Adjacency List Matchings — An Ideal Genotype for Cycle Covers*

Benjamin Doerr and Daniel Johannsen

Max-Planck-Institut für Informatik
Campus E1.4, 66123 Saarbrücken, Germany

Abstract

We propose and analyze a novel genotype to represent walk and cycle covers in graphs, namely matchings in the adjacency lists. This representation admits the natural mutation operator of adding a random match and possibly also matching the former partners.

To demonstrate the strength of this set-up, we use it to build a simple (1+1) evolutionary algorithm for the problem of finding an Eulerian cycle in a graph. We analyze several natural variants that stem from different ways to randomly choose the new match.

Among other insight, we exhibit a (1+1) evolutionary algorithm that computes an Euler tour in a graph with m edges in expected optimization time $\Theta(m \log m)$. This significantly improves the previous best evolutionary solution having expected optimization time $\Theta(m^2 \log m)$ in the worst-case, but also compares nicely with the runtime of an optimal classical algorithm which is of order $\Theta(m)$. A simple coupon collector argument indicates that our optimization time is asymptotically optimal for any randomized search heuristic.

Categories and Subject Desriptors

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

General Terms

Theory, algorithms, performance

Keywords

Evolutionary algorithm, randomized local search, runtime analysis, cycle cover, Euler tour

1 Introduction

We continue the on-going analysis of how to solve the Euler tour problem using evolutionary computation. This problem is particularly interesting because choosing a good representation of the individuals is highly non-trivial.

The main contribution of this paper is a new representation for walk and cycle covers and its successful application to the Eulerian cycle problem. By comparing variants of an appropriate mutation operator, we obtain, besides considerable other theoretical insight, an evolutionary algorithm for the Euler tour problem having an expected optimization time of $\Theta(m \log m)$ in worst-case. This clearly beats the previous best $\Theta(m^2 \log m)$ solution. Here and in the remainder of the paper, m shall always denote the number of edges of the input graph.

1.1 The Euler Tour Problem

Let $G = (V, E)$ be a connected, undirected graph with m edges. A closed walk using each edge exactly once is called an *Euler tour*. There is an Euler tour in G if and only if each vertex v of G has even degree $d(v) := |\{e \in E \colon v \in e\}|$. This famous result, often referred to as the solution of the *Seven Bridges of Königsberg* problem, is due to Euler [4] and considered by many to be the birth of graph theory.

Euler's proof does not reveal an efficient way to actually compute an Euler tour. A reasonably simple $O(m)$ time algorithm can be deduced from Hierholzer [5].

1.2 Evolutionary Computation

Even though a theoretically optimal algorithm for the Euler tour problem was given in [5], there is a strong motivation to study generic approaches to the problem like, for example, randomized local search (RLS) or evolutionary algorithms (EAs).

One motivation is that in most applications generic approaches are less costly to implement and need less expertise on the underlying problem. Another reason to favor a generic approach is that, given it proved to be effective for a particular problem, it can often be adapted easily to solve a generalization of the problem. See, e. g., [3] for generalizations of the Euler tour problem.

The concept of evolutionary computation is to decompose the problem of designing an algorithm into an appropriate choice of nature-inspired building blocks like representation of the individuals, fitness function and variation operators.

For many problems, this choice suggests itself. If, for example, a pseudo-boolean function $f\colon \{0,1\}^n \to \mathbb{N}_0$ has to be maximized, then taking f as the fitness and performing bit-flips as mutation are natural choices. For graph-theoretic problems, these choices are much harder as there are a number of natural ways to represent the underlying objects and to define variation operators. A problem that attracted attention in this respect is the Euler tour problem.

The first work addressing this problem is due to Neumann [6]. He represented the individuals by permutations of the edges. Given such a permutation $(e_1, \ldots, e_m)$ of the edges, he defines its fitness to be the largest ℓ such that $e_1, \ldots, e_\ell$ form a walk. Hence, a fitness of m is equivalent to having an Euler tour. Neumann investigated the runtime behavior of RLS and the (1+1)-EA using two different mutation operators.

The canonical *exchange operator* swaps the positions of two randomly chosen edges in the permutation. For this, Neumann presented a graph such that the expected optimization times of RLS and the (1+1)-EA are exponential.

More appropriate for the problem was the *jump operator*. For $i, j \in \{1, \ldots, m\}$, the operation $jump(i, j)$ moves the ith edge in the permutation to position j and cyclically shifts the edges in between in the appropriate direction. For example, if $i < j$, the permutation $(e_1, \ldots, e_m)$ is mutated to $(e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_j, e_i, e_{j+1}, \ldots, e_m)$. For this operator, the expected optimization time is $O(m^5)$.

This result was improved by Doerr, Hebbinghaus and Neumann [1]. They showed that if only jumps of type $jump(i, 1)$ are used, then, surprisingly, the optimization time drops to $O(m^3)$. A closer look at the proofs reveals that there are two issues that each lead to an $O(m)$ factor improvement. First, only jumps in one direction are used, i. e., only operations $jump(i, j)$ with $i > j$. This leads to a speed-up by replacing a random walk behavior by a one-directional search. Second, the first edge in the permutation is always part of the jump. This increases the chance to pick a mutation that is accepted.

Doerr, Klein and Storch [2] introduced a different representation of the individuals (and, in consequence, used a different mutation operator and fitness function). Since an Euler tour is a walk containing all edges, they chose collections of edge-disjoint walks covering all edges as individuals. Naturally, such a collection is an Euler tour if and only if it consists of a single element which is a closed walk. Hence, the fitness function (to be minimized) was chosen to be the total number of components plus (twice) the number of components that are not closed walks. They also defined a mutation operator that allowed the walks to be joined, split and twisted at a common vertex. This setting was shown to be more suitable to work with. In particular, Doerr, Klein and Storch [2] showed an expected optimization time of $\Theta(m^2 \log m)$ for the resulting (1+1)-EA.

Our representation builds on the representation by Doerr, Klein and Storch, but uses more appropriate genotypes inspired by adjacency lists, the classical data-structure used in most graph algorithms.

1.3 Phenotypes and Genotypes

An important concept in evolutionary computation is that we can distinguish between the phenotype of an individual and its genotype.

In [2], Doerr, Klein and Storch distinguished between the "graph theoretical" representation of a solution as collection of edge disjoint walks covering the graph and a "technical" representation as an array of edges each with two pointers pointing to the next and previous edge in the walk. Though not made explicit these two representations may be seen as the phenotype and the genotype of an individual.

The improvement in optimization time achieved in [2] was due to the superiority of their choice of the phenotype and the mutation operator it allowed. However, their genotype representation was not very elaborate, in fact, they did not care too much for this point at all.

In this paper, we shall see that the choice of the genotype representation is equally important in designing evolutionary algorithms. We stress that this is not only for the purpose of implementation, but already for the design of the evolutionary algorithm, in particular, the variation operators.

Classical experience with graph algorithms often suggests to represent a graph via adjacency lists. Here, for each vertex v we keep a list L_v of its neighbors. In contrast to the representation via adjacency matrices, this data-structure has linear space complexity. This is particularly useful for sparse graphs. Note that in practice, almost all graphs we find are sparse.

Adjacency lists also allow fast access to neighbors, again opposed to $\Theta(n)$ time for adjacency matrices or $\Theta(m)$ for edge arrays as implicitly used in [2].

Guided by this insight, we will develop a genotype representation that builds on adjacency lists. Recall that for each vertex v, we have a list L_v of its neighbors.

Suppose we have an Euler tour in our graph. Then whenever it passes through a vertex v, it uses an edge $\{u, v\}$ to reach v and another one $\{v, w\}$ to leave v. In this situation, we call u and w *matched* (as neighbors of v, or in L_v). Since an Euler tour traverses all edges, it induces a perfect matching in L_v, i. e., a partition of L_v into 2–element sets. Clearly, it does so for all vertices v of the graph.

In general, as we shall see, each family of matchings in the lists L_v immediately yields a collection of walks and tours covering all edges, and vice versa. Hence, these are the genotypes we shall work with.

Our genotype representation has several advantages that in the end lead to faster evolutionary algorithms. An advantage that we will not detail further, but that should not be overlooked, is that using a standard data-structure in general and this one in particular (see above) has several advantages from the view-point of actually implementing the algorithm.

For the theoretical analysis we strive for in this work, the representation via matchings in the adjacency lists has two main advantages. The first is that any variation operator building on this representation can only try to match edges that share a vertex. Clearly anything else would not make

sense anyway, but previous algorithms typically spent much time in trying such useless operations.

The second advantage is that we can easily generate an initial individual that already is a cycle cover (by simply greedily choosing perfect matching in each list). In fact, this allows two sub-types of representations, namely arbitrary matchings (representing a cover by walks and cycles) and perfect matchings (representing a cover by cycles only). We show that using the latter is of advantage: Surprisingly, often it takes much longer to find an arbitrary cycle cover starting with a random walk cover than to transform any cycle cover into a single (Eulerian) cycle.

Still, even when starting with the empty matching our algorithms are faster than previous ones. The details (to be made precise in Section 3) depend on how we choose the random edges in the mutation step.

There are three reasonable alternatives, namely (i) starting with a randomly chosen vertex and then trying a random match in its adjacency list, (ii) starting with a randomly chosen edge and then trying to match one of its end-vertices with a random vertex in the corresponding adjacency lists, or (iii) picking a potential match from all lists uniformly at random. All three choices may differ in the respective mutation probabilities, but not in the set of operations available on each individual.

We investigate these variants and prove sharp bounds for their optimization times. It turns out that the particular choice of what randomly means has a strong impact on the resulting optimization time. Surprisingly, there is no unique best way for the random choice, but the two representation sub-types have different best ways for the random choice in the mutations step.

However, all choices lead to a significant improvement of the expected optimization time over the currently best bound of $\Theta(m^2 \log m)$ from [2].

2 A Cycle Cover Representation

The concept of evolutionary computation is to successively modify a set of solution candidates (*population* of *individuals*) by adding new individuals and removing old ones until the population contains an individual with desired properties.

We investigate the two basic evolutionary search strategies *randomized local search* (RLS) and the *(1+1) evolutionary algorithm* (EA). As our main source of progress lies in choosing a suitable representation for the individual, we start with making this point precise, then develop an appropriate fitness function and mutation operator, and finally combine these components to an RLS algorithms and an (1+1) EA.

2.1 Representations

We distinguish between the phenotype and the genotype of an individual. In general, the set of phenotypes (*phenotype space*) consists of the objects among which we search for the solution of a given problem. The *genotype space*, on the other hand, contains *representations* or encodings of these objects. The evolutionary search heuristic itself operates mainly on the genotype space. For example, mutation is applied to the genotype of an individual.

2.1.1 Phenotypes

Motivated by the promising results in [2], we let the phenotype space consist of all walk covers of the input graph G.

Let $G = (V, E)$ be a connected, undirected graph with n vertices and m edges such that all vertices of G have an even degree.

A *walk* in G of length k is defined by an alternating sequence of vertices and edges $v_0, e_1, v_1, e_2, \dots, v_{k-1}, e_k, v_k$ having the following properties. It starts and ends with a vertex. These two vertices are called *end-vertices*.

Each edge in the sequence is incident with its preceding and succeeding vertex. In other words, $e_i = \{v_{i-1}, v_i\}$ for all $1 \leq i \leq k$. In addition (and contrary to the standard terminology), we require that all edges in a walk are different. Vertices may appear several times, in particular the two end-vertices can be the same. A walk has no distinguished direction, i. e., the sequence $v_k, e_k, v_{k-1}, \dots, e_1, v_0$ defines the same walk.

A *tour* in G of length k is defined by an alternating sequence of vertices and edges $v_0, e_1, \dots, v_{k-1}, e_k$, starting with a vertex and ending with an edge, such that the sequence $v_0, e_1, \dots, v_{k-1}, e_k, v_0$ is a walk.

A tour has neither a distinguished direction nor a distinguished starting vertex. For example, the two sequences of vertices and edge $v_{i-1}, e_i, \dots, v_{k-1}, e_k, v_0, e_1 \dots, v_{i-2}, e_{i-1}$ and $v_i, e_i, v_{i-1}, \dots, e_1, v_0, e_k, v_{k-1}, \dots, v_{i+1}, e_{i+1}$ define the same tour for any $1 \leq i \leq k$. Thus, a tour does not have end-vertices and can be represented by $2k$ sequences.

A tour which contains all edges of G is called an *Euler tour*. In the following, we use the term *walk* for both, walks and tours, and distinguish only if necessary. Thus, a *walk cover* of G is a collection of walks and tours such that each edge of the graph appears in exactly one walk or tour.

A *tour cover* of G is a walk cover such that all its walks are tours.

Obviously, a walk cover contains an Euler tour if and only if it is a tour cover with one element.

2.1.2 Genotypes

In the previous section we defined the phenotype of our individuals to be walk covers.

We will now define the *genotype* of the individual to be a collections of matchings in the adjacency lists.

This definition of the genotype is motivated by two simple observations. First, a sequence defining a walk or a tour can be completely described by listing all pairs of subsequent edges. Second, two edges of such a pair share a common vertex. Thus, the definition focusses on how the edges sharing one vertex are matched in a walk cover. Such a matching is best represented on the *adjacency list* of the given graph.

A typical encoding of a graph in application is that of an adjacency list. This structure keeps for every vertex of the graph list of all adjacent vertices. We are not interested in

implementation details, but only need the structural property to gather all edges that share a common vertex.

Hence, we define an *adjacency list* L as a family of n lists, one for every vertex, such that the list L_v corresponding to vertex v consists of all vertices adjacent to v. More precisely, let $L := (L_v)_{v \in V}$ with $L_v := \{w \in V : \{v, w\} \in E\}$.

The edge $\{u, v\}$ of G is represented in L by the two occurrences of its vertices, namely of u in L_v and of v in L_u. Hence, the total number of vertices appearing in all lists of L is $2m$.

A *matching* M_v in L_v is a set of disjoint unordered pairs (two-element sets) of vertices of L_v. A *matching* M in L is a family of n matchings M_v, one for each list L_v. The *size* of M is defined as $|M| := \sum_{v \in V} |M_v|$.

If all vertices in a list L_v are matched, we call M_v a *perfect matching* in L_v. Clearly, we have $|M_v| = d(v)/2$ in this case. If M_v is perfect for all vertices $v \in V$, we call M a *perfect matching* in L. Such a perfect matching is of size $|M| = m$. Note that the definition of a matching in L is different from the notion of a (graph-theoretical) matching in G. In particular, $\{u, w\} \in M_v$ does not imply that $\{u, w\} \in E$, but rather defines the subsequence $\{u, v\}, v, \{v, w\}$ of some walk.

Theorem 1. *There exists a 1-1-correspondence between the phenotype space of walk covers of G and the genotype space of matchings in L. Moreover, this 1-1-correspondence maps tour covers of G to perfect matching in L and vice versa.*

Proof. The main idea is that the unmatched vertices correspond to end-vertices of a walk and matched vertices to interior vertices of a walk or arbitrary vertices of a tour.

We first show that a walk cover of G defines a matching $M = (M_v)_{v \in V}$ in L. Let $v, u, w \in V$ such that $u, w \in L_v$. Then $\{u, w\} \in M_v$, if and only if there exists a walk in the cover such that the edges $\{u, v\}$ and $\{v, w\}$ are subsequent in one of the sequences defining the walk. Since in a walk cover each edge $\{v, w\}$ appears exactly once and in exactly one walk, in M this edge appears at most twice, once in L_v and once in L_w. Thus, M is a proper matching.

In case of a tour cover, M is a perfect matching in L, since there exists a $w \in L_v$ for all $v \in V$ and an $u \in L_v$ such that $\{u, w\} \in M_v$. This is due to the fact that in a tour there is no distinguished starting or end-vertex, in particular u is not such a vertex.

We next show that a matching M in L defines a walk cover of G. This definition is best explained recursively by the number of matched edges in M.

The empty matching in L defines the walk cover consisting of the walks $v, \{v, w\}, w$ for all edges $\{v, w\} \in E$. For a non-empty matching M, we choose a pair $\{u, w\} \in M_v$ for some $v \in V$.

Let M' be M without $\{u, w\}$. Then we can recursively define a walk cover corresponding to M'. Since u and w are not matched in M', there exist either two walks defined by the sequences $v, \{v, u\}, u, \ldots, u'$ and $w', \ldots, w, \{w, v\}, v$ or a single walk defined by the sequence $v, \{v, u\}, u, \ldots, w, \{w, v\}, v$.

In the first case, the walk cover corresponding to M is defined as the walk cover corresponding to M' where the two walks from above are joined to one walk that is defined by the sequence $w', \ldots, w, \{w, v\}, v, \{v, u\}, u, \ldots, u'$.

In the latter case, the new walk cover is defined by replacing the single walk from above by the tour defined by the sequence $v, \{v, u\}, u, \ldots, w, \{w, v\}$.

If the matching is perfect, then no edge is adjacent to an end-vertex and the cover is a tour cover. □

By assumption, all vertices of G are of even degree. Thus, all L_v of L are of even length, and it is always possible to find a perfect matching in all L_v, and hence on L.

In the following, we investigate evolutionary search heuristics on both search spaces. The one of all arbitrary matchings in L and the one of all perfect matchings in L. Note that an initial perfect matching in L can be constructed very easily by matching pairs of vertices in the order of their appearance in the lists of L.

2.2 Fitness Function

The phenotype of an individual is a walk cover of G. As we have seen, two conditions have to be fulfilled for the individual to be an Euler tour. First, no walk of the cover is allowed to have open ends, i. e., the walks have to be tours. Second, the cover has to have exactly one element.

While the second condition is best formulated in terms of the phenotype, we have seen that we can formulate the first condition in terms of the genotype. In particular, the genotype of individual has to be a perfect matching. Thus, a suitable fitness function to be minimized for the problem of finding an Euler tour in G is to take the total number of unmatched vertices in the genotype of an individual plus the number of walks in its phenotype. As the number of vertices that are not matched by a matching M in L is always even, we divide this number by two in the fitness function, which then equals m minus the size of M plus the number of components induced by M.

Formally, for a matching M in L representing a walk cover of G consisting of k walks, the fitness f of M is defined by

$$f(M) = m - |M| + k\,.$$

This fitness function can attain values from one to $2m$, one for individuals representing an Euler tour and $2m$ for the empty matching.

Lemma 2. *Let $1 \leq k \leq 2m$. The phenotype of an individual with fitness k is a walk cover with at least $k/2$ elements.*

Proof. In a matching each unmatched vertex is an end-vertex of a walk. Thus, for every two such vertices there exists a walk in the cover. In other words, the number of elements in the walk cover contribute at least as much to the fitness function than the number of unmatched vertex pairs. □

For the genotype space of perfect matchings the same fitness function applies, but only takes values between one and $m/3$, as the number of unmatched vertices is zero and each tour has at least three edges.

2.3 Mutation Operators

An evolutionary search heuristic based on a population of size one has to make use of a *mutation operator*. This operator transforms one individual to another based on a randomized change in the genotype of that individual.

We want this choice to be as local as possible, i. e., to affect one or at most two pairs of matched vertices. A natural way to do this is the following, even if the following definition is quite technical.

Given a matching $M = (M_v)_{v \in V}$ in L, the mutation operator φ defines a new matching $\varphi(M)$ as follows. First, we randomly choose two vertices u and w from one of the lists L_v. The distribution of this random choice is discussed later. If $u = w$ then M is not changed. Otherwise, we separate u and w from their current matches (if existent), match u and w (unless they were matched before), and match the former partners of u and w if possible. More precisely, we do the following:

- If u and w are unmatched, then add $\{u, w\}$ to M_v.
- If u and w are matched to each other, then remove $\{u, w\}$ from M_v.
- If u is matched to some w' and w is unmatched, then remove $\{u, w'\}$ from M_v and add $\{u, w\}$ to M_v.
- If u is unmatched and w is matched to some v', then remove $\{u', w\}$ from M_v and add $\{u, w\}$ to M_v.
- If u is matched to some w' and w is matched to some v', then remove $\{u, w'\}$ and $\{u', w\}$ from M_v and add $\{u, w\}$ and $\{u', w'\}$ to M_v.

If we operate on the space of perfect matchings, then the pair $\{u, w\}$ is not removed from M_v if u and w are already matched. Instead, M remains unchanged.

It is not immediately clear how to randomly choose the the ordered pair (u, w). In fact, there are three natural ways to do so.

Recall that $d(v)$ is the degree of the vertex v as defined in Section 1.1 and let $d(G)$ be the average degree of G, $\Delta(G)$ the maximum degree of G, and $\tilde{d}(G) := \frac{1}{2m} \sum_{v \in V} d(v)^2$.

2.3.1 Vertex-based distribution

First choose a vertex $v \in V$ uniformly at random. Then independently choose u and w uniformly at random from L_v. The probability p to choose the pair (u, w) in list L_v satisfies

$$p = \frac{1}{d(v)^2\, n} = \frac{d(G)}{2\, d(v)^2\, m} \geq \frac{d(G)}{2\, \Delta(G)^2\, m}\,.$$

2.3.2 Edge-based distribution

Choose the vertex u uniformly at random from all $2m$ vertices in all lists in L. Suppose u is in list L_v. Choose the vertex w uniformly at random from L_v. The probability p to choose the pair (u, w) in list L_v satisfies

$$p = \frac{1}{2\, d(v)\, m} \geq \frac{1}{2\, \Delta(G)\, m}\,.$$

2.3.3 Pair-based distribution

Choose the pair (u, w) uniformly at random from all ordered pairs of the $2m$ vertices in all lists in L with both vertices in the same list L_v. The probability p to choose the pair (u, w) in list L_v satisfies

$$p = \frac{1}{2\, \tilde{d}(G)\, m}\,.$$

The three distributions are equal for the special case of regular graphs, where $d(G) = \tilde{d}(G) = \Delta(G)$. In general, $d(G) \leq \tilde{d}(G) \leq \Delta(G)$ and thus the lower bound of the edge-based distribution also applies to the vertex-based distribution and the lower bound of the pair-based distribution also applies to the edge-based distribution. If we compare the distributions in detail, then we observe that a pair of vertices (u, w) in a large adjacency list L_v is strongly favored by the vertex-based distribution, moderately favored by the edge-based distribution, and treated independently of the list size in the pair-based distribution. We will see that this bias of the distributions depending on the degree sequence of G has a strong impact on the expected optimization time.

2.4 RLS and the (1+1) EA

To allow an easy comparison with previous works, we focus on the two basic randomized search strategies: randomized local search (RLS) and the (1+1) evolutionary algorithm (EA). In both cases we have a population of size one.

We use the genotype spaces, fitness function and mutation operators introduced in the previous subsections.

The choice whether to search the genotype space of all matchings or the space of perfect matchings is done during initialization the algorithms.

For the genotype space of all matchings in L, the natural choice for the initial individual of this population is the empty matching. This corresponds to the phenotype of the walk cover that consists of m walks each formed by a single edge. Alternatively we can choose a random matching. However, this does not seem to give any advantage.

For the genotype space of perfect matchings in L we choose as initial individual a perfect matching in L at random. The corresponding phenotype is a random tour cover of G. Actually, we can as well choose an arbitrary individual by matching pairs of vertices in the order of their appearance on the lists of L, since the choice does not affect the runtime analysis of the proposed search heuristics.

Randomized local search combines the given components in a straightforward way. Starting with a population of one initial individual, in each step the mutation operator is applied to the current individual to produce a new individual. The new individual replaces the old one if its fitness is at least as good as that of old one. Otherwise, the old individual is kept. The algorithm runs until it finds an Euler tour.

RLS

```
1  Initialize(M)
2  while (f(M) > 1)
3        do
4           M' := φ(M)
5           if f(M') ≤ f(M)
6              then M := M'
```

The *(1+1) evolutionary algorithm* is slightly more sophisticated. For the canonical genotype of n-bit string, the (1+1) EA flips each bit independently with probability $1/n$. Since such a simultaneous independent change is not possible with a complex mutation operator, the (1+1) EA applies such a mutation operator successively for a number of times given by the Poisson distribution Pois(λ) with $\lambda = 1$ (see [2, 6] for a discussion on this approach). Afterwards, the fitness of the new individual is checked and the population modified as in RLS.

(1+1) EA

```
1  Initialize(M)
2  while (f(M) > 1)
3        do
4           M' := M.
5           for i := 0 to Pois(1)
6                 do M' := φ(M')
7           if f(M') ≤ f(M)
8              then M := M'
```

3 Runtime Analysis

In the previous section we described several alternatives to build an evolutionary search strategy for Euler tours.

Genotype space:	arbitrary or perfect matchings in L.
Distribution:	vertex-, edge- or pair-based.
Generic strategy:	RLS or (1+1) EA.

In this section, we prove bounds for the expected optimization time of the resulting twelve combinations. More precisely, we first prove upper bounds for the twelve variants in Section 3.1.

Then, in Section 3.2 we show that several of these bounds are tight and in particular that the discrepancies indicated by them are real.

As usual, the optimization time of a randomized search heuristic is defined as the number of fitness evaluations. In our set-up, this equals the number of times the while-loop is executed (plus one).

3.1 Upper Bounds for the Optimization Time

We analyze the expected optimization time first with arbitrary matchings as genotypes, then with perfect matchings. In both cases, we regard the three different distributions introduced in Section 2.3. For the vertex- and edge-based distribution, the bounds for arbitrary matchings are a factor of Δ higher than for perfect matchings. As our lower bounds in Section 3.2 show, this gap is real.

We start with analyzing the optimization time of RLS. In Section 3.1.2, we show that identical bounds hold for the (1+1) EA.

3.1.1 Randomized Local Search

We prove the following bound for arbitrary matchings.

Theorem 3 (Arbitrary Matchings)**.**
The expected optimization time of randomized local search on the genotype space of arbitrary matchings as genotypes is

$O(\frac{\Delta(G)^2}{d(G)}\, m \log m)$ *for the vertex-based distribution,*

$O(\Delta(G)\, m \log m)$ *for the edge-based distribution, and*

$O(\tilde{d}(G)\, m \log m)$ *for the pair-based distribution.*

These bounds are independent of the initial individual.

Proof. Recall from Section 2.3 that the probability p that a particular pair (u, w) in a particular list L_v is chosen by the mutation operator, satisfies

$$p \geq q := \begin{cases} \frac{d(G)}{2\,\Delta(G)^2\, m} & \text{for the vertex-based distribution,} \\ \frac{1}{2\,\Delta(G)\, m} & \text{for the edge-based distribution,} \\ \frac{1}{2\,\tilde{d}(G)\, m} & \text{for the pair-based distribution.} \end{cases}$$

We show that the expected time to improve the fitness of an individual having fitness k is $O(\frac{1}{k\,q})$.

By Lemma 2, a fitness of k implies that the corresponding phenotype consists of at least $k/2$ components. If a component is a closed cycle, there is at least one pair (u, w) in a list L_v such that the corresponding mutation connects the cycle with some other component. If the component is not a closed cycle, again there is at least one pair (u, w) in a list L_v such that the corresponding mutation either connects the component to some other component or closes it to a cycle.

Since in both cases the involved edges $\{u, v\}$ and $\{w, v\}$ belong to at most two different components, we over-count by at most a factor of two. Hence, there are $\Omega(k)$ different mutations that increase the fitness. In consequence, the probability to do so in a single step is $\Omega(kq)$, and hence the expected time to reach this improvement is $O(1/kq)$.

Now any initial solution has a fitness of at most $2m$. It follows that the expected optimization time is uniformly bounded by $\sum_{k=1}^{m} \frac{1}{k\,q} = \frac{1}{q} \sum_{k=1}^{m} \frac{1}{k} = O(\frac{1}{q} \log m)$. Replacing q by the values given above finishes the proof of the statement. □

We now analyze RLS for the situation where we work with perfect matchings as genotypes. Recall that this is equivalent to saying that the corresponding phenotypes are cycle covers.

Surprisingly, for the vertex- and edged-based distributions we obtain superior runtime bounds than before. This indicates that finding an entire cycle cover is more difficult than joining its cycles to to an Euler tour. This will be proven in Section 3.2.

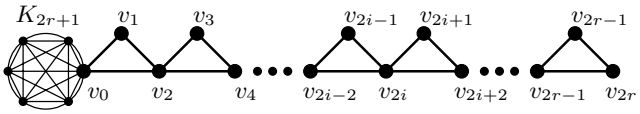

Figure 1: The graph on $2(r+s)+1$ vertices showing the lower bound for the pair-based distribution.

Theorem 4 (Perfect Matchings)**.**
The expected optimization time of randomized local search on the genotype space of perfect matchings as genotypes is

$O(\frac{\Delta(G)}{d(G)}\, m \log m)$ *for the vertex-based distribution,*

$O(m \log m)$ *for the edge-based distribution, and*

$O(\tilde{d}(G)\, m \log m)$ *for the pair-based distribution.*

These bounds are independent of the initial individual.

Proof. Let $M = (M_v)_{v \in V}$ be a perfect matching in L. Then M represents a tour cover of G consisting of $f(M) = k$ tours. Again, we bound the expected time needed to improve the fitness (necessarily by joining two tours).

Two tours are joined if the mutation operator φ is applied to two vertices of a list L_v that belong to different tours. Hence, we estimate the probability to choose such a pair.

Suppose that v is shared by exactly $s \geq 2$ tours $t_1, \ldots, t_s$. We count the number of pairs (u, w) in L_v such that $\{u, v\}$ and $\{v, w\}$ are in different tours of $t, \ldots, t_s$. To do so, we have to know how many edges of each tour are incident with v, in other words, how many of the vertices in L_v belong to the tour.

For $1 \leq i \leq s$ let d_i vertices of L_v belong to tour t_i. Then $\sum_{i=1}^{s} d_i = d(v)$, the d_i are even, and, in particular, $d_i \geq 2$. The number of vertex pairs of L_v belonging to different tours is $d(v)^2 - \sum_{i=1}^{s} d_i^2 \geq s\, d(v)$. The probability that any of these pairs is chosen, equivalently, that two tours are joined in v, is at least sq with

$$q := \begin{cases} \frac{d(G)}{2\,\Delta(G)\,m} & \text{for the vertex-based distribution,} \\ \frac{1}{2\,m} & \text{for the edge-based distribution,} \\ \frac{1}{2\,\tilde{d}(G)\,m} & \text{for the pair-based distribution.} \end{cases}$$

Next, we sum over the probability of all such pairs in the graph. Each of the k tours has at least one vertex it shares with another tour. Hence, if $s(v)$ is the number of tours containing the vertex v, and V' is the set of vertices contained in more than one tour, then $\sum_{v \in V'} s(v) \geq k$.

In consequence, the probability to join any of the k tours is at least $\sum_{v \in V'} s(v)\, q \geq kq$ and the expected number of steps needed for this improvement is at most $1/kq$.

Any initial perfect matching has a fitness of at most $m/3$, since each tour consists of at least three edges. Hence, the expected number of steps needed to join all tours is bounded by $\sum_{k=1}^{m/3} \frac{1}{k\,q} \leq \frac{1}{q} \sum_{k=1}^{m} \frac{1}{k} = O(\frac{1}{q} \log m)$. Replacing q by the bounds given above completes the proof. □

3.1.2 The (1+1) Evolutionary Algorithm

In this section, we show that the (1+1) EA fulfills the same runtime bounds as RLS. The main reason for this is that in Section 3.1.1 we actually proved bounds for the expected times RLS needs to obtain a fitness improvement. These bounds only depended on the current fitness, but were independent of the particular current solution. Hence, we may re-use these bounds here.

Theorem 5. *The (1+1) evolutionary algorithm on the genotype space of arbitrary matchings has an expected optimization time bounded as in Theorem 3. The (1+1) evolutionary algorithm on the genotype space of perfect matchings has an expected optimization time bounded as in Theorem 4.*

Proof. The probability that the (1+1) EA applies the mutation operator exactly once in a given step is $1/e$. Since our bounds for the expected times to increase the fitness in the proofs in section 3.1.1 only depended on the current fitness, we may completely ignore the fact that the (1+1) EA may change the current solution in different ways than RLS. Also, since in expectation the (1+1) EA performs a step with only one mutation every e-th step, we see that the expected time to improve the fitness from a certain level k is at most e times the one for RLS. In consequence, the expected optimization times of the (1+1) EA satisfy the same bounds as proven for RLS. □

3.2 Lower Bounds for the Optimization Time

In this section, we give several lower bounds for the optimization times of our algorithms.

In Section 3.2.1 we show that for the vertex- and edge-based distributions there is indeed a gap between the expected optimization times of finding a cycle cover and of uniting such a cover to an Euler tour. That there is no such gap for the pair-based distribution is shown in Section 3.2.2. Finally, in Section 3.2.3 we sketch an elementary coupon collector argument which indicates that $\Omega(m \log m)$ is a lower bound for the optimization time of any evolutionary algorithm for the Eulerian cycle problem, even if we already start with a cycle cover.

We restrict the analysis to RLS and omit the analysis of the (1+1) EA as the bounds can be obtained similarly and the proofs yield no further insight.

3.2.1 Finding a cycle cover is hard for the vertex- and edge-based distributions

In this section we give worst-case lower bounds that show that Theorem 3 is asymptotically tight.

We consider d–regular graphs (all vertices have degree d with $2 \leq d \leq n-1$ even). For such a d–regular graph G the vertex-, edge-, and pair-based distribution are equal. All three choose a fixed ordered pair of vertices in a fixed list with probability $1/2dm$. Since $\Delta(G) = d(G) = d$, we cover all possible values of $d(G)$ and $\Delta(G)$. Furthermore, the three upper bounds in Theorem 3 coincide.

Theorem 6. *Let $d \geq 2$ be even. For a d–regular graph G, randomized local search initialized with a random*[1] *or empty matching needs an expected number of $\Omega(d\,m\log m)$ steps until its current solution is a perfect matching for the first time.*

If G is a complete graph on n vertices (n odd to ensure existence of Euler tours), then RLS needs $\Theta(n^3 \log n)$ steps just to find a perfect matching. However, if we use the vertex- or edge-based distribution, it then only needs $O(n^2 \log n)$ steps to find the Euler tour afterwards. It seems that the (for deterministic algorithms trivial part of) finding a perfect matching is relatively hard for randomized search heuristics.

Proof. For an individual M, let $\ell(M)$ be the number of lists $L_v, v \in V$, that are not yet perfectly matched. Note that RLS does never increase $\ell(M)$, and in each step decreases it by at most one.

A list can only become perfectly matched if before it has exactly two vertices u and w not matched to other vertices. In this case, the probability that the pair (u,w) or (w,u) is chosen by the mutation operator is exactly $1/dm$. Hence, the probability that $\ell(M)$ decreases in a single step is at most $\ell(M)/dm$. Consequently, the expected number of steps needed for this is at least $dm/\ell(M)$. The expected time needed to completely match some solution M is at least $\sum_{k=1}^{\ell(M)} \frac{dm}{k} = \Omega(d\,m\log(\ell(M)))$. If our initial M is the empty set, then clearly $\ell(M) = n$. If M is a random matching, then with probability $1-\exp(-\Omega(n))$, at least $n/4$ of the lists are not perfectly matched. Thus, in both cases $\ell(M) = \Theta(n)$ and $\log(\ell(M)) = \Theta(\log m)$, which shows the claim. □

3.2.2 Uniting a cycle cover to an Eulerian cycle is hard for the pair-based distribution

Theorem 6 also holds for the pair-based distribution and yields an lower bound of $\Omega(\tilde{d}(G)\,m\,\log m)$ for RLS on arbitrary matchings. But, in contrast to the other two distributions, finding an Euler tour in the perfect matching model seems much harder for this distribution. Theorem 4 yields only an upper bound of $O(n^3 \log n)$ for this second step. This is surprising, as the pair-based distribution was the best among the three in the general matching model. However, this suboptimal behavior is true. Below, we present a graph having $\Theta(n^2)$ edges, for which RLS with pair-based distribution and perfect matchings needs $\Theta(n^3 \log n)$ steps.

Obviously, this time we can not start with an arbitrary perfect matching, as this could already represent an Euler tour in G. Hence, we start with a random perfect matching $M = (M_v)_{v\in V}$ in L. Note that this is canonically defined by choosing M_v uniformly at random among all perfect matchings in L_v for every vertex v.

[1] A matching drawn uniformly at random from the set of all matchings. However, the result remains true for any way of generating a random matching that has the property that each list L_v has a probability of at least $1/2$ of not being perfectly matched, independently for all $v \in V$. Note that this is a very weak requirement. All reasonable random matchings would leave the list L_v not perfectly matched with probability at least $1 - \Theta(1/d)$.

Theorem 7. *There exists a graph G with $\tilde{d}(G) \geq \frac{n}{4}$ such that randomized local search using the pair-based distribution initialized with a random perfect matching finds an Euler tour in G in expected optimization time $\Omega(\tilde{d}(G)\,m\log m)$.*

Proof. We omit the proof due to lack of space, the bound is obtained for the graph depicted in Figure 1. □

3.2.3 The coupon-collector bounds

We now sketch a coupon collector argument that indicates that all reasonable evolutionary algorithms for the Eulerian cycle problem have an expected optimization time of at least $\Omega(m \log m)$ in the worst-case. This looks natural, but is less simple if we start with a cycle cover.

Consider a graph consisting of a sequence of k triangles (the right half of the graph depicted in Figure 1). Then n and m are both $\Theta(k)$. We call a vertex belonging to two triangles *critical*. There are $k-1 = \Theta(k)$ critical vertices.

We argue that if we start with a reasonably random initial individual, a constant fraction of the critical vertices (or the edges incident with them) will need further modification. This is clear for a random cycle cover since (using the 1-1-correspondence between cycle covers and adjacency matchings) there are three perfect matchings on the adjacency list of a critical vertex, and one of them leaves the two triangles disconnected. For other random initial individuals, things are even worse as there are more ways how the two triangles may be disconnected.

Assuming a reasonably natural mutation operator, we see that with high probability there are $\Theta(k)$ critical vertices such that each of them needs the attention of one of a certain set of mutations. These sets are disjoint. In consequence, in each iteration we can fix at most one of these critical vertices.

This makes the problem of fixing all critical vertices a coupon collector problem. Now we do not know the particular distribution on the set of mutations, but it is well known that the coupon collector has the smallest expected optimization time if all coupons are equally likely. Hence even in this best case, we need $\Omega(k \log k)$ steps.

This in particular shows the $O(m \log m)$ bound for the strongest variant of our algorithm to be asymptotically tight, as well as the one for the vertex-based distribution and perfect matchings in the case that $\Delta(G) = O(d(G))$.

4 Conclusion

We proposed a novel genotype representation for walk and cycle covers in graphs, namely matchings in the adjacency lists. Since a perfect matching in all adjacency lists can be computed in a straight-forward way, the variant of only using perfect matchings is equally feasible. We showed that there is a natural one-one correspondence (i) between the genotypes of all matchings in adjacency lists and the phenotypes of all walk covers in the graph, and (ii) between the set of all perfect matchings and the set of all cycle covers.

Besides being superior from the viewpoint of implementation, our representation yields faster evolutionary algo-

	perfect matchings	arbitrary matchings
vertex-based	$O(\frac{\Delta(G)}{d(G)}\, m \log m)$	$\Theta(\frac{\Delta(G)^2}{d(G)}\, m \log m)$
edge-based	$O(m \log m)$	$\Theta(\Delta(G)\, m \log m)$
pair-based	$\Theta(\tilde{d}(G)\, m \log m)$	$\Theta(\tilde{d}(G)\, m \log m)$

Table 1: Bounds on the Expected Optimization time of RLS and (1+1) EA. Columns: genotype spaces. Lines: probability distributions.

rithms. To demonstrate this, we analyzed randomized local search (RLS) and the (1+1) evolutionary algorithm (EA) obtained from our representation together with the simple mutation operator of adding a random match (and possibly matching the former partners). This mutation operator resembles the 2-OPT heuristics for the travelling salesman problem.

There are three natural ways of picking a random match. Since they lead to a quite different runtime behaviors, we analyzed all three distributions for both sub-types of the representation (arbitrary and perfect matchings). The results, which are identical for RLS and the (1+1) EA, are summarized in Table 1.

Most notably, the variant working on the genotype space of perfect matchings and using the edge-based distribution has an expected optimization time $\Theta(m \log m)$ in the worst-case. This is clearly superior to the optimization time of $\Theta(m^2 \log m)$ presented in [2], the so-far best known optimization time for an evolutionary algorithm for the Euler tour problem. In fact, all six variant we investigated have an expected optimization time better than this previous solution. Also, as indicated in Section 3.2, the optimization time of $\Theta(m \log m)$ is likely to be best possible for any randomized search heuristics, due to the coupon collector phenomena arising in a typical run.

Comparing the optimization times of the six variants, we see that finding an Euler tour in the space of arbitrary matchings takes at least as long as in the space of perfect matchings. For the vertex-based and edge-based distribution, the optimization times differ by a factor of $\Delta(G)$. This discrepancy is real, in fact, we proved that these larger times are already necessary to simply find any cycle cover. In other words, uniting an existing cycle cover to an Euler tour is much easier than finding any cycle cover.

For the pair-based distribution this discrepancy does not exist. This stems from the surprising result that the pair-based distribution is the best among the three for the genotypes of arbitrary matching, but the worst for the genotypes of perfect matchings.

In summary, this research produced a valuable genotype representation for walk covers, the current best evolutionary algorithm for the Eulerian cycle problem, and the insight that the interplay of the three natural distributions and the two representation sub-types leads to some unexpected differences between the six resulting variants of our evolutionary algorithm.

5 Acknowledgment

We thank an unknown referee for several hints to improve the presentation.

References

[1] B. Doerr, N. Hebbinghaus, and F. Neumann. Speeding up evolutionary algorithms through restricted mutation operators. In *Proc. of the 9th International Conference on Parallel Problem Solving From Nature (PPSN)*, volume 4193 of *LNCS*, pages 978–987. Springer, 2006.

[2] B. Doerr, C. Klein, and T. Storch. Faster evolutionary algorithms by superior graph representation. In *Proc. of the 2007 IEEE Symposium on Foundations of Computational Intelligence (FOCI)*, pages 245–250. IEEE, 2007.

[3] J. Edmonds and E. L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[4] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741.

[5] C. Hierholzer. Ueber die Möglichkeit, einen Linenzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6:30–32, 1873.

[6] F. Neumann. Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. In *Proc. of the 2004 IEEE Congress on Evolutionary Computation (CEC)*, pages 904–910. IEEE, 2004.