A Theoretical Runtime and Empirical Analysis of Different Alternating Variable Searches for Search-Based Testing

Joseph Kempka
Dept. of Computer Science
University of Sheffield
Sheffield S1 4DP, UK

Phil McMinn
Dept. of Computer Science
University of Sheffield
Sheffield S1 4DP, UK

Dirk Sudholt
Dept. of Computer Science
University of Sheffield
Sheffield S1 4DP, UK

ABSTRACT

The Alternating Variable Method (AVM) has been shown to be a surprisingly effective and efficient means of generating branch-covering inputs for procedural programs. However, there has been little work that has sought to analyse the technique and further improve its performance. This paper proposes two new local searches that may be used in conjunction with the AVM, Geometric and Lattice Search. A theoretical runtime analysis shows that under certain conditions, the use of these searches is proven to outperform the original AVM. These theoretical results are confirmed by an empirical study with four programs, which shows that increases of speed of over 50% are possible in practice.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Non-numerical Algorithms and Problems

Keywords

Search-based software engineering, test data generation, local search, runtime analysis, theory

1. INTRODUCTION

First proposed by Korel [8], the *Alternating Variable Method* (AVM) is a simple local search strategy has been shown to be surprisingly effective for covering branches of procedural programs. In a recent empirical study by Harman and McMinn with a range of C programs, the AVM was able to cover the majority of branches faster than a Genetic Algorithm [6]. This suggests that the underlying fitness landscape for covering individual program branches is relatively simple most of the time, with more "heavy-weight" population-based approaches like Genetic Algorithms only required in a minority of cases [11]. Despite this, there has been relatively little work devoted to analysing and improving the performance of the AVM technique.

The AVM can be regarded as a general framework in which a local search strategy is applied to each individual input vector variable in turn. In this paper, we view the local search strategy to be a component of the overall framework that may be substituted for another. The original AVM applied an accelerated hill climb that we refer to as *Iterated Pattern Search* (IPS), where "exploratory" moves in a direction of fitness improvement are proceeded by larger "pattern" steps in the same direction. In this paper, we propose to replace IPS with two new approaches for exploring individual dimensions of the input vector—*Geometric Search* and *Lattice Search*. Geometric and Lattice Search are elimination searches that are able to find the optimum of a one dimensional function that is unimodal on a given interval. They work by comparing the fitness values of two points at predetermined positions, using the result to select a new but smaller sub-range. The algorithm iterates until it is left with one point. Whereas Geometric Search splits the range in two by comparing the middle two positions, Lattice Search compares points that are offset by Fibonacci numbers.

We examine all three variants of the AVM theoretically and empirically. While prior theoretical runtime analyses of the original AVM with IPS involved specific programs and branches [1, 2], we furnish a more general result, proving that for all unimodal functions Geometric and Lattice Search are faster than IPS, when used in the framework of AVM. In a more general sense, Geometric and Lattice Search converge faster to local optima than IPS. These theoretical results are complemented by an empirical study on open source programs. On most branches our new local searches perform significantly better than IPS. This includes unimodal landscapes, in agreement with our theory, as well as non-unimodal ones, where the assumptions of our theory are not met. This indicates that faster convergence to local optima is beneficial on a broad range of instances. The only departure from this pattern was found for one complex landscape of a type for which—as observed in prior studies [6, 11]—a Genetic Algorithm is significantly better than AVM.

The contributions of this paper are therefore as follows:

1. Two new input variable search strategies for use with the AVM, *Geometric Search* and *Lattice Search* (Section 5).
2. A theoretical runtime analysis of the AVM with Geometric and Lattice Search, with extended results for the original AVM approach employing IPS (Section 4). Geometric and Lattice Search are proved to have faster runtimes for unimodal fitness landscapes (Sections 5.1 and 5.2).
3. An empirical analysis of the AVM, comparing IPS, Geometric and Lattice Search on four programs, including unimodal and non-unimodal functions, complementing our theoretical results on unimodal functions and providing additional evidence that our local searches also speed up search on non-unimodal functions, possibly due to their faster convergence to local optima (Section 6).

We begin by giving important background to search-based testing, the AVM, as well as introducing theoretical runtime analysis.

2. BACKGROUND

2.1 Representation and Fitness Function

The fitness function for covering individual program branches is a multivariate function $mf(\vec{x}) \rightarrow \mathbb{R}$, that takes an input vector $\vec{x} = [\, x_1 \; x_2 \; \cdots \; x_n \,]$, i.e. an ordered list of arguments that are passed to a procedure. In this paper, we assume $\vec{x}$ can be modelled as a sequence of integers (i.e., floating point values are permissible so long as a finite precision is used). The fitness function measures how "close" an input vector was to executing a target branch. It is minimised by the search, with a zero value indicating an input that covers the branch. The fitness function has two components. The *approach level* relates to the decision points in the program appearing en route to reaching the target branch. In terms of the program's control dependence graph, the approach level is equal to the number of nodes unexecuted by $\vec{x}$ on which the branch is transitively control dependent. The *branch distance* is computed from the values of variables at predicates where control flow diverged from the target branch. As an example, the branch distance of the predicate $x_1 = x_2$, is given by the formula $|x_1 - x_2|$ (see the survey paper by McMinn [12] for a full list of rules and formulas). Because the maximum branch distance is not known, the normalization function $norm(d(\vec{x})) = 1 - 1.001^{-d(\vec{x})}$ is used, where $d(\vec{x})$ is the raw branch distance and $norm(d(\vec{x}))$ is the normalized branch distance. The full fitness value is computed by normalizing the branch distance and adding it to the approach level, i.e. $mf(\vec{x}) = l(\vec{x}) + norm(d(\vec{x}))$ [18].

2.2 The Alternating Variable Method (AVM)

The AVM can be viewed as a general framework that proceeds from a random starting point in the search space, and works by calling a local search function on each element of the input vector in turn. That is, while the local search is performing "moves" on one component of the vector, the values for all other dimensions remain fixed. If, during this search, a fitness of zero is found, the AVM terminates with the branch-covering input. If, however, a local optimum is reached, the AVM advances to the next element. If fitness cannot be improved after cycling through all elements, the AVM restarts from another randomly-generated input, continuing the search for a branch-covering input until the number of fitness function evaluations exceeds a predefined maximum.

Algorithm 1 describes the AVM framework more formally. The algorithm transforms the multivariate fitness function *mf* into a one dimensional projection f (line 4). The function f is equivalent to evaluating *mf* with an input vector where all components except x_i are set to constants and x_i is substituted by the free parameter x. The function f is passed to a local search algorithm called **local_search**, along with x_i, the starting point for the search. The fitness function keeps track of the number of fitness evaluations, maintaining a mapping of previously evaluated vectors to their corresponding fitness values. Once a branching-covering input vector is found, or when the number of evaluations exceeds the maximum (i.e., the search has failed), an exception is raised to terminate the search (not shown in our algorithms for space and simplicity).

2.3 Runtime Analysis

Runtime analysis has established itself as a leading theory in randomised search heuristics, with many new results in the last 10–15 years [16, 3]. Problems studied in search-based software engineering include computing input-output sequences [9, 10], test input generation [2], and project scheduling [14].

Arcuri, Lehre, and Yao [2] were the first to present an runtime analysis of search-based input generation. They focussed on the

Algorithm 1 The AVM Framework

```
1:  while true do
2:    let $\vec{x} := \mathbf{random}()$, $i = 1$, $c = 0$
3:    while $c < \mathrm{size}(\vec{x})$ do
4:      let $f : mf \mapsto mf((\vec{x} \setminus x_i) \cup \{x\})$
5:      let $\vec{x'} := \mathbf{local\_search}(f, x_i)$
6:      if $mf(\vec{x'}) < mf(\vec{x})$ then
7:        let $\vec{x} := \vec{x'}$, $c := 0$
8:      else
9:        let $c := c + 1$
10:     let $i := i \bmod (\mathrm{size}(\vec{x})) + 1$
```

triangle classification problem, which involves three integer variables describing side lengths of a potential triangle. The task is to classify the input as a scalene, equilateral, or isosceles triangle, or not representing a triangle. Their analysis was limited to the time for covering the *equilateral* branch of the problem. If the range contains n numbers, the expected time of random search is $\Theta(n^2)$. Hill climbing needs expected time $\Theta(n)$, i. e., in every iteration a single variable is increased or decreased by 1. For the AVM they proved an upper bound of $O((\log n)^2)$ and a weaker lower bound of $\Omega(\log n)$ [2].

Later on, Arcuri [1] extended the analysis of the AVM to all branches of the triangle classification problem, showing that the expected running time is bounded from above by $O((\log n)^2)$ on all branches. Branches with many global optima only need $O(\log n)$ time. In this paper we extend his results by proving an upper bound of $O((\log n)^2)$ for *all* strictly unimodal functions (and functions with further global optima).

3. PRELIMINARIES

A function f is called *strictly unimodal* if it only has a single local optimum. If f is to be minimised (the case of maximisation is symmetric) this means that

$$f(\ell_1) > f(\ell_2) > f(\mathrm{opt}) < f(r_1) < f(r_2)$$

for all $\ell_1 < \ell_2 < \mathrm{opt} < r_1 < r_2$, where opt is the global minimum of f. In other words, if D is the domain of f, f is strictly decreasing on $D \cap (-\infty, \mathrm{opt}]$ and strictly increasing on $D \cap [\mathrm{opt}, \infty)$.

We consider the running time of local searches used within the framework of AVM. Thereby we count the number of function evaluations or fitness evaluations made by local search until an optimum is found for the first time. The motivation for considering fitness evaluations is that such an evaluation is the most costly operation as it involves simulating the program. In some cases we count *unique fitness evaluations*, i. e., the number of *different* search points evaluated. This reflects the fact that it is easy to cache past evaluations, so evaluating the same point twice only incurs insignificant additional cost.

In our analyses we consider minimising a function $f \colon D \rightarrow \mathbb{R}$ for a finite domain $D \subset \mathbb{Z}$. For ease of presentation, we assume $f(x) = \infty$ for all $x \notin D$. This includes settings where f is the branch distance before or after normalisation. The precise choice of a normalisation function is irrelevant in our work; all local searches analysed in this work only use information about the ranks of search points. So every strictly increasing normalisation function leads to the same sequences of search points queried and hence the same performance.

In previous work [1, 2] the authors derived performance results with regard to the size n of the domain, e. g., $\{1, \dots, n\}$ or $\{-n/2 + 1, \dots, n/2\}$. Here we consider the initial distance d to the optimum instead, as this distance governs the running time of all local searches considered in this work. As $d \leq n$ upper running time bounds using d are generally stronger than those using n.

4. ANALYSIS OF THE ORIGINAL AVM

4.1 Original AVM with IPS

The original AVM due to Korel [8] uses the following local search, shown in Algorithm 2, that we name *Iterated Pattern Search* (IPS). Starting at x, IPS first evaluates points $x-1$ and $x+1$ to identify a gradient. Unless x is a local optimum, it then performs a so-called pattern search, moving in the direction of decreasing f-values. The step size doubles with each step, so when the gradient is towards increasing indices IPS traverses the points

$$x, x+1, x+1+2, x+1+2+4, \dots, x+1+2+4+\dots+2^j.$$

Since $\sum_{i=0}^{j} 2^j = 2^{j+1}-1$ this sequence is equal to

$$x, x+2^1-1, x+2^2-1, x+2^3-1, \dots, x+2^{j+1}-1.$$

Pattern search stops if the next point does not improve the fitness, which happens on unimodal functions when the optimum is being overshot. This process is iterated; that is, IPS then starts another exploration. If the function is unimodal, IPS gets close to the optimum over time. However, this line search can be relatively slow. The reason is that IPS accelerates during exploration, but after overshooting the optimum IPS starts from scratch.

Algorithm 2 Iterated Pattern Search, starting at $x \in D$

```
1: while true do
2:     if $f(x-1) \geq f(x)$ and $f(x+1) \geq f(x)$ return $x$
3:     if $f(x-1) < f(x+1)$ then let $k := -1$ else let $k := 1$
4:     while $f(x+k) < f(x)$ do
5:         let $x := x+k, k := 2k$
```

4.2 Upper Bound for Original AVM

We start our investigations with Iterated Pattern Search, the local search used in the original AVM [8]. An upper bound $O(\log^2 n)$ (for domain size n) was proven for AVM with IPS in [2], for the special case that there is a linear relationship between the function value and the distance to the optimum. The following statement holds for arbitrary strictly unimodal functions.

THEOREM 1. *Consider iterated pattern search on a strictly unimodal function f where d denotes the initial distance from the starting point to the optimum. Then IPS finds an optimum after querying at most $4 + 8\log d + (\log d)^2$ values. This also holds for functions f that result from a strictly unimodal function f' and assigning function values $\arg\min(f')$ to further points.*

The last statement implies that we get an upper bound of order $O((\log d)^2)$ for many further common functions. Examples are functions where opt is a global optimum and all points $x > \text{opt}$ or $x < \text{opt}$ are global optima as well. In particular, all functions considered in Arcuri's work [1] are covered by this statement. However, for functions with many global optima the upper bound may not be tight [1].

Proof of Theorem 1. We allow the algorithm to traverse points outside of D, but assume that all $x' \notin D$ are worse than all $x \in D$.

We consider *passes* of IPS, corresponding to one iteration of the outer while loop in Algorithm 2: a pass starts with an exploratory search examining the two neighbouring solutions of the current point (index ± 1). It then performs a pattern search, doubling the distance travelled in each step. Note that a pass starting with the optimum makes exactly 3 queries. If a pass queries points up to a distance of $\sum_{j=0}^{i} 2^j = 2^{i+1}-1$ from the initial value, it queries $i+3$ values.

Without loss of generality assume that the current position is 0 and the optimum is at d. If $d=1$ we need 4 queries, hence we assume in the following that $d \geq 2$. We claim that within at most 2 passes the distance to the optimum has been reduced to at most $\lfloor d/2 \rfloor$. Let i be the unique integer such that $2^i - 1 \leq d < 2^{i+1}-1$. Note that pattern search queries 2^i-1 and $2^{i+1}-1$ as the points are strictly improving in $[0, 2^i-1]$. We consider two cases. First assume that 2^i-1 is better than $2^{i+1}-1$. Then pattern search stops at 2^i-1 and since $d \leq 2^{i+1}-2$ the new distance to the optimum is at most $\lfloor d/2 \rfloor$. The number of queries made is at most $i+3$, and since $d \geq 2^i - 1 \geq 2^{i-1}$ this is at most $\lfloor \log d \rfloor + 4$.

Now consider the case that 2^i-1 is worse than $2^{i+1}-1$. This implies $2^i \leq d \leq 2^{i+1}-2$. Then pattern search will query $2^{i+2}-1$ and stop at $2^{i+1}-1$ as $2^{i+2}-1$ is worse than $2^{i+1}-1$. The second pass will traverse positions $2^{i+1}-2^1, 2^{i+1}-2^2, 2^{i+1}-2^3 \dots$. Since by unimodality all points in $[0, 2^i]$ are increasingly better, pattern search will stop at some $2^{i+1}-2^j$ with $0 \leq j \leq i$. As the optimum must be within $[\max(2^i, 2^{i+1}-2^{j+1}), 2^{i+1}-2^{j-1}]$, and the new current point is $2^{i+1}-2^j$, the distance between the current point and the optimum has decreased to at most 2^j for $j < i$ and 2^{i-1} for $j = i$. In both cases the new distance is bounded by $\lfloor d/2 \rfloor$.

In the worst case, we need one pass querying up to $i+4$ values, and a second pass querying up to $i+3$ points. The total is $2i+7 \leq 2\lfloor \log d \rfloor + 7$ as $d \geq 2^i$.

The total number of queries made, $T(d)$, is then subject to the following recurrence: $T(0) = 3$, $T(1) = 4$, and $T(d) \leq 2\lfloor \log d \rfloor + 7 + T(\lfloor d/2 \rfloor)$ for $d > 1$. Due to all floor functions, we get the same recurrence for $T(d)$ as for $T(2^{\lfloor \log d \rfloor})$. Solving the latter gives

$$\begin{aligned} T(d) &\leq \sum_{k=1}^{\lfloor \log d \rfloor} (2k+7) + T(1) \\ &\leq 7\lfloor \log d \rfloor + 2 \cdot \frac{\lfloor \log d \rfloor (\lfloor \log d \rfloor + 1)}{2} + T(1) \\ &\leq 4 + 8\log d + (\log d)^2. \end{aligned}$$

The last remark of the statement holds true since adding further global optima can only decrease the expected time until some global optimum is found. □

4.3 Original AVM is Slow in the Worst Case

The following result shows that the lower bound from Theorem 1 is asymptotically tight. Both bounds together show that the worst-case running time of IPS is of order $\Theta((\log d)^2)$, when initial points up to distance d are allowed.

THEOREM 2. *Consider iterated pattern search minimising an arbitrary unimodal function f. If there are feasible starting points with distances $0, 1, \dots, d$ to the optimum, the worst case number of unique fitness evaluations is at least*

$$\frac{(\log d)^2}{10} - O(\log d).$$

Proof. We can assume w. l. o. g. that the optimum is at position 0. If the domain D is bounded on one side, we consider an extended function f' where the domain is $\mathbb{Z}$ and $f(x) = \infty$ for all $x \notin D$.

Define $T_s(\ell, r)$ as the number of different search points evaluated when IPS starts in s, counting evaluations from the set $\{\ell, \dots, r\}$ only. If $s \notin D$ we let $T_s(\ell, r) := \infty$. Let $T(\ell, r) := \min\{T_\ell(\ell, r), T_r(\ell, r)\}$ be the fastest time starting from ℓ or r.

Define $\ell_0 = r_0 = 0$ and $T(\ell_0, r_0) = 1$. Assume we have $\ell_i \leq 0 \leq r_i$ for some $i \in \mathbb{N}_0$, such that ℓ_i and r_i are not both outside f's domain. Let $\Delta_i := 2^{\lfloor \log(r_i - \ell_i) \rfloor + 1}$ for $i \in \mathbb{N}$ and $\Delta_0 := 1$ be the smallest power of 2 such that $\Delta_i > r_i - \ell_i$.

We define new points ℓ_{i+1}, r_{i+1} according to the following case distinction. First assume $f(\ell_i) > f(r_i)$, which implies $f(x) > f(r_i)$ for all $x \leq \ell_i$. It also implies that r_i exists. Let $r_{i+1} := r_i + \Delta_i - 1$ and $\ell_{i+1} := r_i - 2\Delta_i + 1$. If IPS starts at r_{i+1}, it will sample points at

$$r_{i+1}, r_{i+1} - (2^1 - 1), \ldots, r_{i+1} - (2^{\log(\Delta_i)} - 1) = r_i, r_i - \Delta_i$$

and since the fitness improves in every step but the last one, IPS will stop at r_i and restart exploration from there. All points but $r_i - \Delta_i$ are guaranteed to exist and are contained in $\{\ell_{i+1}, \ldots, r_{i+1}\}$; so IPS evaluates $\Delta_i + 1$ different search points from that set before restarting exploration. From r_i IPS needs time at least $T(\ell_i, r_i) - 1$ since so far IPS has evaluated a single search point from the set $\{\ell_i, \ldots, r_i\}$, namely r_i. We thus have established the recurrence

$$T_{\ell_{i+1}}(\ell_{i+1}, r_{i+1}) \geq \Delta_i + T(\ell_i, r_i). \quad (1)$$

Similarly, if ℓ_{i+1} exists and IPS starts from there, it will sample points at

$$\ell_{i+1}, \ell_{i+1} + 2^1 - 1, \ldots, \ell_{i+1} + 2^{\log(\Delta_i)} - 1 = r_i - \Delta_i, r_i, r_i + 2\Delta_i.$$

The fitness improves in each step but the last one, and so IPS will stop at r_i and start exploration from there. Not counting the evaluation of $r_i + 2\Delta_i$, IPS evaluates $\Delta_i + 2$ search points, hence as above we get

$$T_{r_{i+1}}(\ell_{i+1}, r_{i+1}) \geq \Delta_i + T(\ell_i, r_i) + 1. \quad (2)$$

Putting (1) and (2) together, we have shown

$$T(\ell_{i+1}, r_{i+1}) \geq \Delta_i + T(\ell_i, r_i).$$

This also holds when $\ell_{i+1} \notin D$ as then $T_{\ell_{i+1}}(\ell_i, r_i) = \infty$. The case $f(\ell_i) < f(r_i)$ is symmetric, and if $f(\ell_i) = f(r_i)$ we also get the same recurrence as IPS stops at r_i as in the case $f(\ell_i) > f(r_i)$ when starting from ℓ_{i+1} and IPS stops at ℓ_i as on the other case when starting from r_{i+1}.

It follows that

$$T(\ell_i, r_i) \geq \sum_{j=1}^{i-1} \Delta_j + T(\ell_0, r_0) \geq 1 + \sum_{j=1}^{i-1} \log(r_j - \ell_j).$$

Note that for $i \geq 1$ the difference $r_{i+1} - \ell_{i+1}$ does not depend on whether $f(\ell_i) > f(r_i)$ or not, hence w. l. o. g. we use the definition of ℓ_{i+1}, r_{i+1} from the case $f(\ell_i) > f(r_i)$. Along with $\Delta_i \geq r_i - \ell_i + 1$ we get

$$r_{i+1} - \ell_{i+1} = r_i + \Delta_i - 1 - (r_i - 2\Delta_i + 1) = 3\Delta_i - 2 > 3(r_i - \ell_i).$$

Expanding and using $\ell_1 - r_1 = 2$ yields

$$r_{i+1} - \ell_{i+1} > 2 \cdot 3^i.$$

Thus,

$$\begin{aligned} T(\ell_i, r_i) &\geq 1 + \sum_{j=1}^{i-1} \log(2 \cdot 3^{j-1}) \\ &= i + \sum_{j=0}^{i-2} \log(3^j) \\ &= i + \log(3) \cdot \frac{(i-2)(i-1)}{2} > \frac{\log(3)}{2} \cdot i^2 - O(i). \end{aligned}$$

It is easy to verify by induction that $r_i \leq 7^{i-1}$ and $|\ell_i| \leq 7^{i-1}$ for all $i \in \mathbb{N}$. Putting $i := \lfloor \log_7(d) \rfloor + 1$ then gives a lower bound of

$$\frac{\log(3)}{2} \cdot (\log_7(2))^2 \cdot (\log d)^2 - O(\log d) > \frac{(\log d)^2}{10} - O(\log d).$$

□

4.4 Original AVM is Slow on Average

The bad worst-case performance of IPS is not simply due to few unlucky choices of the initial point. In fact, most starting points lead to a running time of order $\Theta((\log d)^2)$. For the specific function $f(x) = |x|$ we show that when the target is chosen such that the distance between starting point and target is uniform in some interval, then we still get a lower bound of order $(\log d)^2$.

Note that $f(x) = |x|$ is quite an easy function as points closer to the optimum 0 are better than points that are further away from it. This encourages IPS to stop at the closest point to the optimum traversed in a pattern search, but we still get a time of $\Omega((\log d)^2)$.

THEOREM 3. *Consider iterated pattern search minimising the function $f(x) = |x|$ such that the target is chosen uniformly at random from $\{-2^i, \ldots, 2^i - 1\}$, for some $i \in \mathbb{N}_0$. The expected number of unique fitness evaluations is at least $\frac{i^2}{6}$.*

Proof. Let $T(i)$ denote the expected number of different search points queried when the target is chosen uniformly at random from $\{-2^i, \ldots, 2^i - 1\}$. The claim $T(i) \geq i^2/6$ is trivial for $i = 0$ and $i = 1$.

If IPS starts at some value $x < 0$ (the case $x > 0$ is symmetric), IPS will start a pattern search exploring points with higher indices, querying points at $x_1 := x + 2^0, x_2 := x + 2^0 + 2^1, x_3 := x + 2^0 + 2^1 + 2^2$, etc. (We do not count a potential evaluation of the point at $x - 1$ since it might not be in the domain of feasible search points.) Let $x_j := x + \sum_{\ell=0}^{j-1} 2^\ell = x + 2^j - 1$ for $1 \leq j \leq i$ be the first search point queried where $x_j \geq 0$. Now IPS will stop pattern search and continue with either x_{j-1} or x_j, depending on which is better. If x_j is better, IPS will also query x_{j+1}; but as this might be out of range, we do not count a potential evaluation of x_{j+1}.

Due to the fitness function used, the point with the smaller absolute value from either x_{j-1} or x_j is better. Note that their index difference is $x_j - x_{j-1} = 2^{j-1}$, so $x_j \in \{0, \ldots, 2^{j-1} - 1\}$. If $x_j \in \{0, \ldots, 2^{j-2} - 1\}$, x_j is better than x_{j-1}, and IPS starts another pass at $\{0, \ldots, 2^{j-2} - 1\}$. Otherwise, x_{j-1} is better and IPS will start another pass at $\{-2^{j-2}, -2^{j-2} + 1, \ldots, -1\}$. All these positions are attained with the same probability, hence we are in the same setting as described in the statement, with $j - 2$ in place of i.

The probability of stopping at x_j being the first point where $x_j \geq 0$ ($x_j \leq 0$ when starting at $x > 0$), for $1 \leq j \leq i$, is $2 \cdot 2^{j-1}/2^{i+1} = 2^{j-i-1}$ as there are $x_j - x_{j-1} = 2^{j-1}$ positions for x where this happens when $x < 0$ and the same holds for $x > 0$. Recall that all initial positions are chosen uniformly at random, and there are 2^{i+1} feasible positions.

While getting to x_j IPS has queried at least $j + 1$ mutually different points $x, x_1, \ldots, x_j$. Then the remaining time is at least $T(j-2) - 1$; the reason for subtracting 1 is that we have already queried x_j. Defining $T(-1) := 0$, we have established the following recurrence

$$\begin{aligned} T(i) \geq& \sum_{j=1}^{i} 2^{j-i-1} \cdot (j + 1 + T(j-2) - 1) \\ &= \sum_{j=1}^{i} j \cdot 2^{j-i-1} + \sum_{j=1}^{i} 2^{j-i-1} \cdot T(j-2) \\ &= \sum_{j=0}^{i-1} (j+1) \cdot 2^{j-i} + \sum_{j=0}^{i-2} 2^{j-i+1} \cdot T(j) \\ &= i - 1 + 2^{-i} + \sum_{j=0}^{i-2} 2^{j-i+1} \cdot T(j) \end{aligned}$$

having used $\sum_{j=0}^{i-1} (j+1) \cdot 2^{j-i} = i - 1 + 2^{-i}$.

Assume for an induction that $T(j) \geq j^2/6$ all $0 \leq j < i$. Then

$$\begin{aligned} T(i) &\geq i - 1 + 2^{-i} + \sum_{j=0}^{i-1} 2^{j-i} \cdot \frac{j^2}{6} \\ &= i - 1 + 2^{-i} + \frac{1}{6} \cdot \sum_{j=0}^{i-1} 2^{j-i} \cdot j^2 \\ &= i - 1 + 2^{-i} + \frac{1}{6} \cdot (-4i + 6 + i^2 - 6 \cdot 2^{-i}) \\ &= \frac{i^2}{6} + \frac{i}{3} \end{aligned}$$

which implies the claim. □

5. NEW LOCAL SEARCHES FOR THE AVM

We now show that other local searches used in the framework provided by AVM only require $\Theta(\log d)$ evaluations instead of $\Theta((\log d)^2)$. This yields significant speedups over AVM's original local search method IPS, if the initial distance d to the optimum is not very small.

Our results formally only hold for unimodal functions, but they also indicate more generally that our new local searches converge faster to local optima. The reason is that the basin of attraction around a local optimum has the properties of a unimodal function. So, our analysis is partly applicable in a much wider context; exploring this further is left for future work.

5.1 AVM with Geometric Search

We propose more clever local searches that locate the optimum of a unimodal function more efficiently after the first exploration. The following *Geometric Search* uses a variant of binary search. The idea is to perform a pattern search, and then to use binary search to home in on the target. We will see that then the optimum of any unimodal function is found in time logarithmic in the initial distance. We call it "Geometric Search" since the initial pattern search is performed with a geometric sequence of numbers.

Algorithm 3 Geometric search, starting at $x \in D$

```
 1: if $f(x-1) \geq f(x)$ and $f(x+1) \geq f(x)$ return $x$
 2: if $f(x-1) < f(x+1)$ then let $k := -1$ else let $k := 1$
 3: while $f(x+k) < f(x)$ do
 4:     let $x := x + k, k := 2k$
 5: let $\ell := \min(x - k/2, x + k), r := \max(x - k/2, x + k)$
 6: while $\ell \neq r$ do
 7:     if $f(\lfloor(\ell + r)/2\rfloor) < f(\lfloor(\ell + r)/2\rfloor + 1)$ then
 8:         $r := \lfloor(\ell + r)/2\rfloor$
 9:     else
10:         $\ell := \lfloor(\ell + r)/2\rfloor + 1$
11: return $\ell$
```

Geometric Search uses the same "geometric" pattern search as IPS, but afterwards uses a variant of binary search to narrow down the optimum. Thereby we are using that if pattern search queries search points x_{j-1}, x_j, x_{j+1}, stopping at x_j, we know that $f(x_{j-1}) > f(x_j) \leq f(x_{j+1})$. This implies that, if f is unimodal, the global minimum must lie in the set $\{x_{j-1}, \dots, x_{j+1}\}$.

THEOREM 4. *Consider a one-dimensional search on a unimodal function where d denotes the initial distance from the starting point to the optimum. Then Geometric Search finds an optimum after querying at most* $3 \log d + 5$ *search points.*

Proof. Let i be such that $2^i \leq d < 2^{i+1}$. By the same arguments as in the proof of Theorem 1 pattern search stops at either $2^i - 1$ or $2^{i+1} - 1$ after querying at most $i + 3$ points. We pessimistically assume that it stops at $2^{i+1} - 1$, which results in the algorithm putting $\ell := 2^i - 1$ and $r := 2^{i+2} - 1$. We claim that each iteration of binary search updates ℓ, r towards ℓ', r' such that $r' - \ell' \leq \lfloor(r - \ell)/2\rfloor$. If $r' = \lfloor(\ell + r)/2\rfloor$ we have

$$r' - \ell' = \left\lfloor \frac{\ell + r}{2} \right\rfloor - \ell = \left\lfloor \frac{r - \ell}{2} \right\rfloor.$$

Otherwise, $\ell' = \left\lfloor \frac{\ell+r}{2} \right\rfloor + 1$ and using $-\lfloor x \rfloor - 1 \leq \lfloor -x \rfloor$ for $x \in \mathbb{R}$ we get

$$r' - \ell' = r - \left\lfloor \frac{\ell + r}{2} \right\rfloor - 1 \leq r + \left\lfloor -\frac{\ell + r}{2} \right\rfloor = \left\lfloor \frac{r - \ell}{2} \right\rfloor.$$

Initially $r - \ell < 2^{i+2}$, and due to the floor functions we get the same recurrence as for 2^{i+1}. Two queries are needed to replace the current distance by its floored half, ending at 0 with no further queries. Hence we need an additional amount of $2i + 2$ queries, leading to a total of $3i + 5 \leq 3 \log d + 5$ queries. □

5.2 AVM with Lattice Search

Lattice Search [15] is a refinement of Fibonacci Search [7] for integer domains. It can find the minimum of a unimodal function on a domain of integers $\{1, \dots, F_n - 1\}$ using $n - 2$ function evaluations [15, page 190], where F_n is the n-th Fibonacci number: $F_1 = 1, F_2 = 1$, and $F_{n+2} = F_n + F_{n+1}$ for $n \in \mathbb{N}$ (Monahan [15, page 190] uses the definition $F_0 = 1, F_1 = 1, F_2 = 2 \dots$). Search points are evaluated according to Fibonacci numbers in such a way that only one new search points needs to be evaluated in each iteration. This makes Lattice Search faster than Geometric Search.

Our local search using Lattice Search is as follows.

Algorithm 4 Lattice search, starting at $x \in D$

```
 1: if $f(x-1) \geq f(x)$ and $f(x+1) \geq f(x)$ return $x$
 2: if $f(x-1) < f(x+1)$ then let $k := -1$ else let $k := 1$
 3: while $f(x+k) < f(x)$ do
 4:     let $x := x + k, k := 2k$
 5: let $\ell := \min(x - k/2, x + k), r := \max(x - k/2, x + k)$
 6: let $n := \min\{n \mid F_n \geq r - l + 2\}$
 7: while $n \neq 3$ do
 8:     if $\ell + F_{n-1} - 1 \leq r$ and $f(\ell + F_{n-2} - 1) \geq f(\ell + F_{n-1} - 1)$
        then
 9:         let $\ell := \ell + F_{n-2}$
10:     let $n := n - 1$
11: return $\ell$
```

Note that the initial pattern search is done in a geometric fashion, i.e., increasing the step size geometrically as in IPS and Geometric Search. There is a related search technique called *Fibonaccian Searching* [4] (not to be confused with Fibonacci Search) where pattern search is done by means of Fibonacci numbers. The reason that we are using geometric pattern search is that it is generally faster.

Lattice Search further improves the leading constant preceding the $\log d$ term; using Fibonacci numbers to search for the optimum in the interval identified by pattern search is more efficient than the binary search used in our Geometric Search procedure.

THEOREM 5. *Consider a one-dimensional search on a unimodal function where d denotes the initial distance from the starting point to the optimum. Then Lattice Search finds an optimum after querying at most* $2.45 \log d + O(1)$ *search points.*

Due to space restrictions, we omit the proof. It follows from previous arguments, the analysis of Lattice Search [15, page 190] and rewriting the following closed formula for Fibonacci numbers:

$$F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n \cdot \sqrt{5}}.$$

6. EMPIRICAL EXPERIMENTS

We now provide an empirical comparison of AVM using our different local searches to complement our theoretical results. We first show results on a simple function, designed to provide empirically confirmation of our proofs on a unimodal function, and actual differences in performance in practice. However, real-world programs are not so straightforward, involving fitness landscapes of varying shapes. Therefore, following an initial simple experiment, we continue onto performing experiments with four real-world programs.

6.1 Experiments with "is_zero" ($f(x) = |x|$)

We first study a simple and illustrative problem with just a single variable, as shown in the following program below:

```
void is_zero(int x) {
  if (x == 0) { /* TARGET BRANCH */ }
}
```

The goal is to find an input x that is equal to zero. The approach level is zero (since the branch is always reached), and the branch distance is $norm(|x|)$. Because our local searches do not rely on comparing absolute values, but instead their corresponding ranks, this problem is equivalent to minimizing the fitness function $f(x) = |x|$ (where the branch distance is not normalised) as analysed in Theorem 3.

Studying this simple program allows us to isolate the impact of the range and the initial distance d from the optimum on the running time of AVM. Our theoretical results show that, when the starting point is chosen uniformly at random from a set $\{-d, \ldots, d-1\}$ for d a power of 2, AVM with IPS will take $\Theta((\log d)^2)$ steps whereas AVM with Geometric Search or Lattice Search will succeed in only $\Theta(\log d)$ steps.

The performance of IPS, Geometric and Lattice in optimising the objective function, $f(x) = |x|$ was measured over the following ranges $[-d, d-1]$, where $d \in \{1, 2, 4, 8, \ldots 2^{31}\}$. For each range 100 runs of each local search were performed and the number of fitness evaluations to find the global optimum at 0 was counted. In each run the starting position, x_1 was chosen uniformly at random from the corresponding range (including boundaries).

For each range the runtime distributions of the local searches were compared in a pairwise manner using the Mann-Whitney U test. In addition to p-values, the non-parametric Vargha-Delaney statistic $\hat{A}_{12}$ [17], which is computed from mean ranks, is reported as a measure of effect size. It is possible to interpret $\hat{A}_{12}$ as the probability that a run of the first search algorithm takes a larger number of fitness evaluations compared to that of the second search algorithm. The implication is that if $\hat{A}_{12} < 0.5$, then the first local search performs better overall whereas the opposite is true if $\hat{A}_{12} > 0.5$. Also, depending whether the absolute difference: $|\hat{A}_{12} - 0.5|$ is > 0.21, > 0.14, > 0.06 or ≤ 0.06, the corresponding effect size can be divided into the following categories: large, medium, small and negligible.

The results show that for all ranges where $d \geq 2048$, IPS is worse than Geometric ($p \leq 0.0054$ and $\hat{A}_{12} \geq 0.61$) and also worse than Lattice ($p \leq 4.1 \cdot 10^{-8}$ and $\hat{A}_{12} \geq 0.72$). It was also found that for the same ranges Geometric is worse than Lattice ($p \leq 7.9 \cdot 10^{-8}$ and $\hat{A}_{12} \geq 0.72$).

Figure 1 shows how the average performance of each local search scales with increasing domain size. Here the variable, i is related to the logarithm of the distance, i. e. $i = \log_2(d)$.

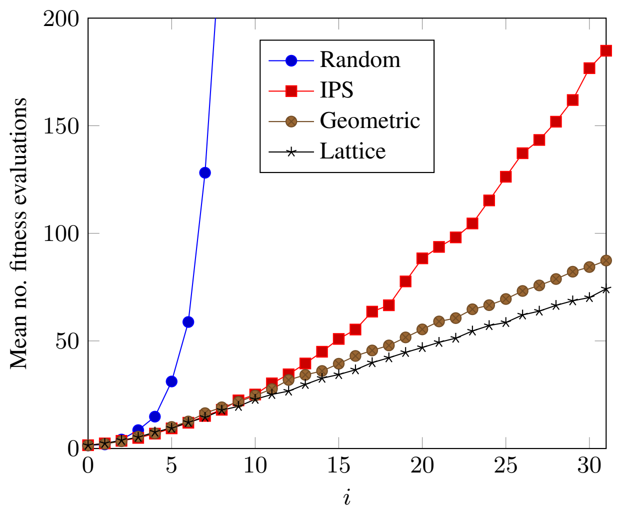

Figure 1: Mean number of fitness evaluations for optimising $f(x) = |x|$ with various local searches. The domain is chosen as $\{-2^i, \ldots, 2^i - 1\}$ for $i \in \{0, \ldots, 31\}$.

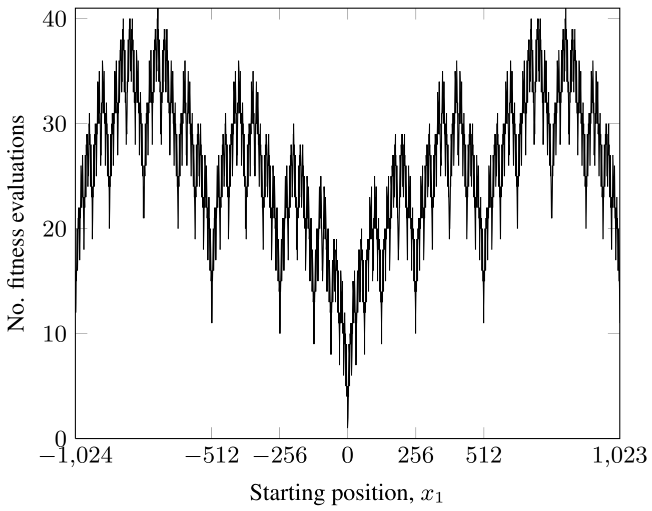

Figure 2: Number of fitness evaluations for optimising $f(x) = |x|$ with IPS for each starting position, $x_1 \in [-1024, 1023]$.

The empirical results agree with our theoretical results as one can clearly see a different scaling behaviour between IPS and our two new local searches, Geometric and Lattice. For small d the performance is similar, but as d grows the differences become obvious.

A second order polynomial was fit to the average running times of IPS using a weighted non-linear regression and then the χ^2 test was used to assess the goodness of fit. The equation of the fitted curve is as follows: $T(i) = 1.5189 + 0.717115 \cdot i + 0.169623 \cdot i^2$, where T is the mean number of fitness evaluations. Note that the leading constant 0.169623 almost exactly matches the constant $1/6 = 0.166666\ldots$ from Theorem 3. For the fit, the obtained value of χ^2 is 27.6887 and the number of degrees of freedom is $32 - 3 = 29$. Since $P(\chi^2 > 27.6887) = 0.5345$, it suggested that the fit is of high quality and explains the experimental results very well.

In Figure 2 we additionally show how the performance of iterated pattern search depends on the precise choice of the starting point. The performance is symmetric around 0 (modulo tiny differences in tie-breaking) and the pattern looks like a fractal structure.

This reflects the recursive nature of IPS, which also became visible in the recurrence arguments used in our proofs from Section 4. In accordance with our average-case analysis from Theorem 3, many starting points lead to rather high running times.

6.2 Experiments with Real-World Test Objects

In order to compare the performance of the local searches in a practical setting, we implemented our new AVM searches into and conducted our experiments with the IGUANA toolset [13], and selected four test objects, details of which are shown in Table 1. Each test object is written in C, and its source code was automatically instrumented by IGUANA for the purposes of collecting fitness information. The *clip_to_circle* function is from the graphical front-end of the *SPICE* electronic circuit simulator. The functions *gimp_rgb_to_hsv_int* and *gimp_hsv_to_rgb_int* are colour space converters from the *GIMP* image editor. Finally, *validate_card* is an implementation of the Luhn algorithm for checking 16 digit credit card numbers. Each test object consists of nested control structures in the form of *if* statements, *switch* statements and loops. IGUANA creates a pair of true and false branches when it detects that control flow diverges in the source code of a C function. Because there can be multiple pairs of branches in a single test object, each pair is labelled numerically.

In the experiments each local search was applied to each branch and the number of unique fitness evaluations needed to cover each branch was counted. This process was repeated 100 times with a different seed being used to initialize the random number generator in each run. For practical reasons the maximum number of fitness evaluations to cover any one branch was capped at 100,000. For each branch pairwise comparisons of the runtime distributions corresponding to different local searches were performed using the Mann-Whitney U test. The results including raw p-values and non-parametric effect sizes, $\hat{A}_{12}$, are shown in Table 2.

Only branches where random search performed notably worse than at least one of the local searches (i. e. $p < 0.05$ (significant) and $\hat{A}_{12} > 0.56$ (at least small effect size)) are considered. The purpose of filtering was to remove branches that are covered easily by any search as there is no reason to design a better algorithm for these branches. Similarly, there were 11 branches which were either infeasible (i. e. no inputs from the input domain execute it) or so hard that none of the tested algorithms found an optimum. In total, 24 out of 82 branches satisfied the selection criteria.

We used a sampling approach to investigate how many fitness landscapes presented to a local searcher are unimodal (details are omitted due to a lack of space). Across all branches listed in Table 2, 13% of landscapes were strictly unimodal and 62% had a similar property: all local optima were also global optima. Only 25% of all sampled settings contained local optima, which were not globally optimal; almost all of these belonged to *SPICE*.

For 14 out of the 24 branches, both Geometric and Lattice perform better than IPS. There were in total 16 branches where Geometric or Lattice were significantly better than IPS, and on 12 of these the effect size was medium or large. In a similar way it is observed that Lattice generally outperforms Geometric, however the effect sizes for such comparisons are typically smaller. Another important result is that the difference in runtime performance between two local search algorithms varies according to the specific branch. On some branches all local searches appear to perform equally well, whereas on other branches one local search is clearly better than another.

The largest improvements were observed with the *clip_to_circle* function (from the *SPICE* project). For the other functions, however, branches were covered relatively quickly regardless of the local search used, leading to a difference of only one or two evaluations on average. This is likely due to their function's small input domain size (see Table 1). While results for these functions were significant, the difference in runtime from a practical standpoint is almost negligible. There are also branches where significant differences between the runtime distributions of local searches exist but the calculated effect sizes are marginally small. Because no correction was made for multiple comparisons, it is likely that a few of the "less significant" results are false positives.

Table 1: Details of the test objects used in the experiments

Function name	No. of branches	Inputs and Ranges
clip_to_circle	42	$\{x_1 \ldots x_7\} \in [-2^{31}, 2^{31}-1]$
gimp_rgb_to_hsv_int	14	$\{x_1 \ldots x_3\} \in [0, 255]$
gimp_hsv_to_rgb_int	16	$x_1 \in [0, 360]$, $\{x_2, x_3\} \in [0, 255]$
validate_card	10	$\{x_1 \ldots x_{16}\} \in [0, 9]$

Furthermore, there is one branch (namely branch 14T of the *gimp_rgb_to_hsv_int* test object) in which IPS performs far better than both Geometric and Lattice. It is clear that this branch is difficult to cover because it was the only branch in all test cases which local searches failed to cover within the allowed number of fitness evaluations. The success rates for this branch are 0% for Random, 93% for IPS, 6% for Geometric and 2% for Lattice. Additional experiments with 10 runs per search and without a limitation on the maximum number of fitness evaluations gave the median number of fitness evaluations to achieve coverage of this branch to be $923,862$ for Geometric and $539,127.5$ for Lattice. We observed that all searches frequently resorted to restarting, with Geometric and Lattice only able to hit the target when a restart produces a solution with at least 2 out of 3 variables already optimised by chance. We found that the fitness landscape of the branch contains several plateaux, and IPS seems to perform better on this branch because it has a higher tendency to explore these plateaux. Further investigations with this branch and the Wegener Genetic Algorithm [6, 18] revealed the GA was much more efficient at finding a solution, requiring only $4,795$ evaluations as a mean average (median = 4633.5), compared to approximately $31,000$ for the AVM with IPS. This is a significant result ($p = 7.9 \times 10^{-18}$), with a large effect size ($\hat{A}_{12} = 0.85$). This result fits with those from previous studies in search-based test input generation, where the AVM works most efficiently for simple fitness landscapes with "obvious" optima, whereas diversifying GAs are more efficient at navigating less smooth landscapes generated by more difficult branches [6].

6.3 Threats to Validity

We briefly consider some threats to validity associated with our study. From the point of view of *external threats*, the test objects in our experiments may not generalise in practice, however, care was taken to select them from real-world open source examples. These examples go beyond the bounds of our theory, but still show positive results in the majority of cases. From the point of view of *internal threats*, possible errors come from our implementation of the techniques. However, as shown with the simple and controlled `is_zero` example, empirical results closely matched those expected from our theoretical observations. Furthermore we used non-parametric statistical tests to analyse our results, i.e. the Mann-Whitney U test and the Vargha-Delaney $\hat{A}_{12}$ statistic, both of which do not have assumptions regarding normality of the sample means, avoiding a further potential source of error from our analysis.

7. CONCLUSIONS AND FUTURE WORK

We have analysed the performance of the original AVM incorporating Iterated Pattern Search (IPS), proposing to replace the

Table 2: Results of test case experiments. The p-values formatted in bold indicate significance at the 0.05 level. Similarly, effect sizes that are large, medium, small and negligible are distinguished by bold, underlined, italic and normal formatting respectively.

		Median no. of fitness evaluations			IPS v Geometric		IPS v Lattice		Geometric v Lattice	
Function	**Branch**	**IPS**	**Geometric**	**Lattice**	**p-value**	**$\hat{A}_{12}$**	**p-value**	**$\hat{A}_{12}$**	**p-value**	**$\hat{A}_{12}$**
clip_to_circle	7F	401.5	181.0	150.5	$\mathbf{1.1 \times 10^{-12}}$	**0.79**	$\mathbf{5.1 \times 10^{-17}}$	**0.84**	$\mathbf{5.8 \times 10^{-6}}$	$\underline{0.69}$
	10F	408.0	184.0	155.5	$\mathbf{3.0 \times 10^{-11}}$	**0.77**	$\mathbf{3.0 \times 10^{-19}}$	**0.87**	$\mathbf{3.0 \times 10^{-9}}$	**0.74**
	55T	1515.0	1296.0	1098.0	$\mathbf{2.4 \times 10^{-2}}$	*0.59*	$\mathbf{4.1 \times 10^{-3}}$	*0.62*	4.2×10^{-1}	0.53
	55F	552.5	303.5	271.0	$\mathbf{9.9 \times 10^{-9}}$	**0.73**	$\mathbf{6.1 \times 10^{-11}}$	**0.77**	1.6×10^{-1}	0.56
	57F	550.5	361.0	295.0	$\mathbf{1.5 \times 10^{-5}}$	$\underline{0.68}$	$\mathbf{9.1 \times 10^{-10}}$	**0.75**	$\mathbf{1.7 \times 10^{-2}}$	*0.60*
	61T	484.5	267.5	209.0	$\mathbf{3.5 \times 10^{-10}}$	**0.76**	$\mathbf{1.8 \times 10^{-12}}$	**0.79**	6.9×10^{-2}	*0.57*
	66T	544.0	287.0	288.0	$\mathbf{2.2 \times 10^{-5}}$	$\underline{0.67}$	$\mathbf{3.6 \times 10^{-6}}$	$\underline{0.69}$	7.7×10^{-1}	0.51
	68T	2455.0	1535.5	1307.5	$\mathbf{1.4 \times 10^{-3}}$	*0.63*	$\mathbf{5.4 \times 10^{-6}}$	$\underline{0.69}$	2.0×10^{-1}	0.55
	68F	833.0	422.5	423.0	$\mathbf{1.9 \times 10^{-5}}$	$\underline{0.68}$	$\mathbf{2.0 \times 10^{-5}}$	$\underline{0.67}$	7.3×10^{-1}	0.49
	70F	710.5	523.0	427.0	$\mathbf{2.8 \times 10^{-2}}$	*0.59*	$\mathbf{1.7 \times 10^{-4}}$	$\underline{0.65}$	9.2×10^{-2}	*0.57*
	74T	736.0	450.5	343.0	$\mathbf{9.0 \times 10^{-4}}$	*0.64*	$\mathbf{4.2 \times 10^{-6}}$	$\underline{0.69}$	9.6×10^{-2}	*0.57*
gimp_rgb_to_hsv_int	14T	31779.0	>100000.0	>100000.0	$\mathbf{2.2 \times 10^{-31}}$	**0.05**	$\mathbf{2.7 \times 10^{-34}}$	**0.04**	1.5×10^{-1}	0.48
	17T	36.0	34.5	33.0	5.1×10^{-1}	0.53	2.2×10^{-1}	0.55	3.4×10^{-1}	0.54
gimp_hsv_to_rgb_int	4T	21.5	23.0	19.0	8.5×10^{-1}	0.49	$\mathbf{3.2 \times 10^{-4}}$	$\underline{0.65}$	$\mathbf{6.4 \times 10^{-10}}$	**0.75**
	11T	21.0	22.0	16.0	4.8×10^{-1}	0.53	$\mathbf{1.3 \times 10^{-11}}$	**0.78**	$\mathbf{2.2 \times 10^{-17}}$	**0.85**
validate_card	5F	6.0	5.0	5.0	6.2×10^{-1}	0.52	5.9×10^{-1}	0.52	9.3×10^{-1}	0.50
	7T	6.5	6.0	6.0	1.7×10^{-1}	0.56	5.5×10^{-1}	0.52	4.8×10^{-1}	0.47
	7F	6.0	5.0	6.0	$\mathbf{2.7 \times 10^{-2}}$	*0.59*	4.9×10^{-1}	0.53	1.5×10^{-1}	0.44
	9T	6.5	5.5	5.5	$\mathbf{9.7 \times 10^{-3}}$	*0.61*	6.1×10^{-1}	0.52	8.1×10^{-2}	*0.43*
	9F	7.0	5.0	5.0	$\mathbf{2.7 \times 10^{-4}}$	$\underline{0.65}$	$\mathbf{4.0 \times 10^{-3}}$	*0.62*	4.5×10^{-1}	0.47
	11T	6.0	5.5	6.0	9.6×10^{-1}	0.50	8.7×10^{-1}	0.49	6.8×10^{-1}	0.48
	11F	5.0	6.0	6.0	4.2×10^{-1}	0.47	1.1×10^{-1}	*0.44*	5.5×10^{-1}	0.48
	14T	13.0	13.0	13.5	9.5×10^{-1}	0.50	2.7×10^{-1}	0.46	2.0×10^{-1}	0.45
	14F	7.0	6.0	6.0	1.5×10^{-1}	0.56	1.0×10^{0}	0.50	2.0×10^{-1}	0.45

latter with faster local searches, Geometric and Lattice Search. On strictly unimodal functions, these searches provably need less time than IPS. IPS requires time $\Theta((\log d)^2)$ while our new searches need time $\Theta(\log d)$, where d is the initial distance to the optimum. These theoretical results were confirmed with empirical experiments optimising the easy function $f(x) = |x|$.

We further empirically analysed Geometric and Lattice Search on test objects that gave rise to unimodal as well as multimodal functions. For multimodal functions there are no non-trivial performance guarantees for any local search; our experiments therefore extend the realm of what can be proven theoretically. Considering branches where any variant of AVM performed significantly better than random search, we found that both Geometric and Lattice performed better than IPS on a majority of branches. There was only one particular branch where IPS performed better (probably due to its better exploration). However, this branch was handled more efficiently by a Genetic Algorithm, as is generally the case for more complex landscapes. Local searches excel in simple conditions, and our paper instead concentrates on improving the AVM for these cases, which have been shown in the test input generation literature to be very common for procedural C programs [6].

With respect to applying the results of our paper, it is not clear what the fitness landscape looks like in advance of test input generation in practice. While further research is required to investigate this problem, our new local searches for the AVM may be used to further improve results with Memetic Algorithms (MAs) [5, 6], which combine diversifying GA searches with intensifying local search algorithms. Such an approach was found to provide the "best of both worlds" for test input generation in Harman and McMinn's study [6]. Thus, further work is needed to investigate the performance of our new local searches with the AVM when integrated into an MA.

Acknowledgement. This work is funded in part by the EPSRC project "RE-COST" (grant no. EP/I010386).

8. REFERENCES

[1] A. Arcuri. Full theoretical runtime analysis of alternating variable method on the triangle classification problem. In *SSBSE*, 2009.

[2] A. Arcuri, P. K. Lehre, and X. Yao. Theoretical runtime analyses of search algorithms on the test data generation for the triangle classification problem. In *SBST*, 2008.

[3] A. Auger and B. Doerr, editors. *Theory of Randomized Search Heuristics – Foundations and Recent Developments*. Number 1 in Series on Theoretical Computer Science. World Scientific, 2011.

[4] D. E. Ferguson. Fibonaccian searching. *Comm. of the ACM*, 3(12), 1960.

[5] G. Fraser, A. Arcuri, and P. McMinn. Test suite generation with memetic algorithms. In *GECCO*, 2013.

[6] M. Harman and P. McMinn. A theoretical and empirical study of search based testing: Local, global and hybrid search. *IEEE Trans. Soft. Eng.*, 36(2).

[7] J. Kiefer. Sequential minimax search for a maximum. *Amer. Math. Soc.*, 4, 1953.

[8] B. Korel. Automated software test data generation. *IEEE Trans. on Soft. Eng.*, 16(8), 1990.

[9] P. K. Lehre and X. Yao. Crossover can be constructive when computing unique input-output sequences. *Soft Computing*, 15, 2011.

[10] P. K. Lehre and X. Yao. Runtime analysis of the (1+1) EA on computing unique input output sequences. *Information Sciences*, 2013. (To appear).

[11] P. McMinn. An identification of program factors that impact crossover performance in evolutionary test input generation for the branch coverage of C programs. *Inf. and Soft. Tech.*, 55(1).

[12] P. McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2).

[13] P. McMinn. IGUANA: Input generation using automated novel algorithms. a plug and play research tool. Technical Report CS-07-14, Uni. Sheffield, 2007.

[14] L. L. Minku, D. Sudholt, and X. Yao. Evolutionary algorithms for the project scheduling problem: Runtime analysis and improved design. In *GECCO*, 2012.

[15] J. F. Monahan. *Numerical Methods of Statistics*. Cam. Univ. Press, 2nd edition, 2011.

[16] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer, 2010.

[17] A. Vargha and H. D. Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal on Educational and Behavioral Statistics*, 25(2), 2000.

[18] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Inf. and Soft. Tech.*, 43(14), 2001.