Crossover Can Provably be Useful in Evolutionary Computation*

Benjamin Doerr
Max-Planck-Institut für Informatik
Campus E 1 4
66123 Saarbrücken, Germany

Edda Happ
Max-Planck-Institut für Informatik
Campus E 1 4
66123 Saarbrücken, Germany

Christian Klein
Max-Planck-Institut für Informatik
Campus E 1 4
66123 Saarbrücken, Germany

ABSTRACT

We show that the natural evolutionary algorithm for the all-pairs shortest path problem is significantly faster with a crossover operator than without. This is the first theoretical analysis proving the usefulness of crossover for a non-artificial problem.

Categories and Subject Descriptors: F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity.

General Terms: Theory, Algorithms.

Keywords: Evolutionary Computation, Crossover, Analysis, Combinatorial Optimization.

1. INTRODUCTION

The paradigm of evolutionary computation is to use principles inspired by nature, e.g., mutation, crossover and selection, to build algorithms. Evolutionary algorithms (EA) and genetic algorithms (GA) are two prominent examples. Together with related approaches like randomized local search (RLS), the Metropolis algorithm [19], and simulated annealing [15] they all belong to a class of algorithms known as randomized search heuristics.

Whereas early hopes that these ideas might make notoriously hard problems become tractable did not fulfill, randomized search heuristics nowadays are frequently used as a generic way to obtain algorithms. Naturally, such generic approaches cannot compete with a custom-tailored algorithm. Practitioners still like to use them, because they are easy and cheap to implement, need fewer analysis of the problem to be solved, and can be reused easily for related problems.

While most research on evolutionary computation is experimental, the last ten years produced a growing interest in a theory-founded understanding of these algorithms. However, contrary to the very positive experimental results, theoretical insight seems much harder to obtain. One of the most fundamental questions still not answered in a satisfying way is whether crossover, that is, generating a new solution from two parents, is really useful. So far, apart from a few artificial examples, no problem is known where an evolutionary algorithm using crossover and mutation is superior to one that only uses mutation.

We answer this question positively. We show that for the classical all-pairs shortest path problem on graphs, the natural evolutionary algorithm using mutation has a worst-case optimization time of $\Theta(n^4)$ with high probability. However, if we add crossover to this algorithm, its expected optimization time is $O(n^{3.5+\varepsilon})$.

Clearly, both variants cannot compete with the classical algorithms for this problem, and also, the $\Omega(n^{\frac{1}{2}-\varepsilon})$ improvement is not ground-shaking. However, this is the first non-artificial answer to a problem discussed in the literature since the existence of evolutionary computation.

1.1 Evolutionary Computation

Evolutionary computation is algorithmics inspired by the evolution principle in nature. Typically, we have a set ("population") of solution candidates ("individuals"), which we try to gradually improve. Improvements may be generated by applying different variation operators, most notably mutation and crossover, to certain individuals. The quality of solutions is measured by a so-called fitness function. Based on their fitness value, a selection procedure removes some individuals from the population. The cycle of variation and selection is repeated until a solution of sufficient fitness is found. See, e.g., [7] for a short introduction to genetic algorithms.

One strength of this general approach is that each component can be adapted to the particular problem under consideration. This adaptation can be guided by an experimental evaluation of the actual behavior of the algorithms or by previously obtained experience. Also, not all evolutionary algorithms need to have all components describes above. For

*This work greatly benefited from various discussions at the Dagstuhl seminar 08051 on Theory of Evolutionary Algorithms.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

example, the simple variants of randomized local search and the (1+1) evolutionary algorithm have a population size of only one, and consequently, no crossover operator.

Using such evolutionary approaches has proven to be extremely successful in practice (see, e.g., the Proceedings of the annual ACM Genetic and Evolutionary Computation Conferences (GECCO)). In contrast to this, a theoretical understanding of such methods is still in its infancy. One reason for this is that genetic algorithms can be seen as non-linear (namely quadratic) dynamical systems, which are inherently more powerful, but "usually impossible to analyze" (c.f. [21]).

Nevertheless, the recent years produced some very nice theoretical results, mostly on convergence phenomena and runtime analyses. Since we will present a runtime analysis, we point the reader interested in some convergence results to [21–23]. Another interesting theoretical result is Wegener's [28] solution to the "outstanding open problem" [13] on whether simulated annealing is superior to the Metropolis algorithm.

1.2 Need for Crossover?

The paradigm of nature-inspired computing suggests to use both a mutation operator and a crossover operator. Mutation means that a new individual is generated by slightly altering a single parent individual, whereas the crossover operator generates a new individual by recombining information from two parents. Most evolutionary algorithms used in practice use both a mutation and a crossover operator.

In contrast to this, there is little evidence for the need of crossover. In fact, early work in this direction suggests the opposite. In [20], Mitchell, Holland and Forrest experimentally compared the run-time of a simple genetic algorithm (using crossover) and several hill-climbing heuristics on so-called *royal road functions*. According to Holland's [9] *building block hypothesis*, these functions should be particularly suited to be optimized by an algorithm employing crossover. The experiments conducted in [20], however, clearly demonstrated that this advantage does not exist. In fact, an elementary randomized hill-climbing heuristic (repeated mutation and selection of the fitter one of parent and offspring) was found to be far superior to the genetic algorithm.

The first theoretical analysis indicating that crossover can be useful was given by Jansen and Wegener [11] in 1999 (see also [29]). For $m < n$, they defined a pseudo-Boolean *jump* function $j_m : \{0,1\}^n \to \mathbb{R}$ such that (more or less) $j_m(x)$ is the number of ones in the bit-string x if this is at most $n - m$ or equal to n, but small otherwise. A typical mutation based evolutionary algorithm (flipping each bit independently with probability $1/n$) will easily find an individual x such that $j_m(x) = n - m$, but will need expected time $\Omega(n^m)$ to flip the remaining m bits (all in one mutation step). However, if we add the uniform crossover operator (here, each bit of the offspring is randomly chosen from one of the two parents) and use it sufficiently seldom compared to the mutation operator, then the run-time reduces to $O(n^2 \log n + 2^{2m} n \log n)$. While the precise computations are far from trivial, this behavior stems naturally from the definition of the jump function.

The work of Jansen and Wegener [11,29] was subsequently extended by different authors in several directions [12, 25], partly to overcome the critique that in the first works the crossover operator necessarily had to be used very sparingly. While these works enlarged the theoretical understanding of different crossover operators, they could not resolve the feeling that all these pseudo-Boolean functions were artificially tailored to demonstrate a particular phenomenon. In [12], the authors state that "It will take many major steps to prove rigorously that crossover is essential for typical applications."

The only two works (that we are aware of) that address the use of crossover for other problems than maximizing a pseudo-Boolean function are "Crossover is Provably Essential for the Ising Model on Trees" [26] by Sudholt and "The Ising Model on the Ring: Mutation Versus Recombination" [5] by Fischer and Wegener. They show that crossover also helps when considering a simplified Ising model on special graph classes, namely rings and trees. The simplified Ising model, however, is equivalent to looking for a vertex coloring of a graph such that all vertices receive the same color. While it is interesting to see that evolutionary algorithms have difficulties addressing such problems, proving "rigorously that crossover is essential for typical applications" remains an open problem.

1.3 Our Result

In this work, we present the first non-artificial problem for which crossover provably reduces the order of magnitude of the optimization time. This problem is the all-pairs shortest path problem (APSP), that is, the problem to find, for all pairs of vertices of a directed graph with edge lengths, the shortest path from the first vertex to the second. This is one of the most fundamental problems in graph algorithms, see for example the books by Mehlhorn and Näher [18] or Cormen et al. [3].

There are two classical algorithms for this problem. The Floyd-Warshall algorithm ([6, 27]) has a cubic runtime and is quite easy to implement. In contrast, Johnson's algorithm [14] is more complicated, but has a superior runtime on sparse graphs. Since the problem is NP-hard [8] if negative cycles exist and simple paths are sought, we will always assume that all weights are non-negative. Though not the focus of this theory-driven paper, we note that path problems do find significant attention from the evolutionary algorithms community, see e.g. [1, 10, 16, 17].

We present a natural evolutionary algorithm for the APSP problem. It has a population consisting of at most one path for every pair of vertices (connecting the first to the second vertex). Initially, it contains all paths consisting of one edge. A mutation step consists of taking a single path from the population uniformly at random and adding or deleting a random edge at one of its endpoints. The newly generated individual replaces an existing one (connecting the same vertices) if it is not longer. Hence our fitness function (which is to be minimized) is the length of the path.

We analyze this algorithm and prove that, in the worst case, it has with high probability an optimization time of $\Theta(n^4)$, where n is the number of vertices of the input graph. Note that the performance measure we regard, as common in the EA community, is the optimization time. This is defined to be the number of fitness evaluations in a run of the algorithm.

We then state three different crossover operators for this problem. They all take two random individuals from the

population and try to combine them to form a new one. In most cases, of course, this will not generate a path. In this case, we define the fitness of the new individual to be infinite (or some number larger than n times the longest edge). Again, the new individual replaces one having the same endpoints and not smaller fitness.

Using an arbitrary constant crossover rate for any of these crossover operators, we prove an upper bound of $O(n^{3.5+\varepsilon})$ for the expected optimization time. Hence for the APSP problem, crossover leads to a reduction of the optimization time. While the improvement of order $n^{0.5-\varepsilon}$ might not be too important, this work solves a long-standing problem in the theory of evolutionary computation. It justifies to use both a mutation and crossover operator in applications of evolutionary computation.

While our proofs seem to use only simple probabilistic arguments, a closer look reveals that we also invented an interesting tool for the analysis of evolutionary algorithms. A classical problem in the analysis of such algorithms is that the mutation operator may change an individual at several places (multi-bit flips in the bit-string model). Hence unlike for the heuristic of randomized local search, with evolutionary algorithms we cannot rely on the fact that our offspring is in a close neighborhood of the original search point. While this is intended from the view-point of algorithm design (to prevent being stuck in local optima), this is a major difficulty in the theoretical analysis of such algorithms. Things seem to become even harder, when (as here) we do not use bit-strings as representations for the individuals. We overcome these problems via what we call *c-trails*. These are hypothetical ways of how to move from one individual to another using simple mutations only. While still some difficulties remain, this allows to analyze the evolutionary algorithms we consider in this paper. We employ methods similar to the ones used in [4] to obtain a tight analysis for the single source shortest path problem.

2. A GENETIC ALGORITHM FOR THE APSP PROBLEM

Let $G = (V, E)$ be a directed graph with $n := |V|$ vertices and $m := |E|$ edges. Let $w : E \to \mathbb{N}$ be a function that assigns to each edge $e \in E$ a weight $w(e)$. Then the APSP problem is to compute a shortest path from every vertex $u \in V$ to every other vertex $v \in V$. A *walk* from u to v is a sequence $u = v_0, v_1, \ldots, v_k = v$ of vertices such that $(v_{i-1}, v_i) \in E$ for all $i \in [1..k]$. The walk is called *path* if it contains each vertex at most once. We will usually describe a walk by the sequence $(e_1, \ldots, e_k)$, $e_i = (v_{i-1}, v_i)$, of edges it traverses. The weight of a walk is defined as the sum of the weights of all its edges.

One of the strengths of evolutionary computation is that the algorithms are composed of generic components like mutation, crossover and selection. We now give the different components needed for a genetic algorithm that solves the APSP problem.

2.1 Individuals and Population

Genetic algorithms usually keep a set (*population*) of solution candidates (*individuals*), which is gradually improved. In the APSP problem we are aiming for a population containing a shortest path for each pair of distinct vertices. Hence it makes sense to allow paths or walks as individuals. To have more freedom in defining the crossover operator, an individual will simply be a sequence of edges, $(e_1, \ldots, e_k), e_1, \ldots e_k \in E, k \in \mathbb{N}$. However, the selection operator (see below) will ensure that only individuals that are walks can enter the population.

For the APSP problem, a natural choice for the initial population is the set $\mathcal{I} := \{(e) \mid e \in E\}$ of all paths consisting of one edge.

2.2 Fitness and Selection

A second standard component of evolutionary algorithms is *selection*. The aim is to prevent the population from growing too big as well as to get rid of individuals that are not considered to be useful solution candidates anymore. Typically, selection is guided by a fitness function assigning each individual a non-negative *fitness*. Strict selection operators, like truncation, eliminate the unfittest individuals, whereas fitness proportionate (also called roulette-wheel) or tournament selection favor fitter individuals more moderately. The first can lead to a faster fitness increase in the population, the latter has the advantage of a higher degree of diversity in the population.

For our problem, diversity is an issue in the sense that we need to end up with one path for each pair of vertices. However, this is better achieved not by fitness proportionate selection, but rather by ensuring directly that we do not eliminate all paths between a pair of vertices. Such an approach is called *diversity mechanism*. Ensuring diversity this way, we can be strict in the selection otherwise. In fact, for each pair (u, v) of vertices we eliminate all but the fittest individual connecting u to v.

With this strict selection principle, we only need to compare the fitness of individuals having identical start and end vertices. The natural choice for the fitness (which in this case has to be minimized) is the length of the walk represented by the individual. As a result of a crossover operation (see below), we may generate individuals that are not walks. These shall have fitness ∞ and never be included in the population.

2.3 Mutation and Crossover

In evolutionary computation, new individuals are generated by *variation operators*, namely by mutation or crossover (or both).

A *mutation* of an individual changes it slightly at some random positions. For the classical case of bit-strings of length n, mutation is often performed by flipping each bit of an individual independently with probability $\frac{1}{n}$. As this might be infeasible for more complex representations, this behavior must be simulated.

In [24], Scharnow, Tinnefeld and Wegener propose the following method to do this. First, a number s is chosen at random according to a Poisson distribution $\text{Pois}(\lambda = 1)$ with parameter $\lambda = 1$. An individual is then mutated by applying the following elementary mutation $s+1$ times. Let $(u, v) \in E$ be the first edge of the individual and $(u', v') \in E$ be the last edge. Pick an edge e from the set of all edges incident to u or v' uniformly at random. If this edge is (u, v) or (u', v'), remove it from the individual, otherwise append it at the corresponding end of the individual. The

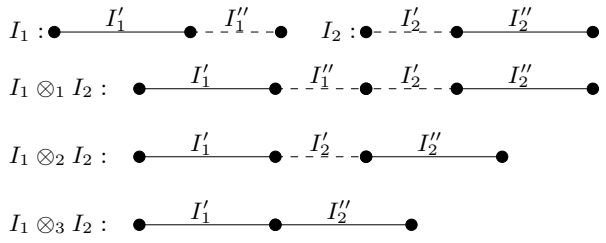

Figure 1. The effects of the three crossover operators.

use of the Poisson distribution is motivated by the fact that it is the limit of the binomial distribution for n trials with probability $\frac{1}{n}$ each.

A *crossover* of two individuals combines parts of them to a new individual. In this paper we consider three variations of the so-called 1-point crossover. For individuals that are bit-strings of length n, it is defined by picking a random position and merging the initial part of the first individual up to the chosen position with the ending part of the second individual starting from the chosen position. Since we don't represent individuals as bit-strings, this cannot be applied directly. Instead, we propose the following three crossover operators to combine two individuals I_1, I_2 containing ℓ_1 and ℓ_2 edges respectively. The crossover operator $\otimes_1$ simply combines both individuals by appending I_2 to I_1. The second operator, $\otimes_2$, chooses a random number $i \in [0..\ell_1]$ and appends I_2 to the first i edges of I_1. Finally, the operator $\otimes_3$ chooses two random numbers $i \in [0..\ell_1]$ and $j \in [0..\ell_2]$. The new individual created by this operator consists of the first i edges of I_1 and the last $\ell_2 - j$ edges of I_2. In Figure 1 the effects of the three crossover operators are depicted.

Observe that, unlike mutation, crossover may combine two individuals representing walks to a new individual that no longer represents a walk, and hence has infinite fitness.

2.4 $(\mu + 1)$-EA and GA

The algorithms we consider repeatedly apply variation and selection to a set of individuals. We study both an algorithm that only uses mutation and an algorithm that uses both mutation and one of the crossover operators.

Both algorithms share the following common framework. First, the population $\mathcal{I}$ is initialized. Then, depending on the kind of algorithm, it is decided randomly with a certain probability if a mutation or a crossover step should be done. If a mutation step is done, the algorithm picks an individual uniformly at random from the population and applies the mutation operator to it to generate a new individual. If a crossover step is done, the algorithm picks two individuals uniformly at random from the population and applies a crossover operator to generate a new individual. Afterwards, it checks if there is an individual in the population that connects the same two points as the newly generated individual. If not, the new individual is added to the population. If yes, the old individual is replaced if it is not fitter than the new one. These variation and selection steps are then repeated forever.

If only mutation is used, we get a classical evolutionary algorithm, the so-called $(\mu + 1)$-EA. This means that the population consists of up to μ individuals, and each step one new individual is generated by mutation. Since the population will consist of up to $n(n-1)$ individuals, namely one shortest path candidate for each pair of distinct vertices, $\mu = n(n-1)$ in our case. For sake of simplicity, we will still call the algorithm $(\mu + 1)$-EA instead of $((n(n-1)) + 1)$-EA. If also crossover is used, we get a more general genetic algorithm, which we simply call GA.

When analyzing evolutionary algorithms, one is interested in the expected optimization time. This is defined as the number of fitness function evaluations needed until the population only consists of optimal individuals. Since we are only interested in the asymptotic optimization time, it suffices to analyze the number of iterations needed.

3. ANALYSIS OF THE $(\mu + 1)$-EA

In this section we show that with high probability the optimization time of the $(\mu + 1)$-EA is $\Theta(n^4)$ in the worst case.

We need the following classical result from probability theory (cf. [2]).

THEOREM 1 (CHERNOFF BOUNDS). *Let $X_1, \ldots, X_t$ be mutually independent variables with $0 \leq X_i \leq b$ for all $i \in [1..t]$. Let $X = \sum_{i=1}^{t} X_i$. Then for all $\alpha > 1$,*

$$\Pr[X \geq \alpha \mathbb{E}[X]] < (e^{\alpha-1}\alpha^{-\alpha})^{\mathbb{E}[X]/b}.$$

3.1 Upper Bound on the Optimization Time

We omit the formal proof of the upper bound of $O(n^4)$ for reasons of space, but sketch the main ideas only.

Being pessimistic, we may assume that shortest paths are found exclusively by adding edges to other already found shortest paths. Since now it is obviously irrelevant which (of the possibly several) shortest path has been found, to simplify the proof we may assume that for every pair of vertices there exists a unique shortest path. Then, to find a shortest path consisting of ℓ edges, it suffices if the $(\mu+1)$-EA chooses ℓ times the correct sub-path already in the solution (with probability $O(n^{-2})$) and adds the appropriate edge (with probability $O(n^{-1})$). If $\ell \geq \log n$, the time needed for this is that sharply concentrated around the mean of $\Theta(\ell n^3)$, that we may use a union bound argument to obtain the following lemma.

LEMMA 2. *Let $\ell \geq \log(n)$. With high probability, the $(\mu+1)$-EA finds all shortest paths with at most ℓ edges in $O(\ell n^3)$ steps.*

For $\ell = n$, Lemma 2 yields the following upper bound.

THEOREM 3. *With high probability, the optimization time of the $(\mu + 1)$-EA is $O(n^4)$.*

3.2 Lower Bound on the Optimization Time

For the lower bound analysis consider the complete directed graph $K_n = ([1..n], \{(u, v) | u, v \in [1..n], u \neq v\})$ with edge lengths

$$w(u, v) = \begin{cases} 1 & \text{if } v - u = 1, \\ n & \text{else.} \end{cases}$$

We call edges with $w(u,v) = 1$ *good* and the other edges *bad*. For two distinct vertices u, v the unique shortest path is $P = (u, \dots, v)$ if $u < v$ and (u, v) otherwise. Because of the edge weights of K_n, the fitness function and selection step assure that at any time all individuals in the population consist of an initial edge (u, v) or of the shortest path from u to v.

The proof of the following lemma is omitted for reasons of space.

LEMMA 4. *With high probability*[1]*, during its optimization time the $(\mu+1)$-EA will only accept mutation steps that consist of a constant number of elementary mutations.*

Assume that, during a mutation step, a single *bad* edge is added. Let P_l be the path e was added to, and P_r be the edges added to e (if any). If the mutation step is accepted, there has to be an elementary mutation that deletes e. Either (i) all edges of P_r or (ii) all edges of P_l have to be deleted prior to deleting e. In case (i) we ignore all elementary mutations that add and delete the edges of P_r. The remaining elementary mutations of the mutation step create a sequence of intermediate shortest paths $(P_1, \dots, P_c)$ (called *reduced* sequence) where P_{i+1} can be created from P_i by a single elementary mutation for every $1 \le i < c$. In case (ii) the path resulting from the mutation step cannot have more than s edges where s is the constant from Lemma 4.

If more than one bad edge is added during a mutation step, we repeat the above construction until no bad edges remain. This leaves us with either a reduced sequence or a path having at most s edges.

Consider a possible sequence of mutation steps that create a shortest path P and disregard the first steps until (ii) happens for the last time. Combining the (possibly reduced) sequences of intermediate shortest paths from all these mutation steps we get a sequence of shortest paths $(P_1, \dots, P_\ell)$ that fulfills the requirements of the following definition.

DEFINITION 5 (s-TRAIL). *Let P be a shortest path. An s-trail T of P is a sequence of* shortest *paths $(P_1, P_2, \dots, P_\ell)$ such that P_1 consists of at most s edges, $P_\ell = P$, and P_{j+1} can evolve from P_j by an elementary mutation. We call $\ell = |T|$ the length of T.*

If P consists of ℓ edges, there exist s-trails of P of length m for $m \in \{\ell, \ell+1, \ell+2, \dots\}$. Since there are at most 4 possibilities to get from one shortest path to another shortest path by a single mutation (adding or deleting an edge at the beginning or end of the path), there are at most 4^m s-trails of P of length m.

THEOREM 6. *With high probability, the optimization time of the $(\mu+1)$-EA on K_n is $\Omega(n^4)$.*

PROOF. Consider the path $P = (1, \dots, n)$ and let s be the constant from Lemma 4. In order to create P the $(\mu+1)$-EA obviously has to perform all elementary mutations that create all shortest paths of one of the s-trails of P. First, we will show that the number of steps the $(\mu+1)$-EA needs to follow one particular s-trail of P is $\Omega(n^4)$ with high probability. Then, we will prove that with high probability the $(\mu+1)$-EA will not follow any of the s-trails of P in less than $\Omega(n^4)$ steps.

Fix one s-trail $T = (P_1, \dots, P_\ell)$ of P. We call a mutation step a *c-improvement* in T if it has the paths $(P_{i+1}, \dots, P_{i+c})$ as intermediate paths for some $i \le \ell - c$. Note that by Lemma 4 $c \le s$ with high probability. If a sequence of c-improvements in s-trail T has as intermediate paths exactly all P_i of T, we say that the $(\mu+1)$-EA has followed T to its end.

Let t' be the number of steps the $(\mu+1)$-EA needs to follow T to its end. Define the random variables X_i for $i \in [1..t']$ as $X_i = c$ if the i-th mutation step is a c-improvement of T. An accepted c-improvement first has to pick the right individual with probability $\frac{1}{n(n-1)}$ and then the c correct edges have to be added with probability $\frac{1}{2(n-1)}$ each. Thus, the probability that $X_i = c$ is bounded by $\Pr[X_i = c] < \frac{1}{n^{2+c}} + \frac{1}{(n-1)^{3+c}} + \frac{1}{(n-1)^{4+c}} + \dots < \frac{2}{n^{2+c}}$. The later summands cover the additional probability of at most $\frac{1}{n-1}$ for deleting a bad edge. For $i > t'$ define X_i as $\Pr[X_i = c] = \frac{2}{n^{2+c}}$. Then the X_i are mutually independent. The expected value of $\sum_{i=1}^{t} X_i = X$ is

$$\mathbb{E}[X] = \sum_{i=1}^{t} \mathbb{E}[X_i] = \sum_{i=1}^{t}\sum_{c=1}^{s} c \Pr[X_i = c]$$
$$< \sum_{i=1}^{t}\sum_{c=1}^{s} c\frac{2}{n^{2+c}} < \sum_{i=1}^{t} s^2 \frac{2}{n^3} = ts^2\frac{2}{n^3}.$$

If the $(\mu+1)$-EA has found P in t steps by following s-trail T to its end, then obviously $\sum_{i=1}^{t} X_i \ge |T| \ge n-1-s$. Hence, the probability of finding P on s-trail T in t steps is

$$\Pr[P \text{ found in } t \text{ steps on } T] = \Pr[\sum_{i=1}^{t} X_i \ge |T|].$$

Define $\alpha := \frac{|T|}{\mathbb{E}[X]}$ and $t := \frac{1}{4s^2 5^s e} n^4$. Then for large enough n it holds that

$$\alpha = \frac{|T|}{\mathbb{E}[X]} > \frac{2 \cdot 5^s e(n-1-s)}{n} \ge 5^s e > 1.$$

Hence, by Theorem 1, the probability of finding P in $t = \frac{1}{4s^2 5^s e} n^4$ steps following s-trail T to its end is bounded by

$$\Pr[P \text{ found in } t \text{ steps on } T] = \Pr[\sum_{i=1}^{t} X_i \ge \alpha \mathbb{E}[X]]$$
$$< (e^{\alpha-1}\alpha^{-\alpha})^{\mathbb{E}[X]/s}$$
$$\le (\frac{e}{\alpha})^{\alpha\mathbb{E}[X]/s} < \frac{1}{5^{s|T|/s}} = \frac{1}{5^{|T|}}.$$

Since the $(\mu+1)$-EA has to follow one of the s-trails of P in order to find P, the probability that the $(\mu+1)$-EA finds P in $t = \frac{1}{4s^2 5^s e} n^4$ steps is bounded by

$$\Pr[P \text{ found in } t \text{ steps}] \le \sum_{T \in \mathcal{T}} \Pr[P \text{ found in } t \text{ steps on } T]$$
$$= \sum_{\ell=n-s-1}^{\infty} \sum_{T \in \mathcal{T}, |T|=\ell} \Pr[P \text{ found in } t \text{ steps on } T]$$
$$< \sum_{\ell=n-s-1}^{\infty} \sum_{T \in \mathcal{T}, |T|=\ell} \frac{1}{5^\ell}$$
$$\le \sum_{\ell=n-s-1}^{\infty} 4^\ell \frac{1}{5^\ell} \le 5(\frac{4}{5})^{n-s-1}.$$

[1]If we say "with high probability", we mean with probability of $1 - O(n^{-k})$ for an arbitrary but fixed constant k.

Here $\mathcal{T}$ denotes the set of all s-trails of P. In the penultimate line we used the fact that there are at most 4^ℓ s-trails of P of length ℓ. Since the $(\mu+1)$-EA has to find P to solve the APSP it needs with high probability at least $\Omega(n^4)$ steps. □

Observe that this theorem implies an expected optimization time of $\Omega(n^4)$.

4. UPPER BOUND ON THE OPTIMIZATION TIME OF THE GA

We now prove that if we use the GA for the APSP problem, that is, we enrich the $(\mu+1)$-EA with a crossover operator, then the expected optimization time drops to $O(n^{3.5+\varepsilon})$.

While it seems natural that the additional use of powerful variation operators should speed up computation, this behavior could so far not be proven for a non-artificial problem. Several reasons for this have been discussed in the literature. In our setting, the following aspect seems crucial. The hoped for strength of the crossover operator lies in the fact that it can advance a solution significantly. E.g., it can combine two shortest paths consisting of ℓ_1 and ℓ_2 edges to one consisting of $\ell_1+\ell_2$ edges in one operation. On the negative side, for this to work, the two individuals we try to combine have to fit together. Thus with relatively high probability, the crossover operator will produce an invalid solution (here, no path at all). Often, this disadvantage seems to outnumber the chance of faster progress.

Our analysis shows that this does not happen in our setting. In fact, from the point on when our population contains all shortest paths having $O(n^{1/2+\varepsilon})$ edges, crossover becomes so powerful that we would not even need mutation anymore.

We proposed 3 crossover operator which all pick two paths uniformly at random. The $\otimes_1$-crossover operator concatenates the two paths to a new one. The $\otimes_2$-crossover operator additionally picks a non-negative integer i not exceeding the length of the first path uniformly at random. It then concatenates the first i edges of the first path and the second path to a new path. The $\otimes_3$-crossover operator picks two non-negative integers i not exceeding the length of the first path and j not exceeding the length of the second path uniformly at random. It then combines the first i edges of the first path with the last j edges of the second path to a new path.

Note that even though we can prove the upper bound for all three crossover operators, as the crossover operators we use become more elaborate we need to comply to a number of restrictions. Given these restrictions, we prove for each crossover operator a certain probability that it successfully creates a longer path by combining two shorter paths. Using these success probabilities we prove the expected optimization time of $O(n^{3.5+\epsilon})$.

LEMMA 7. *Let $k>1$. Assume that for any pair of vertices for which a shortest path having at most k edges exists, the population $\mathcal{I}$ contains a shortest path. Let u,v be two vertices for which a shortest path from u to v having $\ell \in [k+1..2k]$ edges exists. Then the following holds.*

a) A single step of the $\otimes_1$-operator generates a shortest path from u to v with probability $\Omega(\frac{2k+1-\ell}{n^4})$.

b) Assume that among two shortest paths the fitness function prefers the one having fewer edges. If any shortest path from u to v has at least ℓ edges, then a single step of the $\otimes_2$-operator generates a shortest path from u to v having ℓ edges with probability $\Omega(\frac{(2k+1-\ell)^2}{kn^4})$.[2]

c) Assume the input graph has unique shortest paths. Then a single step of the $\otimes_3$-operator generates the shortest path from u to v with probability $\Omega(\frac{(2k+1-\ell)^3}{k^2n^4})$.

PROOF. We exemplarily show the proof for b) and omit the rest for reasons of space. To generate a shortest path from u to v, it suffices that the $\otimes_2$-operator picks a path P_u starting in u, a path P_v ending in v, and a number $i \in [0..|P_u|]$ such that the first i edges of P_u together with P_v form a path from u to v. The probability that a particular triple (P_u,P_v,i) with $|P_u|\leq k, |P_v|\leq k, i\leq |P_u|$ is chosen is at least $(n(n-1))^{-2}(k+1)^{-1} = \Omega(\frac{1}{kn^4})$.

It remains to count how many such triples generate a shortest path from u to v. Let $P=((u,w_1),\dots,(w_{\ell-1},v))$ be such a shortest path having ℓ edges. If $\ell-k\leq j\leq k$, then $\mathcal{I}$ contains a shortest path $P_u=((u,w'_1),\dots,(w'_{j-1},w_j))$ from u to w_j having j edges. If we cut P_u after vertex w'_i for $i\in[\ell-k..j]$, then $\mathcal{I}$ contains a shortest path P_v from w'_i to v since $\ell-i\leq k$. Obviously, the first i edges of P_u combined with P_v form a shortest path from u to v. Hence, the total number of triples yielding a shortest path from u to v having ℓ edges is at least

$$\sum_{j=\ell-k}^{k} (j-(\ell-k)+1) = \Omega((2k+1-\ell)^2).$$

Thus, the probability that $\otimes_2$ generates such a path in a single step is at least $\Omega(\frac{(2k+1-\ell)^2}{kn^4})$. □

COROLLARY 8. *Let $\ell=\frac{3k}{2}$. With the assumptions of Lemma 7, the following holds.*

a) For any two vertices that can be connected by a shortest path of at most ℓ edges, a single crossover step will create a shortest path connecting them with probability at least $\Omega(\frac{k}{n^4})$.

b) The expected number of steps until $\mathcal{I}$ contains a shortest path for all pairs of vertices having a shortest path of up to ℓ edges is $O(\frac{n^4\log(n)}{k})$.

PROOF. The first part follows directly by plugging ℓ into Lemma 7. The second part can be proven as in the coupon collectors theorem (cf. [2]) since there are at most $O(n^2)$ paths that need to be considered. □

THEOREM 9. *Let $i\in[1..3]$. If the conditions for the $\otimes_i$-operator hold, then the GA using mutation and $\otimes_i$-crossover needs an expected number of $O(n^{3.5}\sqrt{\log(n)})$ steps to solve the APSP problem.*

PROOF. Let $\ell := \sqrt{n\log(n)}$. We assume that the algorithm runs in two phases. In the first phase, only mutation is used to construct all shortest paths of up to ℓ edges. In the second phase, only the $\otimes_i$-crossover is used to finish the computation. Since both $\otimes_i$ and mutation happen with

[2]Note that Lemma 2 still holds if the fitness function prefers shortest paths having fewer edges.

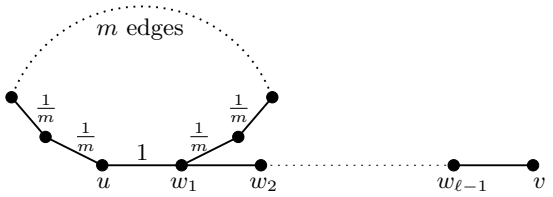

Figure 2. An example where the analysis of the $\otimes_2$-operator fails if the fitness function does not prefer individuals with fewer edges.

constant probability and neither of them can decrease the fitness of the population, this is a valid restriction. According to Lemma 2, the first phase needs an expected number of $O(n^{3.5}\sqrt{\log(n)})$ steps to finish.

For the second phase we apply Corollary 8 repeatedly until $\mathcal{I}$ contains all shortest paths of size up to n. Hence the expected number of steps is

$$\begin{aligned}\sum_{i=\log_c(\ell)}^{\log_c(n)} O\left(\frac{n^4\log(n)}{c^i}\right) &= O\left(n^4\log(n)\sum_{i=\log_c(\ell)}^{\log_c(n)}\frac{1}{c^i}\right)\\ &= O\left(\frac{n^4\log(n)}{c^{\log_c(\ell)}}\sum_{i=0}^{\log_c(\frac{n}{\ell})}\frac{1}{c^i}\right)\\ &= O\left(n^{3.5}\sqrt{\log(n)}\right)\end{aligned}$$

where $c := \frac{3}{2}$. □

4.1 Necessity of the Restrictions

We now demonstrate where the proof of the optimization time would fail without the additional constraints for $\otimes_2$ and $\otimes_3$.

To see that the assumption that the fitness function prefers paths with fewer edges is essential for the $\otimes_2$-operator, consider the graph depicted in Figure 2. Consider the computation of a shortest path from u to v using the $\otimes_2$-operator. Two such shortest paths exist, namely P_1 which uses the edge (u, w_1) of cost 1 and has ℓ edges and P_2 which uses the m edges of cost $\frac{1}{m}$ and has $\ell - 1 + m$ edges. Assume as in Lemma 7b) that for all vertices w, w' for which a shortest path of at most k edges exists a shortest path is in the population $\mathcal{I}$, and that $\ell \in [k+1..2k]$ and $m \in \Omega(n)$. If $\mathcal{I}$ contains for the paths from u to w_i for $i \in [1..k]$ the paths using the edge (u, w_1), the proof of Lemma 7b) works. However, if $\mathcal{I}$ contains the paths using the m edges of cost $\frac{1}{m}$, the probability that the $\otimes_2$-operator picks a convenient triple (P_u, P_v, i) drops from $\Omega(\frac{1}{kn^4})$ to $\Omega(\frac{1}{n^5})$ since there are $\Omega(n)$ possible positions to cut P_u.

The assumption that the shortest paths are unique is essential for the proof for the $\otimes_3$-operator. To see this, consider the graph depicted in Figure 3. Observe that there are many different shortest paths connecting two vertices, all having an equal number of edges. Assume that $\mathcal{I}$ contains the shortest paths from u to w_i and w'_i for $i \in [1..k]$ and from w_j and w'_j to v for $j \in [\ell - k..\ell - 1]$ as given in the figure. Then for any shortest path from u to v the population will not contain all sub-paths of length up to k, as needed by Lemma 7c). Even more, any pair of paths, one starting in u, the other ending in v, will only overlap on at most two vertices.

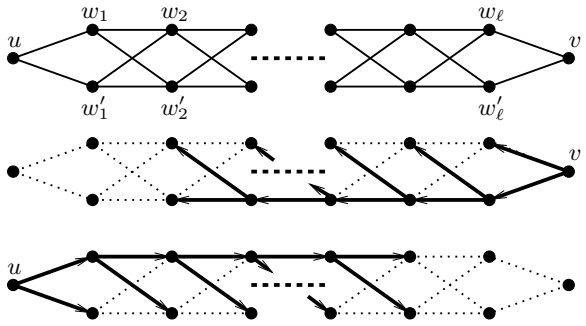

Figure 3. An example where the analysis of the $\otimes_3$-operator fails, since the shortest paths are not unique.

5. EXPERIMENTAL RESULTS

We implemented the algorithm given in Section 2.4 and the three different crossover operators discussed in this paper. To study the behavior of our approach, we ran it on the following three graphs. The first graph is the graph K_n from Section 3.2 that has edge weights 1 for all edges $(i, i+1), i \in [1..n-1]$ and weight n for all other edges. The second and third graph are the graphs given in Figures 2 and 3 that are used in Section 4.1 to show why we need additional conditions for the proofs for $\otimes_2$ and $\otimes_3$. The graphs are complete and the weights of the edges not shown in the figures are chosen in such a way that they are too expensive to be included in the optimal solution. We ran our algorithm on all three graphs, once using mutation only and once for each crossover operator with crossover probability $\frac{1}{4}$. Note that, although we put restrictions on $\otimes_2$ and $\otimes_3$ in the proofs, our implementation does not prefer paths with fewer edges nor does it need unique shortest paths when applying $\otimes_3$. The optimization times for all experiments averaged over 50 runs are shown in Figure 4. It can clearly be seen that adding any of the crossover operators indeed speeds up the computation quite noticeably. Also, it shows that the "bad graphs" from Section 4.1 are not hard to solve for the corresponding crossover operators.

6. CONCLUSIONS

In this work, we presented the first non-artificial problem for which a natural evolutionary algorithm using only mutation is provably outperformed by one using mutation and crossover. By a rigorous analysis of the optimization time, we proved that the all-pairs shortest path problem can be solved by an evolutionary algorithm using crossover in an expected optimization time of $O(n^{3.5+\varepsilon})$, whereas the corresponding algorithm using only mutation needs an expected optimization time of $\Omega(n^4)$ in the worst case. While this clearly does not beat the best classical algorithms, we do feel that this result gives a much better theoretical foundation for the use of crossover in practical applications than previous results on artificially defined pseudo-boolean functions.

Acknowledgments

The authors would like to thank an unknown referee for finding an error in a previous version of this paper. Thanks also to Thomas Jansen and Frank Neumann for many useful discussions.

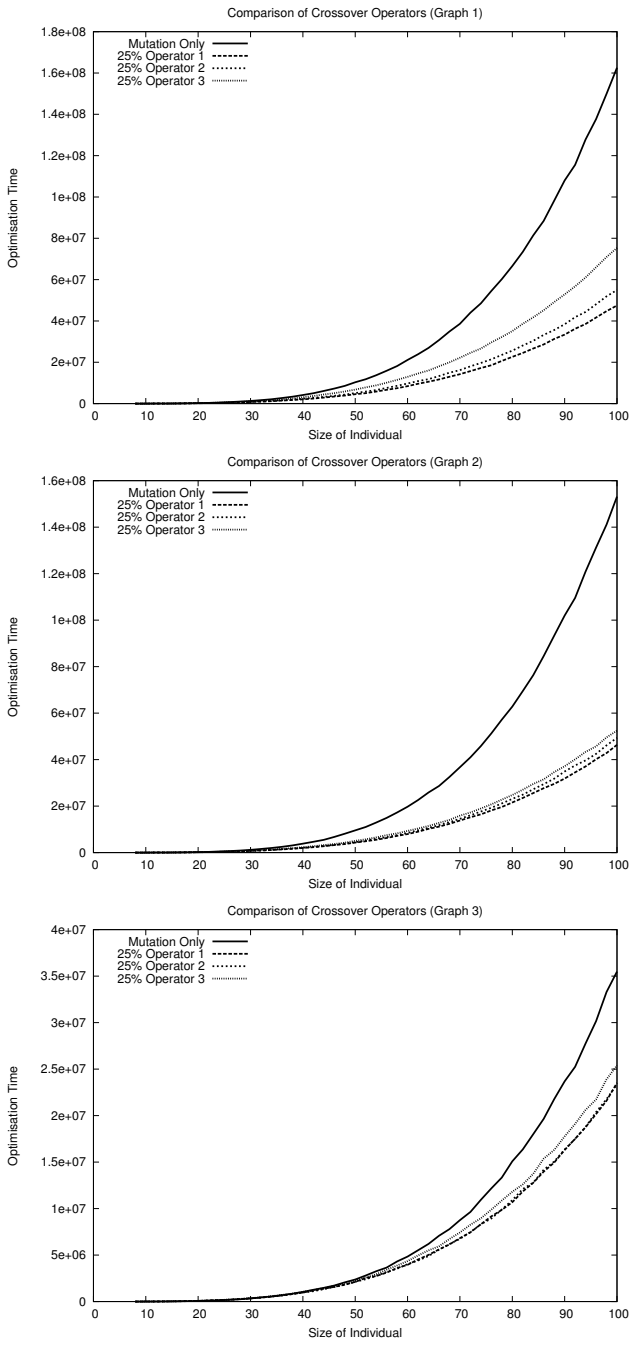

Figure 4. Optimization time for the various crossover operators on different graphs.

References

[1] C. W. Ahn and R. Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Trans. Evolutionary Computation*, 6:566–579, 2002.

[2] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley, 2nd edition, 2000.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[4] B. Doerr, E. Happ, and C. Klein. A tight bound for the (1+1)-EA on the single source shortest path problem. In *Proc. of CEC '07*, pages 1890–1895. IEEE Press, 2007.

[5] S. Fischer and I. Wegener. The Ising model on the ring: Mutation versus recombination. In *Proc. of GECCO '04*, volume 3102 of *LNCS*, pages 1113–1124. Springer, 2004.

[6] R. W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5:345, 1962.

[7] S. Forrest. Genetic algorithms: Principles of natural selection applied to computation. *Science*, 261:872–878, 1993.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[9] J. H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.

[10] J. Inagaki, M. Haseyama, and H. Kitajima. A genetic algorithm for determining multiple routes and its applications. In *Proc. of ISCAS '99*, pages 137–140. IEEE Press, 1999.

[11] T. Jansen and I. Wegener. On the analysis of evolutionary algorithms – a proof that crossover really can help. In *Proc. of ESA '99*, volume 1643 of *LNCS*, pages 184–193. Springer, 1999.

[12] T. Jansen and I. Wegener. Real royal road functions – where crossover provably is essential. *Discrete Applied Mathematics*, 149:111–125, 2005.

[13] M. Jerrum and A. Sinclair. The markov chain monte carlo method: An approach to approximate counting and integration. In D. Hochbaum, editor, *Approximations for NP-hard Problems*, pages 482–520. PWS Publishing, 1996.

[14] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24:1–13, 1977.

[15] S. Kirkpatrick, D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[16] S. Liang, A. N. Zincir-Heywood, and M. I. Heywood. Intelligent packets for dynamic network routing using distributed genetic algorithm. In *Proc. of GECCO '02*, pages 88–96. Morgan Kaufmann, 2002.

[17] S. Liang, A. N. Zincir-Heywood, and M. I. Heywood. Adding more intelligence to the network routing problem: AntNet and Ga-agents. *Appl. Soft Comput.*, 6:244–257, 2006.

[18] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[19] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *J. Chemical Physics*, 21:1087–1091, 1953.

[20] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing? In *Proc. of NIPS '93*, volume 6 of *Advances in Neural Information Processing Systems*, pages 51–58. Morgan Kaufmann, 1993.

[21] Y. Rabani, Y. Rabinovich, and A. Sinclair. A computational view of population genetics. In *Proc. of STOC '95*, pages 83–92. ACM Press, 1995.

[22] Y. Rabinovich and A. Wigderson. An analysis of a simple genetic algorithm. In *Proc. of ICGA '94*, pages 215–221, 1991.

[23] Y. Rabinovich and A. Wigderson. Techniques for bounding the convergence rate of genetic algorithms. *"Random Structures & Algorithms"*, 14:111–138, 1999.

[24] J. Scharnow, K. Tinnefeld, and I. Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *J. Math. Model. Algorithms*, 3:349–366, 2004.

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

[26] D. Sudholt. Crossover is provably essential for the Ising model on trees. In *Proc. of GECCO '05*, pages 1161–1167. ACM Press, 2005.

[27] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9:11–12, 1962.

[28] I. Wegener. Simulated annealing beats metropolis in combinatorial optimization. In *Proc. of ICALP '05*, volume 3580 of *LNCS*, pages 589–601. Springer, 2005.

[29] I. Wegener and T. Jansen. The analysis of evolutionary algorithms – a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002.