Expected Runtimes of Evolutionary Algorithms for the Eulerian Cycle Problem

Frank Neumann
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
24098 Kiel, Germany
fne@informatik.uni-kiel.de

Abstract—Evolutionary algorithms are randomized search heuristics, which are applied to problems whose structure is not well understood, as well as to problems in combinatorial optimization. They have successfully been applied to different kinds of arc routing problems. To start the analysis of evolutionary algorithms with respect to the expected runtime on these problems, we consider the Eulerian cycle problem. We show that a variant of the well known (1+1) EA working on the important encoding of permutations is able to find an Eulerian tour of an Eulerian graph in expected polynomial time. Altering the operator used for mutation in the considered algorithms, the expected optimization time changes from polynomial to exponential.

I. INTRODUCTION

The aim of this paper is to start the analysis of randomized search heuristics on arc routing problems with respect to the expected time until they consider an optimal search point. Evolutionary algorithms belong to the class of randomized search heuristics and have found many applications in solving real-world problems whose structure is not known very well. They also have produced good results on a wide class of problems belonging to the class of NP-hard problems. For general results of randomized search heuristics we refer to Papadimitriou, Schäffer, and Yannakakis [8]. In recent years methods have been developed to analyze simple evolutionary algorithms with respect to the expected runtime and success probability after a fixed number of steps (see Droste, Jansen, and Wegener [1] for an overview).

To get a deeper understanding how evolutionary algorithms work on combinatorial optimization problems, we have to start the analysis of simple problems belonging to the complexity class P. We do not hope to beat the best proposed algorithms for a considered problem. Our main purpose is to get a deeper understanding how evolutionary algorithms work on these simple problems. Such an understanding can help to develop better problem-specific evolutionary algorithms for hard problems as well as help to analyze such algorithms with respect to approximability in expected polynomial time.

The analysis of evolutionary algorithms for combinatorial optimization problems has been started only recently. See Scharnow, Tinnefeld, and Wegener [9] for sorting as the minimization of unsortedness and on shortest paths problems, and Giel and Wegener [3] for maximum matchings.

Here we consider the well-known problem of computing an Eulerian cycle of a given graph with $n$ vertices and $m$ edges (see Skiena [10] for example). This is the simplest problem belonging to the wide class of arc routing problems. It can be solved in worst case run time of $O(m + n)$ by the algorithm of Hierholzer [4].

As already argued, we do not and cannot hope to compete with the best algorithms for the Eulerian cycle problem. This can be different for generalizations of the problem. For example the problem of finding the largest Eulerian subgraph of a given graph and the Mixed Chinese Postman Problem [2] are NP-hard and evolutionary algorithms have a good chance to be competitive on these problems. For other NP-hard variants like the Capacitated Arc Routing Problem evolutionary algorithms have been developed and successfully applied (see e.g. Lacomme et al. [6]). Before we are able to analyze evolutionary algorithms on such problems, we think that it is important to understand evolutionary algorithms that work on simple arc routing problems.

In this paper we consider two simple randomized search heuristics that use a permutation of the edges of the given graph to find an Eulerian cycle. The representation of permutations has succesfully been applied to difficult combinatorial optimization problems like the traveling salesman problem (see Michalewicz and Fogel [7] for an overview). It is important to understand how evolutionary algorithms, using this encoding, work on simple problems. This is one step in the direction to analyze more complicated problems and algorithms using such an encoding. Starting with the first edge in the permutation, we can build up a path. One goal is to estimate the expected time until a path found by the algorithm is lengthend. If the current path is a cycle there may be $\Theta(m)$ operations necessary to lengthen this path. In such a case the algorithm has to walk on a plateau of constant fitness. Jansen and Wegener [5] have studied how evolutionary algorithms can cope with plateaus in the binary search space. In the search space of permutations plateaus have a different structure. Our analysis of the considered algorithms, using jumps in the mutation step, points out the structure of plateaus for the the Eulerian cycle problem in the search space of permutations and how evolutionary algorithms can leave such a plateau. Changing the operator used for mutations, we point out that such a plateau can change to a local optimum with a
large inferior neighbourhood.

After having motivated to analyze randomized search heuristics on the Eulerian cycle problem, we describe in Section II our model of the Eulerian cycle problem. In Section III, we introduce the randomized search heuristics based on jump-operations which we will consider in Section IV and V. In Section IV, we show that a local variant of an evolutionary algorithms is able to compute an Eulerian cycle in expected polynomial time. The analysis is extended to a real evolutionary algorithm in Section V. In Section VI we consider our algorithms using exchanges instead of jumps in the mutation step and prove that the expected optimization times can be in this case infinite and exponential. We finish with concluding remarks.

II. EULERIAN CYCLES

Euler initiated the study of graph theory in 1736 with the famous Seven Bridges problem. The generalization of the Seven Bridges problem can be described as follows and is known as the Eulerian cycle problem. Given an undirected connected graph \( G = (V, E) \) on \( n \) vertices and \( m \) edges, compute a cycle such that every edge is used exactly one time. Euler proved that a tour of all edges in a connected graph without repetition is possible iff the degree of each vertex is even. Such graphs are known as Eulerian graphs. If an Eulerian cycle exists we call \( G \) Eulerian. To find such a cycle we use a permutation of the edges of \( G \). The fitness of a permutation \( \pi \) is the length of the path implied by \( \pi \) when we start with the first edge of it.

The search space \( S_m \) contains all permutations of the edges of \( G \). A search point \( \pi \in S_m \) corresponds to the order to use the edges for the Eulerian tour. Usually a permutation does not correspond to an Eulerian tour. It normally describes a path \( p \) which is part of such a tour. The ideas can be used to define the fitness function \( path(\pi) \) which is appropriate for the Eulerian cycle problem.

We will investigate the following fitness function:

\[
\text{path}(\pi) := \text{length of the path implied by } \pi
\]

where we start with the first edge in \( \pi \) and extend the path, if the edge on the second position has one vertex with the first edge of \( \pi \) in common. This path can be further extended if the third edge has one vertex which is equal to the "free" vertex of the second edge. We can extend the procedure to build up a path of length \( l \) implied by \( \pi \). In the rest of this paper the path will be named by \( p \) and the part of \( \pi \) consisting of the positions \( i = 1, \ldots, m \) will be named by \( q \).

To our opinion the proposed fitness function is the most intuitive one that can be described for the Eulerian cycle problem. The fitness function describes the processing order to use the edges for a tour starting with the edge on position 1. Another advantage of this fitness function is that it can be easily evaluated. If the resulting path is short most of the edges in the permutation do not have to be considered.

For the Eulerian cycle problem algorithms have been designed that compute an Eulerian cycle in linear time. To analyze randomized search heuristics we use the knowledge that has been put into these algorithms. The following algorithm proposed by Hierholzer [4] computes an Eulerian cycle of a given Eulerian graph \( G \) and contains ideas which will be later used in the analysis of our algorithms.

Algorithm 1 (Eulerian Cycle):
1) Find a cycle \( C \in G \)
2) Delete the edges of \( C \) from \( G \)
3) If \( G \) is not empty go to step 1.
4) Construct the Eulerian cycle from the cycles produced in step 1.

Randomized search heuristics do not have the knowledge that the problem can be solved by computing cycles and building up the solution by putting the cycles together. We will see that they are able to compute a single cycle and integrate another cycle if the solution is not optimal. Hence, they follow the idea of the algorithm without having this global knowledge.

III. RANDOMIZED LOCAL SEARCH AND THE (1+1) EA

Randomized local search (RLS) and the (1+1) EA are perhaps the simplest randomized search heuristics that can be considered. They work on a population of size 1 and only use mutation to produce one single new individual in each generation. We consider the jump operator for mutation which has already been discussed by Scharnow, Tinnefeld, and Wegener [9] for the sorting problem. A jump operation \( jump(i,j) \) executed on a permutation \( \pi \) produces a new permutation \( \pi' \) by putting the element on position \( i \) at position \( j \) and shifting the remaining elements in the appropriate direction. For RLS in one mutation step exactly one jump operation is executed. The executed jump is chosen according to the uniform distribution. We can describe RLS as follows.

Algorithm 2 (Randomized Local Search (RLS)):
1) Choose \( \pi \in S_m \) uniform at random.
2) Choose \( i \) and \( j \) uniform at random and define \( \pi' \) by jumping the element at position \( i \) to position \( j \) and shifting the elements between position \( i \) and position \( j \) in the appropriate direction, e.g. \( jump(2,4) \) applied to \( (2,1,3,5,4) \) produces \( (2,3,5,1,4) \).
3) Replace \( \pi \) by \( \pi' \) if \( path(\pi') \geq path(\pi) \).
4) Repeat Steps 2 and 3 forever.

Real evolutionary algorithms use a mutation operator where more than one operation is possible in a mutation step. The (1+1) EA using the encoding of permutations is adopted from the well-known (1+1) ES (evolution strategy) and differs from RLS by the chosen mutation operator.

Algorithm 3 (Mutation operator of (1+1) EA):

1) Find a cycle \( C \in G \)
2) Delete the edges of \( C \) from \( G \)
3) If \( G \) is not empty go to step 1.
4) Construct the Eulerian cycle from the cycles produced in step 1.
2') Define \( \pi' \) in the following way. Choose \( s \) according to a Poisson distribution with parameter \( \lambda = 1 \) and perform sequentially \( s + 1 \) jump operations to produce \( \pi' \) from \( \pi \).

In applications, we need a stopping criterion. Here we are interested in the expected number of fitness evaluations \( E[T] \) until \( \pi \) represents an Eulerian tour. This is the expected optimization time of the considered algorithms.

Note that we can solve the decision variant, whether a given graph is Eulerian, by the proposed randomized search heuristics with an error probability, that depends on the number of steps that are executed when \( E[T] \) is known. By Markov inequality the probability to have more than \( r \cdot E[T] \) steps on an Eulerian graph to reach the optimum can be bounded by \( 1/r \). If we stop the algorithm after this number of steps and give the answer that there is an Eulerian cycle in the case that the fitness value is \( m \) and otherwise give the answer no, we can only give the wrong answer in the case that \( G \) is Eulerian. In this case the probability to err is upper bounded by \( 1/r \).

### IV. Analysis of RLS

In the following we show an upper bound of \( O(m^5) \) on the expected optimization time for RLS to produce an Eulerian cycle working on the proposed fitness function.

**Theorem 1:** The expected time until RLS working on the fitness function \( \text{path} \) constructs an Eulerian cycle is bounded by \( O(m^5) \).

**Proof:** A search point \( \pi \) can take values from \( \{1, \ldots, m\} \), where the optimum is reached if \( \text{path}(\pi) \) equals \( m \). W.l.o.g. the path \( p \) implied by \( \pi \) is of the form \((v_0, v_1), \ldots, (v_{l+1}, v_l)\) and has length \( l \in \{1, \ldots, m - 1\} \). If \( v_0 \neq v_l \) holds there is an edge incident to \( v_l \) which can be placed after \((v_l, v_0)\). Such a jump lengthens the path and has probability \( \Omega(1/m^2) \). In this case the expected time for an improvement of \( \pi \) is bounded by \( O(m^2) \).

If \( v_0 \) and \( v_l \) are equal the analysis for an improvement is more complicated. In this case \( p \) is a cycle \( C \). If the graph is Eulerian and \( p \) is not an Eulerian tour there is at least one vertex \( v_k \) on \( C \) which is also a vertex on another cycle \( C' \) having only \( v_k \) with \( C \) in common (see figure 1). We want to show that a path \( p \) with length \( l \) can be lengthened to a path of length at least \( l + 1 \) in expected time \( O(m^4) \).

**Claim 1:** Let \( p \) be a cycle described by \( \pi \) of length \( l \in \{1, \ldots, m - 1\} \). The expected time to produce a path of length at least \( l + 1 \) is bounded by \( O(m^4) \).

**Proof:** We consider \( cm^4 \) steps of RLS, where \( c \) is a constant large enough. If we can lower bound the probability to produce a path of length at least \( l + 1 \) in \( cm^4 \) steps by a constant \( \alpha \), the expected time for an improvement is at most \( \frac{cm^4}{\alpha} = O(m^4) \).

We inspect the case where \( p \) is a cycle which corresponding permutation \( \pi \) does not start with the vertex \( v_k \). Jumps that do not affect the position \( 1, \ldots, l + 1 \) in \( \pi \) do not change the fitness value. A jump which affects position \( l + 1 \) has the possibility to lengthen the path and is essential for an improvement. If we have jumps that place elements of \( p \), not at position \( 1 \) or \( l \), at another position we get a shorter path. These steps are not accepted by the algorithm. The jump of the edge at position \( 1 \) to a position \( j \), \( j \neq l \), shortens the path by at least one and is also not accepted. The same holds for the jump of an edge at position \( i \in \{l + 1, \ldots, m\} \) to a position \( j \in \{1, \ldots, l\} \). Hence, the algorithm only accepts the two jump operations \( j\text{ump}(1,l) \) and \( j\text{ump}(l,1) \).

We call a mutation step relevant, if the current path \( p \) is changed and accepted. As we have exactly \( 2 \) jumps which are relevant we get a probability of \( \frac{2}{m(m-1)} \) for a relevant step. The expected number of relevant steps in \( cm^4 \) steps is therefore \( \frac{2cm^4}{m(m-1)} \). Thus, we can bound the probability to have less than \( c'm^2 \), \( c' \) a positive constant depending on \( c \), relevant steps in \( cm^4 \) steps by \( e^{-\Omega(m^2)} \).

We estimate the probability to construct a permutation in \( M = c'm^2 \) relevant steps, which starts with the edge \((v_k, v_{k+1})\). It is our aim to construct a path \( p^* = (v_k, v_{k+1}), \ldots, (v_{l+1}, v_l), \ldots, (v_k, v_{k+1}) \). If we have not reached such a path there is exactly one jump which places \((v_k, v_{k+1})\) one position further to the left and one which places \((v_k, v_{k+1})\) one position further to the right. The probability to place \((v_k, v_{k+1})\) further to the left is in each relevant step \( \frac{b}{2} \).

Let \( X \) be the number of steps to the left in \( M \) steps, and let \( b \) be a constant. The probability of exactly \( r \) steps to the left in \( M \) relevant steps equals

\[
\Pr(X = r) = \left( \frac{M}{r} \right) \left( \frac{1}{2} \right)^r \left( 1 - \frac{1}{2} \right)^{M - r}
\]

\[
\leq \left( \frac{M}{M/2} \right)^{M/2} \left( \frac{b}{M/2} \right)^{M/2+1/2} \frac{1}{2^{M/2+1/2}}
\]

\[
= \frac{b \cdot M^{1/2}}{2M}
\]

The probability to have less than \( \frac{M}{2} + \frac{m}{2} \) steps to the left is bounded above by \( \frac{1}{2} + \frac{m}{2} + \frac{1}{2m} = \frac{1}{4} \). Hence, we get a probability of at least \( \frac{1}{4} \) to produce \( p^* \).
As \( v_k \) is also a vertex in another cycle \( C' \) there are two edges \((v_k, v_s)\) and \((v_k, v_t)\) in \( C' \). If we place one of these edges at position \( l + 1 \) we have lengthended the path and achieved an improvement. On the other hand there are exactly two relevant jumps which place \((v_k, v_{k+1})\) at a position other than position 1. This yields a probability of \( \frac{2}{m} \) for an improvement in the next relevant step. Altogether we get a probability of at least \( \alpha = 1 - \left( \frac{2}{m} + \frac{2}{m} + o(1) \right) = \frac{2}{m} - o(1) \) to lengthen the path in \( cm^4 \) steps which implies an expected time for an improvement of \( O(m^4) \).

As there are at most \( m - 1 \) improvements we get an upper bound of \( O(m^5) \) for the expected optimization time of RLS.

The most costly improvements are improvements where a cycle has to be revolted. There are instances of the problem where such an operation is essential for an improvement. Consider the graph \( G' = (V', E') \) consisting of \( n \) vertices and \( m = n + 1 \) edges. The edge set consists of two cycles \( C' \) and \( C '' \), both of length \( \Theta(m) \), having only the vertex \( v_k \) in common.

**Theorem 2:** Running RLS on the instance \( G' \) an improvement revolving a cycle by \( \Theta(m) \) operations is necessary with probability at least \( \frac{1}{m} - o(1) \).

**Proof:** After random initialization each edge of \( G' \) has probability \( 1/m \) to be on the first position of \( \pi \). As there are \( o(m) \) edges which have a distance of \( o(m) \) to \( v_k \), the probability to have an edge with distance \( o(m) \) to \( v_k \) is bounded above by \( o(1) \). Hence, with probability \( 1 - o(1) \) we have at position 1 an edge which has a distance \( \Omega(m) \) to \( v_k \). W.l.o.g. we assume that the edge on the first position is part of the cycle \( C = \{e_1, \ldots, e_r\} \).

Each edge of \( G' \) has at most 4 edges with which it has a vertex in common. The probability that the edges at position 1 and 2 have after initialization one vertex in common is bounded by \( O(1/m) \). The probability to have a path of length at least 2 is therefore bounded by \( O(1/m) \). Hence, we start with probability \( 1 - o(1) \) with a path of length 1 that has a distance \( \Omega(m) \) to \( v_k \). If we start with a path of length 1 we accept in the next step each jump operation. The expected time to produce a path \( p \) of length 2 is bounded by \( O(m^2) \). As there are \( o(m) \) edges which have a distance of \( o(m) \) to \( v_k \), the probability to have a path \( p \) of length 2 which consists of such an edge is bounded by \( o(1) \).

We consider the case where \( p \) has length 2. A path of lengths at least 3 is produced within \( O(m^2) \) steps of the algorithm. Let \( e_{i_1} \) and \( e_{i_{i+1}} \) be the edges on positions 1 and 2 if we start with a path of length 2. The jump operations \( jump(p, 1, 2) \) and \( jump(p, 2, 1) \) are relevant but only improving if the edge on the third position of \( \pi \) is either \( e_{i_1} \) for \( jump(p, 1, 2) \) or \( e_{i_{i+2}} \) for \( jump(p, 2, 1) \). Such an operation can shortend the distance of the start or end vertex of \( p \) to \( v_k \) by at most 1. Furthermore, the jump of edge \( e_{i_1} \) to position 2 is relevant but only improving if the edge \( e_{i_{i+2}} \) is already on the third position. Each of these non improving jumps can shortend the distance of the start or end of \( p \) to \( v_k \). The expected number of these relevant steps in \( O(m^2) \) steps of the algorithm equals \( \Theta(1) \). By Markovs’ inequality the probability to have \( \omega(1) \) steps that shorten the distance to \( v_k \) is bounded by \( o(1) \).

If we have a path of length \( l, l \geq 3 \), that does not start or end with an edge containing \( v_k \) there are exactly two jumps (one jump to position 1 and one to position \( l + 1 \)) that lengthen the path. If \( p \) has length \( l \) and ends with the vertex \( v_k \), there are three possibilities to lengthen the path at \( v_k \). One jump that lengthens \( p \) by the other edge of \( C \) incident to \( v_k \) and two possibilities to lengthen the path by an edge of \( C' \). Hence, the probability to use the edge of \( C \) for the extension of \( p \) at \( v_k \) is \( \frac{1}{m} \). After this extension has at least 2 edges, we only accept changes of \( p \) if we lengthen the path. The cycle \( C \) having length \( r \) is produced within \( O(r) \) improvements. As an improvement can only be reached by a jump to the first postion or a jump to position \( l + 1 \), the expected number of improvements changing the edge on the first position of \( \pi \) equals \( \frac{m}{r} \). By Chernoff bounds the probability to have more than \( \frac{m}{r} \) improvements with jump to position 1 is bounded by \( e^{\frac{\Omega(m)}{m}} \). Hence, the probability to construct a cycle \( C \) where start and end points have distance \( \Omega(m) \) to the vertex \( v_k \) is at least \( \frac{1}{m} - o(1) \). To improve the solution we have to revolve the cycle by \( \Theta(m) \) operations to one direction. Hence, an improvement revolving a cycle by \( \Theta(m) \) operations is necessary with a probability of at least \( \frac{1}{m} - o(1) \).

V. Analysis of the (1+1) EA

To analyze the expected optimization time of the (1+1) EA on the fitness function \( path \) we have to consider the effect of more than one jump. We work under the assumption that \( p \) is a cycle which is not optimal. Otherwise, the probability of an improvement in the next step is at least \( \frac{1}{m+1} \), because single jump can lengthen the path. This gives an expected time for an improvement of \( O(m^2) \). To get an upper bound on the expected optimization time, it is important to analyze relevant mutations with \( k > 1 \) jumps that affect the part \( p \) of \( \pi \). We say that we have a relevant mutation with \( k \) jumps, if the mutation changes \( p \) by \( k \) jumps and is accepted. First, we consider relevant mutations with more than 3 jumps in one mutation step. Later, we analyze relevant mutations with 2 and 3 jumps in one step.

**Lemma 1:** Let \( p \) be a cycle, which is not an Eulerian cycle. The probability to have in \( O(m^2) \) steps a relevant mutation, which changes \( p \) by at least 4 jumps and does not improve the fitness value, is bounded above by \( o(1) \).

**Proof:** The probability to have a mutation with exactly \( k \) jumps equals \( \frac{1}{m(k^2-1)} \). Hence, the probability to have \( k = \Omega(m) \) jumps in one mutation step is exponentially small and this also holds for a polynomial number of mutation steps. In the rest of this proof, we work under the assumption that \( k = o(m) \) holds.

We consider the different cases of relevant jumps \( jump(i, j) \) that change \( p \). There may be other jumps in the part \( q \) of \( \pi \),
but these additional jumps only lead to a smaller probability.

If \( i \in \{1, \ldots, l \} \wedge j \in \{1, \ldots, l \} \) holds the jump has only influence on the path \( p \). If we have 1 jump in the mutation step we get a probability \( \Theta(1/m^3) \) for a relevant step that shifts \( p \) to the left or to the right. If there is more than one jump in one mutation step \( p \) is eventually shifted more than one position to the left or to the right. W.l.o.g. we assume that \( p \) is shifted to the left by putting the first \( r \) edges \( e_1, \ldots, e_r \) of \( p \) to the end of \( p \). The first edge in this sequence of jumps has to jump to position \( l \). Edge \( e_i, i \geq 2 \), has to jump to the position of \( e_j \), where \( e_j \) is the edge with the greatest index smaller than \( i \), that has already been placed at the end of \( p \). If such an \( e_j \) has not yet been placed at the end of \( p \), \( e_i \) has to jump to the position before the edge \( e_k \), where \( k \) is the smallest index greater than \( i \). If both edges \( e_j \) and \( e_k \) already exist at the end of \( p \), the position to which \( e_i \) has to jump is for both cases the same. Hence, for each edge \( e_i \) there is exactly one possible position to jump to and we can estimate the probability of a success in one step by

\[
\frac{r!}{e(r-1)!(m(m-1))^r} = \frac{r}{e(m(m-1))^r}
\]

If \( r \geq 3 \) the probability of such a mutation in \( O(m^4) \) steps is bounded by \( O(1/m^3) \).

If \( p \) contains cycles of the kind \( c = ((u_1, u_2), (u_2, u_3), \ldots, (u_r, u_1)) \) there is another possibility to have sequences of relevant jumps that are not improving. Such a mutation is accepted if it changes the order of cycles in \( p \). As we are considering undirected graphs cycles have length at least 3. A sequence of \( t \geq 3 \) jumps is accepted if it displaces a cycle \( c \) from \( p \) at another position of \( p \). There are \( O(l) \) possible positions where \( c \) can be placed. The number of possible cycles in \( p \) is bounded by \( O(l) \). We inspect the displacement of a cycle \( c \) with length \( t \) in more detail. \( c \) has to “jump over” another cycle, so we have to execute at least \( t \) jumps. There are \( t! \) possibilities to order the \( t \) edges in \( c \) with respect to the jumps. W.l.o.g. we assume that \( c \) is shifted to the right in \( p \). Let \( e_i \) be the edge in the first jump. As we want to have a cycle after the mutation step, \( e_i \) has to be adjacent to \( u_1 \) after the first jump. This is only possible if we jump \( e_i \) to a position that contains an edge with a vertex \( u_1 \). The second jump of our sequence determines how to traverse the cycle. The edge can be placed before or behind \( e_i \). As the direction to traverse the cycle is determined by the second jump, the positions for the other edges to jump to are fixed. Hence, the probability to displace a cycle at another fixed position in \( p \) is at most

\[
\frac{2 \cdot t!}{e(t-1)!(m(m-1))^t} = \frac{2 \cdot t}{e(m(m-1))^t}
\]

As the number of cycles as well as the number of possible positions is bounded by \( O(l) \) the probability to displace a cycle of length \( t = \Theta(1) \) in \( O(m^4) \) steps is bounded by \( O(1/m^{2t}) \). If \( t = \omega(1) \) holds, this probability is bounded by \( O(1/m^{2t}) \). It follows that the probability to have a mutation in \( O(m^4) \) steps, which displaces a cycle of length \( t \geq 4 \), is bounded above by \( O(1/m^2) \).

There may be combinations of shifting \( p \) to the left or to the right and a displacement of a cycle. Such a mutation has to consist of a displacement of at least one cycle of length \( 3 \) and at least one jump operation that shifts \( p \) to the left or to the right. For such an event we get a probability of \( O(1/m^2) \) in \( O(m^4) \) steps because there are only two possibilities for the shifting operation.

We examine the case where \( i \in \{1, \ldots, l \} \wedge j \in \{l + 1, \ldots, m \} \) holds. In this case an edge from \( p \) jumps to a position of \( q \). \( p \) describes a cycle iff it displaces a cycle from \( p \). Consider a cycle \( c \) with \( t \) edges that should be displaced from \( p \). For each edge there are \( O(m-l) \) position to jump to. Hence, the probability to displace the edges of \( c \) is upper bounded by

\[
\frac{e(t-1)!(m(m-1))^t}{e(t)(m(m-1))^t} = \frac{t(m-l)^t}{e(m(m-1))^t}
\]

For each \( t \geq 3 \) this is bounded by \( O(1/m^3) \). As there are at most \( O(l) \) cycles that can be displaced from \( p \), the probability to displace a cycle from \( p \) in the next step is bounded by \( O(1/m^2) \). To accept such a displacement, edges of \( q \) have to be integrated into \( p \) and form a cycle. There are \( O(m-l) \) cycles that can be generated from the edges of \( q \). Cycles can be generated at \( O(l) \) positions of \( p \). Hence, the probability that this happens in the next step is bounded by

\[
\frac{O(l)O((m-l)t)!}{e(t-1)!(m(m-1))^t} = \frac{t}{\Omega(m^{2(t-1)})}
\]

For each \( t \geq 3 \) this is upper bounded by \( O(1/m^4) \). Thus, the probability to displace one cycle from \( p \) and integrate a new cycle into \( p \) in \( O(m^4) \) steps is bounded by \( O(m^2) \).

If \( i \in \{l + 1, \ldots, m \} \wedge j \in \{1, \ldots, l \} \) holds there is an edge that jumps from \( q \) to \( p \). Such a sequence of jumps can only be accepted if it includes a new cycle \( c' \) into \( p \). To have a relevant mutation that does not lead to an improvement, the fitness value has to be unchanged. To achieve this we have to displace a cycle from \( p \) to \( q \) or to displace the edge which is after the integration of \( c' \) at position \( l+1 \). The displacement of the edge at position \( l+1 \) can only be done by the operations \( jump(p, l+1) \) and \( jump(l+1, *) \) where \( * \in \{l+2, \ldots, m \} \) holds. The probability of such a jump equals \( 2^{m(m-1)} \). Hence, the probability to integrate a new cycle into \( p \) and displace one cycle of \( p \) in \( O(m^4) \) steps is bounded by \( O(1/m^2) \) and the probability to integrate one cycle and displace the edge at position \( l+1 \) is bounded by \( O(1/m) \).

As the probability for each \( k \geq 4 \), \( k = o(m) \), to have \( k \) relevant jumps in \( O(m^4) \) mutation steps is bounded by \( O(1/m) \), we can bound the probability to have a mutation with 4 or more jumps that is not improving in \( O(m^4) \) by \( o(m) \cdot O(1/m) = o(1) \).

It remains to examine the effect of relevant mutations with 2 or 3 jumps that change \( p \). Our aim is to bound the expected number of such mutations in \( O(m^4) \) steps.
from above by a constant. Then we will see later that the
(1+1) EA is able to compensate the effect within \( O(m^4) \) steps.

**Lemma 2:** Let \( p \) be a cycle, which is not an Eulerian cycle.
The expected number of non improving relevant mutations in
\( O(m^4) \) steps with 2 or 3 jumps that affect \( p \) is bounded by
\( O(1) \) with probability \( 1 - o(1) \).

**Proof:** We only have to consider the case where \( i \in \{1, \ldots, l\} \) and \( j \in \{1, \ldots, l\} \) holds for jump \((i, j)\), because the other cases have already been considered in Lemma 1. If we have 2 jumps in one step we can only shift the beginning of \( p \). If we have 3 jumps in one step we can displace a cycle of length 3 or shift the beginning of \( p \). Taking the arguments of the previous lemma into account, we get an expected number of mutations that displace a cycle of length 3 or shift the beginning of \( p \) by 3 jumps of \( O(1) \). By Markov’s inequality the probability to have \( \omega(1) \) mutations of that kind is bounded by \( 1/\omega(1) = o(1) \) from above.

We have seen that the effect of more than one jump operation is small with probability \( 1 - o(1) \). This can be used to prove an upper bound on the expected runtime of the \((1+1) \) EA.

**Theorem 3:** The expected time until \((1+1) \) EA working
on the fitness function path contracts an Eulerian cycle is
bounded by \( O(m^5) \).

**Proof:** We consider the path \( p \) implied by the current
search point \( \pi \). If it is not a cycle, the probability of an
improvement in the next step is at least \( \frac{1}{m+1} \), which leads
to an expected time for an improvement of \( O(m^2) \). If \( p \) is
a cycle \( C \) and \( p \) is not optimal there is an other cycle \( C' \)
that has one vertex \( v_k \) with \( C \) in common. The probability
to have a relevant mutation with 4 or more jumps that is not
improving in \( O(m^4) \) steps is bounded by \( o(1) \). Non improving
relevant mutations with 2 or 3 jumps shift the edge \((v_k, v_{k+1})\)
in \( O(m^4) \) steps \( O(1) \) positions away from position 1 with
probability \( 1 - o(1) \).

Considering \( cm^4 \) steps we have at least \( M = c'm^4 \) relevant
mutations consisting of one jump. If we have at least \( \frac{M}{m+1} + \frac{1}{m+1} + O(1) \) mutations that shift \((v_k, v_{k+1})\) to the left we reach a path \( p^* \) that starts with \((v_k, v_{k+1}) \). The probability that we do not reach \( p^* \) is bounded by \( \frac{1}{m+1} + O(1/m) \). Hence, the probability to reach \( p^* \) is at least \( \frac{1}{m+1} - o(1) \) and the probability that the next step is improving is at least \( \frac{1}{m+1} - o(1) \). It follows that the probability to have an improvement in \( cm^4 \) steps is at least \( \frac{1}{m+1} - o(1) \) which leads to an expected time of \( O(m^4) \) for an improvement. As the number of improvements until an Eulerian cycle has been constructed is bounded by \( O(m) \), the expected optimization time until \((1+1) \) EA finds an Eulerian cycle is \( O(m^5) \).

As in the case of RLS, improvements revolving a cycle
\( \Theta(m) \) steps into one direction dominate the runtime of
the \((1+1) \) EA. We consider the instance \( G' \) described in
Section IV to prove that such steps are necessary.

**Theorem 4:** Running the \((1+1) \) EA on the instance \( G' \),
an improvement revolving a cycle by \( \Theta(m) \) operations is
necessary with probability at least \( \frac{1}{3} - o(1) \).

**Proof:** As the expected time for an improvement is bounded by \( O(m^2) \) if \( p \) is a path, we construct an Eulerian cycle or one of the cycles \( C \) and \( C' \) in \( O(m^3) \) steps. The probability of a relevant step in \( O(m^3) \) steps of the algorithm, which changes \( p \) by \( t \geq 2 \) jumps is bounded by \( \frac{O(m^3)}{O(m^3)} = O(1/m^2) = O(1/m) \). As the probability to have a mutation consisting of \( \Omega(m) \) jumps is exponentially small, the probability to have a relevant mutation with more than \( 1 \) jump in \( O(m^3) \) is bounded by \( o(m) \cdot O(1/m) + o(1) = o(1) \) from above. Hence, the \((1+1) \) EA behaves with probability \( 1 - o(1) \) like RLS and this proves the theorem.

**VI. MUTATION USING EXCHANGE OPERATIONS**

Many evolutionary algorithms based on the encoding of
permutations use an exchange operation for mutation. The
operation \( exchange(i, j) \) executed on a permutation \( \pi \) consists of
exchanging the elements at position \( i \) and \( j \) in \( \pi \). The
algorithms RLS* and \((1+1) \) EA* can be defined as in Section III
using exchanges instead of jumps. Note, that an exchange
operation \( exchange(i, j), i < j \), can be simulated by the
two jumps \( jump(i, j) \) and \( jump(j-1, i) \). The exchange
of two elements \( i \) and \( j \) has no affect on the positions of other
elements in the permutation.

We have shown in Section IV and V that RLS and the
\((1+1) \) EA have to walk on a plateau of size \( \Theta(m) \) with a
probability bounded below by a constant. Considering RLS*
and the \((1+1) \) EA this plateau changes to a local maximum
with a large neighbourhood having a smaller fitness value.
We show that this leads to an infinite expected optimization
time for RLS* and an exponential optimization time for the
\((1+1) \) EA* on the instance \( G' \).

**Theorem 5:** RLS* working on the fitness function path
has an infinite expected optimization time on \( G' \).

**Proof:** W.l.o.g. we assume the path starts with an edge
of \( C \). With probability \( 1 - o(1) \) the path \( p \) has length 1 and
the edge on the first position of \( \pi \) has distance \( \Theta(m) \) to \( v_k \). After
the next improvement \( p \) has distance \( \Theta(m) \) with probability
\( 1 - o(1) \), because there are \( o(m) \) edges having distance \( o(m) \)
to the vertex \( v_k \).

If \( p \) is a path of length \( l \) that does not contain the vertex \( v_k \)
there is exactly one exchange operation exchanging the edge
at position \( l + 1 \) with another edge of \( q \) that lengthens the
path. If \( p \) ends with an edge containing \( v_k \), there are exactly
3 possibilities to lengthen the path. One of them lengthens the
path by an edge of \( C \). Each of these three possibilities results
in an extension of \( \Theta(m) \) edges. Hence the probability to
use the extension of \( C \) is at least \( \frac{1}{3} - o(1) \). If we use this
extension we construct the cycle $C$ in an expected number of $O(m^3)$ steps. After that the start and end points of $p$ have distance $\Omega(m)$ to $v_k$ with probability at least $\frac{1}{m} - o(1)$. If this happens there is no relevant mutation, because each exchange operation affecting one edge of the cycle destroys the cycle and shortens the path. Hence, the optimization time is infinite with probability at least $\frac{1}{m} - o(1)$ which yields the theorem.

The (1+1) EA* using exchanges instead of jumps has the possibility to leave such a trap. But this only happens with a small probability. We show that the (1+1) EA* has an exponential expected optimization time on the fitness function $p_{th}$.

**Theorem 6:** The (1+1) EA* working on the fitness function $p_{th}$ has an exponential expected optimization time on $G'$.

**Proof:** As RLS* the (1+1) EA* constructs $p$ describing a cycle which is not Eulerian and whose start vertex has distance $\Omega(m)$ to $v_k$ in $O(m^3)$ steps with probability at least $\frac{1}{m} - o(1)$.

W.l.o.g. we assume that the cycle $C$ has been constructed. An exchange $(i, j)$ only affects the positions $i$ and $j$. Edges at other positions are left unchanged. There are only two possibilities to construct another path $p'$ that differs from $p$ and is accepted. The first possibility is to change the position of at least one edge in $p$. To have a relevant step that changes the position of at least one edge, we have to displace all the other edges of $C$. Hence, for such a step $\Theta(m)$ exchange operations are necessary. The probability that this happens in one step is exponentially small.

The second possibility is to integrate the circle $C'$ into the path. This is only possible if all the edges of $C'$ are placed between the two edges containing the vertex $v_k$ in $p$. As $C'$ has length $\Theta(m)$ the probability that this happens in the next step is exponentially small. Hence, the probability that the next step is accepted is exponentially small, if we have reached the local optimum. The local optimum is reached with a probability at least $\frac{1}{m} - o(1)$ and this proves the theorem.

VII. Conclusions

The Eulerian cycle problem is a fundamental problem in graph theory belonging to the wide class of arc routing problems. Several important problems belonging to this class are difficult, and evolutionary algorithms have a good chance to be competitive on these problems. The analysis shows how evolutionary algorithms based on the encoding of permutations work. This is also important because many evolutionary algorithms based on this encoding have been proposed for the traveling salesman problem and different NP-hard scheduling problems.

As a first step towards the analysis of evolutionary algorithms for the mentioned problems, upper bounds on the expected runtime of RLS and (1+1) EA for the Eulerian cycle problem have been proven. We have also presented a instance where the most costly improvements revolving a cycle by $\Theta(m)$ operations are necessary for both algorithms with probability at least $\frac{1}{m} - o(1)$. Such investigations lead to the result that the variants RLS* and (1+1) EA*, using exchanges for mutation, have an infinite or exponential optimization time on this instance.

**Acknowledgements**

The author would like to thank Ingo Wegener for useful comments that have helped to improve the paper.

**References**