
Proceedings Paper:
Bossek, J., Neumann, F., Peng, P. orcid.org/0000-0003-2700-5699 et al. (1 more author) (2019) Runtime analysis of randomized search heuristics for dynamic graph coloring. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 19). Genetic and Evolutionary Computation Conference (GECCO '19), 13-17 Jul 2019, Prague, Czech Republic. ACM , pp. 1443-1451. ISBN 9781450361118

https://doi.org/10.1145/3321707.3321792

Runtime Analysis of Randomized Search Heuristics for Dynamic Graph Coloring

Jakob Bossek
Information Systems and Statistics
University of Münster, Münster, Germany

Frank Neumann
Optimisation and Logistics
University of Adelaide, Adelaide, Australia

Pan Peng
Department of Computer Science
University of Sheffield, Sheffield, United Kingdom

Dirk Sudholt
Department of Computer Science
University of Sheffield, Sheffield, United Kingdom

ABSTRACT

We contribute to the theoretical understanding of randomized search heuristics for dynamic problems. We consider the classical graph coloring problem and investigate the dynamic setting where edges are added to the current graph. We then analyze the expected time for randomized search heuristics to recompute high quality solutions. This includes the (1+1) EA and RLS in a setting where the number of colors is bounded and we are minimizing the number of conflicts as well as iterated local search algorithms that use an unbounded color palette and aim to use the smallest colors and – as a consequence – the smallest number of colors.

We identify classes of bipartite graphs where reoptimization is as hard as or even harder than optimization from scratch, i. e. starting with a random initialization. Even adding a single edge can lead to hard symmetry problems. However, graph classes that are hard for one algorithm turn out to be easy for others. In most cases our bounds show that reoptimization is faster than optimizing from scratch. Furthermore, we show how to speed up computations by using problem specific operators concentrating on parts of the graph where changes have occurred.

KEYWORDS

Evolutionary algorithms, dynamic optimization, running time analysis, theory.

1 INTRODUCTION

Evolutionary algorithms and other bio-inspired computing techniques have been used for a wide range of complex optimization problems [1, 7]. They are easy to apply to a newly given problem and are able to adapt to changing environments. This makes them well suited for dealing with dynamic problems where components of the given problem change over time [22].

We contribute to the theoretical understanding of evolutionary algorithms in dynamically changing environments. Providing a sound theoretical basis on the behaviour of these algorithms in changing environments helps to develop better performing algorithms through a deeper understanding of their working principles.

Dynamic problems have been studied in the area of runtime analysis for simple algorithms such as randomized local search (RLS) and the classical (1+1) EA. An overview on rigorous runtime results for bio-inspired computing techniques in stochastic and dynamic environments can be found in [26]. Early work focused on artificial problems like a dynamic OneMax problem [9], the function Balance [23] where rapid changes can be beneficial, the function MAZE that features an oscillating behavior [15] and problems involving moving Hamming balls [6].

In terms of classical combinatorial optimization problems, prominent problems such as single-source-shortest-paths [16], makespan scheduling [18], and the vertex cover problem [19, 20, 27] have been investigated in a dynamic setting. Furthermore, the behaviour of evolutionary algorithms on linear functions with dynamic constraints has been analyzed in [28] and experimental investigations for the knapsack problem with a dynamically changing constraint bound have been carried out in [24]. These studies have been extended in [25] to a broad class of problems and the performance of an evolutionary multi-objective algorithm has been analyzed in terms of its approximation behaviour dependent on the submodularity ratio of the considered problem.

We consider graph coloring, a classical NP-hard optimization problem. In the context of problem specific algorithms, algorithms have been designed to update solutions after a dynamic change has happened. Dynamic algorithms have been proposed to maintain proper coloring for graphs with maximum degree at most Δ[1], with the goal of using as few colors as possible while keeping the (amortized) update time small [4, 5]. There exist algorithms that aim to perform as few (amortized) vertex recolorings as possible in order to maintain a proper coloring in a dynamic graph [3, 29]. There have also been studies of *k-list coloring* in a dynamic graph such that each update corresponds to adding one vertex (together with the incident edges) to the graph (e.g. [11]). The related problem of maintaining a coloring with minimal total colors in a *temporal* graph has recently been studied [17]. From a practical perspective, incremental algorithms or heuristics have been proposed that update the graph coloring by exploring a small number of vertices [21, 32].

Graph coloring has been studied for specific local search and evolutionary algorithms in [10, 30, 31]. Fischer and Wegener [10] studied a problem inspired by the Ising model in physics that on bipartite graphs is equivalent to the vertex coloring problem. They showed that on cycle graphs the (1+1) EA and RLS find optimal colorings in expected time $O(n^3)$. This bound is tight under a sensible assumption. They also showed that crossover can speed up the optimization time by a factor of n. Sudholt [30] showed that on complete binary trees the (1+1) EA needs exponential expected time, whereas a Genetic Algorithm with crossover and fitness sharing finds a global optimum in $O(n^3)$ expected time. Sudholt and

[1] In such graphs, there always exist a proper $(\Delta + 1)$-vertex coloring. Furthermore, such a proper coloring can be found in linear time.

Zarges [31] considered a different representation with unbounded-size palettes, where the goal is to use small color values as much as possible. They considered *iterated local search (ILS)* algorithms with operators based on so-called *Kempe chains* that are able to recolor large connected parts of the graph, while maintaining feasibility. This approach was shown to be efficient on paths and for coloring planar graphs of bounded degree ($\Delta \leq 6$) with 5 colors. The authors also gave a worst-case graph, a tree, where Kempe chains fail, but a new operator called *color elimination* that performs Kempe chains in parallel, succeeds in 2-coloring all bipartite graphs efficiently. Table 1 (top row) gives an overview over previous results.

We revisit these algorithms and graph classes for a dynamic version of the graph coloring problem. We assume that the graph is altered by adding T edges to it. This may create new conflicts that need to be resolved. Note that deleting edges from the graph can never worsen the current coloring, hence we focus on adding edges only[2]. Our goal is to estimate the expected reoptimization time, that is, the time to rediscover a proper coloring after T edges have been added, and how this time depends on T and the size of the graph, n. Our results are summarized in Table 1 (center row).

We start by considering bipartite graphs in Section 3. We find that even adding a single edge can create a hard symmetry problem for RLS and the (1+1) EA: expected reoptimization times for paths and binary trees are as bad as, or even slightly worse, than the corresponding bounds for optimizing from scratch. In contrast, ILS with Kempe chains or color elimination reoptimizes these instances efficiently. While ILS with color eliminations reoptimizes every bipartite graph in expected time $O(\sqrt{T}n \log n)$ or better, ILS with Kempe chains needs expected time $\Theta(2^{n/2})$ even when connecting a tree with an isolated edge. This instance is easy for all other algorithms as they all have reoptimization time $O(n \log^+ T)$ (where $\log^+ T = \max\{1, \log T\}$ is used to avoid expressions involving a factor of $\log T$ becoming 0 when $T = 1$).

In Section 4 we show that ILS with either operator is also able to efficiently rediscover a 5-coloring for planar graphs with maximum degree $\Delta \leq 6$ in expected time $O(n \log^+ T)$.

Finally, in Section 5 we design mutation operators that focus on the areas in the graph where a dynamic change has happened. We show that such conflict-aware approaches can reoptimize most graph classes in time $O(1)$ after inserting one edge, however they cannot prevent exponential times in cases where the algorithm is inefficient. All our results are shown in Table 1 (bottom row).

2 PRELIMINARIES

Let $G = (V, E)$ denote an undirected graph with vertices V and edges E. We denote by $n := |V|$ the number of vertices in G. A *vertex coloring* of G is an assignment $c : V \to \{1, \ldots, n\}$ of color values to the vertices of G. Let $\deg(v)$ be the degree of a vertex v and $c(v)$ be its color in the current coloring. Every edge $(u, v) \in E$ where $c(v) = c(u)$ is called a *conflict*. A color is called *free* for a vertex $v \in V$ if it is not assigned to any neighbor of v. The chromatic number $\chi(G)$ is the minimum number of colors that allows for a conflict-free coloring. A coloring is called *proper* is there is no conflicting edge.

[2]In general, the chromatic number of a graph can decrease when removing edges. We focus on graphs that can be colored with 2 or 5 colors, respectively. For 2-colorable graphs the chromatic number can only decrease if the graph becomes empty. For our results on 5-coloring graphs the real chromatic number will be irrelevant.

2.1 Algorithms with Bounded-Size Palette

In this representation, the total number of colors is fixed, i.e., the color palette has fixed size $k \leq n$. The search space is $\{1, \ldots, k\}^n$ and the objective function is to *minimize the number of conflicts*.

We assume that in the static setting all algorithms are initialized uniformly at random. In a dynamic setting we assume that a proper k-coloring x has been found. Then the graph is changed dynamically and x becomes an initial solution for the considered algorithms.

We define the dynamic (1+1) EA for this space as follows. Assume that the current solution is x. We consider all algorithms as infinite processes as we are mainly interested in the expected number of iterations until good solutions are found or rediscovered.

Algorithm 1 (1+1) EA (x)

1: **while** optimum not found **do**
2: Generate y by deciding to mutate each component x_i with probability $1/n$: if yes, choose a new value $y_i \in \{1, \ldots, k\} \setminus \{x_i\}$ uniformly at random.
3: If y has no more conflicts than x, let $x := y$.

We also define randomized local search (RLS) as a variant of the (1+1) EA where exactly one component is mutated.

Algorithm 2 RLS (x)

1: **while** optimum not found **do**
2: Generate y by choosing an index $i \in \{1, \ldots, n\}$ uniformly at random, choosing a new value $y_i \in \{1, \ldots, k\} \setminus \{x_i\}$ uniformly at random and setting $y_j = x_j$ for all $j \neq i$.
3: If y has no more conflicts than x, let $x := y$.

2.2 Algorithms with Unbounded-Size Palette

In this representation, the color palette size is sufficiently large (say has size n). Our goal is to maintain a *proper* vertex coloring (i.e., without any conflicting edge) such that the color-occurrence vector is as close to optimum as possible (see Definition 2.1 from [31]).

Definition 2.1. [[31]] For x, y we say that x is better than y and write $x \geq y$ iff

- x has fewer conflicting edges than y or
- x and y have an equal number of conflicting edges and their color frequencies are lexicographically ordered as follows. Let $n_i(x)$ be the number of i-colored vertices in x, then $n_i(x) < n_i(y)$ for the largest index i with $n_i(x) \neq n_i(y)$.

As remarked in [31], decreasing the number of vertices with the currently highest color (and not introducing yet a higher color) yields an improvement. If this number decreases to 0, the number of colors has decreased.

Setting	Graph class	Bounded-size palette (1+1) EA	Bounded-size palette RLS	Unbounded-size palette: ILS with… Kempe Chains	Unbounded-size palette: ILS with… Color Eliminations
Static	paths	$O(n^3)$ [10]	$O(n^3)$ [10]	$O(n)$ [31]	$O(n \log n)$ [Thm 3.5]
	binary trees	$\exp(\Omega(n))$ [30]	∞	$O(n \log n)$ [Thm 3.3]	$O(n \log n)$ [Thm 3.3]
	depth-2 star	$O(n \log n)$ [Thm 3.8]	$O(n \log n)$ [Thm 3.8]	$\exp(\Omega(n))$ [31]	$O(n^2 \log n)$ [31]
	any bipartite	$\exp(\Omega(n))$ [30]	∞	$\exp(\Omega(n))$ [31]	$O(n^2 \log n)$ [31]
	planar, $\Delta \leq 6$			$O(n \log n)$ [31]	$O(n \log n)$ [Thm 4.1]
Adding T edges (conflict-unaware algorithms)	paths	$\Theta(n^3)$ [Thm 3.1]	$\Theta(n^3)$ [Thm 3.1]	$O(n)$ [31]	$O(n \log^+ T)$ [Thm 3.5]
	binary trees	$\Omega(n^{(n-3)/4})$ [Thm 3.2]	∞	$O(n \log n)$ [Thm 3.3]	$O(n \log n)$ [Thm 3.4]
	depth-2 star	$O(n \log^+ T)$ [Thm 3.8]	$O(n \log^+ T)$ [Thm 3.8]	$\Theta(2^{n/2})$ [Thm 3.6]	$O(n \log^+ T)$ [Thm 3.7]
	any bipartite	$\Omega(n^{(n-3)/4})$ [Thm 3.2]	∞	$\Omega(2^{n/2})$ [Thm 3.6]	$O(\min\{\sqrt{T}, \Gamma\} n \log n)$ [Thm 3.4]
	planar, $\Delta \leq 6$			$O(n \log^+ T)$ [Thm 4.1]	$O(n \log^+ T)$ [Thm 4.1]
Adding one edge (conflict-aware algorithms)	paths	$\Theta(n^2)$ [Thm 5.1]	$\Theta(n^2)$ [Thm 5.1]	$O(1)$ [Thm 5.2]	$O(1)$ [Thm 5.3]
	binary trees	$\Omega(n^{(n-7)/4})$ [Thm 5.6]	∞	$O(1)$ [Thm 5.2]	$O(1)$ [Thm 5.3]
	depth-2 star	$O(1)$ [Thm 5.5]	$O(1)$ [Thm 5.5]	$\Theta(2^{n/2})$ [Thm 5.7]	$O(1)$ [Thm 5.3]
	any bipartite	$\Omega(n^{(n-7)/4})$ [Thm 5.6]	∞	$\Omega(2^{n/2})$ [Thm 5.7]	$O(1)$ [Thm 5.3]
	planar, $\Delta \leq 6$			$O(1)$ [Thm 5.4]	$O(1)$ [Thm 5.4]

Table 1: Worst-case expected times for (re-)discovering proper 2-colorings for bipartite graphs and proper 5-colorings for planar graphs in different settings. We use the notation $\log^+ T = \max\{1, \log T\}$.

Grundy local search. We use the same local search operator as in [31] called Grundy local search (Algorithm 3). A vertex v is called a *Grundy vertex* if v has the smallest color value not taken by any of its neighbors, formally $c(v) = \min\{i \in \{1, \dots, n\} \mid \forall w \in \mathcal{N}(v)\colon c(w) \neq i\}$, where $\mathcal{N}(v)$ denotes the neighborhood of v. A coloring is called a *Grundy coloring* if all vertices are Grundy vertices [33]. Note that a Grundy coloring is always proper.

Algorithm 3 Grundy local search [12]

1: **while** the current coloring is not a Grundy coloring **do**
2: Choose a non-Grundy vertex v.
3: Set $c(v) := \min\{i \in \{1, \dots, n\} \mid \forall w \in \mathcal{N}(v)\colon c(w) \neq i\}$.

The analysis in [12] reveals that one step of the Grundy local search can only increase the color of a vertex if there is a conflict; otherwise the color of vertices can only decrease. Sudholt and Zarges [31] point out that the application of Grundy local search can never worsen a coloring. If y is the outcome of Grundy local search applied to x then $y \geq x$. If x contains a non-Grundy node then y is strictly better, i. e., $y \geq x$ and $x \not\geq y$.

We also introduce the *Grundy number* $\Gamma(G)$ of a graph G (also called *first-fit chromatic number* [2]) as the maximum number of colors used in any Grundy coloring. Every application of Grundy local search produces a proper coloring with color values at most Γ.

We consider the *Kempe chain* mutation operator defined in [31], which is based on so-called *Kempe chain* [13] moves. This mutation exchanges two colors in a connected subgraph. By H_{ij} we denote the set of all vertices colored i or j in G. Then $H_j(v)$ is the connected component of the subgraph induced by $H_{c(v)j}$ that contains v.

The Kempe chain operator (Algorithm 4) is applied to a vertex v and it exchanges the color of v (say i) with a color j. We restrict the choice of j to the set $\{1, \dots, \deg(v) + 1\}$ since larger colors will be replaced in the following Grundy local search. In the connected component $H_j(v)$ the colors i and j of all vertices are exchanged. As no conflict within $H_j(v)$ is created and $H_j(v)$ is not neighbored to any vertex colored i or j, Kempe chains preserve feasibility.

Algorithm 4 Kempe chain [31]

1: Choose $v \in V$ and $j \in \{1, \dots, \deg(v) + 1\}$ uniformly at random.
2: Let $i := c(v)$
3: **for** all $u \in H_j(v)$ **do**
4: **if** $c(u) = i$ **then** $c(u) := j$ **else** $c(u) := i$.

An important point to note is that, when the current largest color is $c_{\max}$, Kempe chains are often most usefully applied to *the neighborhood* of a $c_{\max}$-colored vertex v. This can lead to a color in v's neighborhood becoming a free color, and then the following Grundy local search will decrease the color of v. In contrast, applying a Kempe chain to v directly will spread color $c_{\max}$ to other parts of the graph, which might not be helpful.

Sudholt and Zarges [31] introduced a mutation operator called a *color elimination* (Algorithm 5): it tries to eliminate a smaller color i in the neighborhood of a vertex v in one shot by trying to recolor all these vertices with another color j using parallel Kempe chains.

Algorithm 5 Color elimination [31]

1: Choose $v \in V$ uniformly at random.
2: **if** $c(v) \geq 3$ **then**
3: Choose $i, j \in \{1, \dots, c(v) - 1\}$, $i \neq j$, uniformly at random.
4: Let $v_1, \dots, v_\ell$ enumerate all i-colored neighbors of v.
5: **for** all $u \in H_j(v_1) \cup \dots \cup H_j(v_\ell)$ **do**
6: **if** $c(u) = i$ **then** $c(u) := j$ **else** $c(u) := i$.

Iterated local search (ILS, Algorithm 6) repeatedly uses mutation followed by Grundy local search. The mutation operator is not

specified yet, but regarded as a black box. In the initialization every vertex v receives a uniform random color from $\{1, \dots, \deg(v) + 1\}$.

Algorithm 6 Iterated local search (ILS) (x)

1: Replace x by the result of Grundy local search applied to x.
2: **repeat forever**
3: Let y be the result of a *mutation operator* applied to x.
4: Let z be the outcome of Grundy Local Search applied to y.
5: If $z \geq x$ then $x := z$.

2.3 Reoptimization Times

We consider the *batch-update* model for dynamic graph coloring. That is, given a graph $G' = (V, E')$ and its proper coloring, we would like to find a proper coloring of $G = (V, E)$ which is obtained after a batch of T edge insertions to G'. We are interested in the reoptimization time, i.e., the number of iterations it takes to find a proper coloring of the current graph G, given a proper coloring of G'. How does the expected reoptimization time depend on n and T? More precisely, we consider the *worst case reoptimization time* to be the reoptimization time when considering the worst possible way of inserting T edges into the graph.

Note that a bound for the reoptimization time can also yield a bound on the optimization time in the static setting for a graph $G = (V, E)$. This is because the static setting can be considered as a dynamic setting where we start with n isolated vertices and then add all $T = |E|$ edges to the graph. The only small difference is that with unbounded-size palettes, all vertices will have the smallest color 1 when edges are inserted. In cases where we derive static time bounds from dynamic ones, this difference is irrelevant.

3 REOPTIMIZATION TIMES ON BIPARTITE GRAPHS

We start off by considering bipartite graphs, i. e. 2-colorable graphs. For the bounded-size palette, we assume that only 2 colors are being used, i. e. $k = 2$. We also consider unbounded-size palettes where the aim is to eliminate all colors larger than 2 from the graph.

3.1 Paths and Binary Trees

We first show that even adding a single edge can result in difficult symmetry problems. This can happen if two subgraphs are connected by a new edge, and then the coloring in one subgraph has to be inverted to find the optimum. Two examples for this are paths and binary trees.

Theorem 3.1. *If adding an edge completes an n-vertex path, the worst-case expected time for the (1+1) EA and RLS to rediscover a proper 2-coloring is $\Theta(n^3)$.*

Proof. The claim essentially follows from to the proofs of Theorems 3 and 5 in [10] where the authors investigate an equivalent problem on cycle graphs. Hence, we just sketch the idea here. Imagine we link two properly colored paths of length $n/2$ each with an edge (u, v) which introduces a single conflict. The conflict splits the path into two paths that are properly colored and joined by a conflicting edge. Consider the length of the shortest properly colored path. As argued in [10], both RLS and (1+1) EA can either increase or decrease this length in fitness-neutral operations like recoloring one of the vertices involved in the conflict. If it has decreased to 1, the conflict has been propagated down to a leaf node where a single bit flip can get rid of it. Fischer and Wegener calculate bounds for the expected number of steps until this number reaches its minimum 1. This is achieved by estimating the number of so-called relevant steps, which either increase or decrease the length of the shortest properly colored path. The probability for a relevant step is $\Theta(1/n)$. The expected number of relevant steps is $\Theta(n^2)$ since we have a fair random walk on states up to $n/2$. In summary, this results in a runtime bound of $\Theta(n^3)$. □

Theorem 3.2. *If adding an edge completes an n-vertex complete binary tree, the worst-case expected time for the (1+1) EA to rediscover a proper 2-coloring is $\Omega\left(n^{(n-3)/4}\right)$. RLS is unable to rediscover a proper 2-coloring in the worst case.*

Proof. The proof follows arguments from [30]. Let $e = \{r, v\}$ be the added edge with r being the root of the n-vertex complete binary tree. If $c(r) \neq c(v)$ we are done and the coloring is already a proper 2-coloring. Hence, we assume that $c(r) = c(v)$ and there is exactly one conflict. This situation is a worst-case situation in vertex-coloring of complete binary trees, since many vertices must be recolored in the same mutation to produce an accepted candidate solution. Let W be the set of all worst-case colorings (there are exactly 4 such colorings). Further, let A_1 be the set of colorings having exactly one conflict and A_0 be the set of proper colorings. Note that $|A_0| = O(1)$ and $|A_1| = O(n)$ since there are $\binom{n-1}{1} = n - 1$ edges whose incident edges may be in conflict and there are exactly two possible conflicting color assignments for the incident nodes. Thus, there are only $O(n)$ search points which may be accepted if we start with $x \in W$ as in the given dynamic scenario. The Hamming distance from a worst-case coloring to another coloring in $(A_1 \cup A_0) \setminus W$ is at least $(n+1)/4$ (we need to recolor a whole subtree with root v). We use this fact to upper bound the probability of the (1+1) EA to leave W by $O\left(n^{-(n+1)/4+1}\right)$. Hence, the expected number of mutations is at least $\Omega\left(n^{(n-3)/4}\right)$.

It is obvious from the above that RLS is unable to leave W. □

In the above two examples, the reoptimization time is at least as large as the optimization time from scratch. In fact, our dynamic setting even allows us to create a worst-case initial coloring that might not typically occur with random initialization. Theorem 3.1 gives a rigorous lower bound of order n^3 as after adding an edge connecting two paths of $n/2$ vertices each, we start the last "fitness level" with a worst-case initial setup. Fischer and Wegener [10] were only able to show a lower bound under additional assumptions. Also in [30] the probability of reaching the worst-case situation described in Theorem 3.2 was very crudely bounded from below by $\Omega(2^{-n})$. Our lower bounds for dynamic settings are hence a bit tighter and/or more rigorous than those for the static setting.

The reason for the large reoptimization times in the above cases is because for the (1+1) EA and RLS mutations occur locally, and they struggle in solving symmetry problems where large parts of the graph need to be recolored. Mutation operators in ILS like Kempe chains and color eliminations operate more globally, and can easily deal with the above settings.

Theorem 3.3. *Consider a dynamic graph that is a path or a binary tree after a batch of T edge insertions. The expected time for ILS with Kempe chains to rediscover a proper 2-coloring on paths is $O(n)$.*

On binary trees, the expected time for ILS with either Kempe chains or color eliminations to rediscover a proper 2-coloring or to find a proper 2-coloring in the static setting is $O(n \log n)$.

Proof. The statement about paths follows from [31, Theorem 1] as the expected time to 2-color a path is $O(n)$ in the static setting. (It is easy to see that the proof holds for arbitrary initial colorings.)

The Grundy number of binary trees is at most $\Gamma \leq \Delta + 1 \leq 4$. By design of our selection operator, the number of 4-colored vertices is non-increasing over time. For every 4-colored vertex v there must be a Kempe chain operation recoloring a neighboring vertex whose color only appears once in the neighborhood of v. If there are i 4-colored vertices, the probability of reducing this number is $\Omega(i/n)$ and the expected time for color 4 to disappear is $O(n) \cdot \sum_{i=1}^{n} 1/i = O(n \log n)$. Afterwards, the same arguments apply to the number of 3-colored vertices, leading to another $O(n \log n)$ term. □

3.2 Results for General Bipartite Graphs

Sudholt and Zarges [31] showed that ILS with color eliminations can color every bipartite graph efficiently, in expected $O(n^2 \log n)$ iterations [31, Theorem 3]. The main idea behind this analysis was to show that the algorithm can eliminate the highest color from the graph by applying color eliminations to all such vertices. The expected time to eliminate the highest color is $O(n \log n)$, and we only have to eliminate at most $O(n)$ colors. In fact, the last argument can be improved by considering that in every Grundy coloring of a graph G the largest color is at most $\Gamma(G)$. This yields an upper bound of $O(\Gamma(G) n \log n)$ for both static and dynamic settings.

The following result gives an additional bound of $O(\sqrt{T} n \log n)$, showing that the number T of added edges can have a sublinear impact on the expected reoptimization time.

Theorem 3.4. *Consider a dynamic graph that is bipartite after a batch of T edge insertions. Let Γ be the Grundy number of the resulting graph. Then ILS with color eliminations re-discovers a proper 2-coloring in expected $O(\min\{\sqrt{T}, \Gamma\} n \log n)$ iterations.*

If only one conflicting edge is added, the expected time is $\Theta(n)$.

Proof. Consider the connected components of the original graph. If an edge is added that runs within one connected component, it cannot create a conflict. This is because the connected component is properly 2-colored, with all vertices of the same color belonging to the same set of the bipartition. Since the graph is bipartite after edge insertions, the new edge must connect two vertices of different colors. Hence added edges can only create a conflict if they connect two different connected components that are colored inversely to each other.

Consider the subgraph induced by the added edges that are conflicting, and pick a connected component C in this subgraph. Note that all vertices in C have the same color $c \in \{1, 2\}$ before Grundy local search is applied. Now Grundy local search will fix these conflicts by increasing the colors of vertices in C. We bound the value of the largest color $c_{\max}$ used. For Grundy local search to assign a color $c_{\max}$ to a vertex $v \in C$, all colors $1, \ldots, c_{\max} - 1$ must occur in the neighborhood of v in the new graph. In particular, C must contain vertices $v_3, v_4, \ldots, v_{c_{\max}-1}$ respectively colored $3, \ldots, c_{\max} - 1$ that are neighbored to v. This implies that $c_{\max} - 3$ edges incident to v, connecting v to a smaller color, must have been added during the dynamic change. Applying the same argument to $v_3, v_4, \ldots, v_{c_{\max}-1}$ yields that there must be at least $\sum_{j=1}^{c_{\max}-3} j = (c_{\max} - 3)(c_{\max} - 2)/2$ inserted edges in C. Thus $(c_{\max} - 3)(c_{\max} - 2)/2 \leq T$, which implies $(c_{\max} - 3)^2 \leq 2T \iff c_{\max} \leq \sqrt{2T} + 3$. Also $c_{\max} \leq \Gamma$ by definition of the Grundy number.

Now we can argue as in [31, Theorem 3]: the largest color can be eliminated from any bipartite graph in expected time $O(n \log n)$. (Note that these color eliminations can increase the number of vertices colored with large colors, so long as the number of the vertices with the largest color decreases.) Since at most $c_{\max} - 2$ colors have to be eliminated, a bound of $O(c_{\max} n \log n)$ follows. Plugging in $c_{\max} = O(\min\{\sqrt{T}, \Gamma\})$ completes the proof.

Finally, if only one conflicting edge is inserted ($T = 1$) then there will be one 3-colored vertex v after Grundy local search, and a proper 2-coloring is obtained by applying a color elimination to v. The expected waiting time for choosing vertex v is $\Theta(n)$. □

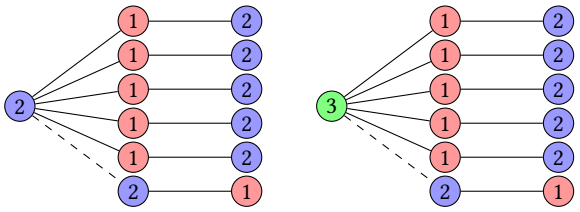

Figure 1: Depth-2 star with $n = 13$ vertices. The dashed line indicates the added edge. Left: coloring with a bounded-size palette, right: coloring after Grundy local search with an unbounded-size palette.

In the case of graphs with Grundy number $\Gamma \leq 3$, the factor of $\log n$ can be replaced by $\log^+ T$. The considered graph class includes graphs of maximum degree $\Delta \leq 2$ (e. g. paths and cycles).

Theorem 3.5. *Consider a dynamic graph that is bipartite and has maximum degree $\Delta \leq 2$ or Grundy number $\Gamma \leq 3$ after a batch of T edge insertions. Then ILS with color eliminations rediscovers a proper 2-coloring in expected time $O(n \log^+ T)$.*

Proof. Note that $\Delta \leq 2$ implies $\Gamma \leq \Delta + 1 \leq 3$. By definition of Γ, after Grundy local search the largest possible color is 3. Every added edge leads to at most one conflict, and each conflict leads to at most one vertex being colored 3 in the Grundy local search.

Following [31, Theorem 3], while there are i vertices colored 3, a color elimination choosing such a vertex will lead to a smaller free color, reducing the number of 3-colored vertices. The expected time for this to happen is at most n/i, hence the total expected time to eliminate all color-3 vertices is at most $\sum_{i=1}^{T} n/i = O(n \log^+ T)$. □

3.3 A Worst-Case Graph for Kempe Chains

While ILS with color eliminations efficiently reoptimizes all bipartite graphs, for ILS with Kempe chains there are bipartite graphs where even adding a single edge connecting a tree with an isolated edge can lead to exponential times.

Theorem 3.6. *For every $n \equiv 1 \bmod 4$ there is a forest T_n with n vertices such that for every feasible 2-coloring the ILS with Kempe chains needs $\Theta(2^{n/2})$ generations in expectation to re-discover a feasible 2-coloring after adding an edge.*

Proof. Choose T_n as the union of an isolated edge $\{u, v\}$ where $c(u) = 2$ and $c(v) = 1$ and a tree where the root r has $N - 1 := (n-3)/2$ children and every child has exactly one leaf (cf. Figure 1). This graph was also used in [31] as an example where ILS with Kempe chains fail in a static setting. Since $n \equiv 1 \bmod 4$, N is an even number. Every feasible 2-coloring will color the root and the leaves in the same color and the root's children in the remaining color. Assume the root and leaves are colored 2 as the other case is symmetric. Now add an edge $\{r, u\}$ to the graph. This creates a star of depth 2 (termed the *depth-2 star* in the following) where the root is the center and the root now has N children.

This creates a conflict at $\{r, u\}$ that is being resolved by recoloring one of these vertices to color 3 in the next Grundy local search. With probability $1/2$, this is the root r.

From this situation, any Kempe chain affecting any vertex in $V \setminus \{r\}$ can swap the colors on an edge incident to a leaf. Let $X_0, X_1, \ldots$ denote the random number of leaves colored 1, starting with $X_0 = 1$. We only consider steps in which this number is changed; note that the probability of such a change is $\Theta(1)$ as every Kempe chain on any vertex except for the root changes X_t if an appropriate color value is chosen. There are $N := (n-1)/2$ leaves and the number of 1-colored leaves performs a random walk biased towards $N/2$: $\Pr(X_{t+1} = X_t + 1 \mid X_t) = (N - X_t)/N$ and $\Pr(X_{t+1} = X_t - 1 \mid X_t) = X_t/N$. This is the Ehrenfest urn model[3].

When $X_t \in \{0, N\}$ then a proper 2-coloring has been found. As long as $X_t \in \{2, \ldots, N-2\}$, all Kempe chain moves involving the root will be rejected as the number of 3-colored vertices would increase. While $X_t \in \{1, N-1\}$ a Kempe chain move recoloring the root with the minority color will be accepted. This has probability $1/n \cdot 1/(N-1) = \Theta(1/N^2)$ (as the color is chosen uniformly from $\{1, \ldots, \deg(r)+1\}$) and then the following Grundy local search will produce a proper 2-coloring. Also considering possible transitions to neighbouring states 0 or N, while $X_t \in \{1, N-1\}$ the conditional probability that a proper 2-coloring is found before moving to a state $X_t \in \{2, N-2\}$ is $\Theta(1/N)$.

For the Ehrenfest model it is known that the expected time to return to an initial state of 1 is $1!(N-1)!/N! \cdot 2^N = 2^N/N$ [14, equation (66)]. It is easy to show that this time remains in $\Theta(2^N/N)$ when considering $N - 1$ as a symmetric target state, and when conditioning on traversing states $\{2, \ldots, N-2\}$.

Along with the above arguments, this means that such a return in expectation happens $\Theta(N)$ times before a proper 2-coloring is found. This yields a total expectation of $\Theta(2^N) = \Theta(2^{n/2})$. □

Notably, this instance is easy for all other considered algorithms.

Theorem 3.7. *On a graph where adding T edges completes a depth-2 star, ILS with color eliminations rediscovers a proper 2-coloring in expected time $O(n \log^+ T)$.*

[3]This simple model was originally proposed to describe the process of substance exchange between two bordering containers of equal size which are separated by a permeable membrane. Consider N particles spread across the containers and denote by $X(t)$ the number of particles in the left container w. l. o. g. at time t. In each step one particle is chosen uniformly at random and swaps sides.

Proof. We argue that the graph's Grundy number is $\Gamma = 3$ as then the claim follows from Theorem 3.5. Since all vertices but the root have degree at most 2, their colors must be at most 3. Assume for a contradiction that the root has a color larger than 3. Then there must be a child v of color 3. But then v has a free color in $\{1, 2\}$, contradicting a Grundy coloring. Hence also the root must have color at most 3, completing the proof that $\Gamma = 3$. □

Theorem 3.8. *On the depth-2 star RLS and (1+1) EA both have expected optimization time $O(n \log n)$ in the static setting and $O(n \log^+ T)$ to rediscover a proper 2-coloring after adding T edges.*

Proof. First note that any conflict can be resolved by one or two mutations. The latter is necessary in the unfavourable situation of $\{r, u\}, \{u, v\} \in E$, r being the root, with $c(r) = 2 = c(u)$ and $c(v) = 1$. Then both u and v need to be recolored simultaneously or in sequence. We show that every conflict has a constant probability of being resolved within the next n steps. Let X_t denote the number of conflicts at time $t \in \mathbb{N}_0$. If $X_t > 0$, the probability of improvement within n steps is at least

$$p \geq \frac{1}{2} \cdot \binom{n}{2} \cdot \left(\frac{1}{n}\right)^2 \cdot \left(\left(1 - \frac{1}{n}\right)^{n-1}\right)^2 \cdot \left(1 - \frac{2}{n}\right)^{n-2} \geq \frac{(n-1)}{4ne^4} = \Omega(1).$$

Here, the term $1/2 \cdot \binom{n}{2}$ describes all combinations of two relevant mutations concerning nodes u and v in sequence. The next two factors indicate that in the selected steps both u and v are recolored and all remaining nodes are left apart. Finally, the last factor is the probability of not mutating both vertices in the remaining $n - 2$ steps. Note that for RLS the penultimate factor disappears. Hence, the expected number of conflicts after n steps is

$$E(X_{t+n} \mid X_t) \leq X_t - X_t p \leq X_t - X_t \cdot \frac{(n-1)}{4ne^4} = X_t \cdot \left(1 - \frac{(n-1)}{4ne^4}\right)$$

and we obtain an expected multiplicative drift of

$$E(X_t - X_{t+n} \mid X_t) \geq X_t - X_t \cdot \left(1 - \frac{(n-1)}{4ne^4}\right) = X_t \frac{(n-1)}{4ne^4}.$$

Applying the multiplicative drift theorem [8] yields an upper bound of $E(T) \leq \frac{8e^2}{1+1/n} \log(1 + x_{\max}) = O(\log^+ x_{\max})$ for the expected number of phases. Here, $x_{\max} \leq n$ in the static setting and $x_{\max} \leq T$ in the dynamic setting denotes the maximum number of conflicts. Hence, the runtime bounds are $O(n \log n)$ and $O(n \log^+ T)$ in the static and dynamic settings, respectively, for RLS and (1+1) EA. □

The conclusion from the above is that reoptimization times strongly depend on the instance and the algorithms considered.

4 REOPTIMIZATION TIMES ON PLANAR GRAPHS

We also consider planar graphs with degree bound $\Delta \leq 6$. It is well-known that all planar graphs can be colored with 4 colors, but the proof is famously non-trivial. Coloring planar graphs with 5 colors has a much simpler proof, and this setting was studied in [31]. The reason for the degree bound $\Delta \leq 6$ is that in [31] it was shown that for every natural number c there exist tree-like graphs and a coloring where the "root" is c-colored, and no Kempe chain or color elimination can improve this coloring. In the following

we only consider the unbounded palette as no results for general planar graphs are known for bounded palette sizes.

Theorem 4.1. *Consider adding T edges to a 5-colored graph such that the resulting graph is planar with maximum degree $\Delta \leq 6$. Then the worst-case expected time for ILS with Kempe chains or color eliminations to rediscover a proper 5-coloring is $O(n \log^+ T)$.*

Proof. Every edge can create at most one conflict, and every conflict can lead to one vertex receiving a higher color than before during Grundy local search. Hence after inserting T edges, there will be at most T vertices colored 6 or 7.

In [31] it was shown that for each vertex v colored 6 or 7, there is a Kempe chain operation affecting a neighbour of v such that a color at v becomes a free color and v receives a color at most 5 after the next Grundy local search. If there are i nodes colored 6 or 7, the probability of a Kempe chain move reducing the number of vertices colored with the highest color is at least $i/(7n)$.

Let n_6 and n_7 denote the number of 6- and 7-colored vertices, respectively. It is obvious that n_7 is non-increasing. However, during an operation reducing n_7, n_6 may increase. We need to argue that n_6 does not increase too much.

Note that n_6 can only increase if n_7 decreases. Consider a vertex v whose color is being decreased from 7 to some color $c < 7$ in one iteration. Since the previous iteration's coloring was a Grundy coloring, all colors $\{1, \ldots, 6\}$ must have been present exactly once in the neighborhood of v. If $c = 6$ then v and its 6-colored neighbour u swap colors. Since u must have all colors $\{1, \ldots, 5\}$ in its neighborhood and u has degree at most 6 (which means that its degree must be exactly 6), u cannot have other 7-colored neighbors and the Kempe chain does not change n_6 and n_7.

Now assume that $c < 6$. Then there must have been a Kempe chain move that has recolored the c-colored neighbour of v to a new color $c' < 7$. If $c' \neq 6$, n_6 is unchanged. Hence assume $c' = 6$. Let u_1 be the unique c-colored neighbour of v. Each 6-colored neighbour of u_1 can only have at most two c-colored neighbours itself, one of which is u_1, as all colors $\{1, \ldots, 5\}$ must appear in the neighbourhood of u_1. Hence each 6-colored node leads on to at most one new c-colored node. (But every c-colored node can lead on to one or more 6-colored nodes.) So in total the number of 6-colored nodes is by at most 1 larger than the number of c-colored nodes. This shows that, if n_7 decreases, n_6 can increase by at most 1.

There are at most T 7-colored nodes initially, and the expected time to recolor them is $O(n \log^+ T)$. Then there are at most T 6-colored nodes, and the same arguments yield another term of $O(n \log^+ T)$. □

5 THE BENEFITS OF USING CONFLICT-AWARE ALGORITHMS

Finally, we consider the performance of the original algorithms, but enhancing them with tailored operators that focus on the region of the graph that has been changed. The assumption here is that only one edge is added at a time, and the algorithms are aware of the vertices involved in the edge addition. Since many of the previous results indicated that algorithm spend most of their time just finding the right vertex to apply mutation to, we expect the reoptimization times to decrease when using conflict-aware operators.

We first define conflict-aware algorithms for bounded-size palettes. The (1+1) EA and RLS are modified so that they take a conflict edge $\{u, v\}$ as additional input and they mutate the end points of said edge with constant probability. They further follow the conflict: if a mutation moves the conflict to another edge $\{u', v'\}$, the algorithm continues with $\{u', v'\}$.

Algorithm 7 Conflict-aware (1+1) EA $(x, (u, v))$

1: **while** optimum not found **do**
2: Generate y by deciding to mutate each x_w with probability $1/2$ for $w \in \{u, v\}$ and with probability $1/n$ for $w \notin \{u, v\}$: if yes, choose a new value $y_w \in \{1, \ldots, k\} \setminus \{x_w\}$ uniformly at random. For $w \notin \{u, v\}$, let $y_w = x_w$.
3: If y has no more conflicts than x, let $x := y$. If y also has a conflict edge, let (u, v) denote the new conflict edge.

Algorithm 8 Conflict-aware RLS $(x, (u, v))$

1: **while** optimum not found **do**
2: Generate y by choosing a vertex w as follows. With probability $1/2$ choose w uniformly at random from $\{u, v\}$, otherwise choose w uniformly at random from all vertices. Choose a new value $y_w \in \{1, \ldots, k\} \setminus \{x_w\}$ uniformly at random and set $y_j = x_j$ for all $j \neq w$.
3: If y has no more conflicts than x, let $x := y$. If y also has a conflict edge, let (u, v) denote the new conflict edge.

For unbounded-size palettes a new edge can lead to a higher color emerging in exactly one vertex, as Grundy local search will increase the color of a vertex involved in a conflict, resolving the conflict. The conflict-aware ILS algorithm applies mutation to this unique vertex v as follows. Color eliminations are applied to v directly. Kempe chains are most usefully applied in the neighborhood of v, hence a neighbor of v is chosen uniformly at random. This is repeated until the largest color has been eliminated from the graph.

Algorithm 9 Conflict-aware ILS $(x, (u, v))$ with color eliminations (resp. Kempe chains)

1: Let c be the largest color currently used.
2: Apply Grundy local search to x.
3: Let $w \in \{u, v\}$ be the vertex with the largest color.
4: **while** $c(w) = c + 1$ **do**
5: Apply a color elimination to w (resp. apply a Kempe chain to a vertex chosen uniformly at random from the neighbors of w) to generate a coloring y.
6: Let z be the outcome of Grundy Local Search applied to y.
7: **if** $z \succeq x$ **then**
8: $x := z$.
9: If there are no $(c + 1)$-colored vertices, stop. Otherwise let w be the unique vertex with color $c + 1$.

Intuitively, the conflict-aware dynamic algorithms can save at least a factor of n in the expected time compared with their conflict-unaware counterparts since the latter need $\Theta(n)$ iterations to discover the conflict vertices u, v and then make progress from there.

This is the case for paths; inspecting the proof of Theorem 3.1 confirms the following result.

Theorem 5.1. *If adding an edge completes an n-vertex path, the worst-case expected time for the conflict-aware (1+1) EA and conflict-aware RLS to rediscover a proper 2-coloring is* $\Theta(n^2)$.

Theorem 5.2. *Consider a dynamic graph that is a path or binary tree after one edge insertion. The expected time for conflict-aware ILS with Kempe chains to rediscover a proper 2-coloring is* $O(1)$.

Proof. Let v denote the unique 3-colored vertex after Grundy local search. For both paths and trees there is a Kempe chain operation applied to a neighbor of v that will create a free color for v, such that color 3 will disappear from the graph. This was mentioned in the proof of Theorem 3.3 for trees and it is easy to see for paths: every Kempe chain recoloring a neighbor u with the unique color from $\{1,2\} \setminus \{c(u)\}$ leaves $c(u)$ as a free color. In both scenarios, an improving Kempe chain occurs with probability $\Omega(1)$. □

Theorem 5.3. *Consider a dynamic graph that is bipartite after one edge insertion. Conflict-aware iterated local search with color eliminations re-discovers a 2-coloring in 1 iteration.*

Proof. The proof follows from the proof of Theorem 3.4 and that a color elimination is applied to the unique 3-colored vertex. □

Theorem 5.4. *Consider adding one edge to a 5-colored graph such that the resulting graph is planar with maximum degree* $\Delta \leq 6$. *Then the worst-case expected time for conflict-aware ILS with Kempe chains or color eliminations to rediscover a proper 5-coloring is* $O(1)$.

Proof. Suppose that there will be a conflict after insertion and let v be the vertex with degree 6 after local Grundy search. Since the algorithm is aware of the v that is colored 6, it performs a Kempe chain on a randomly sampled neighbor. We know from [31] and the proof of Theorem 4.1 that there is a Kempe chain and a color elimination that eliminates the color 6. The probability of applying such a Kempe chain is $\Omega(1)$ as there are only $O(1)$ neighbors and $O(1)$ possible colors. Color eliminations are always applied at v and eliminate color 6 if the right color parameters are chosen. Since two different colors are chosen uniformly at random from $\{1,\dots,5\}$, the probability of a successful color elimination is $\Omega(1)$ as well. □

Theorem 5.5. *On the depth-2 star from Theorem 3.6, conflict-aware RLS and (1+1) EA both have expected optimization time* $O(1)$ *to rediscover a proper 2-coloring after adding one edge.*

Proof. Any conflict can be resolved by one or two mutations and each conflict has a constant probability of being resolved in $O(1)$ steps. The proof then follows from the proof of Theorem 3.8. □

However, conflict-aware operators cannot prevent exponential times as shown for binary trees and depth-2 stars.

Theorem 5.6. *If adding an edge completes an n-vertex complete binary tree, the worst-case expected time for the conflict-aware (1+1) EA to rediscover a proper 2-coloring is* $\Omega\left(n^{(n-7)/4}\right)$. *Conflict-aware RLS is unable to rediscover a proper 2-coloring in the worst case.*

Proof. The proof is similar to proof of Theorem 3.2. The Hamming distance between the worst-case coloring to any acceptable coloring is still at least $\frac{n+1}{4}$. We can save a factor of n as the algorithm will mutate each of the endpoints of the conflict edge (u,v) with $1/2$ probability, rather than with probability $1/n$ as before. □

Theorem 5.7. *On the depth-2 star from Theorem 3.6, conflict-aware ILS with Kempe chains needs* $\Theta(2^{n/2})$ *generations in expectation to rediscover a proper 2-coloring.*

Proof. Conflict-aware ILS with Kempe chains applies a Kempe chain to uniformly chosen neighbors of the root. The transition probabilities still follow an Ehrenfest urn model; the only difference is that no Kempe chain can originate from the root itself. This does not affect the proof of Theorem 3.6, and the same result applies. □

6 CONCLUSIONS

We have studied dynamic graph coloring in a setting where T edges are added to a properly colored graph. Our results in Table 1 show that reoptimization can be much more efficient than optimizing from scratch: in many upper bounds a factor of $\log n$ can be replaced by $\log^+ T$ and we showed tighter general bound for bipartite graphs of $O(\min\{\sqrt{T},\Gamma\} n \log n)$ as opposed to $O(n^2 \log n)$ [31]. However, this heavily depends on the graph class and algorithms. For instance, depth-2 stars led to exponential times for Kempe chains and times of $O(n \log^+ T)$ for all other algorithms. Reoptimization can also be more difficult as we can create difficult initial colorings. On paths and binary trees the dynamic setting allows for negative results that are stronger than those previously published [10, 30].

Conflict-aware repair operators can reduce the efficient runtimes by a factor of n, even down to $O(1)$, but they cannot prevent inefficient runtimes in the considered settings.

ACKNOWLEDGMENTS

J. Bossek acknowledges support by the European Research Center for Information Systems (ERCIS). F. Neumann has been supported by the Australian Research Council (ARC) through grant DP160102401.

REFERENCES

[1] 2012. *Variants of Evolutionary Algorithms for Real-World Applications.* Springer.
[2] J. Balogh, S.G. Hartke, Q. Liu, and G. Yu. 2008. On the first-fit chromatic number of graphs. *SIAM Journal of Discrete Mathematics* 22 (2008), 887–900.
[3] Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. 2017. Dynamic graph coloring. In *Workshop on Algorithms and Data Structures.* Springer, 97–108.
[4] Leonid Barenboim and Tzalik Maimon. 2017. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. *Procedia Computer Science* 108 (2017), 89–98.
[5] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. 2018. Dynamic Algorithms for Graph Coloring. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM, 1–20.
[6] Duc-Cuong Dang, Thomas Jansen, and Per Kristian Lehre. 2017. Populations Can Be Essential in Tracking Dynamic Optima. *Algorithmica* 78, 2 (01 Jun 2017), 660–680.
[7] Kalyanmoy Deb. 2012. *Optimization for Engineering Design - Algorithms and Examples, Second Edition.* PHI Learning Private Limited.
[8] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. 2012. Multiplicative Drift Analysis. *Algorithmica* 64, 4 (01 Dec 2012), 673–697.
[9] S. Droste. 2002. Analysis of the (1+1) EA for a dynamically changing ONEMAX-variant. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC)*, Vol. 1. 55–60.

[10] Simon Fischer and Ingo Wegener. 2005. The One-dimensional Ising Model: Mutation versus Recombination. *Theoretical Computer Science* 344, 2–3 (2005), 208–225.

[11] Sepp Hartung and Rolf Niedermeier. 2013. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science* 494 (2013), 86 – 98.

[12] Stephen T. Hedetniemi, David P. Jacobs, and Pradip K. Srimani. 2003. Linear time self-stabilizing colorings. *Inform. Process. Lett.* 87, 5 (2003), 251–255.

[13] Tommy R. Jensen and Bjarne Toft. 1995. *Graph coloring problems*. Wiley-Interscience.

[14] Mark Kac. 1947. Random Walk and the Theory of Brownian Motion. *The American Mathematical Monthly* 54 (1947), 369–391.

[15] Timo Kötzing and Hendrik Molter. 2012. ACO Beats EA on a Dynamic Pseudo-Boolean Function. In *Parallel Problem Solving from Nature - PPSN XII - 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I (Lecture Notes in Computer Science)*, Vol. 7491. Springer, 113–122.

[16] Andrei Lissovoi and Carsten Witt. 2015. Runtime analysis of ant colony optimization on dynamic shortest path problems. *Theoretical Computer Science* 561 (2015), 73–85.

[17] George B. Mertzios, Hendrik Molter, and Viktor Zamaraev. 2019. Sliding Window Temporal Graph Coloring. In *AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, 2019*.

[18] Frank Neumann and Carsten Witt. 2015. On the Runtime of Randomized Local Search and Simple Evolutionary Algorithms for Dynamic Makespan Scheduling. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. AAAI Press, 3742–3748.

[19] Mojgan Pourhassan, Wanru Gao, and Frank Neumann. 2015. Maintaining 2-Approximations for the Dynamic Vertex Cover Problem Using Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015*. ACM, 903–910.

[20] Mojgan Pourhassan, Vahid Roostapour, and Frank Neumann. 2017. Improved runtime analysis of RLS and (1+1) EA for the dynamic vertex cover problem. In *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017, Honolulu, HI, USA, November 27 - Dec. 1, 2017*. 1–6.

[21] Davy Preuveneers and Yolande Berbers. 2004. ACODYGRA: an agent algorithm for coloring dynamic graphs. In *Symbolic and Numeric Algorithms for Scientific Computing*. 381–390.

[22] Hendrik Richter and Shengxiang Yang. 2013. Dynamic Optimization Using Analytic and Evolutionary Approaches: A Comparative Review. In *Handbook of Optimization - From Classical to Modern Approach*. 1–28.

[23] Philipp Rohlfshagen, Per Kristian Lehre, and Xin Yao. 2009. Dynamic evolutionary optimisation: an analysis of frequency and magnitude of change. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2009*. ACM, 1713–1720.

[24] Vahid Roostapour, Aneta Neumann, and Frank Neumann. 2018. On the Performance of Baseline Evolutionary Algorithms on the Dynamic Knapsack Problem. In *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part I*. 158–169.

[25] Vahid Roostapour, Aneta Neumann, Frank Neumann, and Tobias Friedrich. 2019. Pareto Optimization for Subset Selection with Dynamic Cost Constraints. In *AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, 2019*.

[26] Vahid Roostapour, Mojgan Pourhassan, and Frank Neumann. 2018. Analysis of Evolutionary Algorithms in Dynamic and Stochastic Environments. *CoRR* abs/1806.08547 (2018). http://arxiv.org/abs/1806.08547

[27] Feng Shi, Frank Neumann, and Jianxin Wang. 2018. Runtime analysis of randomized search heuristics for the dynamic weighted vertex cover problem. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. 1515–1522.

[28] Feng Shi, Martin Schirneck, Tobias Friedrich, Timo Kötzing, and Frank Neumann. 2017. Reoptimization times of evolutionary algorithms on linear functions under dynamic uniform constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*. ACM, 1407–1414.

[29] Shay Solomon and Nicole Wein. 2018. Improved Dynamic Graph Coloring. In *26th Annual European Symposium on Algorithms (ESA 2018) (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 112. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 72:1–72:16.

[30] Dirk Sudholt. 2005. Crossover is Provably Essential for the Ising Model on Trees. In *Proc. of GECCO '05*. ACM Press, 1161–1167.

[31] Dirk Sudholt and Christine Zarges. 2010. Analysis of an Iterated Local Search Algorithm for Vertex Coloring. In *21st International Symposium on Algorithms and Computation (ISAAC 2010) (LNCS)*, Vol. 6506. Springer, 340–352.

[32] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. 2017. Effective and Efficient Dynamic Graph Coloring. *Proc. VLDB Endow.* 11, 3 (Nov. 2017), 338–351.

[33] Manouchehr Zaker. 2007. Inequalities for the Grundy chromatic number of graphs. *Discrete Applied Mathematics* 155, 18 (2007), 2567–2572.