

Crossover Can Provably be Useful in Evolutionary Computation^{*†}

Benjamin Doerr, Edda Happ, Christian Klein

Abstract

We show that a 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.

1 Introduction

The paradigm of evolutionary computation is to use principles inspired by natural evolution, e.g., mutation, crossover and selection, to build algorithms. Genetic algorithms (GA), genetic programming (GP) and evolution strategies (ES) are prominent examples. Together with related approaches like randomized local search (RLS), the Metropolis algorithm [21], and simulated annealing [17] 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 at least two parents, is really useful. So far, apart from a few artificial examples, no problem is known

^{*}A preliminary version of this paper won a Best Paper Award at the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO 2008)

[†]This work greatly benefited from various discussions being presented and discussed at the Dagstuhl seminar 08051 on Theory of Evolutionary Algorithms.

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, a natural evolutionary algorithm using mutation has a worst-case optimization time of $\Theta(n^4)$ both in expectation and with high probability. However, if we add crossover to this algorithm, its expected optimization time is $O(n^{3.5}(\log n)^{0.5})$.

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

1.1 Evolutionary Computation

Evolutionary computation is algorithmics inspired by the evolution principle in nature. Typically, we have a collection (“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 this fitness value, a selection procedure may replace some individuals by fitter ones. The cycle of variation and replacement 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 described above. For 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. [24]).

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 [24–26]. Another interesting theoretical result is Wegener’s [31] solution to the “outstanding open problem” [15] on whether simulated annealing is superior to the Metropolis algorithm.

1.2 Need for Crossover?

The paradigm of evolution-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 [22], 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 [22], however, clearly demonstrated that this advantage does not exist. In fact, an elementary randomized hill-climbing heuristic (repeated mutation and survival 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 [12] in 1999 (see also [13]). For $m < n$, they defined a pseudo-Boolean *jump* function $j_m : \{0, 1\}^n \rightarrow \mathbb{R}$ 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 [12, 13] was subsequently extended by different authors in several directions [14, 28], 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 [14], 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" [29] by Sudholt and "The Ising Model on the Ring: Mutation Versus Recombina-

tion” [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 [20] or Cormen et al. [3].

There are two classical algorithms for this problem. The Floyd-Warshall algorithm [6, 30] has a cubic runtime and is quite easy to implement. In contrast, Johnson’s algorithm [16] 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, 18, 19].

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 (Poisson distributed) random number of times an 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 $O(n^{3.5}(\log n)^{0.5})$ 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 we 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 $w: E \rightarrow \mathbb{N}$ 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 $u = v_0, v_1, \dots, v_k = v$ 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, \dots, 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 replacement. 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 collection (*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, $(e_1, \dots, e_k), e_1, \dots, e_k \in E, k \in \mathbb{N}$. However, the replacement 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 for Replacement

A second standard component of evolutionary algorithms is (*selection for replacement*). 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, replacement is guided by a fitness function assigning each individual a non-negative *fitness*. Strict replacement 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 a non-strict replacement mechanism, but rather by ensuring directly that we do not eliminate all paths between a pair of vertices. Such an approach is called a *diversity mechanism*. Ensuring diversity this way, we can be strict in the replacement otherwise. In fact, for each pair (u, v) of vertices we eliminate all but the fittest individual connecting u to v (favoring a newly created offspring over previously generated individuals). This is a form of *truncation selection*.

With this strict replacement 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).

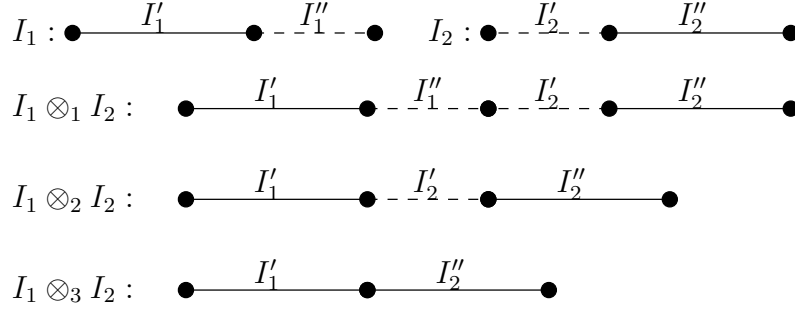


Figure 1: The effects of the three crossover operators.

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 [27], 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. However, if the individual is a single edge (u, v') , pick an edge uniformly at random from the set of all edges incident to u or v' except (u, v') and append it. The 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 do not 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 consisting of 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.

With a truncation selection operator guiding the replacement of individuals, it makes sense to select individuals as parents of mutation and crossover in a way that produces less selection pressure. We therefore choose these individuals uniformly at random from our population (*uniform selection* for breeding).

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

The algorithms we consider repeatedly apply variation and replacement 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.

Replacement is done as follows. If the new individual is not a walk, then it is simply discarded. Otherwise, we check if there is an individual in the population that connects the same two vertices 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 replacement steps are then repeated forever. The pseudo-code in Figure 2 illustrates this procedure.

If only mutation is used, we get an algorithm that strongly resembles the algorithm called $(\mu + 1)$ -EA by various authors [11, 23, 32]. Since our population size is not fixed to, but only bounded by $\mu = n(n - 1)$, we call our algorithm using mutation only $(\leq \mu + 1)$ -EA. If crossover is also used, we get an algorithm strongly resembling a classical genetic algorithm. For this reason, we call this variant $(\leq \mu + 1)$ -GA. We do see that these names are not ideal in the sense that usually EA is a more general concept, including GAs, ESs and GP.

When analyzing evolutionary algorithms, one often is interested in their 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.

ALGORITHM FRAMEWORK

```

    ▷ Initialization:
1   $\mathcal{I} := \{(e) \mid e \in E\}$ 
2  repeat
3      Set  $x := 1$  with probability  $p_x$ 
4      if  $x = 1$ 
5          then
6              ▷ Crossover:
7              Pick two Individuals  $I_1, I_2 \in \mathcal{I}$  u.a.r.
8              Generate a new Individual  $I'$  by applying
9                  a crossover operator to  $I_1$  and  $I_2$ 
10             else
11                 ▷ Mutation:
12                 Pick  $I \in \mathcal{I}$  uniformly at random
13                 Choose  $s$  according to  $\text{Pois}(\lambda = 1)$ 
14                 Generate a new Individual  $I'$  by  $s + 1$ 
15                     times adding or removing an edge from  $I$ 
16             ▷ Replacement:
17             Let  $I'' \in \mathcal{I}$  be the individual with the same
18                 start-vertex and end-vertex as  $I'$ , if any.
19             if  $I'$  is a walk and  $w(I') \leq w(I'')$ 
20                 then
21                     Add  $I'$  to  $\mathcal{I}$  and remove  $I''$  from  $\mathcal{I}$  if it exists.
22             forever

```

Figure 2: Pseudo-Code for the two algorithms studied by us. If p_x is a constant greater than zero, both mutation and crossover are used and the resulting algorithm will be called $(\leq \mu + 1)$ -GA. For $p_x = 0$, only mutation is used as variation operator. We call the resulting algorithm $(\leq \mu + 1)$ -EA.

3 Analysis of the $(\leq \mu + 1)$ -EA

In this section we show that both in expectation and with high probability¹ the worst case optimization time of the $(\leq \mu + 1)$ -EA is $\Theta(n^4)$. We need the following classical results from probability theory (cf. [2]).

Theorem 1 (Chernoff Bounds). *Let X_1, \dots, X_t be mutually independent random variables with $X_i \in \{0, 1\}$ for all $i \in [1..t]$. Let $X := \sum_{i=1}^t X_i$.*

- a) *For all $\alpha < 1$, $\Pr[X < \alpha \mathbb{E}[X]] \leq \exp(-\frac{1}{2}(1 - \alpha)^2 \mathbb{E}[X])$.*
- b) *For all $\beta > 1$, $\Pr[X \geq \beta \mathbb{E}[X]] < (e^{\beta-1} \beta^{-\beta})^{\mathbb{E}[X]}$.*

With Chernoff Bounds we can handle sums of independent random variables. In our proofs, however, we will also encounter sums of correlated variables. To be able to deal with such sums we will use the following lemma. With it, we can approximate the behavior of such sums by using sums of independent random variables. Hence we can, albeit indirectly, apply Chernoff Bounds to certain sums of dependent random variables.

Lemma 2. *Let $X_1, \dots, X_t, X_1^*, \dots, X_t^*$ be binary random variables. Assume that X_1^*, \dots, X_t^* are mutually independent and that for each $i \in [1..t]$, X_i^* is independent of X_1, \dots, X_{i-1} . Then for all $k \geq 0$ the following holds.*

- a) *If for all i and all $x_1, \dots, x_{i-1} \in \{0, 1\}$*

$$\Pr[X_i = 1 | X_1 = x_1, \dots, X_{i-1} = x_{i-1}] \geq \Pr[X_i^* = 1],$$

then

$$\Pr\left[\sum_{i=1}^t X_i \geq k\right] \geq \Pr\left[\sum_{i=1}^t X_i^* \geq k\right].$$

- b) *If for all i and all $x_1, \dots, x_{i-1} \in \{0, 1\}$*

$$\Pr[X_i = 1 | X_1 = x_1, \dots, X_{i-1} = x_{i-1}] \leq \Pr[X_i^* = 1],$$

then

$$\Pr\left[\sum_{i=1}^t X_i \geq k\right] \leq \Pr\left[\sum_{i=1}^t X_i^* \geq k\right].$$

Proof. a) Let $k \geq 0$, let $P_j := \Pr[\sum_{i=1}^j X_i + \sum_{i=j+1}^t X_i^* \geq k]$ for $j \in [0..t]$, and let $\mathcal{X}_k^t = \{(x_1, \dots, x_t) \in \{0, 1\}^t \mid \sum_{i=1}^t x_i = k\}$. Let us compare

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

the outcome of the sequence of events $X_1, \dots, X_j, X_{j+1}^*, \dots, X_t^*$ with the outcome of $X_1, \dots, X_{j+1}, X_{j+2}^*, \dots, X_t^*$. Then we get for $j \in [0..t-1]$

$$\begin{aligned}
P_{j+1} &= \Pr\left[\sum_{i=1}^{j+1} X_i + \sum_{i=j+2}^t X_i^* \geq k\right] \\
&= \Pr\left[\sum_{i=1}^j X_i + \sum_{i=j+2}^t X_i^* \geq k\right] \\
&\quad + \sum_{(x_1, \dots, x_j, x_{j+2}, \dots, x_t) \in \mathcal{X}_{k-1}^{t-1}} \Pr[X_{j+1} = 1 | X_1 = x_1, \dots, X_j = x_j] \\
&\quad \cdot \Pr[X_1 = x_1, \dots, X_j = x_j, X_{j+2}^* = x_{j+2}, \dots, X_t^* = x_t] \\
&\geq \Pr\left[\sum_{i=1}^j X_i + \sum_{i=j+2}^t X_i^* \geq k\right] \\
&\quad + \Pr\left[\sum_{i=1}^j X_i + \sum_{i=j+2}^t X_i^* = k-1\right] \cdot \Pr[X_{j+1}^* = 1] \\
&= \Pr\left[\sum_{i=1}^j X_i + \sum_{i=j+2}^t X_i^* \geq k\right] \\
&\quad + \Pr\left[\sum_{i=1}^j X_i + \sum_{i=j+2}^t X_i^* = k-1 \wedge X_{j+1}^* = 1\right] \\
&= \Pr\left[\sum_{i=1}^j X_i + \sum_{i=j+1}^t X_i^* \geq k\right] \\
&= P_j
\end{aligned}$$

since $\Pr[\sum_{i=1}^j X_i + \sum_{i=j+2}^t X_i^* = k-1] \geq 0$. Thus, we have that

$$\Pr\left[\sum_{i=1}^t X_i \geq k\right] = P_t \geq P_{t-1} \geq \dots \geq P_1 \geq P_0 = \Pr\left[\sum_{i=1}^t X_i^* \geq k\right].$$

b) Let $k \geq 0$ and let $P_j := \Pr[\sum_{i=1}^j X_i + \sum_{i=j+1}^t X_i^* \geq k]$ for $j \in [0..t]$ as above. Using the maximum instead of the minimum over all j -tuples and the fact that $\Pr[X_j = 1 | X_1 = x_1, \dots, X_{j-1} = x_{j-1}] \leq \Pr[X_j^* = 1]$ in the calculation from a), we get that $P_{j+1} \leq P_j$ for $j \in [0..t-1]$. Hence it follows that $\Pr[\sum_{i=1}^t X_i \geq k] \leq P_{t-1} \leq \dots \leq P_1 \leq \Pr[\sum_{i=1}^t X_i^* \geq k]$. \square

In general, a pair of vertices may be connected by more than one shortest path, and these different paths may consist of different numbers of edges. For our purposes, paths consisting of few edges are more important. To ease the language, we introduce the following notation.

Definition 3. Let $G = (V, E)$ be a graph and let $\ell \in \mathbb{N}$. We define

$$V_\ell^2 := \{(u, v) \in V^2 \mid u \neq v \text{ and there exists a shortest path from } u \text{ to } v \text{ consisting of at most } \ell \text{ edges}\}.$$

3.1 Upper Bound on the Optimization Time

The main ideas to prove the upper bound of $O(n^4)$ for the $(\leq \mu + 1)$ -EA are as follows. Being pessimistic, we may assume that shortest paths are found exclusively by adding edges to already found shortest paths, and more specifically, by only adding a single edge in each iteration. Then, to find a shortest path from u to v for $(u, v) \in V_\ell^2$, it suffices that the $(\leq \mu + 1)$ -EA chooses ℓ times the adequate shortest 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 over all $(u, v) \in V_\ell^2$.

Lemma 4. Let $\ell \geq \log(n)$. Within $O(\ell n^3)$ steps, the $(\leq \mu + 1)$ -EA finds with high probability a shortest path from u to v for all $(u, v) \in V_\ell^2$.

Proof. Let $(u, v) \in V_\ell^2$. We first analyze the probability that a shortest path from u to v is not found within a certain time. For the analysis, we fix a path $P = ((u, v_1), (v_1, v_2), \dots, (v_{\ell'-1}, v_\ell = v))$ of length $\ell' \leq \ell$. Note that P will be a technical tool only and we do not aim at finding this particular path.

In the following, we shall only consider mutation steps that perform a single elementary mutation. Note that a mutation consists of a single elementary mutation with probability $\frac{1}{e}$.

We call a mutation step the j -th *pessimistic improvement* in P if (i) it creates a shortest path from u to v_{j+1} out of a shortest path from u to v_j that is already in the population and (ii) the pessimistic improvements $1, \dots, j - 1$ have already been done. Note that this implies that pessimistic improvements appear in ascending order. Obviously, when the $(\leq \mu + 1)$ -EA has performed the $(\ell' - 1)$ -st pessimistic improvement in P , a shortest path from u to v has been found.

Let the random variable t' denote the number of steps the $(\leq \mu + 1)$ -EA executes until it performs the $(\ell' - 1)$ -st pessimistic improvement in P . For $i \in [1..t']$ define the random variable X_i by $X_i = 1$ if the i -th mutation step is a pessimistic improvement in P and $X_i = 0$ otherwise. Then

$$\Pr[X_i = 1] \geq \frac{1}{e} \frac{1}{n(n-1)^2} > \frac{1}{e} \frac{1}{n^3} =: p,$$

independent of the first $i - 1$ steps, since the probability to pick the correct individual is at least $\frac{1}{n(n-1)}$ and the probability to pick the correct edge is

at least $\frac{1}{n-1}$. For every $i > t'$ we independently define X_i by $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$.

Let $t := \epsilon \eta \ell n^3$ for some $\eta > 2$ and let $\mathcal{P}_{uv} := \{P \mid P \text{ is shortest path from } u \text{ to } v\}$. If the $(\leq \mu + 1)$ -EA has not found a shortest path from u to v after t steps, it obviously has not performed the $(\ell' - 1)$ -st pessimistic improvement in P , and thus $X := \sum_{i=1}^t X_i < \ell'$. For every $i \in [1..t]$ the random variable X_i fulfills $\Pr[X_i = 1 \mid X_1 = x_1, \dots, X_{i-1} = x_{i-1}] \geq p$ for all $x_1, \dots, x_{i-1} \in \{0, 1\}$. Define the mutually independent binary random variables X_i^* by $\Pr[X_i^* = 1] = p$ for $i \in [1..t]$. Using Lemma 2 and Theorem 1 with $\alpha := \frac{\ell'}{E[X^*]} \leq \frac{\ell}{pt} = \frac{1}{\eta}$, the probability of not finding a shortest path from u to v in t steps can be bounded by

$$\begin{aligned}
\Pr[\text{no } P \in \mathcal{P}_{uv} \text{ found in } t \text{ steps}] &\leq \Pr[X < \ell'] \\
&= 1 - \Pr[X \geq \ell'] \\
&\leq 1 - \Pr[X^* \geq \ell'] \\
&= \Pr[X^* < \ell'] \\
&= \Pr[X^* < \alpha \mathbb{E}[X^*]] \\
&\leq \exp(-\frac{1}{2}(1 - \alpha)^2 \mathbb{E}[X^*]) \\
&\leq \exp(-\frac{1}{2}(1 - \alpha)^2 pt) \\
&\leq \exp(-\frac{1}{8}\eta \ell).
\end{aligned}$$

Now a simple union bound argument shows that the probability that the $(\leq \mu + 1)$ -EA does not find for all vertex pairs $(u, v) \in V_\ell^2$ a shortest path connecting them ($P \in \mathcal{P}_{uv}$) in t steps is bounded by

$$\begin{aligned}
&\Pr[\text{there exists } (u, v) \in V_\ell^2 \text{ such that no } P \in \mathcal{P}_{uv} \text{ was found in } t \text{ steps}] \\
&\leq \sum_{(u,v) \in V_\ell^2} \Pr[\text{no } P \in \mathcal{P}_{uv} \text{ found in } t \text{ steps}] \\
&\leq n(n-1) \exp(-\frac{1}{8}\eta \ell) \\
&< n^2 \exp(-\frac{1}{8}\eta \log(n)) \\
&= n^{2-\frac{\eta}{8}}.
\end{aligned} \tag{1}$$

For any constant k we can choose $\eta := 8(k + 2)$. Thus, with probability $1 - O(n^{-k})$ the optimization time is at most $\epsilon \eta \ell n^3$. Note that we did not try to optimize the constant η . \square

For $\ell = n - 1$, Lemma 4 yields the following upper bound.

Theorem 5. *With high probability, the optimization time of the $(\leq \mu + 1)$ -EA is $O(n^4)$.*

From the strong concentration bound of equation (1) we also derive an $O(n^4)$ bound for the expected optimization time.

Theorem 6. *Let $\ell \geq \log(n)$. The expected number of steps until the $(\leq \mu + 1)$ -EA finds a shortest path from u to v for all $(u, v) \in V_\ell^2$ is $O(\ell n^3)$. In particular it holds that the expected optimization time of the $(\leq \mu + 1)$ -EA is $O(n^4)$.*

Proof. Let t_ℓ be the number of steps until the $(\leq \mu + 1)$ -EA has found for all $(u, v) \in V_\ell^2$ a shortest path from u to v . In the proof of Lemma 4 we showed that the probability that t_ℓ is greater than $e\eta\ell n^3$ is $\Pr[t_\ell > e\eta\ell n^3] \leq n^{2-\frac{\eta}{8}}$. Let $\eta := \nu n^i$ for some $\nu \geq 24$ and $i \in \mathbb{N}$. Then,

$$\begin{aligned} \Pr[ev\ell n^{4+i} \geq t_\ell > ev\ell n^{3+i}] &\leq \Pr[t_\ell > ev\ell n^{3+i}] \\ &\leq n^{2-\frac{\nu n^i}{8}} \\ &\leq n^{2-3n^i}. \end{aligned}$$

For $n \geq 2$, the expected number of steps $\mathbb{E}[t_\ell]$ needed to find for all $(u, v) \in V_\ell^2$ a shortest path from u to v is thus

$$\begin{aligned} \mathbb{E}[t_\ell] &= \sum_{t'=1}^{\infty} t' \cdot \Pr[t_\ell = t'] \\ &\leq ev\ell n^3 + \sum_{i=0}^{\infty} \sum_{t'=ev\ell n^{i+3}+1}^{ev\ell n^{i+4}} t' \cdot \Pr[t_\ell = t'] \\ &\leq ev\ell n^3 + \sum_{i=0}^{\infty} ev\ell n^{i+4} \cdot n^{2-3n^i} \\ &= ev\ell n^3 \left(1 + \sum_{i=0}^{\infty} n^{i+3-3n^i}\right) \\ &\leq ev\ell n^3 \left(2 + \sum_{i=1}^{\infty} n^{-n^i}\right) \\ &\leq ev\ell n^3 (2 + 2). \end{aligned}$$

Setting $\ell = n$ we get the upper bound for the expected optimization time. \square

3.2 Lower Bound on the Optimization Time

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

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

For two distinct vertices u, v the unique shortest path from u to v is $((u, u + 1), \dots, (v - 1, v))$ if $u < v$ and $((u, u - 1), \dots, (v + 1, v))$ otherwise. These edge lengths, together with our initialization and selection for replacement, ensure that at any time all individuals in the population consist of a single edge or are a shortest path.

Definition 7. *The distance of two paths is the minimal number of elementary mutations needed to mutate one path into the other. A mutation step crosses a distance of c if the path it chooses to mutate and the one it creates have distance c .*

Note that for the graph K_n with edge lengths w the distance of two shortest paths P_1, P_2 is the size $|E(P_1) \Delta E(P_2)|$ of the symmetric difference Δ of the set of edges $E(P_1), E(P_2)$ of the two paths.

Lemma 8. *For any $c \in \mathbb{N}$, the probability that a mutation step crosses a distance of c is at most $\frac{4c}{e(n-2)^c} \frac{n-2}{n-3} = O(cn^{-c})$.*

Proof. Let P_1 be the shortest path the mutation step chooses for mutation and let P_2 be a shortest path that has a distance of c to P_1 . Each elementary mutation of a sequence of elementary mutations applied to P_1 either decreases or increases the distance of the resulting solution to P_2 . Hence a shortest path P_2 in distance c from P_1 can only be obtained via a sequence of $c + 2i$ elementary mutations for some $i \in \mathbb{N}_0$. In this case, $c + i$ of them decrease and i of them increase the distance of the intermediate solution to P_2 . The probability that a certain of the $c + 2i$ elementary mutations decreases this distance is at most $(n - 2)^{-1}$, since there are at most 2 additions/deletions that achieve the distance reduction out of at least $2(n - 2)$ possible elementary mutations.

Assume in this paragraph that our mutation consists of exactly $c + 2i$ elementary mutations. Then there are at most $\binom{c+2i}{i}$ choices for the $c + i$ ones that reduce the distance to P_2 . In consequence, the probability to end up with P_2 is at most $\binom{c+2i}{i} (n - 2)^{-(c+i)}$.

It is easy to see that there are at most $2c$ shortest paths P_2 that are in distance c of P_1 . Thus, the probability to end up with any shortest path P_2 in distance c of P_1 is at most $2c \binom{c+2i}{i} (n - 2)^{-(c+i)}$.

Recall that the probability that our mutation consists of $c + 2i$ elementary mutations is $(e(c + 2i - 1)!)^{-1}$. Hence the probability that a single mutation

step crosses a distance c is at most

$$\begin{aligned}
\sum_{i=0}^{\infty} \frac{1}{e(c+2i-1)!} \binom{c+2i}{i} \frac{2c}{(n-2)^{c+i}} &= \frac{2c}{e(n-2)^c} \sum_{i=0}^{\infty} \frac{c+2i}{i!(c+i)!} \frac{1}{(n-2)^i} \\
&\leq \frac{4c}{e(n-2)^c} \sum_{i=0}^{\infty} \frac{1}{(n-2)^i} \\
&\leq \frac{4c}{e(n-2)^c} \frac{n-2}{n-3} \\
&= O(cn^{-c}).
\end{aligned}$$

□

Lemma 9. *For any constant k , there exists a constant $c := c(k)$ such that with probability $O(n^{-k})$, during its optimization time the $(\leq \mu + 1)$ -EA will only accept mutation steps that cross at most a distance of c .*

Proof. We know from Theorem 5 that the $(\leq \mu + 1)$ -EA has with high probability an expected optimization time of $O(n^4)$ and from Lemma 8 that a distance of c is crossed with probability $O(cn^{-c})$. Thus, the probability that during the optimization time a step crossing a distance of c is accepted is at most $O(n^{4-c})$. Choosing c appropriately, this probability turns into $O(n^{-k})$. □

Let $P^* := ((1, 2), (2, 3), \dots, (n-1, n))$ be the shortest path in K_n with edge length w from 1 to n . Consider a sequence of mutation steps (each changing at least one edge) that may create P^* . Of these steps consider the last $\lfloor \frac{n-3}{c} \rfloor$ where c is the constant from Lemma 9. Let the paths that are created during these steps be $P_0, P_1, \dots, P_{\lfloor \frac{n-3}{c} \rfloor} = P^*$. Since $|P^*| = n-1$ and since P_j has at most c edges more than P_{j-1} , we have that $|P_0| \geq 2$ and thus all P_j are shortest paths. Thus, these paths fulfill the requirements of the following definition.

Definition 10 (*c*-Trail). *A c -trail $T := (P_0, P_1, \dots, P_{\lfloor \frac{n-3}{c} \rfloor})$ of P^* is a sequence of shortest paths such that P_0 consists of at least 2 edges, $P_{\lfloor \frac{n-3}{c} \rfloor} = P^*$, and for all $j \in [1.. \lfloor \frac{n-3}{c} \rfloor]$, P_{j-1} and P_j have a distance of at most c .*

Since there are at most $(2c)^2$ shortest paths that have a positive distance of at most c from P_j , there are at most $(4c^2)^{\lfloor \frac{n-3}{c} \rfloor}$ such c -trails.

Theorem 11. *With high probability, the optimization time of the $(\leq \mu + 1)$ -EA on K_n with edge lengths w is $\Omega(n^4)$.*

Proof. Let c be the constant from Lemma 9. In order to create P^* the $(\leq \mu + 1)$ -EA has to perform all $\lfloor \frac{n-3}{c} \rfloor$ mutation steps that create P_j out of

P_{j-1} for $j \in [1.. \lfloor \frac{n-3}{c} \rfloor]$ of one of the c -trails of P^* (and the mutation steps leading to P_0 , which we will ignore in this proof). First, we will analyze the number of steps the $(\leq \mu + 1)$ -EA needs to follow one particular c -trail of P^* . Then, we will prove that with high probability the $(\leq \mu + 1)$ -EA will not follow any of the c -trails of P^* in less than $\Omega(n^4)$ steps.

Fix one c -trail $T = (P_0, P_1, \dots, P_{\lfloor \frac{n-3}{c} \rfloor})$ of P^* . We call a mutation step an *improvement* in T if it creates P_j out of P_{j-1} for some $1 \leq j \leq \lfloor \frac{n-3}{c} \rfloor$. If all $\lfloor \frac{n-3}{c} \rfloor$ improvements in T have been done, we say that the $(\leq \mu + 1)$ -EA has followed T .

Let the random variable t' denote the number of steps the $(\leq \mu + 1)$ -EA needs to follow T . For $i \in [1..t']$ define the binary random variables X_i by $X_i = 1$ if the i -th mutation step is an improvement in T . An improvement changes at least 1 and at most c edges of a path. In order to change c' edges, it first has to pick the right individual with probability $\frac{1}{n(n-1)}$ and then change the c' edges with probability $\frac{4c'}{e(n-2)^{c'}} \frac{n-2}{n-3}$ (cf. Lemma 8). Thus, for $n \geq 6$ and for all $x_1, \dots, x_{i-1} \in \{0, 1\}$ we have that

$$\begin{aligned} & \Pr[X_i = 1 | X_1 = x_1, \dots, X_{i-1} = x_{i-1}] \\ & \leq \sum_{c'=1}^c \frac{1}{n(n-1)} \frac{4c'}{e(n-2)^{c'}} \frac{n-2}{n-3} \\ & \leq \frac{4}{en(n-1)(n-2)} \cdot \frac{n-2}{n-3} \cdot \sum_{c'=0}^{c-1} \frac{c'}{(n-2)^{c'}} \\ & < \frac{4c}{en(n-1)(n-2)} \cdot \frac{n-2}{n-3} \cdot \frac{n-2}{n-3} \\ & < \frac{8c}{e(n-1)^3} \end{aligned}$$

for all $i \in [1..t]$. For $i > t'$ define X_i by $\Pr[X_i = 1] = \frac{8c}{e(n-1)^3}$ and for $i \in [1..t]$ define the binary random variables X_i^* by $\Pr[X_i^* = 1] = \frac{4}{e(n-1)^3}$. Let $t := \frac{1}{80c^4}(n-1)^4$. The expected value of $X^* := \sum_{i=1}^t X_i^*$ is

$$\mathbb{E}[X^*] = \sum_{i=1}^t \Pr[X_i^* = 1] = t \frac{8c}{e(n-1)^3} = \frac{n-1}{10ec^3}.$$

If the $(\leq \mu + 1)$ -EA has found P^* in t steps by following the c -trail T , then obviously $X := \sum_{i=1}^t X_i \geq |T| = \lfloor \frac{n-3}{c} \rfloor$. Hence,

$$\Pr[P^* \text{ found in } t \text{ steps by following } T] = \Pr[X \geq |T|].$$

Let $\beta := \frac{|T|}{\mathbb{E}[X^*]}$. Then for $n \geq 5 + 2c$ it holds that

$$\beta \geq \lfloor \frac{n-3}{c} \rfloor \cdot \frac{10ec^3}{n-1} \geq \frac{n-3-c}{c} \cdot \frac{2c}{n-1} \cdot 5ec^2 \geq 5ec^2.$$

Hence, by Lemma 2 and Theorem 1, the probability of finding P^* in $t = \frac{1}{80c^4}(n-1)^4$ steps by following c -trail T is bounded by

$$\begin{aligned}
\Pr[X \geq |T|] &\leq \Pr[X^* \geq |T|] \\
&= \Pr[X^* \geq \beta \mathbb{E}[X^*]] \\
&< (e^{\beta-1} \beta^{-\beta})^{\mathbb{E}[X^*]} \\
&\leq \left(\frac{e}{\beta}\right)^{\beta \mathbb{E}[X^*]} \\
&\leq (5c^2)^{-|T|} \\
&= (5c^2)^{-\lfloor \frac{n-3}{s} \rfloor}.
\end{aligned}$$

Since the $(\leq \mu + 1)$ -EA has to follow one of the c -trails of P^* in order to find P^* , the probability that the $(\leq \mu + 1)$ -EA finds P^* in $t = \frac{1}{80c^4}(n-1)^4$ steps is bounded by

$$\begin{aligned}
\Pr[P^* \text{ found in } t \text{ steps}] &\leq \sum_{T \in \mathcal{T}} \Pr[P^* \text{ found in } t \text{ steps by following } T] \\
&\leq \sum_{T \in \mathcal{T}} (5c^2)^{-\lfloor \frac{n-3}{c} \rfloor} \\
&= \left(\frac{4}{5}\right)^{\lfloor \frac{n-3}{c} \rfloor}.
\end{aligned}$$

Here \mathcal{T} denotes the set of all c -trails of P^* . In the penultimate line we used the fact that there are at most $(4c^2)^{\lfloor \frac{n-3}{c} \rfloor}$ c -trails of P^* . Since the $(\leq \mu + 1)$ -EA has to find P^* to solve the APSP it needs with high probability at least $\Omega(n^4)$ steps. \square

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

4 Upper Bound on the Optimization Time of the $(\leq \mu + 1)$ -GA

We now prove that if we use the $(\leq \mu + 1)$ -GA for the APSP problem, that is, we enrich the $(\leq \mu + 1)$ -EA with a crossover operator, then the expected optimization time drops to $O(n^{3.5+\varepsilon})$ for any $\varepsilon > 0$.

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 can prove the claimed upper bound for all three crossover operators introduced in Section 2.3. However, as the crossover operators we use become more elaborate, for the proof we need to comply to the following restrictions.

R1: Among two shortest paths the fitness function prefers the one consisting of fewer edges. (Needed for \otimes_2 .)

R2: The input graph has unique shortest paths. (Needed for \otimes_3 .)

Given these restrictions, we show 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 $O(n^{3.5+\varepsilon})$.

Lemma 12. *Let $k > 1$. Assume the population \mathcal{I} contains a shortest path for any pair of vertices $(u', v') \in V_k^2$. Let $\ell \in [k + 1..2k]$ and $(u, v) \in V_\ell^2$. Then the following holds.*

- a) *A single execution 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 for all $(u', v') \in V_k^2$, \mathcal{I} contains a shortest path from u' to v' consisting of at most k edges. A single execution of the \otimes_2 -operator generates a shortest path from u to v having at most ℓ edges with probability $\Omega(\frac{(2k+1-\ell)^2}{kn^4})$.*
- c) *Assume R2. A single execution 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. *Claim a)* The \otimes_1 -operator can generate a shortest path from u to v by picking a path P_u starting in u and a path P_v ending in v , such that P_u together with P_v forms a path from u to v . A particular pair (P_u, P_v) is chosen with probability at least

$$(n(n-1))^{-2} = \Omega(\frac{1}{n^4}).$$

This leaves the task of counting the number of pairs that generate a shortest path from u to v . Let $P = ((u, w_1), (w_1, w_2), \dots, (w_{\ell-1}, v))$ be a shortest

path from u to v having ℓ edges. Then, for every vertex $w_i, i \in [\ell - k, k]$, a shortest path from u to w_i and a shortest path from w_i to v are in the population. Hence, there are at least $2k + 1 - \ell$ pairs of paths that the \otimes_1 -operator can combine to a shortest path from u to v . In summary, the probability that a single crossover step generates a shortest path from u to v is at least $\Omega(\frac{2k+1-\ell}{n^4})$.

Claim b) 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. Let $\ell - k \leq j \leq k$. 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. For each $i \in [\ell - k..j]$, \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 $\Omega(\frac{(2k+1-\ell)^2}{kn^4})$.

Claim c) To generate P , the \otimes_3 -operator has to pick a path P_u starting in u , a path P_v ending in v , and numbers $i \in [0..|P_u|], j \in [0..|P_v|]$ such that the first i edges of P_u together with the last j edges of P_v form the path P . The probability that a particular 4-tuple (P_u, P_v, i, j) with $|P_u| \leq k, |P_v| \leq k, i \leq |P_u|, j \leq |P_v|$ is chosen is at least

$$(n(n-1))^{-2}(k+1)^{-2} = \Omega(\frac{1}{k^2n^4}).$$

It remains to count the number of such 4-tuples that generate P . For this, consider two sub-paths of P , one starting at u , the other ending at v . Observe that those sub-paths are also shortest paths. Since we assume all shortest paths to be unique, both sub-paths will be in the population if they consist of at most k edges. If the sum of the numbers of edges of both paths is some $i \in [\ell..2k]$, they have $i - \ell$ edges in common and the number of successful crossover positions is $i - \ell + 1$. The number of pairs of sub-paths that have $i - \ell$ edges in common is $2k + 1 - i$. Hence, the total number of

4-tuples yielding P is at least

$$\begin{aligned} \sum_{i=\ell}^{2k} (i - \ell + 1) \cdot (2k + 1 - i) &= \sum_{i=0}^{2k-\ell} (i + 1) \cdot (2k + 1 - i - \ell) \\ &= \Omega((2k + 1 - \ell)^3). \end{aligned}$$

Thus, the probability that \otimes_3 generates the shortest path P in a single step is at least $\Omega(\frac{(2k+1-\ell)^3}{k^2 n^4})$. \square

Corollary 13. *Let $k > 1$ and $\ell = \frac{3k}{2}$. Assume the population \mathcal{I} contains for any pair of vertices $(u', v') \in V_k^2$ a shortest path. Assuming R1 for \otimes_2 and R2 for \otimes_3 the following holds.*

- a) *Let $(u, v) \in V_\ell^2$. A single execution of the \otimes_i -operator for $i \in \{1, 2, 3\}$ will create a shortest path from u to v with probability at least $\Omega(\frac{k}{n^4})$.*
- b) *The expected number of crossover steps until \mathcal{I} contains for all $(u, v) \in V_\ell^2$ a shortest path from u to v is $O(\frac{n^4 \log(n)}{k})$.*

Proof. Claim a) This follows directly by plugging ℓ into Lemma 12.

Claim b) This proof is similar to the proof of the coupon collector's theorem (cf. [2]). Let $r = |V_\ell^2| - |V_k^2| = O(n^2)$ be the number of paths that have to be found. By Claim a) the first of the sought after paths will be found after an expected number of $O(\frac{n^4}{k} \frac{1}{r})$ steps. If i paths have been found, it will take an expected number of $O(\frac{n^4}{k} \frac{1}{r-i})$ steps until the $(i + 1)$ -st path is found. Hence, to find all r paths takes

$$\sum_{i=0}^{r-1} O\left(\frac{n^4}{k}\right) \frac{1}{r-i} = O\left(\frac{n^4}{k}\right) \sum_{i=1}^r \frac{1}{i} = O\left(\frac{n^4 \log(n)}{k}\right)$$

steps. \square

Theorem 14. *Let $i \in [1..3]$. If the conditions for the \otimes_i -operator hold, then the $(\leq \mu + 1)$ -GA using mutation and \otimes_i -crossover with any constant rate needs an expected number of $O(n^{3.5} \sqrt{\log(n)})$ steps to solve the APSP problem.*

Proof. Let $k := \sqrt{n \log(n)}$. Both the \otimes_i and the mutation operator happen with constant probability and neither can decrease the fitness of the population. Thus, for an upper bound we may consider the steps of one of the operators only. Considering the steps of the mutation operator only, according to Theorem 6, the algorithm will need in expectation at most $O(n^{3.5} \sqrt{\log(n)})$ steps to find for every $(u, v) \in V_k^2$ a shortest path from u to v . (Note that Theorem 6 also holds if a fitness function preferring fewer edges is used.) To find the remaining shortest paths, we only consider the

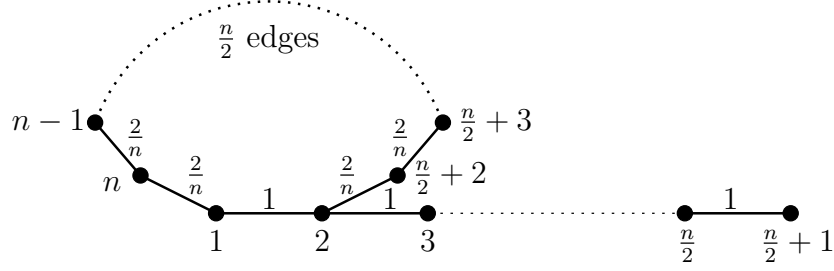


Figure 3: The complete graph K_n with edge lengths w' for which the analysis of the \otimes_2 -operator fails if the fitness function does not prefer individuals with fewer edges. The shown edge lengths apply to both directions of the indicated edge. The edges not shown in the figure are longer than the shortest paths shown.

steps of the \otimes_i -operator and apply Corollary 13 repeatedly until $\ell = n - 1$. Hence the expected number of steps is

$$\begin{aligned}
\sum_{j=\lfloor \log_c(k) \rfloor}^{\lceil \log_c(n) \rceil} O\left(\frac{n^4 \log(n)}{c^j}\right) &= O\left(n^4 \log(n) \sum_{j=\lfloor \log_c(k) \rfloor}^{\lceil \log_c(n) \rceil} \frac{1}{c^j}\right) \\
&= O\left(\frac{n^4 \log(n)}{c^{\lfloor \log_c(k) \rfloor}} \sum_{j=0}^{\lceil \log_c(\frac{n}{k}) \rceil} \frac{1}{c^j}\right) \\
&= O\left(n^{3.5} \sqrt{\log(n)}\right)
\end{aligned}$$

where $c := \frac{3}{2}$. □

The Restrictions R1 and R2

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

To see the necessity of assumption R1 (preference for paths with fewer edges), consider for even n the complete graph $K_n = ([1..n], \{(u, v) \mid u, v \in [1..n], u \neq v\})$ with edge lengths

$$w'(u, v) = \begin{cases} 1 & \text{if } |v - u| = 1 \text{ and } u, v \leq \frac{n}{2} + 1, \\ \frac{2}{n} & \text{if } |v - u| = 1 \text{ and } u, v \geq \frac{n}{2} + 2 \\ \frac{2}{n} & \text{if } (u, v) \in \{(2, \frac{n}{2} + 2), (\frac{n}{2} + 2, 2), (n, 1), (1, n)\} \\ 1 + w_{uv}^2 & \text{else} \end{cases}$$

depicted in Figure 3. Here, w_{uv} is the cost of the shortest path using the edges of length 1 and $\frac{2}{n}$ from u to v .

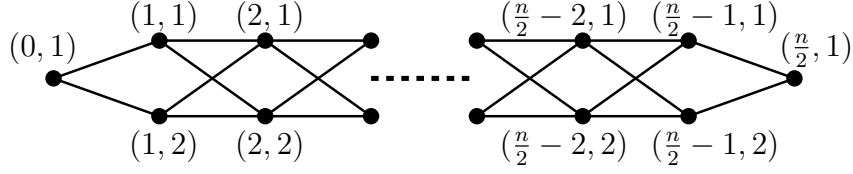


Figure 4: The complete graph K_n'' with edge lengths w'' for which the analysis of the \otimes_3 -operator fails, since the shortest paths are not unique. The edges shown in the figure have length 1 in both directions and the ones not depicted are longer than the shortest paths shown.

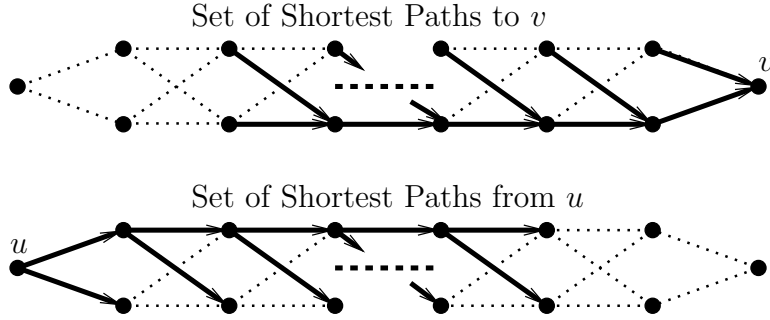


Figure 5: An example for sets of shortest paths in K_n'' that do not overlap enough and thus do not fulfill the requirements for the proof of Lemma 12c).

Assume, as in Lemma 12, that for all vertex pairs $(u, v) \in V_k^2$ a shortest path is in the population \mathcal{I} , and that $\ell \in [k+1..2k]$ and $\ell \leq \frac{n}{2}$. Now consider the computation of a shortest path from $u := 1$ to $v := \ell + 1$ using the \otimes_2 -operator. Two such shortest paths exist, namely P_1 which uses the edge $(1, 2)$ of cost 1 and has ℓ edges and P_2 which uses the $\frac{n}{2}$ edges of cost $\frac{2}{n}$ and has $\ell - 1 + \frac{n}{2}$ edges. If \mathcal{I} contains for the paths from u to i for $i \in [2..k+1]$ the paths using the edge $(1, 2)$, the proof of Lemma 12b) works. However, if \mathcal{I} contains the paths using the $\frac{n}{2}$ edges of cost $\frac{2}{n}$, 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 R2 that the shortest paths are unique is essential for the proof for the \otimes_3 -operator. To see this, consider for even n the complete graph $K_n'' := (V, \{(u, v) | u, v \in V, u \neq v\})$ with $V := [1.. \frac{n}{2} - 1] \times \{1, 2\} \cup \{(0, 1), (\frac{n}{2}, 1)\}$, and with edge lengths

$$w''(u, v) = \begin{cases} 1 & \text{if } |v_1 - u_1| = 1, \\ 1 + w_{uv}^2 & \text{else} \end{cases}$$

depicted in Figure 4. Again, w_{uv} is the length of the shortest path using the edges of length 1 from u to v . Observe that there are many different shortest paths connecting two vertices, all having an equal number of edges. Assume

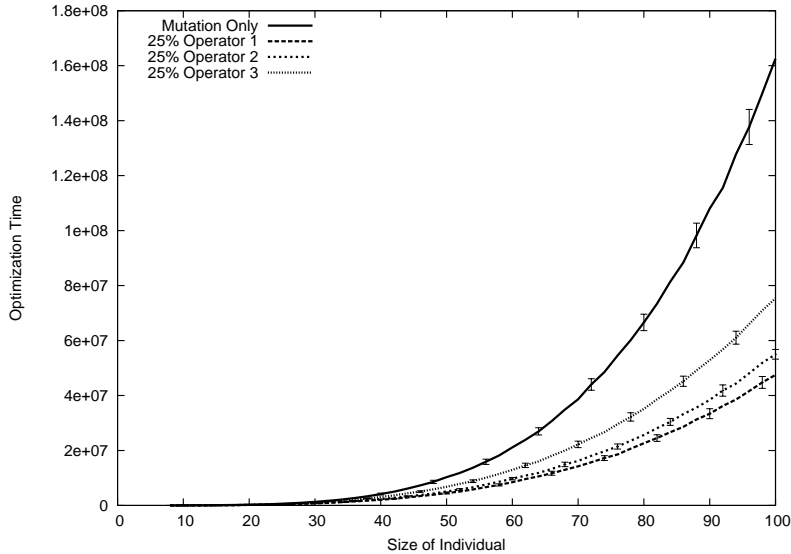


Figure 6: Optimization time for the various crossover operators on the complete graph K_n with edge lengths w (see Section 3.2).

that \mathcal{I} contains the shortest paths from $u := (0, 1)$ to i for $i \in [1..k] \times \{0, 1\}$ and from j to $v := (\frac{n}{2}, 1)$ for $j \in [\frac{n}{2} - k.. \frac{n}{2}] \times \{0, 1\}$ as given in Figure 5. Then for any shortest path from u to v (having $\ell := \frac{n}{2}$ edges) the population will not contain all sub-paths of length up to k , as needed by Lemma 12. Even more, any pair of paths, one starting in u , the other ending in v , will only overlap on at most two vertices.

5 Experimental Studies

In the previous sections we saw that the asymptotic worst case optimization time of the $(\leq \mu + 1)$ -EA is $\Theta(n^4)$, while that of the $(\leq \mu + 1)$ -GA is $O(n^{3.5+\epsilon})$. To show that this difference is in fact noticeable in practice, we implemented the algorithm given in Section 2.4 with the three different crossover operators and ran it on the following three graph classes.

The first are the weighted complete graphs K_n with edge lengths w from Section 3.2, that have edge weights 1 for all edges (u, v) with $|v - u| = 1$, and weight n for all other edges. The second and third graph class are the complete graphs K_n with edge lengths w' and the complete graphs K_n'' with edge lengths w'' used in Section 4 to show why we need additional assumptions in the proofs concerning the operators \otimes_2 and \otimes_3 . 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

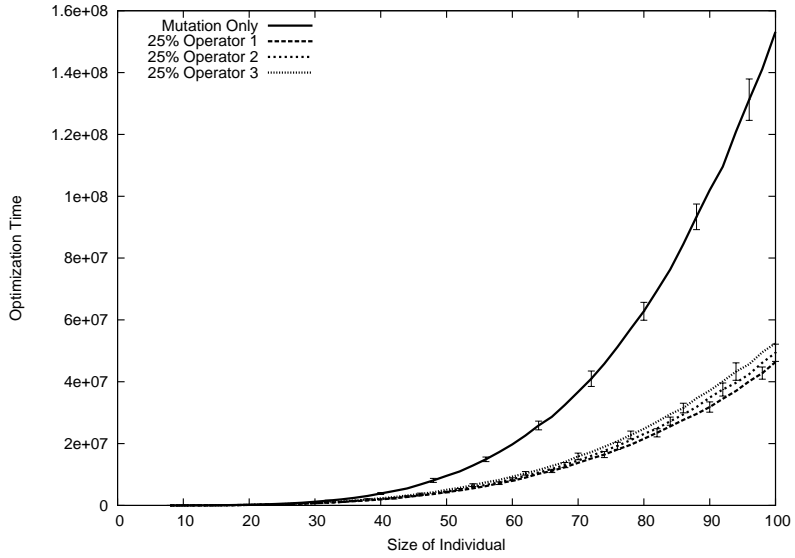


Figure 7: Optimization time for the various crossover operators on the complete graph K_n with edge lengths w' (see Figure 3).

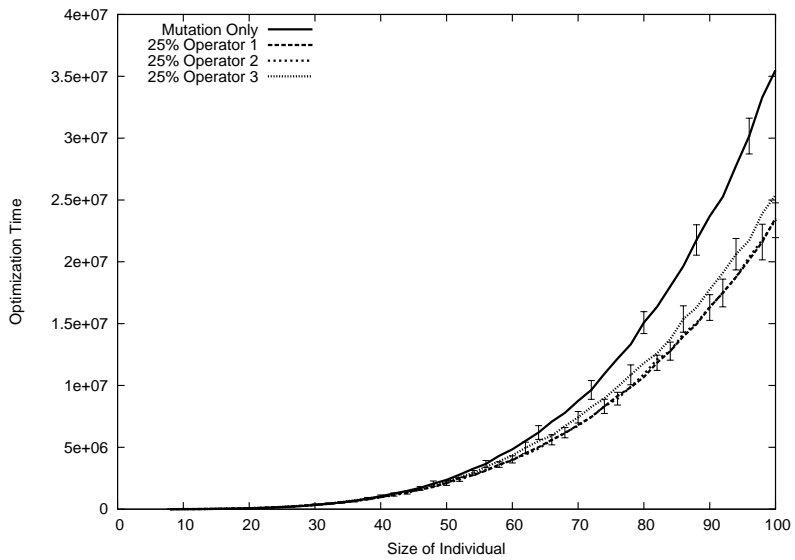


Figure 8: Optimization time for the various crossover operators on the complete graph K_n'' with edge lengths w'' (see Figure 4).

when applying \otimes_3 .

We ran the implementation of our algorithm on all three graph classes

mentioned above, once using mutation only and once for each crossover operator, using it with crossover probability of $\frac{1}{4}$. For all graph classes we considered the graphs having an even number of vertices between 8 and 100. On each instance the algorithm was run 50 times. The average optimization times for the experiments are shown in Figure 6, Figure 7, and Figure 8. To keep the plots legible we plotted the standard deviation for every fourth data point only. For all instances of 40 or more edges, the standard deviation is below 10%.

It can clearly be seen that adding any of the crossover operators does speed up the computation considerably. The results also show that the “bad graphs” K_n with edge lengths w' and K_n'' with edge lengths w'' from Section 4 are not hard to solve for the corresponding crossover operators. In comparison to the other graph classes, the mutation operator is more effective on K_n'' with w'' . The reason is probably that due to the structure of w'' the mutation operator has a lot of possibilities to create shortest paths. Thus, the difference between runs with and without crossover are not quite so noticeable.

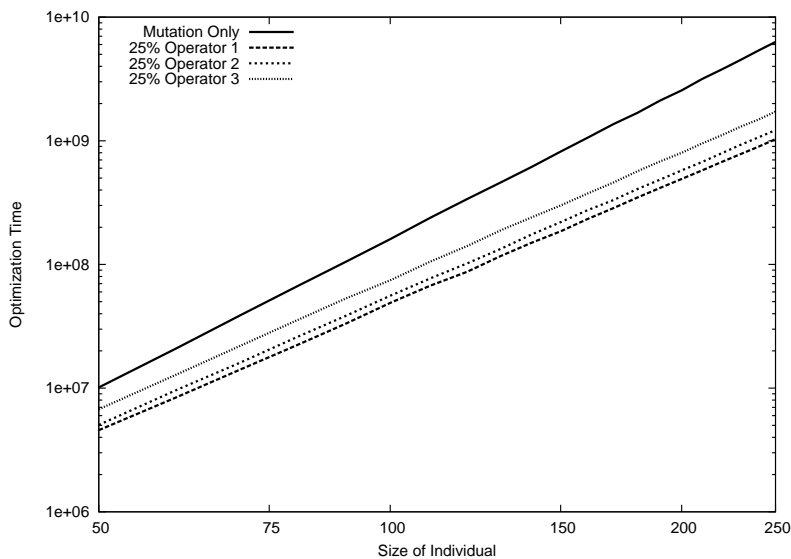


Figure 9: Log-log plots for K_n with edge weights w .

To estimate the different exponents of the runtimes with and without crossover, we additionally ran the algorithm 20 times each on instances of size 50, 60, 70, . . . , 250. We chose these bigger input sizes to weaken the effect of the lower order terms of the runtime. To see the different exponents, we use log-log plots². For any polynomial $f(x) = ax^n + o(x^n)$, a log-log plot

²In other words, both the x and the y -axis are scaled logarithmically.

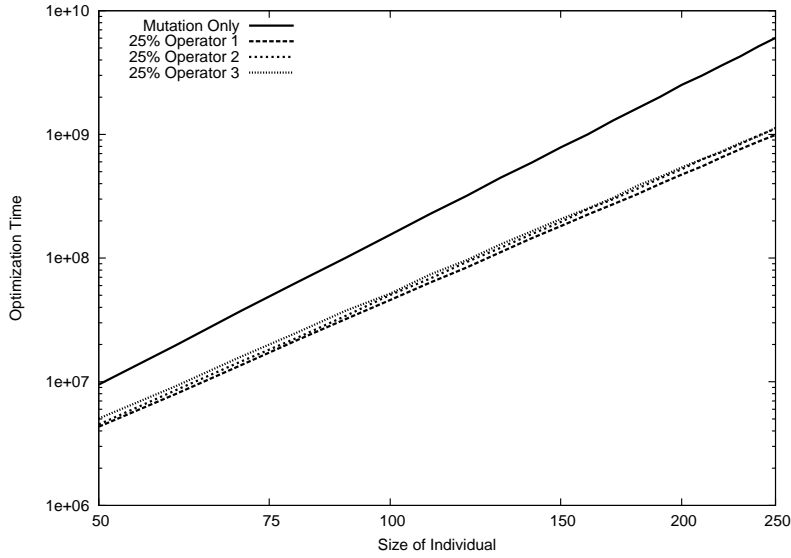


Figure 10: Log-log plots for K_n with edge weights w' .

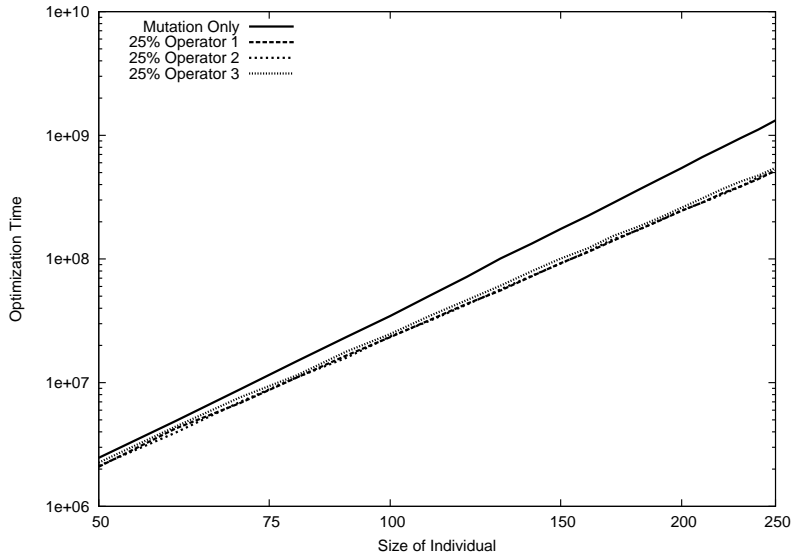


Figure 11: Log-log plots for K_n'' with edge weights w'' .

will plot the function

$$\log(f(\log^{-1}(x))) = \log(a(e^x)^n + o((e^x)^n)) = nx + o(x)$$

hence exposing the exponent of $f(x)$. Figure 9, Figure 10, and Figure 11 show the log-log plots. The difference in the exponent of the runtime between

the mutation-only algorithm and any of the algorithms using crossover can easily be discerned in the plots. We also calculated the slope of the plots. Table 1 shows the results of these calculations. The numbers for K_n using w and w' and K_n'' show that when using only mutation some graphs may indeed cause a quartic runtime. Also, on all three examples crossover seems to be slightly faster than the $O(n^{3.5})$ suggested by our upper bound.

	w	w'	w''
Mutation Only	4.00	4.01	3.90
Crossover (\otimes_1)	3.37	3.38	3.43
Crossover (\otimes_2)	3.41	3.42	3.41
Crossover (\otimes_3)	3.44	3.36	3.41

Table 1: The slope of the log-log plots in Figure 9, Figure 10 and Figure 11.

The experiments also show that \otimes_1 seems to have a slight edge over \otimes_2 which in turn is slightly faster than \otimes_3 . We conjecture that this is caused by the fact that the simpler crossover operators on average combine longer paths than the more complicated ones.

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 $O(n^{3.5+\epsilon})$, 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 algorithm custom tailored for the all-pairs shortest path problem, this result does give a 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, Frank Neumann and Madeleine Theile for many useful discussions.

References

- [1] C. W. Ahn and R. Ramakrishna. A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transactions on 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 *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC)*, pages 1890–1895. IEEE Press, 2007.
- [5] S. Fischer and I. Wegener. The Ising model on the ring: Mutation versus recombination. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation (GECCO)*, volume 3102 of *Lecture Notes in Computer Science*, pages 1113–1124. Springer, 2004.
- [6] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the 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 *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 137–140. IEEE Press, 1999.
- [11] T. Jansen, K. A. De Jong, and I. Wegener. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13:413–440, 2005.
- [12] T. Jansen and I. Wegener. On the analysis of evolutionary algorithms – a proof that crossover really can help. In *Proceedings of the 7th Annual European Symposium on Algorithms (ESA)*, volume 1643 of *Lecture Notes in Computer Science*, pages 184–193. Springer, 1999.

- [13] T. Jansen and I. Wegener. The analysis of evolutionary algorithms – a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002.
- [14] T. Jansen and I. Wegener. Real royal road functions – where crossover provably is essential. *Discrete Applied Mathematics*, 149:111–125, 2005.
- [15] 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.
- [16] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24:1–13, 1977.
- [17] S. Kirkpatrick, D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [18] S. Liang, A. N. Zincir-Heywood, and M. I. Heywood. Intelligent packets for dynamic network routing using distributed genetic algorithm. In *Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO)*, pages 88–96. Morgan Kaufmann, 2002.
- [19] S. Liang, A. N. Zincir-Heywood, and M. I. Heywood. Adding more intelligence to the network routing problem: AntNet and Ga-agents. *Applied Soft Computing*, 6:244–257, 2006.
- [20] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
- [21] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.
- [22] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing? In *Proceeding of the 7th Neural Information Processing Systems Conference (NIPS)*, volume 6 of *Advances in Neural Information Processing Systems*, pages 51–58. Morgan Kaufmann, 1993.
- [23] P. S. Oliveto, J. He, and X. Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4:281–293, 2007.
- [24] Y. Rabani, Y. Rabinovich, and A. Sinclair. A computational view of population genetics. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (STOC)*, pages 83–92. ACM Press, 1995.

- [25] Y. Rabinovich and A. Wigderson. An analysis of a simple genetic algorithm. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 215–221, 1991.
- [26] Y. Rabinovich and A. Wigderson. Techniques for bounding the convergence rate of genetic algorithms. *Random Structures & Algorithms*, 14:111–138, 1999.
- [27] J. Scharnow, K. Tinnefeld, and I. Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3:349–366, 2004.
- [28] T. Storch and I. Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320:123–134, 2004.
- [29] D. Sudholt. Crossover is provably essential for the Ising model on trees. In *Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO)*, pages 1161–1167. ACM Press, 2005.
- [30] S. Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9:11–12, 1962.
- [31] I. Wegener. Simulated annealing beats Metropolis in combinatorial optimization. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 589–601. Springer, 2005.
- [32] C. Witt. Runtime analysis of the $(\mu + 1)$ EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14:65–86, 2006.