAL HEURISTICS OBTAINED AT LEVEL 13, WITH RANDOM HILL-CLIMBING, AND NEIGHBOURHOOD SIZE 3. THE BEST HEURISTIC IS AT THE TOP, SO THE QUALITY DROPS AS GOING DOWN THE ROWS. (ONLY THE LOWER LEFT IS SHOWN AS THE MATRIX IS SYMMETRIC.)

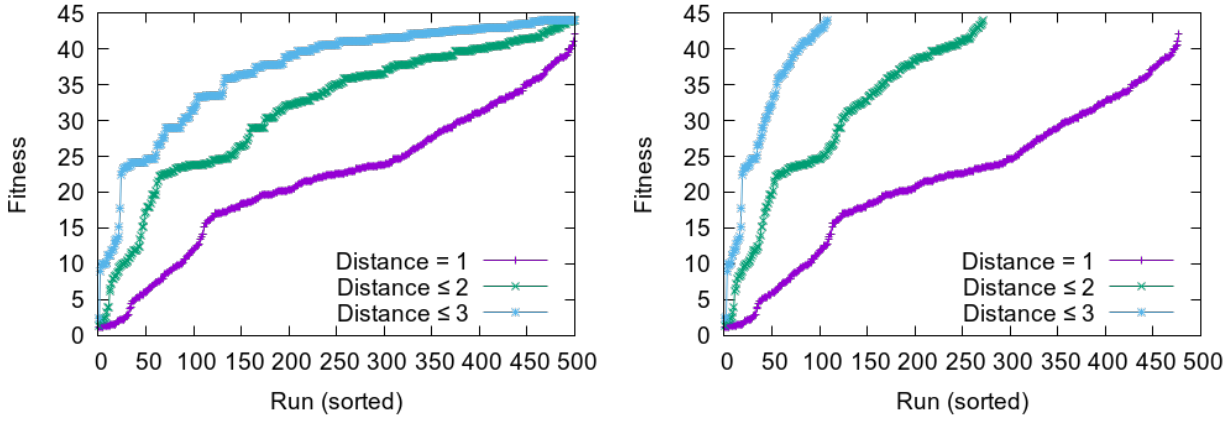

Fig. 6. Graph of the final heuristics returned after 500 runs of random-greedy local search on all heuristics at depth 14 or below, terminated when no neighbours that are within tree-edit distance of 1,2 or 3 of the candidate solution. Left: all results, Right: duplicates removed

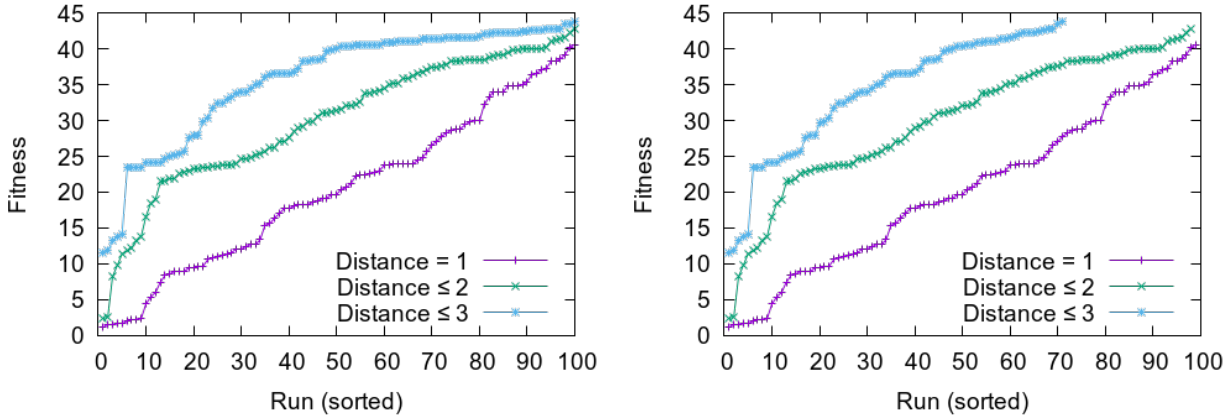

Fig. 7. Graph of the final heuristics returned after 100 runs of random-greedy local search on all heuristics at depth 15 or below, terminated when no neighbours that are within tree-edit distance of 1,2 or 3 of the candidate solution. Left: all results, Right: duplicates removed

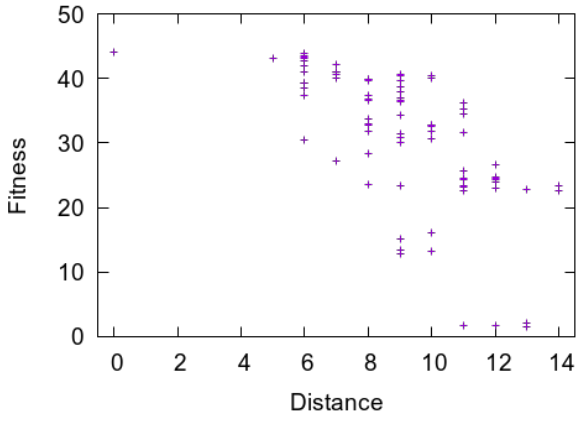

Fig. 8. Distance against fitness for the 80 unique heuristics from 1,000 runs of random-greedy local search with distance ≤ 3. The fitness distance correlation [11] - that is, the relationship between the fitness and the distance is -0.645876264 suggesting some correlation between fitness and distance

A. Future Work

Naturally, future work could be aimed at a deeper understanding of the landscape. Potentially, using a study of the space of local optima, e.g. by extending the methods discussed in [6], [17] (and references therein). It is worth noting that that such previous work was directly on the solution space; and so impractical for full-sized instances. However, this work concerns the landscape of the search space of heuristics. A 'solution' is then a heuristic and so is no longer directly tied to be the size of the underlying problem instance; the SAT heuristics are much smaller than the SAT problems that they are working on. This gives a direct relevance of the landscape properties of the heuristic search space, to realistic sizes of problems instances. The landscape analysis should be further extended with studies of different distance functions. Potentially, the analysis could result in generating a better distance function that gives neighbourhoods resulting in a more effective (cheaper) hill-climbing.

We also intend to apply a variety of machine learning or statistical analysis techniques to the data-set in order to 'data-mine' for properties of heuristics that are correlating with good (or bad) performance. The intent would be to generate insight and understanding of the space of heuristics, and use such knowledge to guide the search for better heuristics, and also

the workings of evolutionary algorithms (such as GP) on the space of heuristics.

Note that emulating the local search via caching the fitnesses allowed much faster study of the comparative value of different local search methods. Of course, in practice the local search would need to evaluate the fitness on demand, and then cache the values; however these results do still suggest that such methods could well be practical. The design of search algorithms working on the space of heuristics could take many standard ideas from metaheuristics and mix these together with a "GP" style. That is, extend crossover and mutation with neighbourhood search. Combining such local search with GP could give what one might call "Memetic Programming" (MP); where local search is added to GP. Previous work in this area of memetic genetic programming [7] has been proposed, where sub-trees of a candidate program are refined by local search through the use of fitness cases - pairs of input and desired output. In our domain, this is not applicable due to the nature of heuristics typically not having obvious examples of fitness cases. Our method does not use domain knowledge and instead works solely on the semantics of the language used. However, the success of memetic algorithms in general, and from the results from [7], suggest there is a potential role for such "memetic programming on the space of heuristics".

ACKNOWLEDGEMENTS

Andrew Burnett would like to thank the EPSRC for financial support, via the Doctoral Training Grant (DTG), grant number EP/L50502X/1. The data produced in this work, the set of heuristics and their associated fitness values, will be made publicly available[4].

REFERENCES

[1] Asta, S., Özcan, E., Parkes, A.J.: CHAMP: creating heuristics via many parameters for online bin packing. Expert Syst. Appl. 63, 208–221 (2016), http://dx.doi.org/10.1016/j.eswa.2016.07.005

[2] Balint, A., Schöning, U.: Choosing probability distributions for stochastic local search and the role of make versus break. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 16–29. Springer (2012)

[3] Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society 64(12), 1695–1724 (2013)

[4] Burnett, A.W., Parkes, A.J.: Systematic search for local-search SAT heuristics. In: Proceedings of META2016, 6th International Conference on Metaheuristics and Nature Inspired computing (2016)

[5] Cai, S., Su, K.: Local search with configuration checking for SAT. In: Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on. pp. 59–66. IEEE (2011)

[6] Chicano, F., Daolio, F., Ochoa, G., Vérel, S., Tomassini, M., Alba, E.: Local optima networks, landscape autocorrelation and heuristic search performance. In: Parallel Problem Solving from Nature - PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part II. pp. 337–347 (2012)

[7] Ffrancon, R., Schoenauer, M.: Memetic semantic genetic programming. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1023–1030. ACM (2015)

[8] Fukunaga, A.S.: Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation 16(1), 31–61 (2008)

[9] Gu, J., Purdom, P.W., Franco, J., Wah, B.W.: Algorithms for the satisfiability (SAT) problem. In: Handbook of Combinatorial Optimization, pp. 379–572. Springer (1999)

[10] Jiang, Y., Kautz, H., Selman, B.: Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In: 1st International Joint Workshop on Artificial Intelligence and Operations Research (1995)

[11] Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proceedings of the 6th International Conference on Genetic Algorithms. pp. 184–192. Morgan Kaufmann Publishers Inc. (1995)

[12] Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection, vol. 1. MIT press (1992)

[13] McAllester, D., Selman, B., Kautz, H.: Evidence for invariants in local search. In: Proceedings of National Conf. on Artificial Intelligence (AAAI). p. 459465 (1997)

[14] O'Reilly, U.M.: Using a distance metric on genetic programs to understand genetic operators. In: Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on. vol. 5, pp. 4092–4097. IEEE (1997)

[15] Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the Twelfth National Conf. on Artificial intelligence (vol. 1). pp. 337–343. American Association for Artificial Intelligence (1994)

[16] Selman, B., Levesque, H.J., Mitchell, D.G., et al.: A new method for solving hard satisfiability problems. In: Proceedings of National Conf. on Artificial Intelligence (AAAI). vol. 92, pp. 440–446 (1992)

[17] Tayarani-N., M.H., Prügel-Bennett, A.: On the landscape of combinatorial optimization problems. IEEE Transactions on Evolutionary Computation 18(3), 420–434 (June 2014)

[4] Via http://www.cs.nott.ac.uk/~pszajp/AB/