Fast Building Block Assembly by Majority Vote Crossover

Tobias Friedrich
Hasso Plattner Institute
Potsdam, Germany

Timo Kötzing
Hasso Plattner Institute
Potsdam, Germany

Martin S. Krejca
Hasso Plattner Institute
Potsdam, Germany

Samadhi Nallaperuma
Dept. of Computer Science
The University of Sheffield
Sheffield, United Kingdom

Frank Neumann
School of Computer Science
The University of Adelaide
Adelaide, Australia

Martin Schirneck
Hasso Plattner Institute
Potsdam, Germany

ABSTRACT

Different works have shown how crossover can help with building block assembly. Typically, crossover might get lucky to select good building blocks from each parent, but these lucky choices are usually rare. In this work we consider a crossover operator which works on three parent individuals. In each component, the offspring inherits the value present in the majority of the parents; thus, we call this crossover operator *majority vote*.

We show that, if good components are sufficiently prevalent in the individuals, majority vote creates an optimal individual with high probability. Furthermore, we show that this process can be amplified: as long as components are good independently and with probability at least $1/2+\delta$, we require only $\mathrm{O}\big(\log \delta^{-1} + \log\log n\big)$ successive stages of majority vote to create an optimal individual with high probability!

We show how this applies in two scenarios. The first scenario is the Jump test function. With sufficient diversity, we get an optimization time of $\mathrm{O}(n \log n)$ even for jump sizes as large as $\mathrm{O}(n^{1/2-\varepsilon})$. Our second scenario is a family of vertex cover instances. Majority vote optimizes this family efficiently, while local searches fail and only highly specialized two-parent crossovers are successful.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

Keywords

run time analysis, evolutionary algorithm, island model, majority vote crossover, Jump, vertex cover

1. INTRODUCTION

At the heart of many algorithms within the area of evolutionary computation is the crossover operator, granting the algorithm significant power beyond the hill-climbing abilities of most mutation operators. By now, there are several theoretical run time results that crossover can be beneficial for speeding up the optimization of certain test functions [8, 9, 11, 13, 14, 16, 18]. One key insight from these works is that crossover can be used to combine the specific strengths of different individuals. In all these examples *crossover* means two-parent crossover on bit string individuals. There are also rigorous proofs that crossover is beneficial for some problem-specific algorithms and representations, namely coloring problems inspired by the Ising model [17] and the all-pairs shortest path problem [2].

In this paper we will consider optimization of bit strings of fixed length n, but in contrast to most previous theoretical work, we turn our focus to crossover with three instead of two parents. Note that the combination of good components from the different parents in two-parent crossover is purely a matter of chance: since the crossover operator cannot know which of the parents excels in which part of the bit string, making the right choice is unlikely (in [8, 11] several run times contain a term 2^{2k}, where $2k$ is the number of binary choices that have to be made correctly by crossover). We consider a three-parent crossover which will be much better suited to make the right choice. The operator selects, for each bit position, the bit value which the majority of the parents has in this position; thus, we call this operator the *majority vote crossover*. It was considered in [3] (under the name of *occurrence-based scanning*), but only as a part of an empirical comparison between several multi-parent techniques. The majority vote crossover is deterministic and its efficiency hinges on the assumption that good bit values are more prevalent in the population than bad ones. Note that any (odd) arity of at least 3 could be chosen, with similar results as presented in this paper.

We analyze the efficiency of this crossover operator on two domains: the Jump function (as considered in [8, 11]) as a motivating example; and on the vertex cover problem (as considered in [4, 13]). In both cases, the algorithm will use an initial phase of hill climbing to find local optima. Once all individuals are stuck in (hopefully sufficiently different) local optima, the crossover operator can start assembling better solutions by majority vote. In fact, in Theorem 3.3 we use that, if three independent individuals are correct in any given position with a probability of $1-\mathrm{O}\big(1/n^{1/2-\varepsilon}\big)$, a single application of the majority vote crossover will immediately

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

GECCO '16 July 20-24, 2016, Denver, CO, USA

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4206-3/16/07.

DOI: http://dx.doi.org/10.1145/2908812.2908884

create the optimum! Note that the probability that out of three individuals two having a bad bit in the same position is strongly related to the Birthday Paradox. For the very same reason we see square roots in the bounds on the error probability.

In Section 3 we give our results on the Jump function. We consider three different ways of achieving diversity: low crossover probability as in [8, 11], explicitly maximizing the total Hamming distance, and single-receiver island models as in [13, 18]. For all three cases, we give upper bounds on the expected optimization time in dependence of the dimension size n, the jump size k, the number of individuals μ and the crossover probability p_c. In all cases, the best bounds are achieved for $\mu = 3$. In the case of low p_c, our analysis follows [8, 11] and similarly gets an optimization time of $O\big(n \log n + 2^{ck} n\big)$ for some constant c. For the other two cases, $p_c = 1$ is optimal. When explicitly maximizing the Hamming distance, we can benefit from a wide spread of the individuals and achieve an optimization time of $O(n \log n + nk \log k)$, even for k linear in n (as long as $k \leq n/8$). For the single receiver island model, we cannot hope that our individuals spread as well as with this explicit mechanism; this means that only $k = O(n^{1/2-\varepsilon})$ is possible (for any fixed ε). However, the islands are then successful within $O(n \log n)$ function evaluations.

In Section 4 we consider a family of instances for the vertex cover problem from [4, 13]. These instances are very hard to optimize when only relying on mutation, and they have been used to show the efficiency of crossover [13]. However, the crossover presented in [13] was tailored to the particular problem instances. In particular, it was not *unbiased* with respect to the encoding of the problem (see [12] for a discussion on unbiasedness), but rather expected the problem to be encoded in certain blocks. The crossover was then chosen such that it can combine these blocks. We show that majority vote crossover does not need any such assumptions on the encoding, but can optimize the instances efficiently, using a single receiver island model, where each island runs a simple hill climbing search heuristic (we consider Steepest Ascent and Random Local Search).

Finally, in Section 5 we consider probability amplification by trees of majority vote crossover. These trees are complete ternary trees of some height h. All leaves run a simple search heuristic; all interior nodes apply majority vote on the three children (without performing any kind of selection). We show that, if at each leaf and for each bit position the probability to have found the correct bit setting is at least $1/2+\delta$, we require only a height of $h = O\big(\log \delta^{-1} + \log \log n\big)$ successive stages of majority vote to create an optimal individual with high probability. This corresponds to a tree with $O\big(\delta^{-1} \log n\big)$ many nodes. This strategy can be used in both the setting of the Jump function as well as vertex cover to allow a significantly larger error in the local search heuristic.

2. SETTING AND PROBLEMS

In this paper we are concerned with optimizing pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$ over bit strings of fixed length n. We interpret the value $f(\boldsymbol{x})$ as the fitness of an individual $\boldsymbol{x}$, that is, the quality of the candidate solution $\boldsymbol{x}$ with respect to a given problem. One such function, Jump, was introduced by Jansen and Wegener [8] and is defined, for any positive integer $1 \leq k \leq n$, as

$$\mathrm{Jump}_k(\boldsymbol{x}) = \begin{cases} k + |\boldsymbol{x}|_1\,, & \text{if } |\boldsymbol{x}|_1 \leq n-k \text{ or } |\boldsymbol{x}|_1 = n; \\ n - |\boldsymbol{x}|_1\,, & \text{otherwise;} \end{cases}$$

where $|\boldsymbol{x}|_1 = \sum_{i=1}^{n} \boldsymbol{x}_i$ is the number of 1-bits in $\boldsymbol{x}$. Observe that, for the Jump function, the fitness of an individual depends only on the number of 1s. Any individual $\boldsymbol{x}$ with $|\boldsymbol{x}|_1 = n - k$ is locally optimal; we refer to this area of the search space as the *plateau*. However, the only global maximum is the all-ones string 1^n. Between the plateau and the global optimum is a gap of Hamming distance k, which has to be *jumped* over in order to optimize the function, thus the name.

Furthermore, we investigate the *minimum vertex cover* problem. Given a graph, minimum vertex cover is the problem of finding a set of vertices of minimum size, the cover, such that every edge is incident to at least one of the elements in the set. This optimization problem is well-known to be **NP**-hard [5, 10]. In this work we focus on certain instances of the vertex cover problem defined by Neumann et al. [13]. They consist of m copies of a complete bipartite graph where the two partitions have l and $l+k$ vertices, respectively. More precisely, we fix positive integers k, l, and m; for each i with $1 \leq i \leq m$, let $G_i = (V_i, E_i)$ be a graph as follows. Its set of vertices $V_i = V_i^{(1)} \dot\cup V_i^{(2)}$ is the disjoint union of two subsets with $|V_i^{(1)}| = l$ and $|V_i^{(2)}| = l + k$, $E_i = \{\{u, v\} \mid u \in V_i^{(1)} \wedge v \in V_i^{(2)}\}$ is its set of edges. The graph $G = \bigsqcup_{i=1}^{m} G_i$ is then defined as the disjoint union of the components G_i. We let $s = 2l + k$ denote the number of vertices in each component and $n = ms$ be the total number of vertices in the graph. The discrepancy k serves as a measure of the disparity of the two partitions. Obviously, for any i, a cover of component G_i must contain at least all vertices of $V_i^{(1)}$ or all of $V_i^{(2)}$, the only minimum vertex cover for the whole graph G is $\bigsqcup_{i=1}^{m} V_i^{(1)}$. For any component G_i, we refer to the cover consisting of *only* the vertices in $V_i^{(1)}$ as *globally optimal* and the cover consisting of only the vertices in $V_i^{(2)}$ as *locally optimal*.

Both problems exhibit the common pattern of strong local optima, which are hard to escape during the optimization. In order to tackle this issue, we investigate the performance of the *majority vote crossover operator* [3], Φ_{mv}. This operator, given three bit strings of the same length, outputs a string in which every position is the result of the majority vote of the respective bits in the input. More formally, for any three individuals $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \{0,1\}^n$ and on every position $1 \leq i \leq n$, the crossover is defined as

$$\Phi_{\mathrm{mv}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})_i = \begin{cases} \boldsymbol{x}_i\,, & \text{if } \boldsymbol{x}_i = \boldsymbol{y}_i\,; \\ \boldsymbol{z}_i\,, & \text{otherwise.} \end{cases}$$

The operator is symmetric in its arguments and the result deterministically depends on the input. We point out that the candidate solutions need to be drawn randomly *without replacement* as otherwise a duplicated individual would determine the outcome of the whole vote.

2.1 Algorithms

To achieve the diversity we need, we employ several mechanisms. The first one is a $(\mu + 1)$ GA [8] combined with the majority vote crossover operator, see Algorithm 1. This genetic algorithm maintains a population of μ individuals and,

in each iteration, chooses one of them uniformly at random and mutates it by flipping each bit independently with probability $1/n$. Afterward, if the fitness of the new individual is at least as good as the worst fitness of the population, it replaces one fitness-worst solution drawn uniformly at random. With probability p_c, a majority vote crossover among three (different) randomly chosen individuals occurs and the result is not mutated, as this would defeat the purpose of the majority vote crossover. Note that the maintenance of set $I_{\min}$, the set of indices of fitness-worst individuals, needs μ fitness evaluations during the initialization phase and subsequently comes with no extra cost as the evaluations in line 11 provide all necessary information. It will be clear from the results in Section 3 that this additional effort in the beginning is asymptotically negligible.

Algorithm 1: $(\mu+1)$ GA with majority vote crossover, $I_{\min}$ is the set of all indices of fitness-worst individuals; concept from [8]

```
1  for i ∈ {1,...,μ} in parallel do
2      x^(i) ← individual drawn u.a.r. from {0,1}^n;
3  repeat
4      with probability p_c do
5          (i,j,k) ← indices drawn u.a.r. from {1,...,μ};
6          y ← Φ_mv(x^(i), x^(j), x^(k));
7      else
8          i ← index drawn u.a.r. from {1,...,μ};
9          y ← flip each bit in x^(i) with probability 1/n;
10     l ← index drawn u.a.r. from I_min;
11     if f(y) ≥ f(x^(l)) then
12         x^(l) ← y;
13 until one individual is optimal;
```

For Theorem 3.2 we equip this algorithm with a different selection policy, namely, we discard the fitness-worst individual which has minimal Hamming distance to the rest of the population (see Section 3 for details).

As a third diversity mechanism, we use the *single receiver island model* as given by Watson and Jansen [18]. That is, we run μ independent instances, the islands, of a given iterative search heuristic A. Furthermore, in each iteration, we apply the majority vote crossover operator to three islands chosen uniformly at random and store the solution in the so-called *receiver island*. See Algorithm 2 for a detailed description.

We want to point out that throughout this paper we consider μ, i.e., the population size or the number of islands, respectively, to be bounded from above by some polynomial in the length n of the bit string.

The search heuristic A manipulates the currently best solution as a bit string, possibly in a randomized fashion. In this paper we compare as choices for A the method of *Steepest Ascent* [15] and *Random Local Search* (RLS) [7]. Steepest Ascent examines *all* 1-neighbors of the currently best solution and chooses one that maximizes the fitness increase, i.e., a neighbor with the best fitness. Random Local Search also flips exactly one bit of the given input string, but this time it is chosen uniformly at random among all possible bit positions.

Algorithm 2: Single receiver model with μ islands running A; concept from [18].

```
1  for i ∈ {1,...,μ} in parallel do
2      x^(i) ← individual drawn u.a.r. from {0,1}^n;
3  repeat
4      for i ∈ {1,...,μ} in parallel do
5          y^(i) ← A(x^(i));
6          if f(y^(i)) ≥ f(x^(i)) then
7              x^(i) ← y^(i);
8      (i,j,k) ← indices drawn u.a.r. from {1,...,μ};
9      x^(0) ← Φ_mv(x^(i), x^(j), x^(k));
10 until x^(0) is optimal;
```

3. THE JUMP FUNCTION

In this section we consider the Jump test function as a first example where the majority vote crossover leads to very good run times. Since we require sufficient diversity to be present in the population, we consider the following three sources of diversity.

First, we consider the $(\mu+1)$ GA (Algorithm 1) and set the crossover probability vanishingly low; this way, the population will make a random walk and gain diversity by random genetic drift. This approach was also taken by Jansen and Wegener [8] for the uniform crossover, and, in fact, our analysis follows closely the one in [8]. Our result is given in Theorem 3.1.

Second, we consider (again) the $(\mu+1)$ GA, this time with a tie-breaking rule in the selection operator which explicitly aims at increasing diversity. In particular, we consider breaking ties such that the total pairwise Hamming distance of the population is maximized. More formally, we let d_H denote the Hamming distance and measure the diversity of a population $P = \{x^{(1)}, \dots, x^{(\mu)}\}$ (considered as a multiset) as

$$D(P) = \sum_{x,y \in P} d_H(x,y).$$

Selection will choose for replacement a fitness-worst individual $x \in I_{\min}$, breaking ties by trying to maximize

$$D(P \setminus \{x\}) = D(P) - 2\sum_{y \in P} d_H(x,y).$$

Further ties are broken by uniform random choice. Our result regarding this mechanism is given in Theorem 3.2. We would like to point out that computing $D(P)$ takes time $O(\mu^2 n)$ per iteration. However, as usual in this setting of optimization, we only measure the computational effort of the algorithms presented in this paper w.r.t. the number of iterations and the number of fitness evaluations.

Finally, we consider a single receiver island model (Algorithm 2), which naturally exhibits significant diversity. Its islands each run an instance of the $(1+1)$ EA [7]. Note that this is a special case of the $(\mu+1)$ GA introduced in the last section with population size $\mu = 1$ and crossover probability $p_c = 0$, that is, it flips every bit of the input independently with probability $1/n$.

Next, we state the results regarding low crossover probabilities and using the tie-breaking procedure described above as an explicit diversity mechanism.

Theorem 3.1. *Let $\mu \geq 3$ and consider the $(\mu+1)$ GA with crossover probability $p_c = k/n$ optimizing Jump_k with $k = o(\sqrt{n})$. Then, in expectation, an optimal solution is sampled within $O\big(\mu n + n \log n + e^{7k}\mu^{2k+3}n\big)$ iterations.*

Theorem 3.2. *Let $k \leq n/8$ and $3 \leq \mu < n/(2k)$. Consider the $(\mu+1)$ GA with crossover probability p_c bounded away from 1 by a constant optimizing Jump_k. Suppose that ties in the selection phase are handled by maximizing the sum of all Hamming distances between any two individuals. Then, the expected number of iterations until an optimal solution is sampled is $O\big(\mu n + n \log n + \mu^2 nk \log(\mu k) + p_c^{-1}\big)$.*

Due to space constraints, the proofs of these two theorems are omitted. The proof of Theorem 3.1 is similar to the one of Theorem 7 of Kötzing et al. [11]. It considers a single run of the optimization process as a chain of conditional events. For each event we then bound the probability that its outcome is favorable. The proof of Theorem 3.2 is a standard drift argument.

We now consider the performance of island models.

Theorem 3.3. *Let $\mu \geq 3$ and consider the single receiver island model running μ instances of the $(1+1)$ EA optimizing Jump_k with $k = O\big(n^{\frac{1}{2}-\varepsilon}\big)$ where $\varepsilon > 0$ is some arbitrary constant. Then, the majority vote crossover operator Φ_{mv} samples an optimal individual on the receiver island within $O(n \log n)$ iterations w.h.p.[1] This corresponds to a total of $O(\mu n \log n)$ fitness evaluations.*

Proof. The first step requires that three islands, drawn uniformly at random, sample an individual on the plateau after $O(n \log n)$ iterations with high probability; this follows directly from Theorem 7 of Doerr and Goldberg [1] and a union bound.

In the second step we prove that, conditioning on the first step being successful, the crossover operator succeeds again with high probability. Jump_k considers only the number of 1s, the remaining k bits with value 0 in each individual on the plateau are uniformly distributed. We now bound the probability of the event $\mathcal{E}$ that the three individuals chosen in the first step share no 0. If this is the case, the majority vote crossover operator samples the optimum in a *single* step. We have

$$\Pr(\mathcal{E}) = \frac{\binom{n-k}{k}}{\binom{n}{k}} \frac{\binom{n-2k}{k}}{\binom{n}{k}}.$$

Note that, by the constraint imposed on k, we have $n \geq 3k$ for n sufficiently large. We observe

$$\Pr(\mathcal{E}) \geq \left(\frac{\binom{n-2k}{k}}{\binom{n}{k}}\right)^2 = \left(\frac{(n-2k)^{\underline{k}}}{n^{\underline{k}}}\right)^2 \geq \left(\frac{(n-3k)^k}{n^k}\right)^2$$
$$= \left(1 - \frac{3k}{n}\right)^{2k} \geq 1 - \frac{6k^2}{n}$$

by Bernoulli's inequality. Here, $n^{\underline{k}} = \prod_{i=1}^{k}(n+1-i)$ denotes the falling factorial. Due to $k = O\big(n^{\frac{1}{2}-\varepsilon}\big)$, event $\mathcal{E}$ occurs with high probability. Combining the two bounds implies the theorem. □

[1]We use the term *with high probability* for a success probability of at least $1 - n^{-c}$ for some constant $c > 0$.

4. VERTEX COVER

We now turn the discussion to the minimum vertex cover problem. Recall that we investigate instances G consisting of m separate components, each of which is a complete bipartite graph with partitions of size l and $l+k$, respectively. In total, G has $n = ms$ vertices, $s = 2l + k$ per component.

Candidate solutions for the vertex cover problem can be represented as bit strings of length n. Every position corresponds to a specific vertex of the graph, a 0 denotes that this vertex is not chosen for the cover, and a 1 denotes that the vertex is chosen. We use the same fitness function as in [4], i.e., for every individual $\boldsymbol{x} \in \{0,1\}^n$, its fitness is defined as $f(\boldsymbol{x}) = (n+1)\,u(\boldsymbol{x}) + |\boldsymbol{x}|_1$, where $u(\boldsymbol{x})$ denotes the number of uncovered edges in $\boldsymbol{x}$. Thus, in this section we are concerned with minimizing a given function. The choice of the fitness function has two immediate consequences: first, any elitist algorithm optimizing f prioritizes reducing the number of uncovered edges over reducing the number of chosen vertices due to the large penalty of $n+1$ per uncovered edge. Second, every mutation operator manipulating exactly one bit in $\boldsymbol{x}$ alters $|\boldsymbol{x}|_1$ and, hence, the fitness value. Note that in this representation the globally and locally optimal states of a component have Hamming distance $s > 1$.

We consider Steepest Ascent and Random Local Search in turn. For both algorithms, we show that they find a locally optimal state of G where each component is correct with sufficiently high probability; then we conclude that, for the single receiver island model, the majority vote crossover Φ_{mv} constructs the global optimum with high probability at the receiver island. Comparing these two approaches illustrates necessary trade-offs between an (almost) deterministic search and a random one in terms of run time and chance of success.

4.1 Steepest Ascent

Lemma 4.1. *Consider Steepest Ascent running on G. As long as not all edges of G are covered by solution $\boldsymbol{x}^{(i)}$, Steepest Ascent chooses the vertex incident to the most uncovered edges. In particular, in any component, it always chooses a vertex from the partition with fewer unchosen vertices. When all edges are covered, Steepest Ascent, again in every component, discards the vertices from the partition which was not chosen completely. Once all components are either locally or globally optimal, the currently best solution remains unchanged.*

Proof. As mentioned earlier, Steepest Ascent always prefers decreasing the number of uncovered edges. Only afterward the number of vertices in the cover is reduced since every bit flip not changing $u(\boldsymbol{x}^{(i)})$ contributes at most 1 to the total fitness. By construction, the number of uncovered edges incident to a vertex is exactly the number of unchosen vertices in the *other* partition of the same component. This means choosing a vertex from the partition with fewer unchosen vertices covers more edges. Once a cover has been found, i.e., in every component one of the two partitions has been chosen completely, all and only the chosen vertices from the other partition are expendable; other bit flips would increase (worsen) the fitness by 1 or even uncover a previously covered edge. If all components are either locally or globally optimal and there is at least one only locally optimal, this optimum could only be escaped if one switches the partitions in a locally optimal component. This is impossible

using only one-bit flips as the corresponding solutions have Hamming distance s. Note that along the way all ties are broken uniformly at random. □

As a consequence of Lemma 4.1, the method of steepest ascent deterministically finds the minimum vertex cover of G when initialized with the empty subgraph, 0^n. During the search, it flips exactly the bits corresponding to $\bigsqcup_{i=1}^{m} V_i^{(1)}$, hence, finding the optimal solution in time polynomial in n. This supports the assessment in [4] that greedy algorithms perform well on these instances. Even in our probabilistic setting the behavior described above hugely benefits the expected optimization time and the success probability as shown in the next theorem.

Theorem 4.2. *Let $\mu \geq 3$ and consider the single receiver island model running Steepest Ascent optimizing the vertex cover instance G with $k \geq (1+\varepsilon)\ln m + 2l$, where $\varepsilon > 0$ is an arbitrary constant. Then, an optimal solution is sampled within $n+1$ iterations w.h.p., using a total of $\mathrm{O}(\mu n^2)$ fitness evaluations.*

In order to establish this result, we want to use a similar argument as in the proof of Theorem 3.3. Thus, we show that each of the m components of G is either locally or globally optimal after n iterations and, for some suitable positive constant c, at most $\mathrm{O}\big(m^{\frac{1}{2}-c}\big)$ of them are only locally optimal. The last step is then the same as for optimizing the Jump function on bit strings of length m.

Proof. A single run of the hill-climbing process described in Lemma 4.1 is finished in at most n iterations as every bit is flipped at most once. We note that in every step each of the μ islands has to evaluate the fitness of all n neighbors of the currently best solution. After this period, every component of each solution is, deterministically, either globally or locally optimal. One can think of such a solution as bit strings of length m. Every bit represents a component with a 0 standing for a locally optimal component and a 1 standing for a globally optimal one. We have shown in Theorem 3.3 that with high probability Φ_{mv} creates an optimal solution in one step if there are, in each string, at most $\mathrm{O}\big(m^{\frac{1}{2}-c}\big)$ 0s. Furthermore, the outcome of the majority vote of the respective component equals the one for any vertex within this component. Hence, if the above requirement is met, the theorem follows.

We bound the probability of a component being (only) locally optimal after n iterations. The deterministic behavior of Steepest Ascent, see Lemma 4.1, reduces the question of whether the smaller or the larger partition is discovered first to the question which partition has fewer unchosen nodes after the initialization phase in the beginning. We pessimistically assume that, if both partitions have the same score, the component is going to be locally optimal. During the initialization, each vertex is chosen with probability $p = \frac{1}{2}$ independently of the other choices, thus, the number of chosen vertices per partition is binomially distributed. For any natural number a, let $X_a \sim \mathrm{Binom}(a,p)$ be the random variable denoting how many vertices out of a have initially been chosen for the cover. The number of unchosen vertices in $V_i^{(1)}$ therefore equals $l - X_l$ and the number for $V_i^{(2)}$ equals $l + k - X_{l+k}$. The component becomes locally optimal just in case the former value is not smaller than the latter. Observe that due to symmetry, for any two natural numbers a and b, the random variables $(a - X_a)$ and X_a are identically distributed, the same holds for $(X_a + X_b)$ and X_{a+b}. Hence, we get the following equation.

$$\begin{aligned}\Pr(l - X_l \geq l + k - X_{l+k}) &= \Pr(X_l \geq l + k - X_{l+k}) \\ &= \Pr(X_l + X_{l+k} \geq l + k) = \Pr(X_s \geq l + k).\end{aligned}$$

As the success probability p equals $1/2$, we can employ a special form of the Chernoff-Hoeffding Theorem [6] to bound the last probability.

$$\Pr\left(X_s \geq \overbrace{l + \frac{k}{2}}^{=sp} + \frac{k}{2}\right) \leq \mathrm{e}^{-\frac{\left(\frac{k}{2}\right)^2}{2sp(1-p)}} = \mathrm{e}^{-\frac{k^2}{2s}}.$$

We now require that there are in each solution asymptotically less than $\sqrt{m}$ locally optimal components w.h.p. To that end, for any island $1 \leq i \leq \mu$, we define a random variable $Z_{m,i} \sim \mathrm{Binom}\big(m, \exp(-k^2/(4l+2k))\big)$, denoting the number of locally optimal components in $\boldsymbol{x}^{(i)}$. Regarding their expected value, we have

$$\mathrm{E}(Z_{m,i}) = m\,\mathrm{e}^{-\frac{k^2}{4l+2k}} \leq m\,\mathrm{e}^{-\frac{k-2l}{2}} \leq m^{\frac{1-\varepsilon}{2}}.$$

The last inequality is due to $k \geq (1+\varepsilon)\ln m + 2l$. Using Chernoff and union bounds, one verifies that the probability that *none* of the $Z_{m,i}$-s, for any polynomial number of islands, exceeds $m^{\frac{1-\varepsilon}{2}}$ by more than a constant factor is at least $1 - \exp\Big(-\Omega\Big(m^{\frac{1-\varepsilon}{2}}\Big)\Big)$. □

Intuitively speaking, as long as the sizes of the two partitions are not too similar, Steepest Ascent only needs a linear number of iterations but the number of fitness evaluations on each island is quadratic – it spends most of its computational resources in determining the one bit position that maximizes the fitness decrease.

4.2 Random Local Search

In the following we investigate how randomization of the search can help reduce the expected number of fitness evaluations. For this we use the paradigm of Random Local Search, that is, in every step and on each island we flip exactly one bit chosen uniformly at random. Utilizing RLS to optimize vertex cover instances has already been extensively studied in [4]. Namely, it has been shown that RLS covers a single component G_i in expected time $\mathrm{O}(s \log s)$ and the probability of finding the larger partition equals the relative share of the smaller one. The next lemma is a reformulation of these results for our setting. We add the assertion that their run time bound holds not only in expectation but even with high probability.

Lemma 4.3. *Consider RLS running on one island optimizing a single component G_i. Then, after $\mathrm{O}(s \log s)$ iterations it is either globally or locally optimal w.h.p. The probability of G_i being only locally optimal equals l/s.*

Proof. We are only concerned with bounding the probability that G_i is at least locally optimal after $\mathrm{O}(s \log s)$ iterations. The other statements immediately follow from Theorem 1 and Theorem 4 of Friedrich et al. [4].

RLS behaves similarly to Steepest Ascent as described in Lemma 4.1. While not all edges are covered, no bit flips discarding vertices are accepted; on the other hand, every bit flip covering a previously uncovered edge is accepted.

Hence, trying every possible bit flip at least once during this phase is sufficient to find a (not necessarily optimal) vertex cover. A standard argument now shows that this happens with high probability within $O(s \log s)$ iterations: the probability of flipping a fixed bit in one iteration is $1/s$. Hence, the probability of flipping this bit at least once during $(c+1)s \ln s$ iterations, $c > 0$ a constant, equals

$$1 - \left(1 - \frac{1}{s}\right)^{(c+1)s \ln s} \geq 1 - \mathrm{e}^{-(c+1)\ln s} = 1 - s^{-(c+1)}.$$

A union bound over all s bit positions now yields the desired result.

An identical argument shows that, once a cover has been obtained, another phase of $O(s \log s)$ steps suffices to remove all expendable vertices, again w.h.p. For this, note that no new vertices are added during the second phase. These two bounds combined imply that RLS is successful in time $O(s \log s)$ with high probability. □

Theorem 4.4. *Let $\mu \geq 3$ and consider the single receiver island model running RLS optimizing the vertex cover instance G with $k = \Omega\big(l\, m^{\frac{1}{2}+\varepsilon}\big)$, where ε, again, is some arbitrary positive constant. Then, an optimal solution is sampled within $O(n \log n)$ iterations w.h.p., using a total of $O(\mu n \log n)$ fitness evaluations.*

Proof. The concentration result in Lemma 4.3 is not adversely affected by considering all m components after a phase of $O(n \log n)$ iterations. Moreover, a union bound shows that it also holds for three islands chosen uniformly at random. That is, after so many steps all components of the respective solutions are at least locally optimal w.h.p. During this first phase any island uses only a constant number of fitness evaluations per iteration.

For the second phase – the application of the majority vote crossover – to be successful, we require that at most $O\big(m^{\frac{1}{2}-\varepsilon}\big)$ components per island are locally optimal. For each island i with $1 \leq i \leq \mu$, let $Z_{m,i}$ be the random variable denoting the number of locally optimal components on i. Once again, due to Lemma 4.3, the $Z_{m,i}$-s are i.i.d. according to the binomial distribution with parameter l/s. We have, for any i,

$$\mathrm{E}(Z_{m,i}) = m\,\frac{l}{2l+k} = O\Big(m^{\frac{1}{2}-\varepsilon}\Big).$$

The asymptotic estimate is due to k being in the order of $\Omega\big(l m^{\frac{1}{2}+\varepsilon}\big)$. The reasoning why none of the random variables, regarding the islands used in the process, overshoot their expected value by more than a constant factor w.h.p. is the same as in the proof of Theorem 4.2. □

5. PROBABILITY AMPLIFICATION

In the remainder of this work, we introduce a hierarchical architecture in which the majority vote crossover facilitates a rapid amplification of the success probability of a given randomized search heuristic. The setting is motivated by the algorithms we examined in the previous sections, but is not restricted to them. Consequently, we will present this method in more general terms.

We consider a complete ternary tree, called the *mv-tree*, in which every inner node computes the majority vote crossover of its three children; each leaf is running some randomized search heuristic A. The algorithm A does not need to optimize the string entirely. However, there are two essential requirements in order to let the mv-tree amplify the success:

1. For every bit position in the result of A, the probability of this bit being set correctly is strictly larger than $1/2$.
2. The bits which are incorrectly set are uniformly distributed among all n bit positions.

Under these requirements we show that the probability of a bit position being set correctly increases quickly as the result of the crossover is propagated upward in the tree. The question naturally arises how many layers are needed until this probability exceeds a given threshold. As it turns out, the chance of success proliferates extensively under majority vote. In the rest of this section, we prove the following theorem, which is the main result.

Theorem 5.1. *Let $\delta > 0$, possibly dependent on n, be such that the probability for any bit position in the result of algorithm A being set correctly is at least $\frac{1}{2} + \delta$. Then, for any fixed $c > 0$, an mv-tree of height $\Theta\big(\log \delta^{-1} + \log\log n\big)$ suffices to amplify this probability to at least $1 - n^{-c}$.*

As this allows for a cleaner analysis, we focus on minimizing the probability of failure instead of directly quantifying the amplification of the success. By definition, the result of the majority vote crossover deterministically depends on the input strings, hence, we observe the following behavior in each inner node of the tree. Let p denote (an upper bound on) the probability that a bit position in any of the three children is erroneously set. Then the probability that this defect still occurs in the result of the majority vote is

$$3p^2(1-p) + p^3 = 3p^2 - 2p^3.$$

We therefore define the error decreasing function as $\mathrm{dec}(p) = 3p^2 - 2p^3$ for all $p \in [0,1]$. Function dec is monotonically increasing on the unit interval and has fixed points in $\frac{1}{2}$, 0 and 1, of which the latter two are attractive. More precisely, for any $p \in (0, \frac{1}{2})$, we have $\mathrm{dec}(p) < p$; thus, the error probability indeed approaches zero. We establish bounds on the decay of failure under iterated applications of dec, i.e., we bound the mapping $t \mapsto \mathrm{dec}^t(p)$. This task is split into two major parts corresponding to two ranges of p. First, in Lemma 5.2, we investigate initial values which are close to the fixed point $\frac{1}{2}$, namely, $\frac{1}{4} < p < \frac{1}{2}$. The analysis of the second range, $0 < p \leq \frac{1}{4}$, can be found in Lemma 5.3. As we will see, the underlying process is fundamentally different in behavior on these two domains.

As δ denotes the quantity by which the initial success probability exceeds one half, we can denote the error probability by $p = \frac{1}{2} - \delta$. In the following lemma, we only consider values of δ lying strictly between 0 and $\frac{1}{4}$. We introduce some additional technical notation. For all considered δ and natural numbers t, let g_δ be an auxiliary function defined as

$$g_\delta(t) = \frac{1}{2} - \mathrm{dec}^t\left(\frac{1}{2} - \delta\right).$$

Intuitively, g_δ measures how fast the error probability moves away from the fixed point $\frac{1}{2}$ if its initial value was $\frac{1}{2} - \delta$. Note that, for any parameter δ, g_δ is monotonically increasing in t and we have $g_\delta(0) = \delta$. The relevance of this function stems from the fact that in the moment in which $g_\delta(t)$ exceeds $\frac{1}{4}$, $\mathrm{dec}^t\big(\frac{1}{2} - \delta\big)$ falls below this value.

Lemma 5.2. *For any parameter $0 < \delta < \frac{1}{4}$, we define the crossing point as $t^*_\delta = \min\{t \in \mathbb{N} \mid g_\delta(t) \geq \frac{1}{4}\}$. Then, for all $t \leq t^*_\delta$, we have $(11/8)^t\delta \leq g_\delta(t) \leq (3/2)^t\delta$.*

Proof. Function g_δ observes the following recurrence for any natural number t:

$$\begin{aligned} g_\delta(t+1) &= \frac{1}{2} - \operatorname{dec}\left(\operatorname{dec}^t\left(\frac{1}{2} - \delta\right)\right) = \frac{1}{2} - \operatorname{dec}\left(\frac{1}{2} - g_\delta(t)\right) \\ &= \frac{1}{2} - 3\left(\frac{1}{2} - g_\delta(t)\right)^2 + 2\left(\frac{1}{2} - g_\delta(t)\right)^3 \\ &= \frac{3}{2}g_\delta(t) - 2\big(g_\delta(t)\big)^3. \end{aligned}$$

It is now easy to see that $g_\delta(t) \leq \big(\frac{3}{2}\big)^t\delta$ indeed holds.

The more interesting part is bounding $g_\delta(t)$ from below. This is established by an induction over all $t \leq t^*_\delta$. Note that $0 < t^*_\delta$, as $g_\delta(0) = \delta < \frac{1}{4}$, and the inequality $g_\delta(0) \geq \big(\frac{11}{8}\big)^0\delta$ obviously holds. Now assume that the induction hypothesis is true for some $t < t^*_\delta$. In this case, we get

$$\begin{aligned} g_\delta(t+1) &= \left(\frac{3}{2} - 2\big(g_\delta(t)\big)^2\right)g_\delta(t) \\ &> \left(\frac{3}{2} - 2\left(\frac{1}{4}\right)^2\right)\left(\frac{11}{8}\right)^t\delta = \left(\frac{11}{8}\right)^{t+1}\delta. \end{aligned}$$ □

Next, we turn to values of p between zero and one quarter.

Lemma 5.3. *For any natural number t and $0 < p \leq \frac{1}{4}$, we have $(5/8p)^{2^t} \leq \operatorname{dec}^t(p) \leq (3/4)^{2^t}$.*

Proof. Both inequalities are shown by an induction over t. The lower bound clearly holds for $\operatorname{dec}^0(p) = p \leq \frac{1}{4}$. Now assume that it is true for some natural number t. We get

$$\begin{aligned} \operatorname{dec}^{t+1}(p) &= \operatorname{dec}\big(\operatorname{dec}^t(p)\big) \geq \operatorname{dec}\left(\left(\frac{5}{8}p\right)^{2^t}\right) \\ &= \left(3 - \underbrace{2\left(\frac{5}{8}p\right)^{2^t}}_{\leq 2}\right)\left(\left(\frac{5}{8}p\right)^{2^t}\right)^2 \geq \left(\frac{5}{8}p\right)^{2^{t+1}}. \end{aligned}$$

The first inequality uses both the induction hypothesis and the monotonicity of function dec.

The upper bound follows by a similar but somewhat simpler argument using that $\big((3/4)^{2^t}\big)^3$ is always smaller than $(3/4)^{2^{t+1}}$. □

In other words, if the process of error minimization is started with an initial value close to $\frac{1}{2}$, it takes relatively long time to get the error under $\frac{1}{4}$ as the gap only grows exponentially. However, once this crossing point has been passed, the process is massively accelerated, the error probability now tends *double* exponentially toward zero.

Proof of Theorem 5.1. Naturally, this proof is also done in two steps. In the first one, we give matching upper and lower bounds on the time in which the initial error shifts from $\frac{1}{2} - \delta$ to $\frac{1}{4}$, i.e., on the crossing point t^*_δ as defined in Lemma 5.2. In the second step, we do the same for the time the mv-tree needs to further decrease the error probability below n^{-c}, for some prescribed constant $c > 0$. If δ is a constant or the initial error is already below one quarter, the first step is asymptotically irrelevant.

Let $p = \frac{1}{2} - \delta$ be the probability that a bit in the result of algorithm A – the leaves of the mv-tree – is set incorrectly. As we have seen above, $t^*_\delta = \min\{t \mid g_\delta(t) \geq \frac{1}{4}\}$ is the smallest number t such that after t layers the chance of failure is at most 1/4. By Lemma 5.2 we have $\big(\frac{11}{8}\big)^t\delta \leq g_\delta(t)$, therefore,

$$g_\delta(t) < \frac{1}{4} \;\Rightarrow\; t < \log_{\frac{11}{8}} \frac{1}{4\delta}.$$

Thus, choosing $t^*_\delta = \mathrm{O}\big(\log \delta^{-1}\big)$ suffices. A similar calculation, using $\big(\frac{3}{2}\big)^t\delta \geq g_\delta(t)$, verifies the matching lower bound of $t^*_\delta = \Omega\big(\log \delta^{-1}\big)$.

For the second step, we can assume that p is a constant not greater than $\frac{1}{4}$ and, hence, apply Lemma 5.3. That is, we get $\operatorname{dec}^t(p) = \Theta\big(n^{-c}\big)$ for some t with $t = \Theta(\log \log n)$. In total we need $\Theta\big(\log \delta^{-1} + \log \log n\big)$ layers for the optimization. □

To conclude our work, we revisit the optimization processes of the previous sections, in particular, our vertex cover instances. We already know that, after a sufficient number of steps, RLS manages to either locally or globally optimize each component of graph G individually. Furthermore, the probability of a component being locally optimal is l/s. We now show that the optimization by the mv-tree set on top of RLS comes with (asymptotically) no extra cost and is even capable of handling an arbitrarily small discrepancy between the two partitions.

Theorem 5.4. *Consider an mv-tree of height $\mathrm{O}(\log s + \log \log n)$, each of its leaves running an instance of RLS on a vertex cover instance G with $k > 0$. Then, an optimal solution arrives at the root within $\mathrm{O}(n \log n)$ iterations w.h.p., using a total of $\mathrm{O}\big(ms^2(\log ms)^2\big)$ fitness evaluations.*

Proof. We want to employ the above theorem for this proof, however, the result of RLS working on graph G does not meet an important requirement: as one bit in a component is set erroneously, just in case *all* of them are, given that all components are at least locally optimal, the incorrectly set bits are not uniformly distributed. We circumvent this obstacle by again analyzing the time it takes to optimize a string of length m in which every bit position corresponds to a whole component instead of a single vertex. This method is justified by the observation that, according to Theorem 4.4, after $\mathrm{O}(n \log n)$ iterations each component of any solution on a polynomial number of islands (leaves) is either locally or globally optimal w.h.p. In this case the majority vote of the components determines the outcome of the respective votes of the vertices involved. Observe that now the remaining defects in the bit string are indeed uniformly distributed as the components are optimized independently.

Consider a single individual after being treated by RLS for some $\mathrm{O}(n \log n)$ rounds. Lemma 4.3 states that the probability of a component being globally optimal is

$$\frac{l+k}{s} = \frac{1}{2} + \frac{k}{2s} = \frac{1}{2} + \Omega\left(\frac{1}{s}\right).$$

This equals the probability of a bit being set correctly. Thus, even for arbitrarily small discrepancies $k \geq 1$, we have $\delta^{-1} = \mathrm{O}(s)$ in the terms of Theorem 5.1. As a consequence, an

mv-tree of height $O(\log s + \log\log n)$ suffices to amplify the success probability in every position beyond $1 - n^{-c}$ for some constant $c > 0$, this is clearly enough to optimize the whole underlying graph w.h.p. The additional iterations needed to propagate the solution from the leaves to the root is asymptotically negligible compared to the optimization time of $O(n \log n)$. In fact, an analysis of the inequalities in Lemma 5.2 and Lemma 5.3 not neglecting constant factors even shows that we can choose the tree to have height at most $\log_3 s + \log_3 \log_3 n$. Hence, the mv-tree has $\mu = O(s \log n)$ leaves. As stated in Theorem 4.4, the optimization process needs at most $O(\mu n \log n)$ fitness evaluations, which is $O\big(ms^2(\log ms)^2\big)$. □

Applying the same reasoning directly on the level of the underlying bit strings also shows that an mv-tree can optimize Jump_k for any reasonable jump distance with no extra cost (compared to Theorem 3.3).

Theorem 5.5. *Consider an mv-tree of height* $O(\log n)$*, each of its leaves running an instance of the* $(1+1)$ *EA on* Jump_k *with* $k < \frac{n}{2}$*. Then, an optimal solution arrives at the root within* $O(n \log n)$ *iterations w.h.p., using* $O\big(n^2 \log n\big)$ *fitness evaluations.*

6. CONCLUSION

In this work we explored the influence of majority vote crossover in population-based optimization. We equipped several Genetic Algorithms, employing different crossover probabilities and diversity mechanisms, with this operator and then examined their run times optimizing the Jump function and structured vertex cover instances. If the fitness function observes multiple local maxima that need to be combined to find the global optimum, majority vote can significantly speed up the optimization. In these cases, this crossover operator naturally benefits from a high diversity of the underlying population.

From these observations, we derived the hierarchical optimization scheme of mv-trees, in which successive majority votes assemble building blocks, incorporating more and more partial solutions. We showed that in this setting even a small bias towards the optimum is amplified rapidly.

There are two immediate directions for future work. Majority vote crossover appears to work best on individuals drawn randomly from a population with high diversity as this diminishes the influence of stochastic error. Hence, it would be interesting to rigorously investigate the robustness of this operator against noise in the input or in the fitness function itself. Finally, we believe that mv-trees have large potential for applications, since they can be used with an arbitrary given optimization algorithm.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 618091 (SAGE) and from the Australian Research Council under grant agreement DP140103400.

References

[1] B. Doerr and L. Goldberg. Adaptive Drift Analysis. *Algorithmica*, 65:224–250, 2013.

[2] B. Doerr, E. Happ, and C. Klein. Crossover Can Provably be Useful in Evolutionary Computation. *Theor. Comp. Sci.*, 425:17–33, 2012.

[3] A. Eiben, P.-E. Raué, and Z. Ruttkay. Genetic algorithms with multi-parent recombination. In *Proc. of PPSN '94*, pp. 78–87, 1994.

[4] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Approximating Covering Problems by Randomized Search Heuristics using Multi-Objective Models. *Evol. Comp.*, 18:617–633, 2010.

[5] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York City, NY, 1979.

[6] W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Jour. of the ASA*, 58: 13–30, 1963.

[7] T. Jansen. *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. Springer, Berlin, 2013.

[8] T. Jansen and I. Wegener. The Analysis of Evolutionary Algorithms–A Proof That Crossover Really Can Help. *Algorithmica*, 34:47–66, 2002.

[9] T. Jansen and I. Wegener. Real royal road functions—where crossover provably is essential. *Discrete Appl. Math.*, 149:111–125, 2005.

[10] R. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pp. 85–103. 1972.

[11] T. Kötzing, D. Sudholt, and M. Theile. How Crossover Helps in Pseudo-Boolean Optimization. In *Proc. of GECCO '11*, pp. 989–996, 2011.

[12] P. K. Lehre and C. Witt. Black Box Search by Unbiased Variation. In *Proc. of GECCO '10*, pp. 1441–1448, 2010.

[13] F. Neumann, P. Oliveto, G. Rudolph, and D. Sudholt. On the Effectiveness of Crossover for Migration in Parallel Evolutionary Algorithms. In *Proc. of GECCO '11*, pp. 1587–1594, 2011.

[14] C. Qian, Y. Yu, and Z.-H. Zhou. An analysis on recombination in multi-objective evolutionary optimization. *Artificial Intelligence*, 204:99–119, 2013.

[15] J. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Applied Optimization. Springer, New York City, NY, 2005.

[16] T. Storch and I. Wegener. Real royal road functions for constant population size. *Theor. Comp. Sci.*, 320: 123–134, 2004.

[17] D. Sudholt. Crossover is Provably Essential for the Ising Model on Trees. In *Proc. of GECCO '05*, pp. 1161–1167, 2005.

[18] R. Watson and T. Jansen. A Building-Block Royal Road Where Crossover is Provably Essential. In *Proc. of GECCO '07*, pp. 1452–1459, 2007.