

Metrics Are Fitness Functions Too

Mark Harman

Brunel University
Uxbridge, Middlesex
UB8 3PH, UK

Mark.Harman@brunel.ac.uk

John Clark

The University of York
Heslington, York
YO10 5DD, UK.

jac@cs.york.ac.uk

Keywords: Search Based Software Engineering, Fitness, Optimisation, Genetic Algorithms

Abstract

Metrics, whether collected statically or dynamically, and whether constructed from source code, systems or processes, are largely regarded as a means of evaluating some property of interest. This viewpoint has been very successful in developing a body of knowledge, theory and experience in the application of metrics to estimation, predication, assessment, diagnosis, analysis and improvement. This paper shows that there is an alternative, complementary, view of a metric: as a fitness function, used to guide a search for optimal or near optimal individuals in a search space of possible solutions.

This 'Metrics as Fitness Functions' (MAFF) approach offers a number of additional benefits to metrics research and practice because it allows metrics to be used to improve software as well as to assess it and because it provides an additional mechanism of metric analysis and validation.

This paper presents a brief survey of search-based approaches and shows how metrics have been combined with the search based techniques to improve software systems. It describes the properties of a metric which make it a good fitness function and explains the benefits for metric analysis and validation which accrue from the MAFF approach.

1. Introduction

The title of this paper could equally well have been 'Fitness functions are metrics too' since the paper sets out to establish a one-to-one correspondence between fitness functions (used in optimisation and, in particular, in search-based software engineering) and software metrics.

The advantage of establishing this correspondence is that techniques associated with search-based optimisation can be deployed in the analysis and validation of software metrics, while the wealth of research and development of software metrics can be used at the heart of search-based ap-

proaches to software engineering — as the fitness functions which guide the search.

The paper presents a brief overview of search-based approaches to software engineering (SBSE) and the metrics that have already been used in SBSE, before moving on to define the characteristics of a metric that would make it a good candidate for a search-based approach to optimizing a software engineering problem. The paper then gives several examples of the way in which techniques associated with search based optimisation can be used to assist in the analysis and validation of software metrics.

The principal contributions of this paper are as follows:

- The paper shows how concepts from the search based software engineering community can be useful in metrics research. Specifically, it is shown:
 - How the combination of (possibly very different) metrics can be explored using multi objective fitness functions;
 - How fitness landscape visualisation techniques can be used to analyse metrics;
 - How a sequence of optimisation steps can be used as a metric validation technique;
 - How optimisation can yield insight into anomalies, hidden assumptions and similar shortcomings in a putative metric definition.
- The paper also shows that metrics can be very useful in search based software engineering. The one-to-one correspondence between metrics and fitness functions means that many metrics can be used as the guiding force behind the search for optimal or near optimal solutions to many software engineering problems. The paper sets out the problem characteristics which are likely to lead to success in the application of a search-based approach, with the metric as a fitness function.

The rest of the paper is organized as follows: Sections 2 and 3 summarize three popular search techniques and how they have already been combined with familiar software metrics in the goal of optimizing software engineering problems using search-based software engineering. Section 4 describes the properties of a problem and associated metric which make the metric suitable as a fitness function for search based software engineering. The Metrics As Fitness Functions approach is a two-way street. Metrics are invaluable in search based approaches to software engineering. This paper also shows how search-based software engineering can be useful for software metrics research and practice. Specifically, Sections 5 and 6 describe ways in which search based techniques can be used to analyse and validate software metrics. Section 7 concludes.

2. Search Techniques

Search techniques [16, 24, 25, 29, 38, 54, 62] are a set of generic algorithms concerned with searching for optimal or near optimal solutions to a problem within a large multimodal search space.

To make the paper self-contained, this section (briefly) reviews the three primary search techniques which have recently been used to optimise problems in search-based software engineering.

Search techniques have been applied to many different areas of engineering, including mechanical engineering [39, 56], chemical engineering [7, 34], medical and biomedical engineering [53, 55, 65], civil engineering [3, 6, 21, 30] and electronic engineering [5, 14, 46].

All of these techniques define a constrained, guided search, based upon some fitness function and, as such, metrics play a crucial role in all of these techniques. This section focuses on Hill Climbing, Simulated Annealing and Genetic Algorithms simply because these have been most closely associated with software engineering.

2.1. Hill Climbing

In hill climbing, the search proceeds from a randomly chosen point by considering the neighbours of the point. Once a fitter neighbour is found this becomes the 'current point' in the search space and the process is repeated. If there is no fitter neighbour, then the search terminates and a maximum has been found (by definition). The approach is called hill climbing, because when the fitness function is thought of as a landscape, with peaks representing points of higher fitness, the hill climbing algorithm selects a hill near to the randomly chosen start point and simply moves the current point to the top of this hill (climbing the hill).

Clearly, the problem with the hill climbing approach is that the hill located by the algorithm may be a local maximum,

and may be far poorer, in terms of fitness, than the global maximum in the search space.

There are many variations of the hill climbing theme. For example, should the algorithm select the first neighbour it finds with better fitness than the current individual (first ascent hill climbing) or should it consider all of the neighbours and select that with the best fitness improvement (steepest ascent hill climbing)? Whichever variation is chosen, the important characteristic of hill climbing is the property that it will select a local peak, which may be a local optimum and once this local peak is found the algorithm terminates; fast, simple, but prone to sub-optimality. Simulated annealing and tabu search are local search techniques which also search a local neighbourhood within the search space, but each has a different approach to overcoming this tendency to become trapped by a local maximum.

2.2. Simulated Annealing

In simulated annealing, a value, x_1 , is chosen for the solution, x , and the fitness function function, E , is typically minimised. As such, the fitness function represents the 'unfitness' or cost of a solution, with a minimal value denoting the best individual. Clearly, this is just another kind of metric; one which measures undesirability rather than desirability of a candidate solution.

The simulated annealing algorithm then considers a neighboring value of x_1 and evaluates its cost function; what constitutes a neighboring value of x_1 may not be immediately obvious and a neighbour, x_1' , must be defined in an appropriate way. If the cost function for x_1' is reduced, the search moves to x_1' and the process is repeated. However, if the cost function increases, the move to x_1' is not necessarily rejected; there is a small probability, p , that the search will move to x_1' and continue. The probability, p , is a function of the change in cost function, ΔE , and a parameter, T :

$$p = e^{-\frac{\Delta E}{T}}.$$

This probability is of a similar form to the Maxwell-Boltzmann distribution law for the statistical distribution of molecular energies in a classical gas, where ΔE and T are related to energy and temperature respectively.

When the change in cost function is negative (i.e. an improvement), the probability is set to one and the move is always accepted. When ΔE is positive (i.e. unfavourable), the move is accepted with the probability given in the equation above; the probability depends strongly on the values of the ΔE and T . At the start of the simulated annealing, T is high and the probability of accepting a very unfavourable move is correspondingly high. During the search, T is slowly reduced by some function, C , called the 'cooling' function because of its counterpart in the physical world.

```

Initialise  $x$  to  $x_0$  and  $T$  to  $T_0$ 

loop — Cooling
  loop — Local search
    Derive a neighbour,  $x'$ , of  $x$ 
     $\Delta E := E(x') - E(x)$ 
    if  $\Delta E < 0$ 
      then  $x := x'$ 
    else derive random number  $r \in [0, 1]$ 
      if  $r < e^{-\frac{\Delta E}{T}}$ 
        then  $x := x'$ 
      end if
    end loop — Local search
  exit when the goal is reached or a pre-defined
  stopping condition is satisfied
   $T := C(T)$ 
end loop — Cooling

```

Figure 1. Simulated Annealing Algorithm

The effect of ‘cooling’ on the simulation of annealing is that the probability of following an unfavourable move is reduced. In practice, the temperature is decreased in stages, and at each stage the temperature is kept constant until thermal quasi-equilibrium is reached. The set of parameters that determine the temperature decrement (i.e. initial temperature, stop criterion, temperature decrement between successive stages, number of transitions for each temperature value) is called the cooling schedule. The cooling schedule is critical to the success of the optimisation.

The algorithm is described in Figure 1.

2.3 Genetic Algorithms

Genetic algorithms (GAs) [4, 29] operate upon a ‘population’ of individuals. Each individual represents a possible solution to the problem in hand. These ‘candidate solutions’ are combined and mutated to evolve into a new generation of solutions which, because of fitness-based selection, tends to be a ‘fitter generation’.

Crossover (an analog of the mating process in natural genetics) provides a mechanism for mixing genetic material within the population. Mutation introduces new genetic material thereby preventing the search from stagnating. The next population of solutions is chosen from the parent and offspring generations in accordance with a survival strategy that favours fit individuals.

In natural evolution the fitness function is determined by a complex set of properties of the individual, the real world in which it operates and possibly other members of its and other species and their behaviour. In artificial genetic al-

```

Set generation number,  $m := 0$ 
Choose the initial population of candidate solutions,  $P(0)$ 
Evaluate the fitness for each individual of  $P(0)$ ,  $F(P_i(0))$ 
loop
  Recombine:  $P(m) := R(P(m))$ 
  Mutate:  $P(m) := M(P(m))$ 
  Evaluate:  $F(P(m))$ 
  Select:  $P(m+1) := S(P(m))$ 
   $m := m + 1$ 
exit when goal or stopping condition is satisfied
end loop;

```

Figure 2. Genetic Algorithm

gorithms the fitness function is nothing more than a metric which determines the ‘goodness of a candidate solution’.

A typical selection technique is tournament selection, in which two individuals, chosen at random from the population, fight a virtual tournament in order to determine which will go on to reproduce. The tournament is won, simply, by the individual with the best fitness. Therefore, the metric which determines fitness need only be an ordinal scale metric [58], since all that is required is that the metric determines whether one individual is better than another. there is no requirement (for the application of a genetic algorithm) for a metric to be interval, ratio or absolute, since there is no need to know how much better one individual is compared to another.

GAs operate on a population of individuals (often called chromosomes) each of which has an assigned fitness. Those individuals that either undergo recombination or survive are chosen with a probability which depends on fitness in some way.

A generic evolutionary algorithm is presented in Figure 2.

3. Existing MAFF Approaches

This section describes how existing work on search-based software engineering has used fitness functions to guide the search. Some of these fitness functions, such as structural coverage in testing and cohesion and coupling in modularisation, are familiar from the study of metrics, while others, such as test input distance have been invented specifically for the software engineering problem concerned. This suggests that the MAFF association can be beneficial both as a way of providing new metrics to play the role of fitness functions and also as a way of formulating and validating new (and existing) metrics. As can be seen, the work covers a wide range of applications. We shall mix formal and informal descriptions of metrics.

3.1. Metrics in the Design of Process

The need to phase or schedule requirements typically occurs in projects which have a relatively short development time, a high level of user involvement, and often operate in a dynamic environment. This was recognised by Boehm and his co-workers some time ago and is the motivation behind the development of the spiral model into the WinWin spiral model [8].

Clark et al. [11] have shown how this problem can be recast as one of search-based software engineering. Suppose that a customer's view is amalgamated into a single opinion. Assume also that all requirements are independent. At each iteration of the system build there is:

- A set of requirements, R , which have yet to be implemented:

$$R = \{r_1, \dots, r_n\}$$

- A development cost associated with each requirement.
- A resource (a limit on how much the customers wish to spend).
- A set of customers who might have different opinions of what is required in the next phase of the system.

A choice has to be made of which set of requirements is going to be incorporated into the next phase of development. The problem is how to select the set of requirements that is going to be acceptable to as many customers as possible.

A customer's opinion is expressed as a priority or weighting, p_x , associated with each requirement. Suppose there are k customers and n potential requirements. A customer-priority matrix, CP can be defined, such that $p_i^j \in CP$ represents the priority associated with customer i for requirement j ($1 \leq i \leq k, 1 \leq j \leq n$). The values used for each p_i^j will depend upon the nature of the application. If all users are equal, then each could simply rank the requirements, with the most desirable being ranked n , through to the least desirable ranked 1. The value of the j^{th} requirement is defined as

$$tp_j = \sum_{i=1}^k p_i^j$$

Having scored the requirements in some way, the problem is now to find the optimal set of requirements to build. However, this problem is an instance of the 0-1 knapsack problem [33], which is known to be NP-hard [23], which is known to be well-suited to search based techniques such as genetic algorithms.

3.2. Metrics in the Design of Architecture and Infrastructure

It is widely accepted that 'getting the architecture wrong' is very expensive. The architectural design space may be huge and ways need to be found to explore it efficiently. Below we indicate the use of simple metrics as fitness functions to guide heuristic searches. We have taken a flexible view architecture, viewing it loosely as aspects of the design of interacting components.

Distributed System Static Configuration. Many modern software systems are distributed. For critical software applications tasks may be replicated (to provide an element of redundancy) and run concurrently (to achieve real-time performance requirements). Important issues arise in the configuration of such systems. Are response times guaranteed to be within required bounds? The real-time systems community have developed analysis models that allow the worst case response times to be calculated. (These are typically called 'schedulability models'.) The response times depend on which processors particular tasks are hosted, what communications overheads are incurred as a consequence etc. It can be regarded as a form of architectural design.

The determination of particular configurations to achieve the required response time is known to be a computationally hard problem. The worst case response times provided by the schedulability analysis models allow a simple metric to be adopted as part of a heuristic search — the sum of the times by which the tasks in the system overrun their required bounds. A configuration with a value of 0 satisfies all response time requirements. Refinements of this idea, and the use of similar metrics for non-functional properties of critical software systems can be found in [48, 49].

Module Clustering. Hill climbing, simulated annealing and genetic algorithms have also been applied to the problem of restructuring at the module level of abstraction, attempting to find good module decompositions and module hierarchies [26, 40, 41, 42, 45].

As an example, consider the fitness function implemented in the Bunch [42] module clustering tool. The authors refer to their fitness function as *Modularization Quality* (MQ). MQ for a software structure graph G partitioned into k clusters is calculated by summing the *Cluster Factor* (CF) for each cluster of the partitioned graph. The Cluster Factor, CF_i , for cluster i ($1 \leq i \leq k$) is defined as a normalized ratio between the total weight of the internal edges (edges within the cluster) and half of the total weight of external edges (edges that exit or enter the cluster). The weight of the external edges is split in half in order to apply an equal penalty to both clusters that are connected by an external edge. Internal edges of a cluster are referred to as intra-edges. The number of intra-edges for module i is de-

noted by μ_i . The edges between two distinct clusters i and j are the inter-edges.

The number of inter-edges between a module i and a module j is denoted as $\varepsilon_{i,j}$. If edge weights are not provided by G , it is assumed that each edge has a weight of 1. Also, note that $\varepsilon_{i,j} = 0$ and $\varepsilon_{j,i} = 0$ when $i = j$. The formal definition of the MQ calculation is defined as follows:

$$MQ = \sum_{i=1}^k CF_i \quad CF_i = \begin{cases} 0 & \mu_i = 0 \\ \frac{\mu_i}{\mu_i + \frac{1}{2} \sum_{\substack{j=1 \\ j \neq i}}^k (\varepsilon_{i,j} + \varepsilon_{j,i})} & \text{otherwise.} \end{cases}$$

Notice how MQ increases as the cohesiveness of the individual clusters increases and the coupling between all of the clusters decreases. Thus the goal of the search algorithms implemented in Bunch is to maximize MQ . In this way MQ captures the familiar software engineering design principle that we seek high cohesion and low coupling [13]. However, by using search-based software engineering the measurement of cohesion and coupling moves from a passive act of assessment of quality to an active means of automating the search for high quality.

Protocols. Many of the systems we currently rely upon are distributed and the development of reliable and efficient infra-structural frameworks and distributed OS APIs is of great importance. Search can be of benefit here too. Consider, for example, the development of communications protocols. Protocols define rigorously a series of message exchanges between agents. At the end of a protocol run we may require certain goals to have been achieved, e.g. that agents should possess certain data, or else hold certain ‘beliefs’ about other agents’ states. If there are several such goals to be achieved then a simple metric can be defined over the space of feasible protocols to guide the search — simply count the number of required goals achieved by the protocol. Thus, we use fitness functions for a protocol of the form:

$$\sum_{i=1}^M w_i * g_i$$

where M is the number of messages, w_i are weights and g_i is the number of *required goals* achieved after message i . Refinements of this basic idea (in particular taking weighted sums of the numbers of goals achieved after each message) have been adopted to evolve provably secure communications protocols using both genetic algorithms and simulated annealing [12].

3.3. Metrics in the Design of Test Data

Test data generation is basically a constraint problem: find an input vector v such that some property p is achieved.

The basic notion underpinning search based approaches to test data generation is that some test data are clearly better than others; good test data either achieve the stated aim, or else come ‘close to’ achieving the aim. The notion of ‘close to’ can generally be defined in a problem specific way — we can measure how ‘far away’ the current test input vector is from satisfying the aim, and seek to reduce this distance via the search algorithms indicated earlier. The ‘distance’ is of course a metric.

At heart the dynamic test data generation problem can be regarded as ‘find an input vector that causes predicates p_1, \dots, p_n to be true at some identified points in program execution. Thus, if we want a particular path to be followed, the appropriate branch predicates must be satisfied. (Various researchers have addressed such structural coverage issues [31, 32, 44, 47, 50, 61, 64]). If we want to find data that breaks a specification then we would want the pre-condition and the negation of the post-condition to be satisfied. Statements have associated with them ‘healthiness pre-conditions’ which, if not met, might cause abnormal execution (such as underflow or overflow). But this condition is just another predicate to be broken.

We need some metric (fitness function) that can indicate how close we are to satisfying a predicate. The metric used varies between researchers, but the following example gives the common flavour. Suppose you want to cause the assertion $X \leq 20$ to be true at some identified point in the program. Suppose you have three test data inputs v_1, v_2 and v_3 . Execution with v_1 causes X to take the value 40, execution with v_2 causes X to take the value 22, and execution with v_3 causes X to take the value 15. v_3 satisfies the aim and so we would assign it zero cost. v_1 and v_2 do not satisfy the aim, but v_2 comes closer than does v_1 , and this should be reflected in the cost function metric. This can be captured for this predicate by a metric of the form:

$$\text{cost}(v_k) = \max(X_{v_k} - 20, 0)$$

where X_{v_k} is the value of X when v_k is used as input. Where many predicates need to be satisfied, weighted sums of cost functions for each predicate can be used to guide the search.

The above metric is a simplification of what researchers have actually used, but it does capture the essence of what they do.

3.4. Evolution of Metrics Using Genetic Programming

Genetic Programming (GP) [38] is a form of evolutionary algorithm in which the individuals evolved are programs, rather than more simple data types. The programs are represented as abstract syntax trees [2], requiring differ-

ent forms of mutation and crossover operation, compared to the standard genetic algorithm model (see Section 2.3).

Typically, for a genetic program, this fitness function is a set of training data. That is, a set of input output pairs which the evolved program must compute correctly. The evolution effectively searches for a program which satisfies all (or almost all) input-output pairs.

The GP technique works very well for simple programs, such as simple purely functional characterisations of an input-output relationship between a set of scalar values [38]. Many metrics are precisely this: a simple functional relationship. Therefore it makes sense to consider ways in which the use of search—based software engineering could be used to innovate software metrics. This approach was pioneered by Jose Javier Dolado [17, 18], but has also been developed by my Lefley and Burgess [10] and Aguilar et al. [1].

Dolado [17, 18] used GP to discover linear and nonlinear functions which capturing the software project cost in terms of dependent variables, such as size in function points. The fitness function was the mean squared error,

$$mse = \frac{1}{n-2} \sum_{i=1}^n (y_i - \hat{y}_i)^2.$$

There may be many other situations in which metrics can be evolved to explain data in a similar way, such as

1. Fault density (in terms of module size);
2. The Brooks factor [9] (the amount of additional slippage created by adding new staff to a late project;
3. Lines of code (per function point) for various programming languages;
4. Program size (in terms of design diagram size) for diagrammatic designs, such as those found in UML, dataflow Diagramming and SSADM;

The advantage of these approaches is that, rather than adopting a ‘standard conversion factor’, the GP approach allows the organisation to tailor the conversion to their own particular data (which may, for instance, be atypical). Such approaches will be particularly effective (and certainly, by definition more effective than linear regression) where the relationship between the data is complex and/or non-linear.

4. Properties required for MAFF

For a metric to be useful as a fitness function it needs to have the following four properties:

1. Large search space

If the metric is only used to distinguish a *few* individuals from one another, then the value of the metric for each individual can be computed and the search space explored exhaustively. There would be no need to use a search-based technique to sample the search space. Of course, most search spaces are *very* large. That is, most metrics apply to large (conceptually infinite) search spaces, such as the space of all expressible programs in some language or the space of all expressible designs in some design notation.

2. Low computational complexity

Search Based algorithms sample a portion of a very large search space. The portion sampled is typically non-trivial, requiring many thousands (possibly hundreds of thousands) of fitness evaluations. Therefore the computational complexity of the fitness function (the metric) has a critical impact on the overall complexity of the search process. Fortunately, most metrics are relatively cheap to compute, since they are constructed in terms of the structural or syntactic properties of the programs, designs and systems which they assess.

3. Approximate continuity

It is not necessary for a function to be continuous to be useful as a fitness function, but too much discontinuity can mislead a search, because all search—based optimisation approaches rely upon the guidance given by the fitness function. Continuity ensures that this guidance is perfect; the less continuous is the fitness function, the less guidance it gives.

4. Absence of known optimal solutions

If there is a known optimal solution to a problem, then clearly there is no need to use a search-based approach to seek optimal (or near optimal) solutions.

These four properties can be seen to apply in many cases. Most source code level metrics are immediate candidates as they apply to a potentially infinite search space (of possible programs), typically have linear algorithmic complexity and usually measure intangible properties of programs which are highly conceptual and therefore have no known perfect solutions.

Design level metrics, also apply to large search spaces (of candidate designs) and are not typically expensive to compute for a particular design. There are also typically few known ‘best design solutions’ to a given problem.

Process metrics are less likely to fit this set of characteristics and therefore, it would appear that the MAFF approach is ideally targeted at product-oriented metrics rather than process oriented metrics. However, where such processes can be simulated, search techniques can be used to

explore the search space of possible process attributes, configurations and parameters.

One of the appealing features of the search based approach is that many of the search techniques tend to be fairly robust. For many problems, it is possible to make good progress without a great deal of sophistication. Genetic algorithms, for example, have a strong reputation for obtaining good results across a whole range of problems.

For the would-be search-based software engineer, keen to apply a favourite metric to optimize a potential solution, this offers the possibility of rapid proof of concept. There is an armoury of sophisticated search techniques that can subsequently be brought to bear and several excellent toolsets are available. It is perfectly feasible to combine evolutionary and local based searches; indeed, this is often done. Evolutionary approaches often get good results which can then be improved by the addition of local search.

5. Fitness Landscape Analysis

It is common in the search-based algorithm community to attempt to visualize the ‘fitness landscape’ [28, 35, 36, 52, 66]. That is, to use the fitness function values as a measure of height (or vertical co-ordinate), in a landscape where each individual in the search space potentially occupies some location within the horizontal co-ordinates.

Most search problems involve individuals made up of more than two components (or genes in the case of genetic algorithms). Mapping an individual from the search space into a two dimensional plane is therefore non-trivial. There are two possible approaches:

1. Pick two elements from the many which make up an individual. Keep all other elements constant at some arbitrary value, while considering a range of values for each of the two chosen elements. These two element ranges form the values for the *x* and *y* axis of the *projection* of the search landscape, while the fitness at each point produces the associated value for the *z* axis. This approach can only represent a slice through the overall landscape, but it serves to give a feeling for the kinds of properties of its shape, at least for two chosen components of the individuals being optimized.
2. Map all elements from the search space onto a flat plane in such a manner that near neighbours in the *N* dimensional search space (those which are ‘close to one another’) lie near to one another on the 2D plane. This approach involves some necessary distortions of the landscape to squash *N* dimensions onto only two. Therefore, it can only give a feeling for the proximity of peaks of fitness and for the global properties of the landscape, but does not produce a ‘landscape shape’ in any conventional sense.

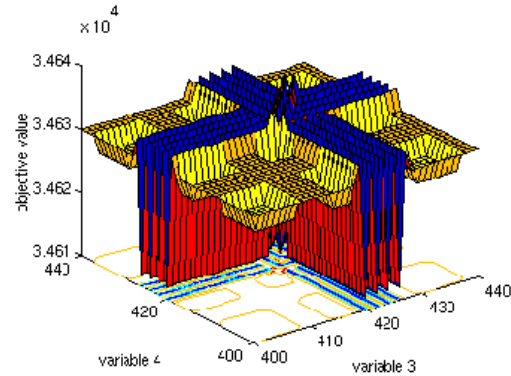


Figure 3. Search space fitness landscape projection for sort input variables 250 and 150

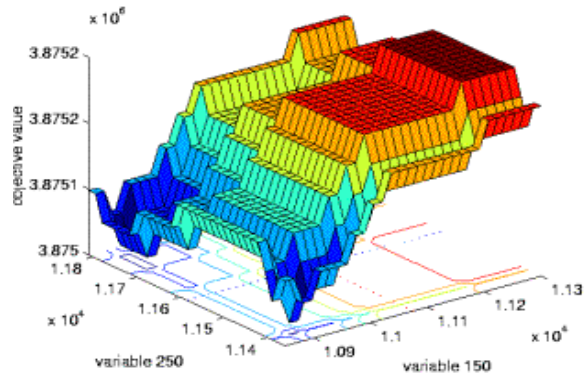


Figure 4. Search space fitness landscape projection for sort input variables 3 and 4

For example consider the two landscapes, in Figures 3 and 4 produced from the application of the DaimlerChrysler Evolutionary testing system [64] to the problem of finding worst case execution time. The two landscapes are for projections of the search space of an array of 250 variables. The program under test is a simple sorting program which uses the familiar bubble sort algorithm to order the elements of the input array in ascending order.

In each case the *x* and *y* axis show the variation in two of the input array’s elements (while the other elements are held constant); the *z* axis shows the fitness value obtained. In this way the graphs produce a projection (onto two of the 250) variables of the search space of the problem.

In this case the landscape projections reveal that the problem is complex and multi model, with discontinuities

and plateaus (reflected by the jagged edges of Figure 3 and the large flat area of Figure 4). From a search technique perspective, this suggests that this problem is one more suited to a genetic algorithm than a local search, such as a hill climber (or a quasi local search like simulated annealing). From a software metric point of view, the landscapes reveal something of the nature of the metric; small changes in individual input variables can, in some input subranges, produce dramatic changes in metric value, while in others, comparatively larger changes produce no change in metric value.

The examples in Figures 3 and 4, were those of a projected landscape. The alternative approach, in which all N dimensions of the search space are reduced to two can also be used. For example consider the plot of peaks in a landscape depicted in Figure 5. This is a visualisation of a landscape from a hill climbing approach to feature subset selection. The features in this instance are software project attributes used in a case-based software project cost estimation system [37, 60]. In this figure, peaks which occur in the best quartile of the hills are denoted by 'x' symbols, those which occur in the next best quartile are denoted by 'o' symbols, those in the third quartile by '+' symbols and those in the worst quartile by '.' symbols. The division of peaks in this way is suitably coarse-grained and approximate in nature, that the approximate nature forced by the requirement to squash the N dimensions (in this case $N = 43$) onto 2 is not misleading. The figure clearly shows that high peaks tend to cluster together (although the dimension squashing makes it impossible to say *where* this occurs on the original landscape).

6. MAFF Metric Validation

Some metrics are absolute representations of a precise measurable aspect of a software product or process. However, metrics are human artifacts, designed to give a concrete and measurable value to some more intuitive underlying notion. As such, it is important to validate the extent to which a proposed metric adequately captures the underlying concept it seeks to measure.

This section shows how approaches to search based software engineering can be used to validate metrics by optimisation (using the metric as a fitness function). This addresses the issue of selection bias when choosing individuals to rank as part of the representation condition and also provides additional insight by highlighting anomalies and implicit assumptions in the formulation of a metric.

6.1. The Representation Condition

The representation condition [58] states that a good metric is one which is truly representative of the property it

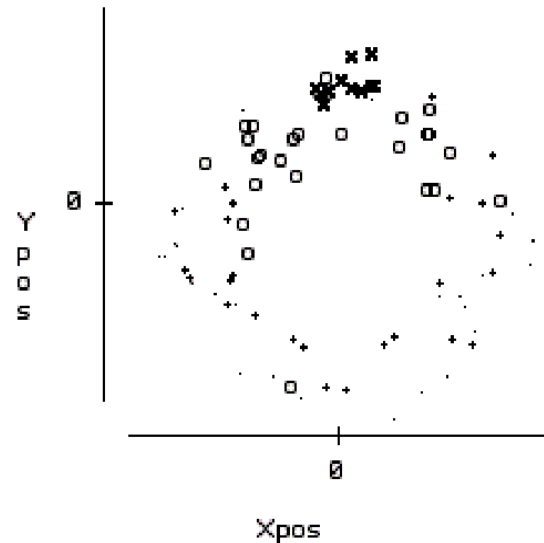


Figure 5. Search Space Fitness Landscape Peak Density

seeks to denote. This is often characterised by supposing that there is an empirical relation R which is to be preserved by some numeric representation (a metric) M . More formally, suppose R is an ordering relation on a set of elements S . The representation condition requires that $\forall x, y. (x, y) \in R \Leftrightarrow M(x) \leq M(y)$ That is, elements lower down the ordering according to R are numerically lower according to their image under M .

Despite its simplicity, this condition is rather hard to test in practice. There is the obvious problem of agreeing what the order imposed by R is, when, typically R is a ill-defined conceptual level ordering capture in some intangible property of the software process or its products. A common approach is to suppose that there is a group of human experts who can provide the ordering [58]. However, for validation purposes this raises the thorny issue of selecting a group of experts. Furthermore, there is the additional problem of identifying a suitable set of objects to measure and order according to the metric. While the MAFF approach cannot help with the selection of experts, it can be a great assistance with the determination of a bias-free set of objects to use in validation.

Consider the situation where a program is to be measured for complexity. The measurement of software complexity is a well-known and controversial area of software metrics research and practice [22, 57, 59]. The controversy arises partly because of the difficulty of agreeing upon whether a proposed complexity metric satisfies the repre-

sensation condition. A difficulty here is in selecting a set of objects (in this case the set of programs to order according to a proposed metric). Different software tasks *require* different levels of complexity and so there is a problem of comparing ‘apples and oranges’; we need to be sure that the inherent complexity of the problem is roughly equivalent, in order to avoid any bias which would force some of the programs to have some innately larger complexity.

To address this problem, search-based program transformation [11, 20, 27] can be used. Search-based transformation seeks to optimise a program by a sequence of transformations according to some fitness function which determines whether one program is better than another. The important point about the transformation process is that it is *meaning preserving*. As the program is optimised, according to the fitness function, each new version of the program produced in the sequence of optimisation steps, performs the *same* task as the original. However, each is expressed in a style that is rewarded by a higher score for the fitness function. The fitness function can be substituted for by the complexity metric under scrutiny.

The sequence of optimisation steps produced by search-base transformation, thereby produces an ever increasing sequence of programs, each of which performs an identical task, but with increasingly attractive structure according to the metric. Because each member of the sequence of programs performs an identical task, there is no problem of comparing ‘apples and oranges’ when coming to assess whether the metric meets the representation condition.

6.2. Identifying False Assumptions

The application of the MAFF approach can provide similar insights into software metrics research. For example, consider again the application of search-based transformation¹ to the problem of validating a (source level) software metric. If the fitness function is the McCabe [43] complexity metric, then the optimizing sequence of transformations in Figure 6 might be produced.

It is not the goal of this paper to stray into the debate concerning the validity of certain complexity metrics, as the issues have been well-documented by previous authors [22, 57]. Rather we wish to show how the search-based transformation approach provides an *objective* way to validate such metrics. By definition, all traditional approaches to program transformation alter the syntax of a program without altering its semantics [15, 51, 63]. Suppose a program p' can be obtained from a program p by a sequence of transformation steps. The meaning preserving nature of transformation means that p' is nothing more than alterna-

¹All transformations in this section will concern side-effect free programs as side effects significantly complicate matters [19] without adding any new insights.

IF A THEN IF B THEN S	IF B THEN IF A THEN S
--------------------------	--------------------------

Figure 7. McCabe Invariant Transformation

tive syntax for p . Surely, p and p' , *by definition*, should be accorded the same measure of complexity by a valid complexity measure?

This observation suggests a property, transformation invariance, for a metric. This property, defined more formally, in Definition 3 below, provides a way of measuring the robustness, or ‘resistance to noise’ for a code-level metric. The search-based transformation technique provides an automated way of assessing compliance.

Definition 1 (Transformation) A transformation rule, t , is a total function from programs to programs.

Definition 2 (Metric) A metric, M , is a total function from programs to \mathbb{R} .

Definition 3 (Transformation Invariant Metric (TIM))

Let T be a set of transformation rules. Let PROG be the set of all programs. A metric M , is a Transformation Invariant Metric with respect to T iff $\forall t \in T, p \in \text{PROG}. M(t(p)) = M(p)$.

We shall write T -Transformation Invariant Metric or simply T -TIM for a metric which is a Transformation Invariant Metric with respect to T according to Definition 3. For example, the four programs in Figure 6 obtain different values for McCabe complexity and so, McCabe is not a T -TIM for the set of transformations which includes the rule which absorbs two nested IF statements to produce a single IF with a conjunction of the two IF predicates. However, consider the two programs in Figure 7. These two programs are produced by the ‘associativity of nested IF transformation’. Suppose this transformation is denoted `ass_iff`, then we can say, formally, that McCabe is a `{ass_iff}`-TIM.

How does all this transformation theory help? Well, we are now able to define a metric together with a set of transformation rules under which the metric is invariant. The transformation rules capture the aspects of the program’s structure, its syntax, which the metric does not *care about*. The metric thereby captures an equivalence class defined by a set of transformation rules.

7. Conclusion

This paper has as its title the bold thesis ‘Metrics are Fitness Functions Too’. We believe this to be a radical statement, with far reaching implications for the development of

<pre> IF A THEN IF B THEN IF C THEN IF D THEN S ENDIF ENDIF ENDIF ENDIF ENDIF </pre>	<pre> IF A THEN IF B THEN IF C AND D THEN S ENDIF ENDIF ENDIF </pre>	<pre> IF A THEN IF B AND C AND D THEN S ENDIF ENDIF </pre>	<pre> IF A AND B AND C AND D THEN S ENDIF </pre>
--	--	--	--

Figure 6. Sequence of optimizing transformations according to McCabe complexity

modern day software systems and have sought to indicate how metrics (both familiar and new) have been and can be used to improve the effectiveness of the software engineering process.

Yet, what we have said should have a direct intuitive appeal to the metrics community. After all, metrics act as projections of qualities of interest of software systems. If an engineer is not happy with what the metric is reporting, then corrective action will be taken; the system will be (manually) altered to improve it (as judged by the metric). But isn't this exactly what is proposed in this paper? Yes! But with the crucial difference that the metric evaluation and system redesign are *automated* happen in **milli-seconds**. Furthermore, through iteration the current 'system' (which may be any software development artifact, e.g. specification, design, code, estimate, test data) evolves towards an excellent solution.

In a sense, we are combining the blindingly obvious (software engineering is a search for high quality solutions), with a radical view: our claim that, for a great many software engineering problems, the search can be automated. All that is needed is a suitable fitness function and sufficient motivation.

The Search Based Software Engineering (SBSE) community has emerged rapidly over the past few years and does not lack motivation! Our observation that metrics are fitness functions, opens up a wealth of potential applications. For many years the metrics community, has effectively been working on the central component for each application of SBSE; the fitness functions which guide the search. Whether they be begged, stolen or borrowed, there is clearly a wealth of metrics research that can be mined to support automated, search-based software engineering. We believe also that the fusing of metrics research and heuristic search may spawn whole new areas of metric research, for example, concerning granularity of information, speed of calculation, how to achieve best tradeoffs between metrics and so on:



Search is what you seek

8. Acknowledgements

This work has been supported, in part, by EPSRC Grants GR/R98938, GR/M58719, GR/M78083 and GR/R43150. the authors would also like to thanks members of the EPSRC SEMINAL Network and the wider Search-Based Software Engineering (SBSE) community for many profitable discussions which undoubtedly influenced the approach to metrics advocated here. Figures 3 and 4 are included here by kind permission of Joachim Wegener of DaimlerChrysler AG, Berlin. Figure 5 is included here by kind permission of Martin Shepperd and Colin Kirsopp of Bournemouth University, UK.

References

- [1] J. Aguilar-Ruiz, I. Ramos, J. C. Riquelme, and M. Toro. An evolutionary approach to estimating software development projects. *Information and Software Technology*, 43(14):875–882, Dec. 2001.
- [2] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, techniques and tools*. Addison Wesley, 1986.
- [3] V. Babovic. Mining sediment transport data with genetic programming. In *Proceedings of the First International*

Conference on New Information Technologies for Decision Making in Civil Engineering, pages 875–886, Montreal, Canada, 11-13 Oct. 1998.

- [4] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.
- [5] F. H. Bennett III, M. A. Keane, D. Andre, and J. R. Koza. Automatic synthesis of the topology and sizing for analog electrical circuits using genetic programming. In K. Miittinen, M. M. Mäkelä, P. Neittaanmäki, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 199–229, Jyväskylä, Finland, 30 May - 3 June 1999. John Wiley & Sons.
- [6] P. J. Bentley and J. P. Wakefield. Generic representation of solid geometry for genetic search. *Microcomputers in Civil Engineering*, 11(3):153–161, 1996.
- [7] R. R. Birge. Protein-based optical computing and memories. *Computer*, 25(11):56–67, Nov. 1992.
- [8] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy. Using the winwin spiral model: A case study. *IEEE Computer*, 31(7):33–44, "July" 1998.
- [9] F. P. Brooks, Jr. *The Mythical Man Month: Essays on Software Engineering*. Addison-Wesley Publishing Company, Reading, MA, USA, 1975.
- [10] C. J. Burgess and M. Lefley. Can genetic programming improve software effort estimation? A comparative evaluation. *Information and Software Technology*, 43(14):863–873, Dec. 2001.
- [11] J. Clark, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd. Reformulating software engineering as a search problem. *IEE Proceedings — Software*, 150(3):161–175, 2003.
- [12] J. A. Clark and J. L. Jacob. Protocols are Programs Too: the Metaheuristic Search for Security Protocols. "Information and Software Technology", December 2001. Special issue on Metaheuristic Search for Software Engineering.
- [13] L. L. Constantine and E. Yourdon. *Structured Design*. Prentice Hall, 1979.
- [14] O. Cordon, F. Herrera, and L. Sánchez. Evolutionary learning processes for data analysis in electrical engineering applications. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 205–224. John Wiley and Sons, Chichester, 1998.
- [15] J. Darlington and R. M. Burstall. A transformation system for developing recursive programs. *J. ACM*, 24(1):44–67, 1977.
- [16] K. De Jong. On using genetic algorithms to search program spaces. In J. J. Grefenstette, editor, *Genetic Algorithms and their Applications: Proceedings of the second international conference on Genetic Algorithms*, pages 210–216, MIT, Cambridge, MA, USA, 28-31 July 1987. Lawrence Erlbaum Associates.
- [17] J. J. Dolado. A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, 26(10):1006–1021, 2000.
- [18] J. J. Dolado. On the problem of the software cost function. *Information and Software Technology*, 43:61–72, 2001.
- [19] J. J. Dolado, M. Harman, M. C. Otero, and L. Hu. An empirical investigation of the influence of a type of side effects on program comprehension. *IEEE Transactions on Software Engineering*, 29(7):665–670, 2003.
- [20] D. Fatiregun, M. Harman, and R. Hierons. Search based transformations. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 2511–2512, Chicago, 12-16 July 2003. Springer-Verlag.
- [21] C.-W. Feng, L. Liu, and S. A. Burns. Using Genetic Algorithms to Solve Construction Time-Cost Trade-Off Problems. *Journal of Computing in Civil Engineering*, 10(3):184–189, 1999.
- [22] N. E. Fenton. Software measurement: A necessary scientific basis. *IEEE Transactions on Software Engineering*, 20(3):199–206, 1994.
- [23] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [24] F. Glover. Tabu search: A tutorial. *Interfaces*, 20:74–94, 1990.
- [25] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [26] M. Harman, R. Hierons, and M. Proctor. A new representation and crossover operator for search-based optimization of software modularization. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1351–1358, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
- [27] M. Harman, L. Hu, R. M. Hierons, J. Wegener, H. Sthamer, A. Baresel, and M. Roper. Testability transformation. *IEEE Transactions on Software Engineering*. To appear.
- [28] E. Hart and P. Ross. GAVEL - a new tool for genetic algorithm visualization. *IEEE-EC*, 5:335–348, Aug. 2001.
- [29] J. H. Holland. *Adaption in Natural and Artificial Systems*. MIT Press, Ann Arbor, 1975.
- [30] W. M. Jenkins. The genetic algorithm-or can we improve design by breeding. In *IEE Colloquium on Artificial Intelligence in Civil Engineering*, pages 1/1–4, London, UK, Jan., 16 1992. IEE.
- [31] B. Jones, H.-H. Sthamer, and D. Eyres. Automatic structural testing using genetic algorithms. *The Software Engineering Journal*, 11:299–306, 1996.
- [32] B. F. Jones, D. E. Eyres, and H. H. Sthamer. A strategy for using genetic algorithms to automate branch and fault-based testing. *The Computer Journal*, 41(2):98–107, 1998.
- [33] H.-W. Jung. Optimizing value and cost in requirements analysis. *IEEE Software*, 15:74–78, 1998.
- [34] C. L. Karr, S. K. Sharma, W. J. Hatcher, and T. R. Harper. Fuzzy control of an exothermic chemical reaction using genetic algorithms. *Engineering Applications of Artificial Intelligence* 6, 6:575–582, 1993.
- [35] Y.-H. Kim and B.-R. Moon. Visualization of the fitness landscape, A steady-state genetic search, and schema traces. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page 686, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
- [36] Y.-H. Kim and B.-R. Moon. New usage of sammon's mapping for genetic visualization. In *Genetic and Evolutionary Computation – GECCO-2003*, volume 2723 of *LNCS*, pages 1136–1147, Chicago, 12-16 July 2003. Springer-Verlag.

- [37] C. Kirsopp, M. Shepperd, and J. Hart. Search heuristics, case-based reasoning and software project effort prediction. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1367–1374, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
- [38] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.
- [39] J. E. Labossiere and N. Turrkan. On the optimization of the tensor polynomial failure theory with a genetic algorithm. *Transactions of the Canadian Society for Mechanical Engineering*, 16(3-4):251–265, 1992.
- [40] R. Lutz. Evolving good hierarchical decompositions of complex systems. *Journal of Systems Architecture*, 47:613–634, 2001.
- [41] K. Mahdavi, M. Harman, and R. M. Hierons. A multiple hill climbing approach to software module clustering. In *IEEE International Conference on Software Maintenance (ICSM 2003)*, pages 315–324, Amsterdam, Netherlands, Sept. 2003. IEEE Computer Society Press, Los Alamitos, California, USA.
- [42] S. Mancoridis, B. S. Mitchell, Y.-F. Chen, and E. R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proceedings; IEEE International Conference on Software Maintenance*, pages 50–59. IEEE Computer Society Press, 1999.
- [43] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976.
- [44] C. Michael, G. McGraw, and M. Schatz. Generating software test data by evolution. *IEEE Transactions on Software Engineering*, (12):1085–1110, Dec. 2001.
- [45] B. S. Mitchell and S. Mancoridis. Using heuristic search techniques to extract design abstractions from source code. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1375–1382, New York, 9-13 July 2002. Morgan Kaufmann Publishers.
- [46] Y. Miyamoto, Y. Miyatake, S. Kurosaka, and Y. Mori. A parameter turning for dynamic simulation of power plants using genetic algorithms. *ELECTRICAL ENGINEERING IN JAPAN*, 115(1):104–113, 1995.
- [47] F. Mueller and J. Wegener. A comparison of static analysis and evolutionary testing for the verification of timing constraints. In *4th IEEE Real-Time Technology and Applications Symposium (RTAS '98)*, pages 144–154, Washington - Brussels - Tokyo, June 1998. IEEE.
- [48] M. Nicholson. Supporting design synthesis for safety-critical systems. *Proc. GECCO 2003, Chicago*, July 2003.
- [49] M. Nicholson and D. Prasad. Design synthesis using adaptive search techniques and multi-criteria decision analysis. *Proc. 2nd ICECCS, Hilton Hotel, Montreal, Canada*, 1996.
- [50] R. P. Pargas, M. J. Harrold, and R. R. Peck. Test-data generation using genetic algorithms. *The Journal of Software Testing, Verification and Reliability*, 9:263–282, 1999.
- [51] H. Partsch. *The CIP Transformation System*, pages 305–322. Springer, 1984. Peter Pepper (ed.).
- [52] H. Pohlheim. Visualization of evolutionary algorithms - set of standard techniques and multidimensional visualization. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 533–540, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.
- [53] R. Poli, S. Cagnoni, and G. Valli. Genetic design of optimum linear and nonlinear QRS detectors. *IEEE Transactions on Biomedical Engineering*, 42(11):1137–41, Nov. 1995.
- [54] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Press, Oxford, UK., 1992.
- [55] E. Sanchez, H. Miyano, and J. P. Brachet. Optimization of fuzzy queries with genetic algorithms. applications to a data base of patents in biomedical engineering. *Proc. Sixth International Fuzzy Systems Association World Congress (IFSA'95)*, 2:293–296, 1995. Sao Paulo.
- [56] M. Sebag, M. Schoenauer, and H. Maitournam. Parametric and non-parametric identification of macro-mechanical models. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, pages 327–340. John Wiley and Sons, Chichester, 1998.
- [57] M. J. Shepperd. A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*, 3(2):177–188, 1988.
- [58] M. J. Shepperd. *Foundations of software measurement*. Prentice Hall, 1995.
- [59] M. J. Shepperd and D. C. Ince. A critique of three metrics. *Journal of Systems and Software*, 26:197–210, 1994.
- [60] M. J. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(11):736–743, 1997.
- [61] N. Tracey, J. Clark, and K. Mander. The way forward for unifying dynamic test-case generation: The optimisation-based approach. In *International Workshop on Dependable Computing and Its Applications (DCIA)*, pages 169–180. IFIP, January 1998.
- [62] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Practice*. Kluwer Academic Publishers, Dordrecht, the Netherlands, 1987.
- [63] M. Ward. Reverse engineering through formal transformation. *The Computer Journal*, 37(5), 1994.
- [64] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology Special Issue on Software Engineering using Metaheuristic Innovative Algorithms*, 43(14):841–854, 2001.
- [65] R. Weiss and J. T. F. Knight. Engineered communications for microbial robotics. In *Proceedings 6th DIMACS Workshop on DNA Based Computers, held at the University of Leiden, Leiden, The Netherlands, 13 - 17 June 2000*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., pages 5–19. Leiden center for natural computing, 2000.
- [66] A. S. Wu, C. L. Ramsey, K. A. De Jong, J. J. Grefenstette, and D. S. Burke. VIS: A genetic algorithm visualization tool. In T. D. Collins, editor, *Evolutionary Computation Visualization*, pages 106–109, Orlando, Florida, USA, 13 July 1999.