Modelling Genetic Improvement Landscapes with Local Optima Networks

Nadarajen Veerapen
Computing Science and Mathematics,
University of Stirling
Stirling, Scotland, UK
nve@cs.stir.ac.uk

Fabio Daolio
Computing Science and Mathematics,
University of Stirling
Stirling, Scotland, UK
fda@cs.stir.ac.uk

Gabriela Ochoa
Computing Science and Mathematics,
University of Stirling
Stirling, Scotland, UK
goc@cs.stir.ac.uk

ABSTRACT

Local optima networks are a compact representation of the global structure of a search space. They can be used for analysis and visualisation. This paper provides one of the first analyses of program search spaces using local optima networks. These are generated by sampling the search space by recording the progress of an Iterated Local Search algorithm. Source code mutations in comparison and Boolean operators are considered. The search spaces of two small benchmark programs, the triangle and TCAS programs, are analysed and visualised. Results show a high level of neutrality, i.e. connected test-equivalent mutants. It is also generally relatively easy to find a path from a random mutant to a mutant that passes all test cases.

CCS CONCEPTS

•Software and its engineering → Search-based software engineering; •Computing methodologies → Randomized search;

KEYWORDS

Fitness landscape, Local Optima Network, Genetic Improvement

ACM Reference format:
Nadarajen Veerapen, Fabio Daolio, and Gabriela Ochoa. 2017. Modelling Genetic Improvement Landscapes with Local Optima Networks. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082518

1 INTRODUCTION

Genetic Improvement (GI) is the use of Search-Based Software Engineering (SBSE) techniques to improve existing software. A number of challenges related to the use of computational search techniques and their specific application to optimising code have been identified by Langdon and Ochoa [5]. These include analysing the structure, or landscape, of program search spaces. Particular features of interest include the distribution of local optima and the neutrality of the landscapes because these usually influence how difficult it is to traverse the landscape and, thus, how difficult it is to solve the problem at hand. This paper uses so-called Local Optima Networks (LONs) to represent the global structure of program search space in order to analyse and visualise these spaces.

Within GI, a program is mutated in an attempt to improve some property that represents the fitness of the program, for instance the number of failed test cases. Neutrality is especially relevant because it is related to the proportion and distribution of test-equivalent mutants. Traditionally, the space of program mutants has been thought to be relatively disjoint, and with few good programs. However, recent work [6, 7, 12, 17] suggests that many changes do not impact the fitness of the mutants. This may either point to the programs being quite robust or to the test suite not providing enough coverage — which is relevant to mutation testing.

Various metaheuristics have been used in the SBSE literature. In this paper, we employ a perhaps less common — in the SBSE context — metaheuristic, which is Iterated Local Search (ILS). Nevertheless, it is a quite powerful yet simple approach based on multiple hill-climber applications and random perturbations. It is especially well-suited to our context since its search trajectories can be modelled by LONs with high fidelity.

The remainder of the paper is organised as follows: Section 2 presents our GI test bench; Section 3 provides definitions for fitness landscapes and LONs; results in terms of both network analysis and visualisations are presented in Section 4; threats to validity are identified in Section 5; and, finally, the conclusion is provided in Section 6.

2 PROGRAM SEARCH SPACE TEST BENCH

In order to study the landscape of program search spaces, we start off with known bug-free programs and introduce random mutations. Starting from these mutants, we try to recover the original bug-free programs or any version that passes all the test cases in the test suite.

We make the assumption that programmers usually create close-to-being-correct programs, i.e. they make minor mistakes such as typos or small logic errors. This is also known as the *competent programmer hypothesis* [2].

We examine two C programs, described in Section 2.1. For the sake of simplicity, in this paper we first consider mutations on comparison operators (<, <=, ==, !=, >=, >), as done by Langdon et al. [4], and then add mutations of Boolean operators with two operands (&&, ||).

Instead of generating new code that needs to be compiled for each new mutant, we use a *super-mutant* program that contains all the possible mutations under consideration [14]. These mutations can then be activated or turned off as desired. In our implementation, each operator is transformed into a function call with four

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3082518

```
if(side1 == side2){
  triang = triang + 1;
}
if(side1 == side3){
  triang = triang + 2;
}
```

(a) Original code snippet

```
if(SM(4,side1,side2,"=="){
  triang = triang + 1;
}
if(SM(5,side1,side3,"=="){
  triang = triang + 2;
}
```

(b) Super-mutant code snippet

Figure 1: Code snippets showing an example of super-mutant transformation where the two equality operators are the fourth and fifth operators in the code.

arguments: an operator id, the two operands of the operator, and a cosmetic final argument string that describes the original operator. This is mostly useful for a fast evaluation of the fitness function. An example of the transformation is shown in Figure 1.

The transformation is carried out using the LibTooling library of Clang–LLVM to parse the programs, build the abstract syntax trees, and rewrite the required nodes. Some additional manual steps are required to build our test harness, as described in Section 2.1.

Note that, in the programs we examine, no two operators are ever part of the same expression when only the comparison operators are considered. When both comparison and Boolean operators are considered, then operator precedence is enforced and left associativity is used for operators of similar precedence when the super-mutant is generated.

2.1 Benchmark Programs

We use two C programs: the triangle program and the TCAS program. Their characteristics are summarised in Table 1.

`triangle.c`: The triangle program is a small program that takes the lengths of the three sides of a triangle and determines if it is scalene, isosceles, equilateral, or not a triangle. We use a simplified version [4], which has been translated into C from the original Fortran version by DeMillo et al. [2].

`tcas.c`: The TCAS, or Traffic Collision Avoidance System, program controls the altitude of an aircraft depending on a number of input parameters. We use version 2.0 from the SIR repository [3][1]. For the sake of simplicity, we do not consider the test cases that do not have all 12 input parameters. This effectively reduces the number of test cases from 1608 to 1578. Array indices are not checked in the original program. We introduce a check to accept valid indices and generate an arbitrary output value for invalid indices. This prevents the program from crashing and improves the efficiency of the sampling process, which is described in the next subsection.

[1] http://sir.unl.edu/content/sir.php

Table 1: Characteristics of benchmark programs

Program	`triangle.c`	`tcas.c`
Lines of code	40	135
No. comparison operators	17	14
No. Boolean operators	7	16
No. input parameters	3	12
No. output values	1	1
No. test cases used (original)	14 (14)	1578 (1608)

2.2 Iterated Local Search

A full enumeration of the search space, or even of the local optima, for the two programs is unmanageable. Therefore a sample of high-quality local optima in the search space is generated.

The sampling algorithm is an Iterated Local Search, or ILS, (Algorithm 1) which starts from a locally-optimal solution and then alternates between a random mutation and a best-improvement hill-climber. The termination criterion is a fixed number of iterations. At each step, only non-worsening local minima are accepted. The fitness, or objective value, of a solution is the number of test cases that it fails. Both the hill-climber and the mutation consider the first degree or 1-move neighbourhood, i.e. neighbouring solutions only differ by a single element.

Algorithm 1 Iterated Local Search

$s_0 \leftarrow$ RandomInitialSolution
$s^* \leftarrow$ HillClimber(s_0)
repeat
  $s' \leftarrow$ RandomMutation(s^*)
  $s'^* \leftarrow$ HillClimber(s')
  if $f(s'^*) \leq f(s^*)$ **then**
    $s^* \leftarrow s'^*$
  end if
until *termination condition met*

3 LOCAL OPTIMA NETWORKS

Fitness landscapes are a commonly-used metaphor to describe the dynamics of evolutionary and local search heuristics. The search space can be regarded as a spatial structure with height representing fitness, forming a surface that can vary from smooth to rugged, and that can contain ridges and plateaus. Formally [13], a landscape is a triplet (S, N, f) where S is a set of potential solutions, i.e. a search space; $N : S \longrightarrow 2^S$, a neighbourhood structure, is a function that assigns to every $s \in S$ a set of neighbours $N(s)$, and $f : S \longrightarrow \mathbb{R}$ is a fitness function that can be pictured as the *height* of the corresponding solutions.

In our study, a potential solution is encoded as a vector of integers of length l, where $l = c + b$ corresponds to the number of comparison operators (c) and Boolean operators (b) in the program under consideration (Table1). There are 6 possible comparison operators and 2 Boolean operators. Therefore, the size of the search space is $|S| = 6^c \times 2^b$. The neighbourhood structure, N, is given by the simplest possible move operator in this landscape, namely,

the value of a single position in a solution is changed: to one of the 5 alternatives if it is a comparison operator, or the opposite Boolean operator. Let us call this operator *1-move*. The size of the neighbourhood induced by 1-move on the given representation is $5 \times c + b$. The fitness function f is given by the number of test cases failed by the program, which is to be minimised.

Local optima are important features of fitness landscapes as they can be seen as obstacles to the progress of heuristic search. Local optima networks (LONs) [9] model the global structure of landscapes as graphs where nodes are local optima and edges represent possible transitions among them with a given search operator. In order to model GI fitness landscapes with local optima networks, we adapted the model with escape edges [16]. To construct these networks, we need to define their nodes and edges. The definitions are related to the search operators used, specifically, the local search (hill-climbing) heuristic to determine the local optima and *escape* operator to transit among them. In our study, the hill-climbing heuristic is a best-improvement approach based on the 1-move operator, and the escape operator is also given by a single application of 1-move. This is possible as, given the problem encoding, a single 1-move provides enough variability to escape from a local optimum basin of attraction.

Local optima. A local optimum, which in GI landscapes is a minimum when considering the number of failed test cases, is a solution s^* such that $\forall s \in N(s^*)$, $f(s^*) \leq f(s)$. Notice that the inequality is not strict, in order to allow the treatment of the neutral landscape case.

The set of local optima, which corresponds to the set of nodes in the network model, is denoted by L. Since the whole set of local optima cannot be determined in realistic search spaces, such as those considered here, a process of sampling is required to estimate L.

Escape edges. Edges are directed and based on the 1-move operator. There is an *escape* edge from local optimum x to local optimum y, if y can be obtained after applying a 1-move random mutation to y followed by the best-improvement hill-climbing. The set of escape edges is denoted by E. Note that during the sampling process we only store edges that correspond to non-worsening transitions.

The type of edge is dependent on the algorithm used to sample the landscape. While escape edges are natural edges for ILS algorithms, other edge types may be used such as crossover and mutation edges in the case of an evolutionary algorithm, as done by Veerapen et al. [15].

Local Optima Network (LON). This is the $LON = (L, E)$ graph where nodes are the local optima L, and edges E are the escape edges. Edges are directed.

4 RESULTS

Sampling was carried out by running an ILS 1000 times. Within each ILS run, the stopping criterion corresponds to 10 000 calls to the hill-climbing procedure.

4.1 Visualisations

Figure 2 shows visualisations of the LONs, only for a subset of the sampled runs since representing large graphs is not straightforward on paper medium. The figures show the first 100 runs of the sampling process and the first 2000 iterations of each of these for the triangle program and the full 10 000 iterations for the tcas program. Nodes are not displayed: there would be roughly 100 000 of them in each figure. However, the edges by themselves provide insight into the nature of the landscapes. Edges between global optima are painted red and edges between local optima of equal fitness are painted grey. Edges between local optima with different fitness are painted black.

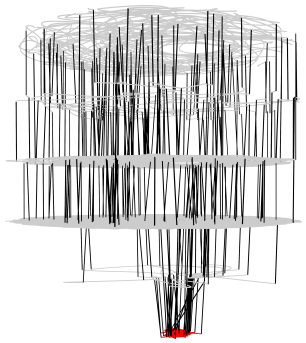
The layouts for the visualisations are first generated in two dimensions using a force-directed layout algorithm [8] and implemented in the igraph library [1]. This algorithm is able to deal with large graphs and tries to utilise space efficiently and to minimise overlap of edges and nodes (even if the latter are not rendered in our context). The third dimension, fitness, is then added to provide the height to the visualisation.

A common characteristic of all four visualisations is the clear presence of a large number of edges that connect nodes of equal fitness. These are plateaus at the LON level, or meta-plateaus. Let us note that meta-plateaus do not necessarily indicate plateaus at the solution level, only that a random mutation in the ILS more often than not leads to a new local optimum with the same fitness as before. These local optima may effectively be on the same plateau at the solution level, or be in two different basins of attraction. Although an abuse of language, we will use the term plateau instead of meta-plateau in the rest of the paper since we are always considering plateaus at the LON level.

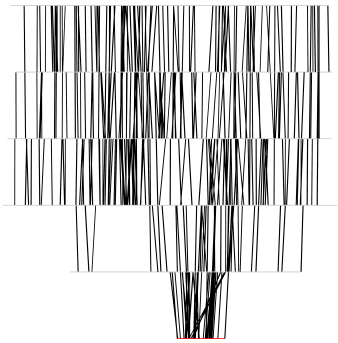
The worst plateau has fitness 5 for the triangle program — and there is a tiny fitness 6, easily escapable, plateau when both comparison and Boolean operators are considered. For the tcas program, the maximum fitness is 264. For both programs, we thus have plateaus that are well below the maximum fitness of 14 and 1578. Since random, not locally-optimal solutions, are often close to those maximum values, this indicates that it is fairly easy to improve the solution fitness with a simple hill-climber, at least initially.

The triangle program LONs are made up of 6 (fitness 0–5) large plateaus relatively well-connected between each other. However, when mutations on Boolean operators are introduced, more ILS runs get stuck at fitness 2 and are not able to progress to the global optima level. The ILS success rate, i.e. the proportion of runs that reach a global optimum, is measurably lower (87 % vs. 31 %).

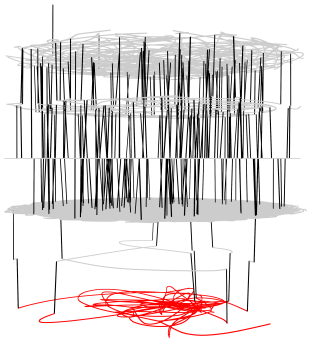
The tcas program LONs display more difference between them. Perhaps surprisingly, the variant that only considers comparison operators shows fairly well-defined plateaus — the three main ones have fitness 0, 144 and 264. This may be an artefact of some interaction between mutations. The study of these interactions is beyond the scope of this paper but seems to be an interesting area for future research. The variant with both comparison and Boolean operators shows more "steps" along the different runs and, therefore, less well-defined plateau structures. This potentially means that finding improving solutions, and ultimately a global optimum, is easier. Whilst the ILS success rate for both variants is quite high, there is a marked improvement for the second variant (from 94 % to 98 %).



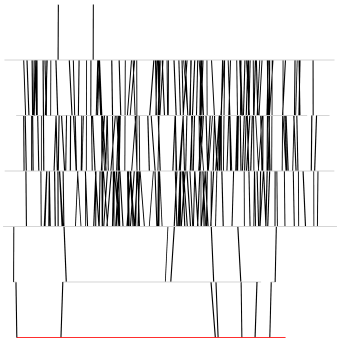
(a) triangle (comparison operators) – 3D view



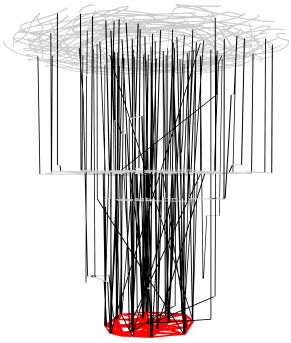
(b) triangle (comparison operators) – profile view



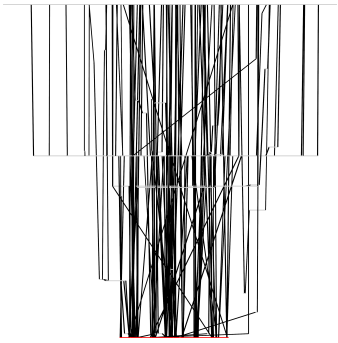
(c) triangle (comparison & Boolean operators) – 3D view



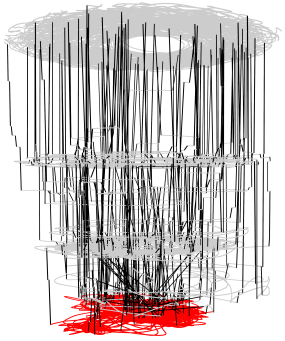
(d) triangle (comparison & Boolean operators) – profile view



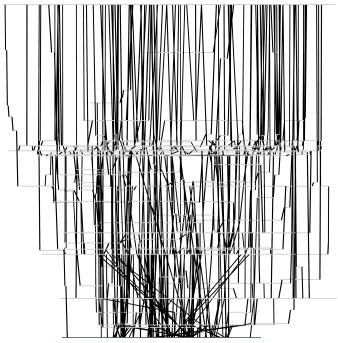
(e) tcas (comparison operators) – 3D view



(f) tcas (comparison operators) – profile view



(g) tcas (comparison & Boolean operators) – 3D view



(h) tcas (comparison & Boolean operators) – profile view

Figure 2: Local optima networks for the two variants of the two programs. Edges between global optima are painted red and edges between local optima of equal fitness are painted grey. Edges between local optima with different fitness are painted black.

Table 2: Network characteristics and ILS performance. The variant that considers only comparison operators is denoted by *c*, while the variant that considers both comparison and Boolean operators is denoted by *c+b*.

Program	`triangle.c`		`tcas.c`	
Variant	c	c+b	c	c+b
No. of nodes	2.4×10^{6}	4.1×10^{6}	8.6×10^{4}	5.0×10^{5}
No. of edges	2.8×10^{6}	4.7×10^{6}	6.4×10^{5}	1.4×10^{6}
No. of global optima	9.2×10^{3}	5.4×10^{4}	2.3×10^{4}	1.1×10^{5}
Network density	4.7×10^{-7}	2.8×10^{-7}	8.7×10^{-5}	5.6×10^{-6}
Clustering coefficient	2.4×10^{-3}	2.4×10^{-3}	4.4×10^{-2}	1.6×10^{-2}
Neutral degree	99.8 %	99.9 %	99.6 %	99.6 %
No. of connected components	3	3	2	12
Relative size of largest connected component	92.6 %	99.8 %	97.3 %	94.9 %
Nodes with path to global optimum	92.5 %	99.4 %	94.8 %	96.4 %
No. of sinks	4	5	3	12
No. of global sinks	1	1	1	5
ILS success rate	87.1 %	31.2 %	94.4 %	98.4 %

4.2 Network Statistics

Table 2 reports the main characteristics of the LON graphs extracted from the benchmark problems described in Section 2.1. The sampling procedure yielded, in all cases, graph sizes in the order of one million edges, which are non-deteriorating transitions between local minima. The actual number of distinct local minima visited during the search, that is, the number of nodes in the graph, is also in the order of one million for `triangle.c`, and one order of magnitude less in the case of `tcas.c`. In particular, allowing mutations to both comparisons operators and Boolean operators, increases the size of the search space and the number of local minima.

Let us observe that there is a high number of global optima, i.e solutions that are test-equivalent to the original programs. This may mean that the programs are quite robust — we have not tested this hypothesis — or that the test suite dose not provide enough coverage.

In all benchmarks, LONs are rather sparse but present patterns of local connectivity. In fact, the *clustering coefficient*, that is, the average proportion of transitive closures among the neighbours of a vertex, is always around four orders of magnitude higher than the overall *network density*. That is, connections between nodes that already share a neighbour, are orders of magnitude more frequent than connections in general, which could be explained by the fact that the LON graph displays the traces of iterated local search trajectories.

However, the great majority of these local connections happen on the plateaus that are well-visible in Figure 2. Indeed, considering the sampled non-deteriorating moves, more than 99% of the times a transition out of a local minimum leads to another local minimum with the same fitness value. That applies to both problems and both mutation operators subsets.

In terms of global connectivity, we can observe that a path between any pair of nodes is not always present, even if we disregard the direction of the edges. That is, the networks break down into a number of weakly-disconnected components, especially when the larger search space of comparison and Boolean operators is considered. Nonetheless, more than 92% of the local minima we observed belong to a single, largest connected component. Moreover, a similar high fraction of all local minima lie on paths that could eventually descend to a global optimum.

By following the steepest descent directions on the LON, we can also detected the presence of multiple attractors with no non-deteriorating transitions around them, which we term *sinks*. Their number is indicative of the *multi-funnel* global structure of the landscapes [11], and may directly relate to the empirical problem hardness from the point of an iterated local search [10]. Among the four benchmark instances, the one with the lowest success rate also has multiple sub-optimal sinks. Indeed, almost all its local minima have access or belong to the funnel containing the global optima, but we hypothesise that, given the ILS stopping criterion, the actual success rate might depend on how easy it is for the search to find exits across plateaus and gain access to better (lower) fitness levels within the budget of function evaluations. As it can be visually appreciated on Figure 2, the "hardest" instance is also, notably, the one with fewer such connections across the lowest fitness levels.

5 THREATS TO VALIDITY

This paper provides some interesting insight into the global structure of program search spaces. It is, however, important to note that, while there may be grounds to believe that other programs may exhibit similar global structure, our analysis does not permit us to draw any broader conclusions. Furthermore, the analysis relies on a sample of the search space which only gives a partial picture of the actual space. This is somewhat mitigated by the fact that the sampling method is an actual solving method. Thus, the subset of the space that is sampled provides the coarse structure of the reachable parts of the space. This leads us to another cautionary point. A fitness landscape, or a local optima network, is not only dependent on the instance, or program, examined but also on the search algorithm used and the operators within. Should a different neighbourhood be used, a different landscape would probably be produced.

6 CONCLUSION

We used Local Optima Networks to investigate the landscapes produced when applying Iterated Local Search to two small programs. Mutations in comparison and Boolean operators were considered, on the assumption that programmers make relatively minor mistakes. The visualisations and analysis of multiple network metrics highlighted the high level of neutrality in the networks but also showed that paths exists from most sampled local optima to some global optimum. Thus neutrality should not necessarily be seen as a negative.

Naturally, it is difficult to draw broad conclusions from a very limited number of exemplars. Future work will remediate this by looking at a wider range of programs and mutation points. Further insight into the neutrality issue is required, which will involve examining the actual modifications brought about by higher-order mutations. This kind of analysis is also potentially useful to provide clues on which algorithms and neighbourhoods would work best for landscapes and programs with specific features.

DATA ACCESS

All data generated during this research are openly available from the Stirling Online Repository for Research Data (http://hdl.handle.net/11667/89).

ACKNOWLEDGMENTS

This work is supported by the Leverhulme Trust (award number RPG-2015-395) and by the UK's Engineering and Physical Sciences Research Council (grant number EP/J017515/1).

REFERENCES

[1] Gabor Csardi and Tamas Nepusz. 2006. The igraph software package for complex network research. *InterJournal* Complex Systems (2006), 1695. http://igraph.org

[2] Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer* 11, 4 (April 1978), 34–41. DOI: http://dx.doi.org/10.1109/C-M.1978.218136

[3] Hyunsook Do, Sebastian Elbaum, and Gregg Rothermel. 2005. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Software Engineering* 10, 4 (Oct. 2005), 405–435. DOI: http://dx.doi.org/10.1007/s10664-005-3861-2

[4] William B. Langdon, Mark Harman, and Yue Jia. 2010. Efficient multi-objective higher order mutation testing with genetic programming. *Journal of Systems and Software* 83, 12 (Dec. 2010), 2416–2430. DOI: http://dx.doi.org/10.1016/j.jss.2010.07.027

[5] William B. Langdon and Gabriela Ochoa. 2016. Genetic improvement: A key challenge for evolutionary computation. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. 3068–3075. DOI: http://dx.doi.org/10.1109/CEC.2016.7744177

[6] William B. Langdon and Justyna Petke. 2017. Software is Not Fragile. In *First Complex Systems Digital Campus World E-Conference 2015*. Springer, Cham, 203–211. DOI: http://dx.doi.org/10.1007/978-3-319-45901-1_24

[7] William B. Langdon, Nadarajen Veerapen, and Gabriela Ochoa. 2017. Visualising the Search Landscape of the Triangle Program. In *Genetic Programming*, James McDermott, Mauro Castelli, Lukas Sekanina, Evert Haasdijk, and Pablo Garcia-Sánchez (Eds.). Lecture Notes in Computer Science, Vol. 10196. Springer, Cham, 96–113. DOI: http://dx.doi.org/10.1007/978-3-319-55696-3_7

[8] Shawn Martin, W. Michael Brown, and Brian N. Wylie. 2007. *Dr.l: Distributed Recursive (graph) Layout.* Technical Report dRl; 002182MLTPL00. Sandia National Laboratories.

[9] Gabriela Ochoa, Marco Tomassini, Sébastien Vérel, and Christian Darabos. 2008. A Study of NK Landscapes' Basins and Local Optima Networks. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*. ACM, New York, NY, USA, 555–562. DOI: http://dx.doi.org/10.1145/1389095.1389204

[10] Gabriela Ochoa, Nadarajen Veerapen, Fabio Daolio, and Marco Tomassini. 2017. Understanding Phase Transitions with Local Optima Networks: Number Partitioning as a Case Study. In *Evolutionary Computation in Combinatorial Optimization*, Bin Hu and Manuel López-Ibáñez (Eds.). Lecture Notes in Computer Science, Vol. 10197. Springer, Cham, 233–248. DOI: http://dx.doi.org/10.1007/978-3-319-55453-2_16

[11] Gabriela Ochoa, Nadarajen Veerapen, Darrell Whitley, and Edmund K. Burke. 2015. The Multi-Funnel Structure of TSP Fitness Landscapes: A Visual Exploration. In *Artificial Evolution*, Stéphane Bonnevay, Pierrick Legrand, Nicolas Monmarché, Evelyne Lutton, and Marc Schoenauer (Eds.). Lecture Notes in Computer Science, Vol. 9554. Springer International Publishing, 1–13. DOI: http://dx.doi.org/10.1007/978-3-319-31471-6_1

[12] Eric Schulte, Zachary P. Fry, Ethan Fast, Westley Weimer, and Stephanie Forrest. 2014. Software mutational robustness. *Genetic Programming and Evolvable Machines* 15, 3 (Sept. 2014), 281–312. DOI: http://dx.doi.org/10.1007/s10710-013-9195-8

[13] Peter F. Stadler. 2002. Fitness Landscapes. *Appl. Math. and Comput* 117 (2002), 187–207.

[14] Roland H. Untch, A. Jefferson Offutt, and Mary Jean Harrold. 1993. Mutation Analysis Using Mutant Schemata. In *Proceedings of the 1993 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '93)*. ACM, New York, NY, USA, 139–148. DOI: http://dx.doi.org/10.1145/154183.154265

[15] Nadarajen Veerapen, Gabriela Ochoa, Renato Tinós, and Darrell Whitley. 2016. Tunnelling Crossover Networks for the Asymmetric TSP. In *Parallel Problem Solving from Nature — PPSN XIV*, Julia Handl, Emma Hart, Peter R. Lewis, Manuel López-Ibáñez, Gabriela Ochoa, and Ben Paechter (Eds.). Lecture Notes in Computer Science, Vol. 9921. Springer International Publishing, 994–1003. DOI: 10.1007/978-3-319-45823-6_93.

[16] Sébastien Verel, Fabio Daolio, Gabriela Ochoa, and Marco Tomassini. 2012. Local Optima Networks with Escape Edges. In *Artificial Evolution, EA 2011*, Jin-Kao Hao, Pierrick Legrand, Pierre Collet, Nicolas Monmarché, Evelyne Lutton, and Marc Schoenauer (Eds.). Lecture Notes in Computer Science, Vol. 7401. Springer Berlin Heidelberg, 49–60. DOI: http://dx.doi.org/10.1007/978-3-642-35533-2_5

[17] Xiangjuan Yao, Mark Harman, and Yue Jia. 2014. A Study of Equivalent and Stubborn Mutation Operators Using Human Analysis of Equivalence. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 919–930. DOI: http://dx.doi.org/10.1145/2568225.2568265