Engineering Dynamic Graph Coloring Algorithms

Marvin Weitz

May 30, 2024

3726675

Bachelor Thesis

at

Algorithm Engineering Group Heidelberg
Heidelberg University

Supervisor:
Univ.-Prof. PD. Dr. rer. nat. Christian Schulz

Co-Supervisor:
Henrik Reinstädtler

Acknowledgments

I would like to acknowledge all the people who supported me, not only during my work on this thesis but especially in the last years of my studies. I am grateful to have met so many like-minded and kind people who always pushed me to my best.

Furthermore, I would like to thank Prof. Dr. Christian Schulz for the supervision of this thesis and for the understanding he showed me in this time. I want to express a special thank-you to Henrik Reinstädtler for his words of affirmation as well as his time and efforts to ease my research process as much as possible.

Hiermit versichere ich, dass ich die Arbeit selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich aus fremden Werken übernommenes als fremd kenntlich gemacht habe. Ferner versichere ich, dass die übermittelte elektronische Version in Inhalt und Wortlaut mit der gedruckten Version meiner Arbeit vollständig übereinstimmt. Ich bin einverstanden, dass diese elektronische Fassung universitätsintern anhand einer Plagiatssoftware auf Plagiate überprüft wird.

Heidelberg, May 30, 2024

Marvin Weitz

Abstract

The Graph Coloring Problem (GCP) asks for assigning as few distinct colors to all vertices of a graph such that no two vertices connected by and edge share the same color. The dynamic GCP concerns graphs whose structure changes over time by insertion or deletion of edges and vertices. Due to the GCP not being solvable in polynomial time, heuristics and approximations are used. We will explore a method to effectively update the coloring of a graph after edge insertions, by optimally solving the GCP on a subgraph created by a breadth-first search. We convert existing Integer Linear Programs (ILPs) used for the static GCP by Jabrayilov and Mutzel [19] to be compatible with the dynamic environment. Furthermore, we address preprocessing techniques. We carry out experiments to evaluate the performance of our dynamic ILPs with regard to runtime and solution quality as well as other metrics, such as the number of occurring color conflicts.

Contents

CHAPTER 1

Introduction

The Graph Coloring Problem (GCP) asks for assigning as few distinct colors to all vertices of a graph such that no two vertices connected by and edge share the same color. It is a well known and thoroughly studied problem that has significant implications in both theoretical and practical domains. In computer science and combinatorics, this problem is a fundamental issue in the context of scheduling [12, 26], register allocation in compilers [9], assigning frequencies in wireless networks [37] and general resource allocation. In the applied context, an optimized graph coloring can enhance the performance of computational systems, improve timetables [21] and assist air traffic control [3].
The dynamic setting of the GCP is of particular interest for almost all mentioned areas, as in real applications circumstances can change quickly. Users accessing a cellular network, a diverted flight, a broken down train in the public transport system or changing connections in a social network; in all these cases the topology of the setting at hand changes and can require reevaluation. Adapting to these changes without the need to recolor the entire graph is important, if not indispensable, for performance. Optimally solving the GCP on large graphs is not feasible due to the NP-hard nature [13] of the problem. The problem's complexity grows exponentially with the graph's size, most of which are huge in real world applications. Therefore, approximations and heuristics are a common tool to have a trade-off between solution quality and runtime.
In this context, Integer Linear Programming (ILP) is a powerful tool that can model the GCP by using variables, constraints and objectives. While solving ILPs exactly is computational expensive, modern solvers like IBM's CPLEX [18] or Gurobi Optimizer [15] utilize a multitude of techniques to lower the efforts needed. The black box design of an ILP makes it a useful tool, as it is easily integrateable and interchangeable.

1.1 Our Contribution

The Graph Coloring Problem (GCP) has been subject to a multitude of studies, also in the dynamic context [4, 38, 33, 34, 16]. To our knowledge, no one has used ILPs to solve the dynamic GCP yet. In this thesis we propose four ILPs adapted from their static versions in Jabrayilov and Mutzel [19]. Edge conflicts are resolved by inducing a subgraph via a breadth-first search. This subgraph will be isolated and colored via an ILP, ensuring that the result does not conflict when reinserting it into the original graph. For this, we fixate the neighborhood of the subgraph and map its colors, as well as applying a number of preprocessing techniques to the subgraph to achieve better performing ILPs.
We conduct experiments, evaluating the impact of search depth on the results regarding runtime, as well as solution quality and other metrics. The ILPs are compared in different scenarios as well as regarding a number of graph instances. These results are matched with the findings of Jabrayilov and Mutzel [19].

1.2 Structure

The remainder of this thesis is organized as follows. In Chapter 2, we introduce the fundamentals and notations used in the rest of the thesis. Chapter 3 explores existing work related to the GCP and dynamic approaches. We cover the current state of ILPs for the GCP as well as related preprocessing in their own chapter (Chapter 4). In Chapter 5, we present our contribution. It opens with the preparation of data, especially the creation of the subgraph and its fixated neighborhood. We continue by showing the mapping of colors of said neighborhood into a different color space. After adapting preprocessing techniques to our setting, we perform a similar color mapping on the remaining vertices. Next, the dynamic ILPs are introduced. This includes their definition, adaptations to turn them dynamic as well as their resulting dimensions. In Chapter 6 we analyze the performances of the ILPs for a number of graph instances as well as comparing them with existing results for static ILPs. We start by exclusively assessing the runtime performance by controlling the number of subgraphs created by each ILP. Secondly, instances of the DIMACS benchmark are used to compare multiple metrics in an applied context. The thesis finishes with Chapter 7 by presenting a conclusion based on the experiments and giving an outlook on potential future work.

CHAPTER 2

Fundamentals

2.1 General Definitions

Graphs. A *graph* is a set of objects V, called *vertices* and identified by a unique natural number $V \subset \mathbb{N}_{>0}$. Vertices can be pairwise connected. These connections are called *edges*. Let $G = (V, E)$ be a graph where V is the set of vertices and $E \subseteq \binom{V}{2}$ is the set of edges. These edges $e = \{u, v\} \in E$ are *undirected*, meaning the order of vertices is indifferent. The number of vertices is represented by $n = |V|$ and the number of edges by $m = |E|$. A graph is *dense* if the number of edges is close to the maximum number of edges possible. Two vertices connected by an edge are called *adjacent* or *neighboring*. If a vertex is not part of any edge, it is *isolated*. A graph $G = (V, E)$ is called a *simple graph* if all its edges connect two distinct vertices $\forall \{u, v\} \in E \mid u \neq v$; there are no self-loops. We consider undirected simple graphs in this thesis. The *neighborhood* of a vertex v is the set of adjacent vertices $N(v) = \{u \in V \mid \{u, v\} \in E\}$. The *degree* of v is the size of its neighborhood $\deg(v) = |N(v)|$.

Subgraphs. A *subgraph* $G_s = (V_s, E_s)$ of a graph $G_o = (V_o, E_o)$ is induced by a subset of its vertices $V_s \subseteq V$ and its edges $E_s \subseteq E$ such that for all edges both vertices are in V_s. If $\forall \{u, v\} \in E_s \iff \{u, v\} \in E_o \mid u, v \in V_s$, the subgraph is called *induced*. In this context we call G_o an *original* or *global* graph and note $G_s \subseteq G_o$.
For a graph $G = (V, E)$ and $V_s \subseteq V$, $G[V_s]$ denotes an induced subgraph $G_s = (V_s, E_s)$. The neighborhood of a subgraph $N(G_s) = \{v \in V \mid \{u, v\} \in E \wedge u \in V_s \wedge v \notin V_s\}$ is the set of vertices adjacent to the subgraphs vertices that are not part of the subgraph themselves.

Cliques. A *clique* C is a subset of vertices of a graph G with all pairs of distinct vertices being adjacent. Such a subgraph $G[C]$ is also called a *complete* graph, as all possible edges

are present. A clique is called *maximal* if no vertex $v \in G(V)$ exists, such that $G[C \cup \{v\}]$ is a complete graph.

Breadth-First Search. A *breadth-first search* (BFS) explores a graph by exploring all vertices $v \in V$ at the present *depth* $d(v)$ in relation to a root vertex v_r before traversing to one depth level deeper. The depth level of a vertex v is the length of the shortest path $(v_r, \ldots, v)$ from v_r to v.

Dynamic Graphs. A dynamic graph $G_{dyn} = G(t) \mid t \in T$ with $G(t) = (V, E(t))$ is an evolving graph for which the set of edges can change over time. The graph can be represented by a sequence of states $G(t_0) \to G(t_1) \to \ldots \to G(t_n)$, each at a specific time $t \in T = \{t_0, \ldots, t_n\}$. $E(t)$ is the set of edges at time t. A non-dynamic graph is called *static*. If not noted otherwise a graph is considered static.

2.2 Graph Colorings

For a graph $G = (V, E)$ a *graph coloring* or simply *coloring* is the mapping of its vertices to a set of colors. A *color* is an identifier that can be assigned to a vertex or edge of a graph. Similar to vertices, for a set of colors C, each color is identified by a unique natural number $C \subset \mathbb{N}_{>0}$.
A coloring using $k = |C|$ colors is called a k-coloring. A k-coloring is represented by a function $\text{col} : V \to 1, \ldots, k$. It is *valid* or *legal* if no adjacent vertices are assigned the same color. An edge $\{u, v\}$ causes a *conflict* if $\text{col}(u) = \text{col}(v)$. If a color c can be assigned to a vertex $v \in V$, it is considered free or *available* to v. The *saturation degree* of v is the number of distinct colors in its neighborhood $\text{sat}(v) = |\{\text{col}(u) \mid u \in N(v)\}|$.
If a graph G is k-colorable, a valid k-coloring for G exists. The *Graph Coloring Problem* (GCP) demands to find the smallest possible value for k - also known as the *chromatic number* χ_G - such that G is k-colorable. We denote such a coloring as $\chi(G)$. The *dynamic GCP* applies the GCP to a dynamic graph by ensuring a proper coloring for each $G(t)$. The GCP is NP-complete [13].
We solve the dynamic GCP by creating a subgraph $G_s = (V_s, E_s)$ for every edge added and solving the static GCP for G_s via an ILP. For this we need to consider the neighborhood $N(G_s)$, meaning we need to solve the GCP on the non-induced subgraph

$$G'_s = G_o[V_s \cup N(G_s)] \setminus \{\{u, v\} \in E_o \mid u, v \in N(G_s)'\}$$

that omits edges whose endpoints are both in the neighborhood of G_s. See Figure 5.2 for a visualization. We will refer to $F'_s = N(G_s)$ as the *fixated neighborhood* or *fixated vertices* of G'_s while $M'_s = V_s$ are the *mutable vertices* of G'_s. Similarly, we will refer to C_F as *fixated* and C_M as *mutable colors*. In Section 5.1.2 the colors $c \in C$ are mapped into a different *color space* $\gamma \in \mathscr{C}$. The mapped color of a vertex v is $\text{col}_\gamma(v)$

2.3 Integer Linear Programming

An *Integer Programming* (IP) is way to formulate a mathematical optimization problem. It uses integer *variables*, whose values are restricted by *constraints*, to achieve the goal of maximizing or minimizing an *objective function*. An example can be found in IP 2.1a. IP is NP-complete [23].
The variables in an IP are a vector $x \in \mathbb{Z}^n$. They are decision points needed to be optimized to fulfill the objective function. A constraint is an equation that uses variables of the IP. It creates restrictions, dependencies and relations between these variables. An IP is *feasible* if there exists a combination of variables such that all constraints are satisfied (see IP 2.1b). The *search space* is the space of all feasible solutions. The objective function is the function to be optimized. It represents the goal of the IP and yields the *optimal solution* S_{IP} of the search space.

$$\begin{aligned} \text{minimize} \quad & 4a - 3b \\ \text{subject to} \quad & a + b \leq 3 \\ & a, b \geq 0 \end{aligned}$$

(a) Basic IP

$$\begin{aligned} \text{maximize} \quad & 2x + y \\ \text{subject to} \quad & x + y \geq 3 \\ & 2x + y \leq 2 \\ & x, y \geq 0 \end{aligned}$$

(b) Infeasible IP

IP 2.1: Example IPs

Integer Linear Programming. In *Integer Linear Programming* (ILP) the constraints as well as the objective function are linear. The only exception are integer constraints. The variables of ILPs used in this thesis are all binary. The *dimension* or *problem size* of an ILP refers to its number of independent variables n_{ILP} and constraints m_{ILP}. A larger dimension ILP (in general) requires more computational resources to solve. Each ILP has an *upper bound* (U_{ILP}) and a *lower bound* (L_{ILP}). The optimal solution χ_{ILP} lies between the two. If $L_{\text{ILP}} = U_{\text{ILP}} \Rightarrow L_{\text{ILP}} = U_{\text{ILP}} = S_{\text{ILP}}$.

ILP Solver. ILP solvers are specialized software that are designed to find an optimal solution for a given ILP. They are highly optimized and use sophisticated mathematical techniques to face the complexity of large ILP.
Branch and bound is a method that looks at branches of a decision tree created by the (decision) variables. It *explores* those which potentially bear a feasible solution better than a current result and *prunes* all that do not lie within the upper and lower bounds of the ILP. By using *heuristics* an ILP solver uses techniques that trade optimality and accuracy for speed. With this, new bounds can be set and used by other methods. Besides the two mentioned, there are many more but - for this thesis - less relevant procedures.

There is a great choice of ILP solvers like IBM's CPLEX [18], GLPK (GNU Linear Programming Kit) [14] or Gurobi Optimizer [15]. We choose the latter for this thesis, as it has excellent documentation and is commonly used in academic papers.

Symmetries. An ILP is symmetric when there are multiple feasible variable permutations for which the result of the objective function is the same. E.g. for the GCP, colors are indifferent and thus interchangeable. This causes redundancy for the ILP solver as it explores multiple equivalent solutions, making it more inefficient.

CHAPTER 3

Related Work

The Graph Coloring Problem (GCP) is amongst the most studied NP-hard problems in graph theory and computer science in general. Its research spans several decades with possibly the most well known result being the Four Color Theorem. Originally conjectured in 1852, it states that any planar graph is 4-colorable. Such a graph can be drawn on a plane without intersecting edges. This was proven in 1976 by Appel and Haken [1]. It was one of the first theorems proving via computer assistance. Many algorithms have been proposed to solve the generalized GCP, mostly optimized for a specific subset of graphs. Nevertheless, only a few *exact* algorithms have been developed because of the significant complexity of the problem. Lund and Yannakakis [27] have shown that it is NP-hard to compute an efficient approximation within a fixed ratio. In this chapter, we will explore some of the work done regarding the GCP.

3.1 Exact Algorithms

Exact algorithms for the GCP aim to find an optimal solution, meaning a coloring with the lowest number of colors necessary, by exploring all possible configurations. These methods guarantee finding the chromatic number of the considered instance, but their calculations are computational expensive due to the GCP being NP-hard.
A well known approach to the problem is DSATUR (degree of saturation) by Brélaz [5]. DSATUR utilizes the color degrees to create a color ordering, choosing a vertex dynamically at each step. The initially colored vertex is determined by maximum degree. Subsequent vertices are chosen by highest saturation degree until all vertices in the graph are colored. The algorithm is exact by a process called backtracking, where all possible colorings are systematically explored. The rule mentioned above is applied recursively until a coloring is found, the size of which is saved. It then tracks back to the last decision made and chooses a different vertex than before. If it is clear that a coloring won't undercut the current minimum, the latest process stops and backtracks, as it won't result in a

lower solution size. Further work was done by Lawler [24] who proposed a dynamic programming algorithm. Mehrotra and Trick [29] combine integer linear programming and column generation techniques. For more in depth examples we are referring the reader to de Lima and Carmo [10].
ILPs are also used exclusively to solve the GCP. Different methods are applied to formulate and solve the problem. There are simple assignment based ILPs, where the basic formulation of the GCP is translated into an ILP, as well as ILPs that utilize the structures of a valid coloring, creating disjoint classes which represent colors. In the latest developments partial ordering based ILPs were proposed that can also be formulated as a hybrid of the assignment based ILPs. We will go into great detail of the state of all these ILPs in Chapter 4.

3.2 Dynamic GCP Algorithms

Dynamic graph coloring algorithms try to keep the coloring size of a graph low while edges are deleted or inserted. Like the static GCP, there has been extensive research regarding the dynamic GCP [34] [33] [38] [16] [4].

Preuveneers and Berbers [34] provide an upgrade to the basic approach of assigning the lowest available color to one of the conflicting vertices after an edge insertion. This method produces a quickly increasing number of colors used. Preuveneers and Berbers' ACODYGRA updates the lower degree vertex based on the lowest available color. Furthermore, if only colors larger than already present in the vertices are assignable, the degrees of saturation in the neighborhood is calculated. The maximum degree of saturation for each color present in the neighborhood is stored. Finally, the color whose vertices show the lowest maximum saturation degree is chosen to be the new color assigned to the initial vertex. This will cause conflicts with neighboring vertices, which are resolved by recursively applying the same coloring algorithm to the conflicting vertices. Beyond this, said coloring algorithm is applied to the involved vertices after an edge removal. This potentially lowers the number of colors.
Henzinger and Peng [16] propose an algorithm that has a constant expected amortized update time for maintaining a $(\deg_{\max} + 1)$ coloring for edge insertion and deletion on a graph whose maximum degree is $\deg_{\max}$. They use a sophisticated rank based system paired with efficient data structures. When a conflict causing edge is added the algorithm recursively recolors vertices based on their rank and the colors present in their neighborhood.

CHAPTER 4

State of the Art ILPs for the GCP

In this chapter we talk about the most well known and performant static ILPs. These are the same that we will adapt to the dynamic GCP version in Chapter 5. Furthermore, we will take a look at the most commonly used preprocessing techniques for the GCP.
Jabrayilov and Mutzel [19] introduce a purely partial-ordering based ILP (POP), as well as a hybrid version (POP2). As a comparison they use two previously established ILPs. Firstly, an assignment-based ILP that was refined by Méndez-Díaz and Zabala [31] to be asymmetrical (ASS). Secondly, a representatives ILP by Campêlo et al. [6].

4.1 Assignment-Based ILP (ASS)

Given a graph $G = (V, E)$, each vertex $v \in V$ is assigned a color c. To avoid symmetries, Méndez-Díaz and Zabala [31] propose the rule that assigning a color c to a vertex is only possible if another vertex has already been assigned the color c_{i-1}.
With the number of colors being upper bound by U, binary variables x_{vc} for each vertex $v \in V$ and color $c = 1, \ldots, U$ are implemented to denote the use of a color for a vertex. $x_{vc} = 1$ if vertex v was assigned the color c and $x_{vc} = 0$ otherwise. Furthermore, binary variables z_c are used to indicate if a color c was assigned to any vertex in the solution. The objective is to reduce this number used colors in the result.
The constraints are defined in such a way as to prevent a vertex being assigned multiple colors (4.2), as well as adjacent vertices not being attributed the same color (4.3). Equation (4.3) also ensures that the binary variable z_c is set to 1 iff any vertex is assigned the color c. Colors are used in order due to constraint (4.4).
The number of variables is $|V|U + U = U(|V| + 1)$. The number of constraints is $|V| + U|E| + (U - 1) = |V| + U(|E| + 1) - 1$.

$$\text{minimize} \quad \sum_{c=1}^{U} z_c \tag{4.1}$$

$$\text{subject to} \quad \sum_{c=1}^{U} x_{vc} = 1 \qquad \forall v \in V \tag{4.2}$$

$$x_{uc} + x_{vc} \leq z_c \qquad \forall \{u, v\} \in E,\ c = 1, \ldots, U \tag{4.3}$$

$$z_{c-1} \geq z_c \qquad \forall c = 2, \ldots, U \tag{4.4}$$

$$x_{vc}, z_c \in \{0, 1\} \qquad \forall v \in V,\ c = 1, \ldots, U \tag{4.5}$$

ILP 4.1: Assignment-Based (ASS), [19, pp. 3-4]

4.2 Representatives ILP (REP)

REP [6] solves the GCP by creating classes which are disjoint. Since any arrangement of disjoint classes creates a valid solution for the GCP by assigning a color to each class, the objective of the ILP is to find the smallest set of disjoint color classes for a graph $G = (V, E)$.
To achieve this, class representatives are used. Let $\bar{N}(u) = \{v \mid \{u, v\} \notin E\}$. For all $u \in V$, $v \in \bar{N}(u)$ the binary variable x_{uv} denotes if v is represented by u. This includes x_{uu} which is 1 iff u is the representative of its color class.

$$\text{minimize} \quad \sum_{u \in V} x_{uu} \tag{4.6}$$

$$\text{subject to} \quad \sum_{u \in \bar{N}(v)} x_{uv} \geq 1 \qquad \forall v \in V \tag{4.7}$$

$$x_{uv} + x_{uw} \leq x_{uu} \qquad \forall u \in V,\ \forall \{v, w\} \in G[\bar{N}(u)] \tag{4.8}$$

$$x_{uv} \in \{0, 1\} \qquad \forall \{u, v\} \notin E \tag{4.9}$$

ILP 4.2: Representatives (REP), [19, p. 4]

Inequality (4.7) ensures that each vertex is represented and therefore part of a color class. (4.8) enforces that a representative of another vertex must represent itself, as it cannot be part of more than one color class and each class can only have one representative.
As noted by Jabrayilov and Mutzel [19], REP is very compact, requiring only $|\bar{E}| + |V|$ variables and at most $|V| + |V||E| = |V|(|E| + 1)$ constraints. $\bar{E} = (V \times V) \setminus E$ being the set of non-adjacent pairs of vertices. A drawback of the ILP is its symmetry. Every member of a color class can be the representative. Jabrayilov and Mutzel [19] acknowl-

edge this but highlight that a possible solution provided by Campêlo et al. [7] results in up to exponentially many constraints. They stick to the symmetric version as described above.

4.3 Pure Partial-Ordering-Based ILP (POP)

For a graph $G = (V, E)$ Jabrayilov and Mutzel [19] create a partial order on $P = V \cup C$ with $C = 1, \ldots, U$ being colors in a linear ordering. Each vertex $vinV$ is set into relation to every color. v can be smaller ($v \prec c$) or larger ($v \succ c$) than a color $c \in C$. The binary variables $\text{prec}_{v,c}$ and $\text{succ}_{c,v}$ result from these comparisons. For each pair (v, c), $\text{prec}_{v,c}$ is 1 iff $v \prec c$ and $\text{succ}_{c,v}$ is 1 iff $v \succ c$. To translate the partial ordering into a coloring, a vertex is assigned a color iff $\text{prec}_{v,c} = 0 \wedge \text{succ}_{c,v} = 0$.
A random vertex $r \in V$ is selected. POP is designed to assign r the largest color in the solution (4.16), representing the size of the coloring.

$$\text{minimize} \quad \sum_{c \in C} \text{succ}_{c,r} \tag{4.10}$$

$$\begin{aligned}
\text{subject to} \quad & \text{prec}_{v,1} = 0 && \forall v \in V && (4.11)\\
& \text{succ}_{U,v} = 0 && \forall v \in V && (4.12)\\
& \text{succ}_{c,v} - \text{succ}_{c+1,v} \geq 0 && \forall v \in V,\ c \in C \setminus \{U\} && (4.13)\\
& \text{succ}_{c,v} + \text{prec}_{v,c+1} = 1 && \forall v \in V,\ c \in C \setminus \{U\} && (4.14)\\
& \text{prec}_{u,c} + \text{succ}_{c,u} + \text{prec}_{v,c} + \text{succ}_{c,v} \geq 1 && \forall \{u, v\} \in E,\ c \in C && (4.15)\\
& \text{succ}_{c,r} - \text{succ}_{c,v} \geq 0 && \forall v \in V,\ c \in C \setminus \{U\} && (4.16)\\
& \text{prec}_{v,c}, \text{succ}_{c,v} \in \{0, 1\} && \forall v \in V,\ c \in C && (4.17)
\end{aligned}$$

ILP 4.3: Partial-Ordering-Based (POP), [19, p. 6]

In addition to a lower and upper color limit for the vertices (4.11, 4.12), the constraints ensure color transitivity for each vertex (4.13, 4.14). (4.15) prevents adjacent vertices from being assigned the same color. The number of variables is $2 \cdot U|V|$. As mentioned by Jabrayilov and Mutzel [19], $|V| + |V| + |V|(U - 1) = |V|(U + 1)$ variables can be eliminated via (4.11), (4.12) and (4.14). This results in $|V|(U - 1)$ variables. The number of constraints is $2|V| + 3 \cdot |V|(U - 1) + |E|U = |E|U + 3 \cdot U|V| - |V|$.

4.4 Hybrid Partial-Ordering-Based ILP (POP2)

Though POP has a lower number of variables than ASS, Jabrayilov and Mutzel [19] point out a drawback that occurs when handling dense graphs. The number of non-zero coefficients outgrows that of ASS. To solve this, they substitute constraint (4.15) with (4.3).

Both inequalities handle the coloring of adjacent vertices, but (4.3) uses only half as many non-zero coefficients. This requires a conversion of the POP variables $\text{prec}_{v,c}$ and $\text{succ}_{c,v}$ to the assignment variables x_{vc} which is done by equation (4.23) and the subsequent addition of these variables into the ILP. Preventing neighboring vertices being assigned the same color is handled similarly to ASS (4.24).

$$\begin{aligned}
&\text{minimize} && \sum_{c \in C} \text{succ}_{c,r} && && (4.18)\\
&\text{subject to} && \text{prec}_{v,1} = 0 && \forall v \in V && (4.19)\\
& && \text{succ}_{U,v} = 0 && \forall v \in V && (4.20)\\
& && \text{succ}_{c,v} - \text{succ}_{c+1,v} \geq 0 && \forall v \in V,\ c \in C \setminus \{U\} && (4.21)\\
& && \text{succ}_{c,v} + \text{prec}_{v,c+1} = 1 && \forall v \in V,\ c \in C \setminus \{U\} && (4.22)\\
& && x_{vc} + (prev_{v,c} + \text{succ}_{c,v}) = 1 && \forall v \in V,\ c \in C && (4.23)\\
& && x_{uc} + x_{vc} \leq 1 && \forall \{u,v\} \in E,\ c \in C && (4.24)\\
& && \text{succ}_{c,r} - \text{succ}_{c,v} \geq 0 && \forall v \in V,\ c \in C \setminus \{U\} && (4.25)\\
& && \text{prec}_{v,c}, \text{succ}_{c,v}, x_{vc} \in \{0,1\} && \forall v \in V,\ c \in C && (4.26)
\end{aligned}$$

ILP 4.4: Hybrid Partial-Ordering-Based (POP2)

$|V| \cdot |C|$ variables and constraints are added respectively, resulting in a total of $|V|(U - 1 + |C|)$ variables and $|E|U + 4 \cdot U|V| - |V|$. As mentioned by Jabrayilov and Mutzel [19] the additional variables do not impact the dimensions of the problem as their value is directly tied to the POP variables.

4.5 Heuristic Preprocessing Techniques

A number of preprocessing techniques are used when facing the GCP. These are used to either pre-color vertices, eliminate vertices to decrease the problem size or calculate better lower or upper bounds. Each of these criteria has a different impact to the time required to calculate a solution, depending on the type of graph. A special focus is set on the lower and upper bound though. The size of the assignment and partial-ordering based ILPs directly depends on the upper bound of colors required in the solution. Furthermore, any narrowing of the solution size by a good lower bound can be used by the ILP solver to improve pruning. We will explain three methods used in static approaches, which are later modified by us in Section 5.1.3 to be used with the dynamic ILPs. Some are also part of the set of preprocessing implemented by Jabrayilov and Mutzel [19].

4.5.1 Calculating a Lower Bound via a Clique

Calculating a clique before solving the GCP is a straight forward way to calculate a lower bound for color usage. Both cliques and graph colorings involve a vertex adjacency relevance by definition. Because a clique is a complete subgraph $G_s = (V_s, E_s)$ of a graph G, solving the GCP on S requires colors $C = 1, \ldots, |V_s|$, each vertex being assigned a different color. The chromatic number of S is $\chi_S = |V_s|$. This subsequently means that for the chromatic number of G it holds that $\chi_G \geq \chi_S$. The larger the considered clique, the better the lower bound for the GCP. Finding the optimal clique is a challenge as the maximum clique problem is NP-hard [13]. While there is extensive research on this topic [40], none of the results are feasible for preprocessing due to their complexity. Heuristic approaches on the other hand provide a good middle ground of runtime and result quality. They will not necessarily find an optimal solution, but provide a more feasible one compared to randomly guessing while keeping the algorithms' complexity, and thus runtime, low. We will use a variation of the procedure described by Segundo [35]. It acts like a heuristic DSATUR (see Algorithm 2) to form an initial coloring. This takes advantage of the observation that in a relatively regular graph, vertices with a higher degree have a higher chance of being part of a large clique. Starting with the highest degree vertex a set is created. The highest degree vertex connected to all vertices in this set is chosen and added to the clique. This process repeats until no vertex with this property exists. Ties in degree are broken lexicographically. The outcome is a clique formed by the chosen vertices. See Algorithm 1 for pseudocode.

The resulting set of clique-vertices can be pre-colored and used to eliminate variables by pre-assigning the corresponding values to the associated coefficients. The size of the set is passed on to the ILP solver as a lower bound for the objective function. Heuristic clique approaches are poor approximation still. This is due to the fact, that cliques can be located anywhere in the graph. A large clique can exist isolated from any parameterizable attributes, like the highest degree nodes, making it impossible to find it without an extensive analysis.

4.5.2 Calculating an Upper Bound via Greedy Coloring

Algorithms performing a greedy coloring assign the smallest available color to a vertex by considering a predetermined strict ordering. This order is of particular importance as it affects the solution size in terms of number of colors used. An optimal ordering would result in an optimal solution for the GCP [17]. This means such an ordering would solve the GCP, thus finding one is NP-hard as well. Every strict ordering results in a valid upper bound for the GCP.

As mentioned in Section 3.1, Brélaz [5] introduced DSATUR which relies on degrees of saturation to pick the next vertex to be colored, after initially doing so with the highest degree vertex.

$$v_{\text{chosen}} = \max_{v \in \bar{V}_c} \text{sat}(v)$$

Algorithm 1: Heuristic clique creation, modified from description of Segundo [35]

Input: graph G
Output: clique-set C
begin
  $C \leftarrow \{\}$
  $v \leftarrow highestDegreeVertex(G)$
  $C \leftarrow C \cup \{v\}$
  while *true* **do**
    `// Get highest degree vertex connected to all clique vertices`
    $v \leftarrow nextCliqueVertex(G, C)$
    if *v* has no *value* **then**
      break
    $C \leftarrow C \cup \{v\}$

with $\bar{V}_c \subset V$ being the set of uncolored vertices. Ties in saturation degree are broken by degree. If the tie persists the lexicographically higher vertex is colored. The heuristic version does not perform backtracking but colors the graph once.
Sewell [36] improved by this with his tie-breaking strategy SEWELL. It uses a similar strategy to DSATUR but tries to minimize the colors available to the uncolored subgraph. It picks the next vertex by highest saturation degree. The difference is in the tie-breaking strategy, that chooses the highest saturation degree vertex which has the most colors commonly available to the uncolored vertices of its neighborhood.

$$v_{\text{chosen}} = \max_{v \in \bar{V}_{\text{sat}}} \sum_{n \in N(v),\, n \in \bar{V}_c} |C_a(v) \cap C_a(n)|$$

where $\bar{V}_{\text{sat}}$ is the set of uncolored vertices with the highest saturation degrees and $C_a(v)$ is the set of colors available to v. A further improvement was done by Segundo [35]. His tie-breaking rule called PASS only considers the uncolored neighborhood with maximum saturation degree.

$$v_{\text{chosen}} = \max_{v \in \bar{V}_{\text{sat}}} \sum_{n \in N(v), n \in \bar{V}_{\text{sat}}} |C_a(v) \cap C_a(n)|$$

The mentioned improvements are considered when using DSATUR, SEWELL and PASS as a base for an exact algorithm. In Section 5.1.3 we will use a variation of the original DSATUR rule to heuristically pre-solve the dynamically created GCP for an upper bound. We choose it due to its simplicity in implementing and adapting to a dynamic environment.

Algorithm 2: Heuristic DSATUR, Brélaz [5]

Input: graph $G = (V, E)$
Output: coloring C
begin
  while *true* **do**
    if G empty **then**
      break
    $\text{sat}_{\text{max}}, \deg_{\text{max}} \leftarrow -1$
    forall $v \in V$ **do**
      if $\text{sat}_G(v) > \text{sat}_{\text{max}}$ **then**
        $v_{\text{chosen}} \leftarrow v$
        $\text{sat}_{\text{max}} \leftarrow \text{sat}_G(v)$
        $\deg_{\text{max}} \leftarrow \deg_G(v)$
      else if $\deg_G(v) > \deg_{\text{max}} \wedge \text{sat}_G(v) = \text{sat}_{\text{max}}$ **then**
        $v_{\text{chosen}} \leftarrow v$
        $\deg_{\text{max}} \leftarrow \deg_G(v)$
    $C(v_{\text{chosen}}) \leftarrow \text{smallestColorAvailable}(v_{\text{chosen}})$
    $G \leftarrow G - v_{\text{chosen}}$

4.5.3 Removing Dominated Vertices

A dominated vertex is a vertex whose neighborhood is a subset of another vertex's neighborhood. The dominators' constraints for a GCP ILP are part of a more strict set compared to set of constraints involving the dominated vertex. The latterly mentioned vertex can be omitted from the problem, as its constraints will be satisfied when the dominators' are. This means a color assigned to the dominator can also be applied to the dominated vertex after solving. Omitting vertices reduces the problem size for the ILP as well as other preprocessing such as the heuristic calculation for an upper bound via DSATUR. This is a wide-spread technique and is commonly used [7, 19, 20, 35].

CHAPTER 5

Creating Dynamic Graph Coloring ILPs

This chapter shows our method of using ILPs on subgraphs to solve the GCP for a dynamic Graph. We start by exploring the concepts and data structures needed to apply a local solution to the global graph. Afterwards we will show how we adjust the static ILPs from Chapter 4 to being capable of using this additional data.
The basic concept of our approach (see Figure 5.1) is to use the place of an edge insertion in the global graph as the center for a subgraph. This subgraph is induced by a breadth-first search (BFS) with both edge endpoints as its root. We fixate the subgraphs' neighborhood to consider its colors to be able to reinsert the subgraph into the original graph after solving the GCP on it via our ILPs.

5.1 Preparing Data

In this section we will illustrate what data, other than the subgraph itself, is needed in our approach. We will explain the structures and algorithms used to obtain and process this data. Furthermore, we will explain the benefits of color mapping as well as preprocessing and its impact on the original graphs coloring.

5.1.1 Neighborhood Creation

Contrary to an ILP for the static GCP, our ILPs need to consider the neighborhood $N(G_s)$ of the subgraph $G_s = (V_s, E_s)$ that is to be colored. This is necessary, as the coloring needs to be re-inserted into the original graph $G_o = (V_o, E_o)$. For this reason the BFS will perform a search of depth $d_{\text{BFS}} + 1$ with the endpoints of the inserted edge as its roots. We mark all vertices $F'_s = \{v \in V_o \mid d(v) = d_{\text{BFS}} + 1\}$ as *fixated*. This is the neighborhood $N(G_s)$. All other vertices $M'_s = \{v \in V_o \mid d(v) \leq d_{\text{BFS}}\} = V_s$ are considered *mutable*. $G'_s = G_o[M'_s \cup F'_s] \setminus \{\{u, v\} \in E_o \mid u, v \in F'_s\}$ is the subgraph that is provided to the ILP to be solved. Note that G_s is an induced subgraph of G_o, while G'_s is not. It is missing

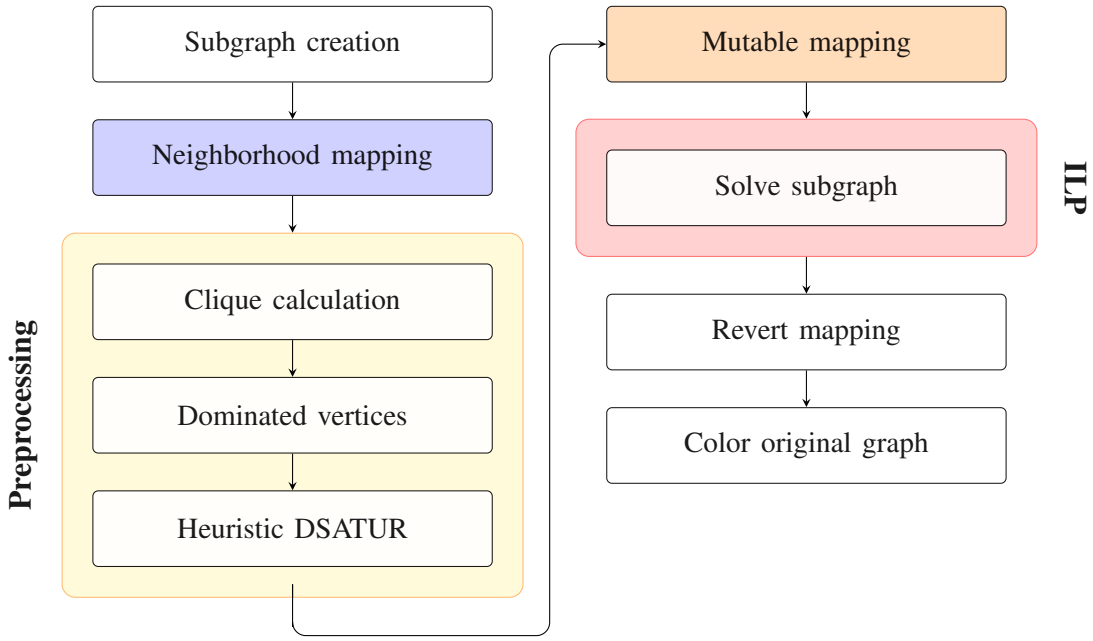


Figure 5.1: Overview of our algorithm for a dynamic graph coloring.

edges between fixated vertices, as they do not impact the feasibility of a coloring. G'_s can be described as divided into different layers, see Figure 5.2. A fixated vertex $f \in F'_s$ will be assigned a color before solving, namely the color $c_f = \text{col}_{G_o}(f)$ it was allocated to in the original graph.

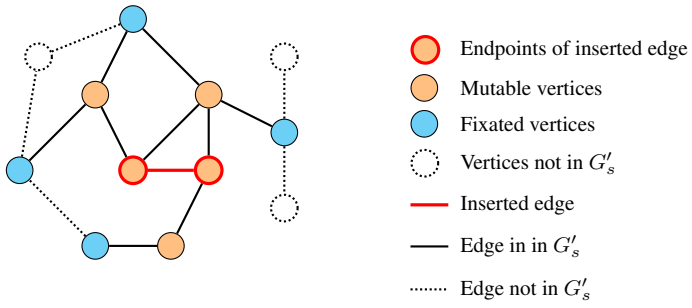


Figure 5.2: Different layers of subgraph G'_s for $d_{\text{BFS}} = 1$

The inclusion of the fixated neighborhood introduces new variables and constraints to an ILP. Though, as their color is pre-assigned, the dimension of the ILP remains unchanged compared to providing purely mutable vertex based subgraph G_s. More constraints usually raise the computational complexity due to the ILP solver being tasked to determine the values in constraints consisting solely of decision variables. With pre-assigned variables however, the solver can use the knowledge to perform better bound calculation, reduce the

Algorithm 3: Subgraph via BFS

Input: original graph G_o, new edge $e = (v_1, v_2)$, BFS-depth d_{BFS}
Output: subgraph G'_s
begin
  `// Queue with (vertex, depth) pairs`
  $Q \leftarrow \text{push}\big(Q, (v_1, 0)\big)$
  $Q \leftarrow \text{push}\big(Q, (v_2, 0)\big)$
  G'_s.vertices $\leftarrow \{v_1, v_2\}$
  G'_s.edges $\leftarrow \{\{v_1, v_2\}\}$
  while not Q.empty() **do**
    $(v, d) \leftarrow Q$.front()
    $Q \leftarrow \text{pop}(Q)$
    if $d > d_{\text{BFS}}$ **then**
      v.isFixated $\leftarrow$ **true**
      continue
    v.isFixated $\leftarrow$ **false**
    forall $a \in G_o$.adjacent(v) **do**
      if a.isVisited **then**
        continue
      a.isVisited $\leftarrow$ **true**
      $Q \leftarrow \text{push}\big(Q, (a, d+1)\big)$
      G'_s.vertices $\leftarrow G'_s$.vertices $\cup \{a\}$
      G'_s.edges $\leftarrow G'_s$.edges $\cup \{\{v, a\}\}$

search space, check the feasibility more easily and enhance numerical stability. We will explore the ILP specific effects in Section 5.2.

We use a queue $Q = \langle (v_1, d(v_1)), \ldots, (v_n, d(v_n)) \rangle$ with $v_i \in V_o$ and $d(v_i)$ being v's depth, to perform the BFS with depth d_{BFS} on the original graph G_o. New entries are *pushed* into the queue via $\text{push}\big(Q, (v_{n+1}, d(v_{n+1}))\big) \rightarrow Q' = \langle (v_1, d(v_1)), \ldots, (v_n, d(v_n), (v_{n+1}, d(v_{n+1}))) \rangle$ and *popped* from the front of the queue with $\text{pop}(Q) \rightarrow Q' = \langle (v_2, d(v_2)), \ldots, (v_n, d(v_n)) \rangle$.
Initially our queue Q contains the endpoints of the inserted edge $e = \{v_1, v_2\}$. We then enter a loop that will continue until the queue is empty (see Algorithm 3 for details). We look at and remove the front pair $(v, d(v))$ of the queue and examine all its neighbors $n \in N_o(v)$ individually. If n was not yet part of the queue, we push $(n, d(n) \mid d(n))$ with $d(v) + 1$ into Q. If we already considered the neighbor, we do nothing. We also skip v if $d(v)$ is larger than d_{BFS}. In this case v is part of the fixated neighborhood and we mark it as such.

5.1.2 Neighborhood Mapping

As shown with the static ILPs, the dimension and number of constraints is in many cases dependent on the upper bound U of colors used in the solution $\chi(G)$. Because the colors of fixated neighborhood vertices F'_s in the subgraph $G'_s = (V'_s, E'_s)$ with $V'_s = F'_s \cup M'_s$ must be predetermined, this upper bound would be

$$U' = \max\big(U, \max_{f \in F'_s}(\text{col}(f))\big)$$

with U being an upper bound for the optimal coloring $\chi(G_s)$ of the mutable subgraph $G_s = (V_s, E_s)$, considering adjacent fixated colors but not including the fixated colors in the bound themselves. This can lead to a larger than necessary upper bound and consequently to a lower performing ILP due to larger dimensions.
We counter this behavior by creating a bijective mapping of the colors assigned to the fixated vertices $f \in F'_s$. We will differentiate these colors by *color spaces*. There is an *original* (C) and a *mapped* ($\mathscr{C}$) color space. Let $\mathscr{C}_F = \{1, \ldots, |F'_s|\}$ be the set of mapped fixated colors with $C_F = \{c_1, c_2, \ldots, c_n\} \mid c_1 < c_2 < ... < c_n$ being the ordered set of colors used in the fixated neighborhood. We define the map function for fixated colors $\text{map}_{\mathscr{C}_F} : C_F \to \mathscr{C}_F$ such that

$$\text{map}_{\mathscr{C}_F}(c_i) = \gamma_i \quad \text{if } c_i \text{ is the } \gamma_i\text{-th element in } C_F$$

This will lower the upper bound to

$$U' = \max(U, |C_F|) = \max(U, |\mathscr{C}_F|)$$

The mapping $\text{map}_{\mathscr{C}_F}(C_F)$ must occur before any preprocessing involving colors, otherwise their calculations would rely on C while the subgraph to be solved is using $\mathscr{C}$ as its color space. This would mean upper and lower bounds are not determined correctly. The colors $\gamma \in \mathscr{C}_F$ could produce conflicts for low color values, not present in the original color space. This potentially increases the highest color value assigned to a mutable vertex $v \in M'_s$, invalidating the previous bounds.

A mapped color γ_i can be *translated* back to the original color space C by accessing the γ_i-th element in C_F. We define the translation function for fixated colors trans_{C_F} such that

$$\text{trans}_{C_F}(\gamma_i) = C_F[\gamma_i]$$

To implement the color spaces, we use a vector $v_{\mathscr{C}} = [c_1, \ldots, c_{|C_F|}]$ whose indices are the mapped colors $\gamma_i \in \mathscr{C}_F$ corresponding to the original colors $c_i = v_{\mathscr{C}}[\gamma_i]$. Initially $v_{\mathscr{C}}$ is an empty vector. We can *append* colors to it via $\text{append}(v_{\mathscr{C}}, c_i) \to v'_{\mathscr{C}} = [c_1, \ldots, c_{|C_F|}, c_i]$. Let

$$\mathscr{M} = (\{c_i : \{f \in F'_s \mid \text{col}(f) = c_i\}\}\ \forall c_i \in C_F)$$

be a key-value map that sorts all fixated vertices f by their color c_i in the original graph G_o. $\mathscr{M}[c_i]$ returns a set of all fixated vertices assigned the color c_i. We can easily fill this

Algorithm 4: Map fixated neighborhood

Input: subgraph $G'_s = (V'_s, E'_s)$, color mapped vertices $\mathscr{M}$
Output: color translation $v_{\mathscr{C}}$
begin
  $v_{\mathscr{C}} \leftarrow [\,]$
  $\gamma_{\text{next}} \leftarrow 1$
  forall $c_i \in \mathscr{M}$.keys() **do**
    $v_{\mathscr{C}} \leftarrow \text{append}(v_{\mathscr{C}}, c_i)$
    forall $f \in \mathscr{M}[c_i]$ **do**
      $\text{col}_\gamma(f) \leftarrow \gamma_{\text{next}}$
    $\gamma_{\text{next}} \leftarrow \gamma_{\text{next}} + 1$

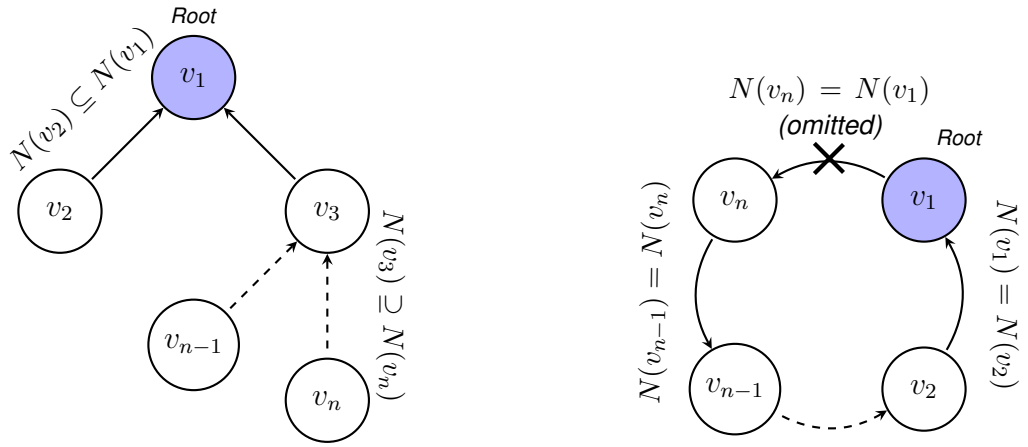
map while creating the subgraph, by adjusting Algorithm 3 to put every fixated vertex f into the set in $\mathscr{M}$ corresponding to its color.
We iterate over the ordered set of keys $c_i \in \mathscr{M}$.keys(). The iteration index γ_i corresponds to the mapped color. We place c_i at $v_{\mathscr{C}}[\gamma_i]$ and assign all vertices $f \in \mathscr{M}[c_i]$ the color γ_i in the subgraph G'_s. See Algorithm 4 for pseudocode.

5.1.3 Preprocessing

Just as with the static GCP, preprocessing can be applied to our method of solving the dynamic version as well. We calculate a clique for a lower bound, identify dominated vertices to be omitted and use Algorithm 2 to get an upper bound. The order of preprocessing is as mentioned above. This is to have a more performant heuristic upper bound calculation as it can omit the dominated vertices.

Some described preprocessing techniques need to take fixated vertices and their mapped colors into account. The process of searching for a clique depends on adjacency only. As the neighborhood vertices can be a part of the clique they should be considered as well to create a more accurate lower bound for the subgraph. No changes to this preprocessing compared to our earlier explanation in Section 4.5.1 needs to be done.

Adapting Dominated Vertices. Dominated vertices must consider fixated vertices as their assigned color is crucial for the calculation of the subgraphs coloring and later reinsertion. For this reason, vertices that are part of the subgraphs' neighborhood are not to be marked as dominated. However, they can become the dominator as well as being considered when examining a neighborhood being subset of another.
Because a dominating vertex can be dominated itself, it is important to keep track of trees of domination shown in Figure 5.3a. Each tree will have a root dominator. This vertex will be references by all dominated vertices when being colored after solving the GCP. The root dominator only changes if is dominated itself. This new dominator will then be the root



(a) Domination tree

(b) Preventing domination cycles

Figure 5.3: Vertex domination pitfalls

for all dominated vertices in the tree. Considering that a vertex u is dominated by a vertex v if $N(u) \subseteq N(v)$, cycles occur for $v_1, \ldots, v_n$ if $N(v_1) = \ldots = N(v_n)$. In this case, the last tracked domination will be omitted to break the cycle (Figure 5.3b). This ensures a root dominator will always be present. The order of domination is determined based on individual implementation, in our case it is lexicographically.

The dominated vertices $\mathscr{D}$ will be removed from the graph $G'_s = (V'_s, E'_s)$, creating a new subgraph $G'_s[V'_s \setminus \mathscr{D}]$. For simplicity, we will continue calling the subgraph for the ILP G'_s. From here on out, this will have its dominated vertices and related edges removed.

Adapting heuristic DSATUR. To ensure the proper heuristic calculation of an upper bound, fixated vertices must be considered. They reduce the set of colors available to a vertex. As mentioned fixated colors are mapped (Algorithm 4) to ensure the largest used color is as small as possible. This also has the effect of guaranteeing the colors used in the heuristic solution - which is in the mapped color space - are as small as possible, while not conflicting with fixated neighbors.
The upper bound usually equals $U = |\mathscr{C}_h|$ with $\mathscr{C}_h$ being the set of colors used in the heuristically produced solution for G'_s. With fixated vertices, this can be lowered to $U = \min(|\mathscr{C}_h|, |\mathscr{C}_M|)$ where $\mathscr{C}_M$ is the set of colors used by the mutable verticies of the subgraph. This is the case, as $|\mathscr{C}_M|$ indicates the maximum number of colors needed for coloring the mutable vertices while considering the fixated neighborhood. A fixated vertex with a color $\gamma_f > |\mathscr{C}_M|$ cannot produce a conflict, as any optimal solution produced by an ILP will assign at most color $|\mathscr{C}_M|$ to a mutable vertex. The fixated vertices with $\gamma_f > |\mathscr{C}_M|$ can be removed from the subgraph problem entirely. Any vertex previously marked as dominated by such a vertex will still be colored by the color of the fixated vertex. This will not increase the number of colors used in the original graph while keeping

the dimension of the ILPs reduced.
By their definition, dominated vertices can be omitted at this stage already without impacting upper bound accuracy but improving runtime of the heuristic algorithm.

5.1.4 Mapping Mutable Vertices

Since we map the fixated colors of the subgraphs' neighborhood, the final coloring will take place in the mapped color space $\mathscr{C}$. After solving the subgraph instance, its colors will be translated from the mapped to the original color space C and assigned to the vertices in the original graph. It is in our interest for the mapped colors $\gamma \in \mathscr{C}$ to be as small as possible even though they might get translated back into a larger (fixated) color. This ensures a small upper bound and thus better performance in the ILP.
With U_M being the upper bound for the mutable vertices of the subgraph in the mapped color space, if $U_M > |\mathscr{C}_F|$, there are $U_M - |\mathscr{C}_F|$ colors left that are potentially used in the solution and need to be translated into the original color space. Let $\mathscr{C}_M$ be the set of these colors. We choose colors as small as possible as translation values. This will keep the average coloring value in the original graph as low as possible. It also means that the subgraph coloring will, after translation into the original color space, introduce at most one new color. This is because every edge conflict in a k-coloring can be solved by assigning one of the endpoints the color $k+1$. As G'_s is a subgraph of k-colorable G_o, G'_s (which includes the inserted edge) must be $k+1$-colorable. As we choose the translated colors in the original color space C_M as small as possible, this means at most colors $1, \ldots, k+1$ will be assigned to the original graph. It will at most have a $k+1$-coloring.
Non-possible colors for the translation of $\mathscr{C}_M$ are those used by the fixated vertices in the original color space C_F. Let $\bar{C_F} = \mathbb{N}_{>0} \setminus C_F$ be the increasingly ordered set of all possible colors. We define the translation function for mutable colors $\text{trans}_{C_M} : \mathscr{C}_M \to C_M$ as

$$\text{trans}_{C_M}(\gamma_i) = \bar{C_F}[\gamma_i]$$

This completes all mappings (see Figure 5.4 for a visualization) necessary to change color spaces for the subgraph to be solved with the lowest upper bound possible, while keeping the coloring in the original graph at most $k+1$.

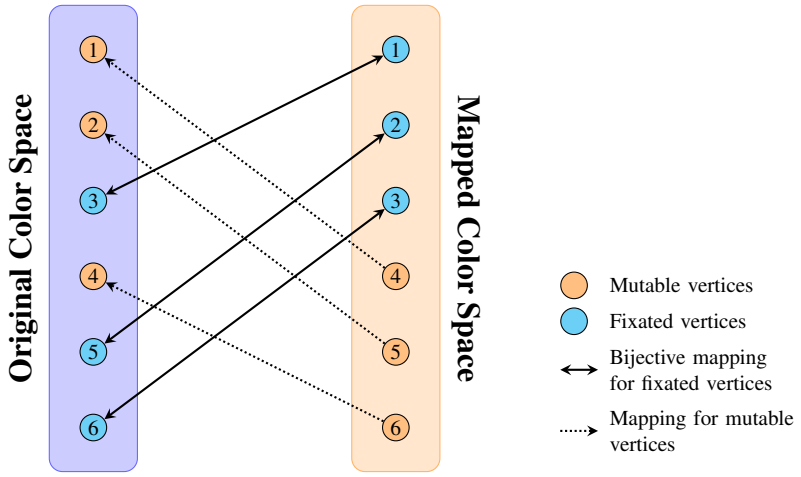


Figure 5.4: Mapping functions for color spaces

5.2 Turning ILPs Dynamic

In this section we will convert the four ILPs from Chapter 4 into being able to use the prepared data. We have calculated lower and upper bounds. The fixated colors of G'_s are converted into the mapped color space $\mathscr{C}$. trans_{C_M} and trans_{C_F} give us the ability to translate both fixated and mutable colors back to the original color space. This means our ILPs do not need to consider the different color spaces. The solution will be in the mapped color space. Still, it is necessary to create extra constraints and variables in the ILPs that factor in the existence of a fixated neighborhood. We will reduce the number of both of these as much as possible.

5.2.1 Include Fixated Neighborhood

Taking into account the colors of the fixated neighborhood is straight forward. We will add binary variables for each vertex v, just as the static ILPs require. If $v \in \mathscr{C}_F$, we will assign a value to the variable, making it a constant. As mentioned, this keeps the dimensions of the ILP the same as if the problem would consist of the mutable subgraph G_s only, as well as creating constraints that can be beneficial for the solution computation.

Dynamic Assignment-Based ILP (DynASS) For the assignment-based ILP, two types of variables are used.

1. $x_{v\gamma}$ to indicate if vertex v is assigned a color γ
2. z_γ to indicate if a color γ is assigned to any vertex in the solution

To assign fixated colors $\mathscr{C}_F$ of the subgraph $G'_s = (V'_s, E'_s)$, $V'_s = F'_s \cup M'_s$ we adjust $x_{v\gamma}$ to be 1 if v is fixated and γ is the color assigned to v. If v has a color that is not γ, $x_{v\gamma}$ is

set to 0. In all other cases it will be a binary variable, as with the static ILP.
We reflect the color usage for all fixated colors as well changing z_γ to be 1 if $\gamma \in \mathscr{C}_F$. Due to $\mathscr{C}_F = \{1, \ldots, |C_F|\}$, we can omit the first $|C_F|$ constraints compared to (4.5). The constraints ensuring only one color is assigned to a vertex can be cut out for all fixated vertices $f \in F'_s$.

$$\text{minimize} \quad \sum_{\gamma=1}^{U} z_\gamma \tag{5.1}$$

$$\text{subject to} \quad \sum_{\gamma=1}^{U} x_{v\gamma} = 1 \qquad \forall v \in M'_s \tag{5.2}$$

$$x_{u\gamma} + x_{v\gamma} \leq z_\gamma \qquad \forall \{u, v\} \in E'_s,\ \gamma = 1, \ldots, U \tag{5.3}$$

$$z_{\gamma-1} \geq z_\gamma \qquad \forall \gamma = |C_F| + 2, \ldots, U \tag{5.4}$$

$$x_{v\gamma} \in \begin{cases} \{1\} & \text{if hasCol}(v) \wedge \\ & \gamma = \text{col}_\gamma(v) \\ \{0\} & \text{if hasCol}(v) \wedge \\ & \gamma \neq \text{col}_\gamma(v) \\ \{0, 1\} & \text{otherwise} \end{cases} \qquad \forall v \in V'_s,\ \gamma = 1, \ldots, U \tag{5.5}$$

$$z_\gamma \in \begin{cases} \{1\} & \text{if } \gamma \in \mathscr{C}_F \\ \{0, 1\} & \text{otherwise} \end{cases} \qquad \gamma = 1, \ldots, U \tag{5.6}$$

ILP 5.1: Dynamic Assignment-Based (DynASS)

The number of variables is $|V'_s|U + U = U(|V'_s| + 1)$, the same as in the static ASS. $|M'_s| \cdot |\mathscr{C}_F| + |\mathscr{C}_F|$ of these are turned into constants, reducing the dimension of the ILP to the same as if G_s, the graph without a neighborhood, was to be considered. The number of constraints is $|M'_s| + U|E'_s| + (U - |C_F| - 1) = |M'_s| + U(|E'_s| + 1) - |C_F| - 1$. There are only as many additional constraints compared to the G_s, as edges $\{u, v\} \in E'_s$ with $u \in M'_s \wedge v \in F'_s$

Dynamic Representatives ILP (DynREP) With a representative based ILP including fixated vertices requires a little more work. Because multiple neighborhood vertices can have the same color γ, but only one representative can exist for γ. To obey this rule, we can use the key-value map $\mathscr{M}$ from Section 5.1.2. For each original color $c_i \in \mathscr{M}$.keys(), we get the mapped color $\gamma_i = \text{map}_{\mathscr{C}_F}(c_i)$. We choose the first vertex in the lexicographical order of vertices to represent the others in their color class. $\mathscr{R} = \{(\gamma_i, \mathscr{M}[c_i].\text{front}())\ \forall c_i \in \mathscr{M}.\text{keys}()\}$ is the set of these fixated representatives paired with their mapped color. There will be $|\mathscr{C}_F|$ such pre-determined color classes. This

reduces the problem symmetry associated with this ILP, as there are fewer representatives to be chosen.
Due to every fixated vertex already being in one color class, we can omit the constraints (4.7) for these vertices. The constraints ensuring no vertex can be the representative of both endpoints of an edge keeps unchanged, as we do not include edges between fixated vertices in the subgraph.

$$\text{minimize} \quad \sum_{u \in V'_s} x_{uu} \tag{5.7}$$

$$\text{subject to} \quad \sum_{u \in \bar{N}(v)} x_{uv} \geq 1 \qquad \forall v \in M'_s \tag{5.8}$$

$$x_{uv} + x_{uw} \leq x_{uu} \qquad \forall u \in V'_s,\ \forall \{v, w\} \in G'_s[\bar{N}(u)] \tag{5.9}$$

$$x_{uv} \in \begin{cases} \{1\} & \text{hasCol}(v) \wedge \\ & \mathscr{R}[\text{col}_\gamma(v)] = u \\ \{0\} & \text{if hasCol}(v) \wedge \\ & \mathscr{R}[\text{col}_\gamma(v)] \neq u \\ \{0, 1\} & \text{otherwise} \end{cases} \qquad \forall \{u, v\} \notin E \tag{5.10}$$

ILP 5.2: Dynamic Representatives (DynREP)

DynREP has the same number of variables as REP but $|\mathscr{C}_F|$ of these are turned into constants. Just as with DynASS, the dimension of DynREP is the same as REP considering only the mutable subgraph G_s compared to G'_s. The number of constraints compared to this is increased by $\sum_{u \in F'_s} \ | \ \{e = \{v, w\} \in G'_s[\bar{N}(u)]\}|$. The higher the density of the graph, the lower the increase in constraints.

Dynamic Partial Ordering Based ILP (DynPOP) Including fixated vertices requires determining their relations in the partial ordering introduced by POP. With $\mathscr{C}_T = \mathscr{C}_F \cup \mathscr{C}_M, |\mathscr{C}_T| = U$, we define $\text{prec}_{f,\gamma} = 1$ iff the color $\gamma_f = \text{col}_\gamma(f)$ is smaller than $\gamma \in \mathscr{C}_T$ for a fixated vertex $f \in F'_s$. Likewise, $\text{succ}_{\gamma,f} = 1$ iff γ_f is larger than γ. In case $\gamma = \gamma_f$, both $\text{prec}_{f,\gamma}$ and $\text{succ}_{\gamma,f}$ are 0, in other words f is assigned the color γ by DynPOP. The static version requires a random vertex r to be chosen from the set of vertices. This is a problem for the dynamic ILP. r cannot be chosen from F'_s as it needs to be variable to adapt to the largest color used. But even when using a mutable vertex, there is the possibility that this will make the ILP infeasible if there are only valid colorings for which $r \in M'_s$ is not assigned the largest coloring. For valid colorings, this restriction also bears the potential of a better solution being omitted by the ILP solver. The solution is to add an isolated vertex i to G'_s. i will only be relevant during the solving of the ILP and removed afterwards.
Looking at the constraints, we can omit $|F'_s|$ from every one except when preventing the

assignment of the same color to adjacent vertices. All pairs (f, γ) with $f \in F'_s$, $\gamma \in \mathscr{C}_T$ are ordered properly. There is no need to ensure transitivity (5.14) or exact relations (5.15) between the pairs (f, γ) for a vertex f. Their upper and lower bound cannot extend out of the allowed range, as the lowest color assigned is 1 and i initially orientates itself at the largest fixated color used. The color assigned to i can only grow.

$$\text{minimize} \quad \sum_{\gamma \in \mathscr{C}_T} \text{succ}_{\gamma,i} \tag{5.11}$$

$$\text{subject to} \quad \text{prec}_{v,1} = 0 \qquad \forall v \in M'_s \tag{5.12}$$

$$\text{succ}_{U,v} = 0 \qquad \forall v \in M'_s \tag{5.13}$$

$$\text{succ}_{\gamma,v} - \text{succ}_{\gamma+1,v} \geq 0 \qquad \forall v \in M'_s,\ \gamma \in \mathscr{C}_T \setminus \{U\} \tag{5.14}$$

$$\text{succ}_{\gamma,v} + \text{prec}_{v,\gamma+1} = 1 \qquad \forall v \in M'_s,\ \gamma \in \mathscr{C}_T \setminus \{U\} \tag{5.15}$$

$$\text{prec}_{u,\gamma} + \text{succ}_{\gamma,u} + \text{prec}_{v,\gamma} + \text{succ}_{\gamma,v} \geq 1 \qquad \forall \{u,v\} \in E'_s,\ \gamma \in \mathscr{C}_T \tag{5.16}$$

$$\text{succ}_{\gamma,i} - \text{succ}_{\gamma,v} \geq 0 \qquad \forall v \in M'_s,\ \gamma \in \mathscr{C}_T \setminus \{U\} \tag{5.17}$$

$$\text{prec}_{v,\gamma} \in \begin{cases} \{1\} & \text{if hasCol}(v) \wedge \\ & \text{col}_\gamma(v) < \gamma \\ \{0\} & \text{if hasCol}(v) \wedge \\ & c < \text{col}_\gamma(v) \\ \{0,1\} & \text{otherwise} \end{cases} \qquad \forall v \in V'_s,\ \gamma \in \mathscr{C}_T \tag{5.18}$$

$$\text{succ}_{\gamma,v} \in \begin{cases} \{1\} & \text{if hasCol}(v) \wedge \\ & c < \text{col}_\gamma(v) \\ \{0\} & \text{if hasCol}(v) \wedge \\ & \text{col}_\gamma(v) < \gamma \\ \{0,1\} & \text{otherwise} \end{cases} \qquad \forall v \in V'_s,\ \gamma \in \mathscr{C}_T \tag{5.19}$$

ILP 5.3: Partial-Ordering-Based (DynPOP)

The number of variables, again, is the same as with the static version POP. $2 \cdot U|F'_s|$ are turned into constant, equaling the amount to the one used by G_s. The number of constraints compared to G_s is increased by the number of edges connecting a fixated and a mutable vertex $\{u,v\} \in E'_s \mid u \in M'_s \wedge v \in F'_s$.

Dynamic Hybrid Partial Ordering Based ILP (DynPOP2) Just as POP2 is a hybrid of ASS and POP, so are the changes to adapt it a combination of their dynamic counter-

parts. The assignment variables $x_{v\gamma}$ are defined as in DynASS, the POP variables $\text{prec}_{v,\gamma}$ and $\text{succ}_{\gamma,v}$ as in DynPOP. Consequently, this also means the number of variables and constants changes as a combination of the two with the same amounts as mentioned with the respective ILP.

$$\text{minimize} \quad \sum_{\gamma \in \mathscr{C}_T} \text{succ}_{\gamma,i} \tag{5.20}$$

$$\text{subject to} \quad \text{prec}_{v,1} = 0 \qquad \forall v \in M'_s \tag{5.21}$$

$$\text{succ}_{U,v} = 0 \qquad \forall v \in M'_s \tag{5.22}$$

$$\text{succ}_{\gamma,v} - \text{succ}_{\gamma+1,v} \geq 0 \qquad \forall v \in M'_s,\ \gamma \in \mathscr{C}_T \setminus \{U\} \tag{5.23}$$

$$\text{succ}_{\gamma,v} + \text{prec}_{v,\gamma+1} = 1 \qquad \forall v \in M'_s,\ \gamma \in \mathscr{C}_T \setminus \{U\} \tag{5.24}$$

$$x_{v\gamma} + (\text{prec}_{v,\gamma} + \text{succ}_{\gamma,v}) = 1 \qquad \forall v \in M'_s,\ \gamma \in \mathscr{C}_T \tag{5.25}$$

$$x_{u\gamma} + x_{v\gamma} \leq 1 \qquad \forall \{u,v\} \in E'_s,\ \gamma \in \mathscr{C}_T \tag{5.26}$$

$$\text{succ}_{\gamma,i} - \text{succ}_{\gamma,v} \geq 0 \qquad \forall v \in M'_s,\ \gamma \in \mathscr{C}_T \setminus \{U\} \tag{5.27}$$

$$\text{prec}_{v,\gamma} \in \begin{cases} \{1\} & \text{if hasCol}(v) \wedge \\ & \text{col}_\gamma(v) < \gamma \\ \{0\} & \text{if hasCol}(v) \wedge \\ & \gamma < \text{col}_\gamma(v) \\ \{0,1\} & \text{otherwise} \end{cases} \qquad \forall v \in V'_s,\ \gamma \in \mathscr{C}_T \tag{5.28}$$

$$\text{succ}_{\gamma,v} \in \begin{cases} \{1\} & \text{if hasCol}(v) \wedge \\ & \gamma < \text{col}_\gamma(v) \\ \{0\} & \text{if hasCol}(v) \wedge \\ & \text{col}_\gamma(v) < \gamma \\ \{0,1\} & \text{otherwise} \end{cases} \qquad \forall v \in V'_s,\ \gamma \in \mathscr{C}_T \tag{5.29}$$

$$x_{v\gamma} \in \begin{cases} \{1\} & \text{if hasCol}(v) \wedge \\ & \gamma = \text{col}_\gamma(v) \\ \{0\} & \text{if hasCol}(v) \wedge \\ & \gamma \neq \text{col}_\gamma(v) \\ \{0,1\} & \text{otherwise} \end{cases} \qquad \forall v \in V'_s,\ \gamma \in \mathscr{C}_T \tag{5.30}$$

(5.31)

ILP 5.4: Hybrid Partial-Ordering-Based (DynPOP2)

Algorithm 5: Translate coloring to original graph

Input: subgraph $G'_s = (V'_s, E'_s)$, graph G_o
Output: color translation $v_{\mathscr{C}}$
begin
  forall $v \in V'_s$ **do**
    $\gamma \leftarrow \text{col}_\gamma(v)$
    if v.isFixated **then**
      $c \leftarrow \text{trans}_{C_F}(\gamma)$
    else
      $c \leftarrow \text{trans}_{C_M}(\gamma)$
    $\text{col}(f) \leftarrow c$

Summary Although, each ILP has to be individually adapted, the changes occurring are similar in each ILP. We see an increase in variables declared, due to the neighborhood F'_s. Because these vertices are pre-colored though, they are turned into constants every time. This keeps the dimension of the ILP the same as when considering G_s. Additional constraints can not be avoided. The increase is in most cases, except DynREP, primarily due to the additional edges between the fixated and mutable variables. This can be beneficial for the ILP solver. Their complexity is lowered, because these constraints include at least one variable of a fixated vertex, which are turned constant. This can also help for different methods applied by the ILP like branch and bound.

5.3 Post Solve Tasks

After solving the ILP on the subgraph, it is necessary to revert the mapping of the colors used in the coloring. As shown in Algorithm 5, this done by our previously defined translation functions trans_{C_M} and trans_{C_F}. We will iterate over each vertex $v \in V'_s$ and translate the assigned color γ_v into the original color space c_v. This color is then assigned to the same vertex in G_o. This will finish the procedure of solving the GCP on the subgraph and including the results in the original graph.

CHAPTER 6

Experimental Evaluation

This chapter shows our results for the experiments. At first, we showcase the hardware used, as well as the graph instances, the experiments were performed on. Afterwards, we will explain the setup and execution of our algorithm. Finally, we will analyze the results and compare the dynamic ILP amongst themselves as well as compared to the results of Jabrayilov and Mutzel [19].

6.1 Methodology

Hardware. All algorithms are implemented in C++ and compiled using g++ version 12.3.0 with optimization level O3. The machine used is equipped with an Intel(R) Xeon(R) Silver 4216 @ 2.10GHz with 16 cores and 96 GB of main memory as well as 22 MiB of L3 cache under Ubuntu 20.04.1 LTS and Linux kernel version 5.4.0-152-generic. We use Gurobi Optimizer [15] 11.0.1 running single-threadedly and with an 80 GB memory limit per instance as our ILP solver.

Experiment Setup. For all our experiments we use the same algorithm setup, except for changing the value of the BFS-depth (Figure 5.1), although we split the graph instances into two types of categories. Graphs derived from finite element methods (FEM) are modified before the experiment, by excluding a set of 100 and 1000 randomly chosen edges. These edges will be used to turn the graphs dynamic, by reinserting them one by one. The reduced graphs are pre-colored using a heuristic DSATUR algorithm (Algorithm 2) and the coloring is saved. All ILPs will use the same coloring as their basis to make them comparable. We want to measure the performance of the ILPs for a fixed number of executions. For this, we will execute our algorithm on *every* edge insertion. Even if there is no conflict. FEM graphs are chosen, because of their irregular structure and large size which leads to a more robust benchmark for a randomly chosen set of edges. The BFS-depths chosen are in a wide range of 0 to 15 to better explore the ILPs' performance when facing different sized subgraphs. All other graph instances will be build from the ground up. We start with a

graph that consists of vertices only and add each edge back into it. The edges are ordered by their smallest endpoint, with ties being broken via the second endpoint, and inserted in ascending order. This ensures a connected graph. We use graphs with a variety of attributes to explore differences in performance for the individual ILP. As we build small graphs from an empty state, the values chosen as depths will be lower, ranging from 0 to 5. Due to the high number of instances, each has a maximal compute time of 30 minutes in both types of experiments.

Instances. The graphs derived from finite element methods (FEM) used in our experiments originate from the 10th DIMACS implementation challenge [2]. They were introduced by Chan et al. [8] and represent 2-dimensional meshes of real world objects. Furthermore, we use a subset of the DIMACS benchmark [39] that is oriented on the set used by Jabrayilov and Mutzel [19]. They are widely used and consist of *FullIns*, derived from graphs based on the Mycielski transformation, *ash*, obtained from matrix partitioning problems, *DSJC* and *DSJR*, which are random and geometric graphs [22], *le450*, Leighton graphs [25], *mugg100*, as almost 3-colorable graphs, *qg*, latin sqaures, *school*, a class scheduling and *wap*, an optical network design problems. A detailed list can be found in Table 6.1 and Table 6.3.

Plots. The results for both experiments include multiple forms of presentation. While some are straight forward to understand, we want to explain how to read *performance profiles*. Introduced by Dolan and Moré [11], this type of graph is a widely used tool to compare the performance of different algorithms across a set of problem instances. It uses a defining metric to create a two-dimensional graph whose x-axis denotes a variable $\tau \in \mathbb{R}_{\geq 1}$. This represents the performance ratio of an algorithm and instance. A value of $\tau = 1$ means that the algorithm performed best on that instance. Higher values relate to this base value, e.g. $\tau = 1.3$ is the value of the best performance $\times$ 1.3.
The y-axis shows the percentage of instances $p(\tau)$ for each algorithm where the performance is within a factor of τ. It accumulates the number of instances, meaning that for $y = 0.5$, 50% of instances for this graph performed better than value of the best performance $\times$ τ. The points $(\tau, p(\tau))$ for an algorithm shape a performance staircase line. Vertical lines are formed at each value of τ where the performance on the next instance is within a factor of τ. This results in an intuitive graphical representation for algorithm comparison, as well as more specific insights. Particular interesting knowledge can be gained at $\tau = 1$ and the point $(\tau, p(\tau) = 1)$. In the first case we can see the percentage of instances solved best by each algorithm. For the latter, we can see the largest factor by which an algorithm is worse compared to the best performance on any of the instances considered. In some of our illustrations, the line never reaches $p(\tau) = 1$. This is the case when instances could not be solved by the ILP in the time limit. If an instance could not be solved by any ILP, the instance is removed from the graph entirely.
We also show graphs where the average value of τ_Δ is presented. For each ILP we calculate the list of differences in τ. The list is generated by finding $\tau_\Delta = \tau - \tau_{\text{best}}$ for each τ_i in the performance profile ILP increases its fraction of instances performing within factor τ_i. This

essentially means, each τ_i where the line forms a vertical line. τ_{best} is the factor at which the best ILP reaches the same performance as the referenced ILP at τ_i. We then multiply each τ_Δ by the fraction of the y-axis for which it is valid and sum up the values to get the average τ_Δ. This metric gives us a condensed numerical value to assign to performance profiles. Please note, that as with the performance profiles, we do not consider instance where an ILP was not able to solve in time. If an ILP has not completed any instance it is marked with a hyphen. All values are rounded to the second decimal place.

6.2 Results

6.2.1 Fixed Number of Conflicts

This section compares our algorithm when using each of the dynamic ILPs introduced in Section 5.2 on a fixed number of edge insertions, disregarding conflicts. The algorithms were run using a BFS-depth of $0, 1, 5, 10$ and 15. The detailed results can be found in Table 6.2. We grouped the depth them into three sections, small $(0, 1)$, medium $(1, 5, 10)$ and large $(10, 15)$. Their performance profiles can be found in Figure 6.1a, Figure 6.1b and Figure 6.1c respectively. We do this to have a smoother transition between groups and to counteract the fact, that we use non-continuous depth values. With gaps in between, evaluating only single depths could create the illusion of edges in the data, where the performance of an algorithm changes more drastically at a certain point. Our goal with this experiment is not to find such an edge, but rather to display the trends of performance for the different ILPs.
As we evaluate the runtime performance of the ILPs on very large graphs derived from finite element methods (see Table 6.1) and insert only a few edges, the size of the resulting global coloring is of no interest for us. We will leave this metric unmentioned for this experiment.

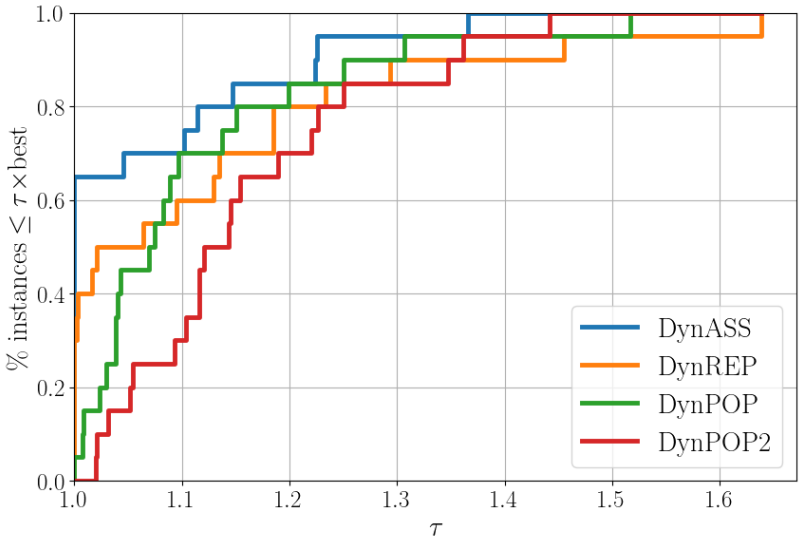
Looking at the performance profiles in Figure 6.1 for the runtimes of the different groups, the most noticeable aspect is DynREP. In the small BFS-depth group of 0 and 1, it is on par with the other algorithms. It is best performing for around 30% of instances and only needs a factor of $\tau = 1.02$ to include 50% of instances. This rate stagnates as τ grows. In the end, it needs the highest factor of all ILPs for the instances considered. For the other groups, this pattern worsens, with its performance line dropping significantly below its competitors. Only on a third of instances DynREP's performance is within a factor of $\tau = 1.8$. This is where our graphic ends for scale reasons. The largest factor needed over all instances in this group is $\tau = 32.6$. DynREP fails to solve five instances for a BFS-depth of 5 and all instances for depths 10 and 15. This is the reason for its line overlapping with the x-axis in the large group's performance profile. The reason for the poor performance on medium and larger problem instances derives from its inherit symmetry mentioned in Section 4.2.

DynASS is performing the best ILP in the small group. While all ILPs perform quite similarly, it consistently keeps its (sometimes shared) top rank regarding percentage of

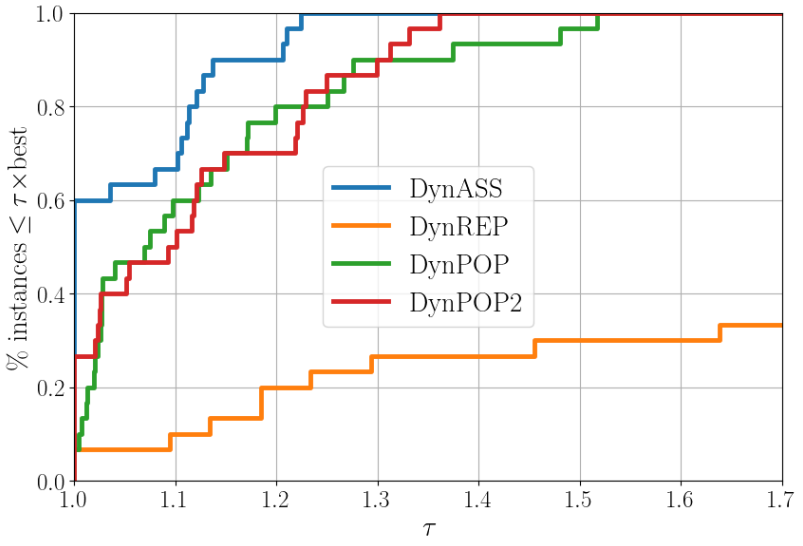
solved instances for a given τ. It is most efficient regarding runtime for 65% of instances and solves 95% within a factor of $\tau = 1.23$. The steep performance line of DynASS is present in the medium group as well. Here its superiority is even stronger with a visible gap to the next best performing ILPs. It finally falls behind both DynPOP version within the large group where its performance drops significantly. With only a factor of $\tau = 1.04$, the first instances can be solved via DynASS. Even though its curve rises steeply after this point it flattens out again quickly, before completing the last third of instances with $1.44 \leq \tau \leq 1.69$. The languishing after 66% is due to the instances with 100 edges and BFS-depth 15. Inserting 1000 edges with this depth was not completed by any ILP.

DynPOP and DynPOP2 perform very similar in all groups. They start on the lower end for small BFS-depths, having a steep initial line. This indicating good performance for the depth range, although both ILPs have a poorer performance for the first 50% of instances compared to the other two ILPs. Especially DynPOP2 struggles to outperform even the symmetry-burdened DynREP, only doing so at $\tau = 1.37$. DynPOP on the other hand competes closely with DynASS for the last 20% of instances. In the middle group both DynPOP and DynPOP2 succumb to DynASS. They are start rivaling each other with DynPOP2 coming out on top in the last 10% of instances. Here the factor gap between the two grows to $\tau_\Delta = 0.15$, similar to the difference of DynPOP2 and DynASS throughout all graphs. Instances created by large BFS-depths are the most efficient for DynPOP and DynPOP2 compared to the other ILPs. While they cannot complete 1000 edge insertions with BFS-depth 15 either, they greatly outperform DynASS for 100 such conflicts, as well as all instances with a depth of 10. DynPOP2 has an advantage over DynPOP, being best performant in 80% of instances. This is almost neglectable as the difference in factor is at most 0.026 for all except one instance.

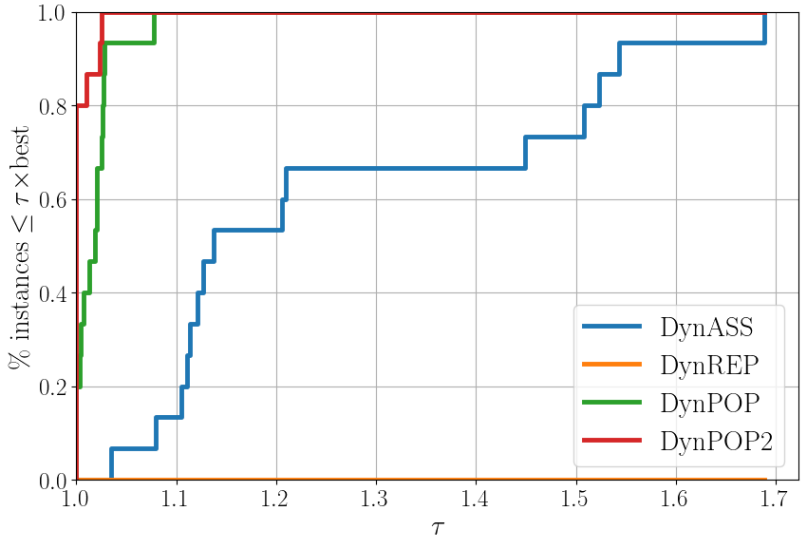
To summarize, DynASS generally performs best in the small and medium BFS-depth groups, showing a strong capability to solve instances efficiently. While not being the best at a depth of 0, it starts to surpass its competition at a depth of 1, which is enhanced going to a depth of 5 as seen in Figure 6.2. With larger depths though, DynPOP and DynPOP2 start to gain the upper hand by a great margin. DynPOP is of no competition except for some instances at very low BFS-depth values.



(a) Small BFS-depths $(0, 1)$

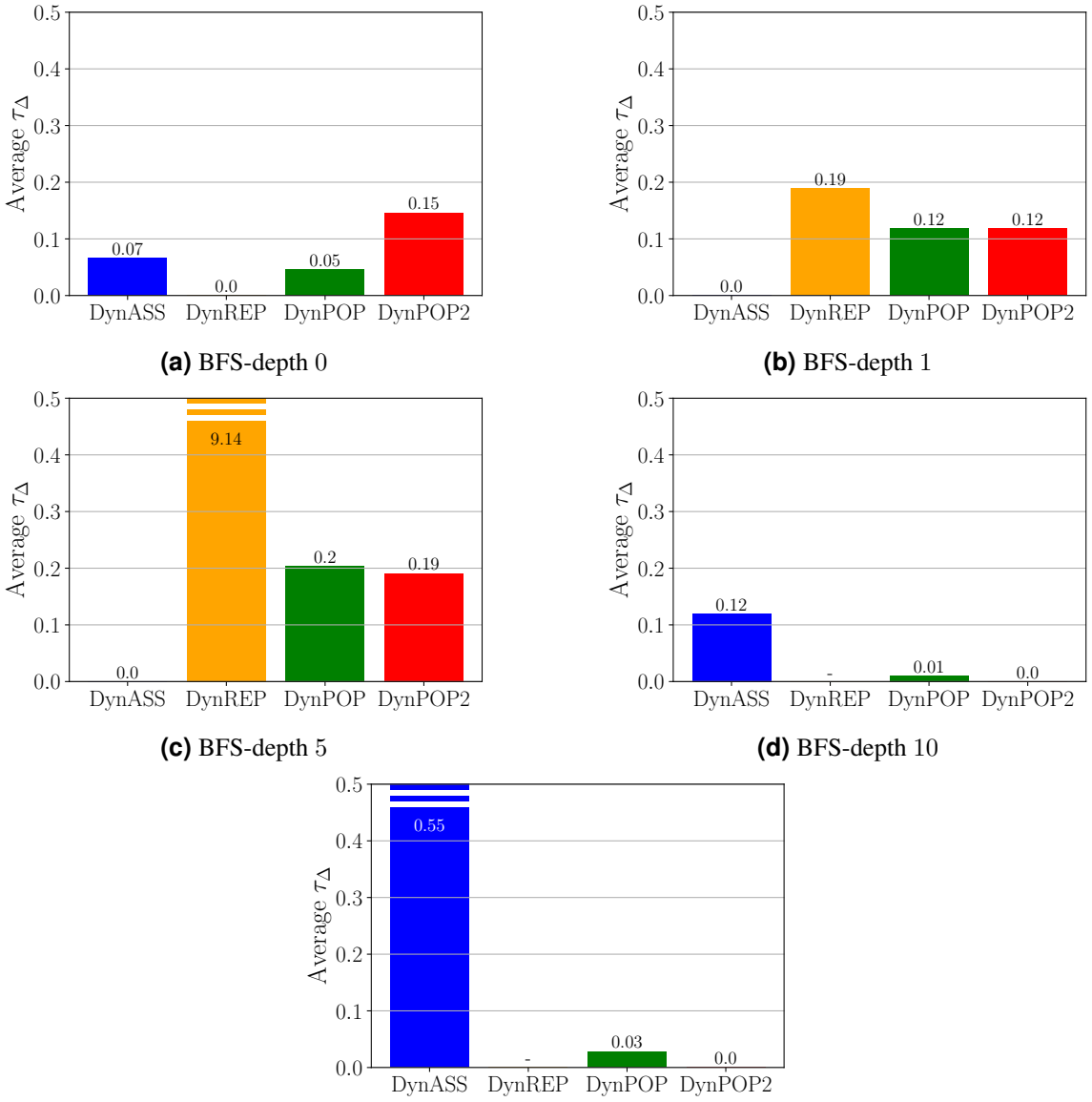


(b) Medium BFS-depths $(1, 5, 10)$



(c) Large BFS-depths $(10, 15)$

Figure 6.1: Performance profiles displaying runtimes of the dynamic ILPs for graphs derived from finite element methods.



(a) BFS-depth 0

(b) BFS-depth 1

(c) BFS-depth 5

(d) BFS-depth 10

(e) BFS-depth 15

Figure 6.2: Average τ_Δ for runtime performance compared to best performing ILP over all percentages of instances.

6.2.2 Evaluation of DIMACS Benchmark

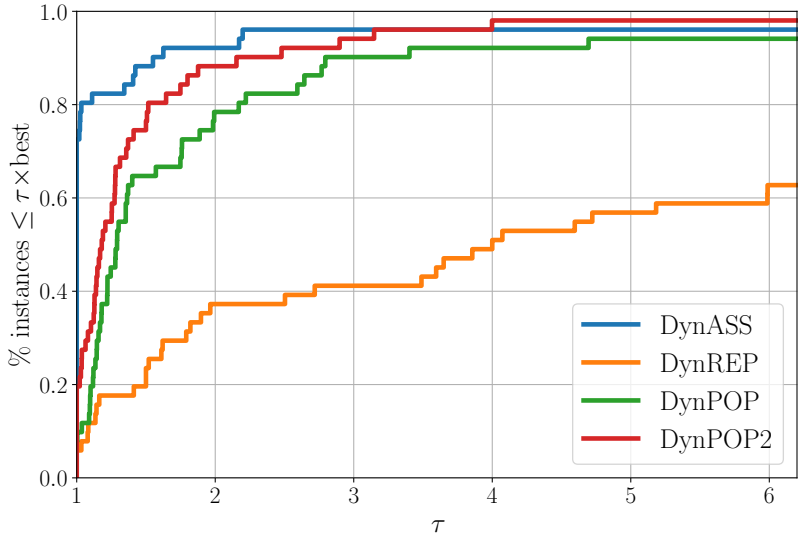
For the set of DIMACS graphs (see Table 6.3) we evaluate the performance metrics by considering only edge insertions that cause a conflict to be solved by our algorithm. With these graphs, smaller VFS-depths of $0, 1, 3$ and 5 were chosen, as there is the possibility of numerous conflicts needing to be resolved. For larger depths this would increase the runtime too much to be viable on the execution of the experiment. Due to the more realistic setting, we measure runtime, solution size, number of edge conflicts, average subgraph density encountered as well as the ratio of fixated versus mutable vertices in the subgraphs created by our algorithm. We want to compare the ILPs in its runtime performance overall as well as its behavior regarding the mentioned metrics with the instances. The results as a whole are displayed in Table 6.4.
Looking at the performance profiles for all instances and BFS-depth in Figure 6.3, one can see the superiority of DynASS when it comes to pure runtime. It is the most performant ILP in 72.55% of instances and only needs a factor of $\tau = 1.04$ to include 80%. Around 90% of instances are solved within $\tau = 1.56$. The closest competitors, DynPOP2 and DynPOP, have a steep line as well, completing 80% of instances within a factor of 1.52 and 2.17 respectively. DynPOP declines around the 65% mark, where it struggles to complete the last 35% compared to the best algorithms. DynPOP2 on the other hand keeps its pace and almost reaches DynASS before $\tau = 2$. DynREP is the least efficient ILP. Not only does it solve fewer instances as shown in Figure 6.4 but performs poorly on those it manages to complete in the time limit. It needs 4 times as long to solve 50% of instances compared to the other ILPs. Making things worse, it also falls behind when it comes to the solution size (see Figure 6.3b). The colorings produced by DynREP are around 1.1 to 1.4 times larger than those of DynASS. It falls behind the DynPOP ILPs in this metric as well. It manages to color the instances best or equivalently best in 53% of cases. After this, the growth disparity is at a similar rate to DynPOP and DynPOP2. The latter managing to perform the best out of all ILPs when it comes to the solution size, solving 80% at $\tau = 1$.

To have a better understanding of why DynASS outperforms DynPOP and DynPOP2, even though its solution size is bigger, one can look at Figure 6.3c where the number of edge conflicts encountered is compared. Here we can see that DynASS produced fewer conflicts than the other ILPs for the same ordering of edge insertions. Although DynASS increases the problem size during the solving of subgraphs, meaning the calculated upper bound is higher in more cases, which in turn results in larger ILPs and an extended calculation time, it has to solve fewer of these problems. Because of this the overall runtime performance is better than the other ILPs. DynPOP and DynPOP2 are, performing similarly on this metric again, with DynPOP causing more conflicts, which is in turn represented in the runtime performance profile in Figure 6.3a. DynREP is once more the worst performing ILP by a large margin.

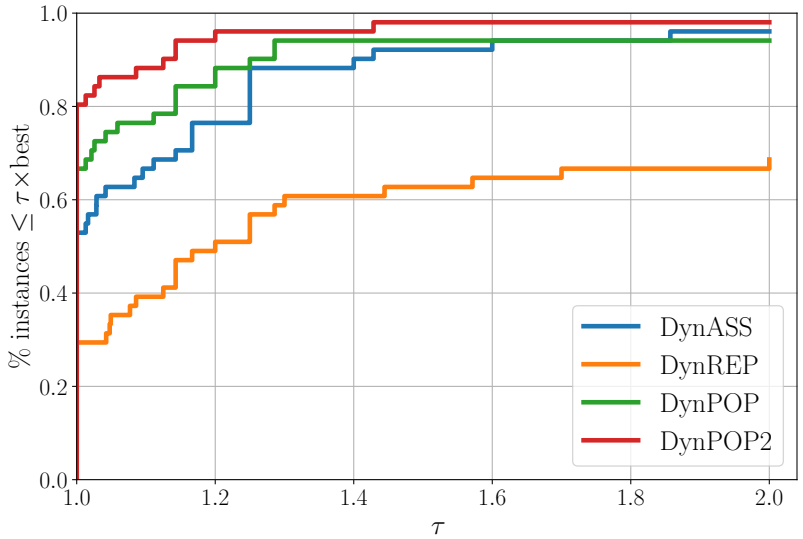
It is noteworthy that the apparent superiority of DynASS is supported by the fact that a lot of instances with large BFS-depth weren't solved by any ILP (Figure 6.4). This means a higher proportion of smaller graphs build the basis for the performance profiles, where -

as we already showed in Section 6.2.1 - DynASS is better performing. For this reason, we want to take a more detailed look at the different depth levels displayed in Figure 6.5. Here we can confirm what we have just stated. For small depth of 0 and 1, DynASS is the most performant when it comes to runtime. But we can also see it struggling with the solution size for depths greater than 0. This only worsens for d_{BFS} of 3 where both DynPOP ILPs solve 100% of instance best. It is also evident, that DynASS struggles to solve the last $17-22\%$ of instances for all BFS-depths except 0. DynPOP2 performs better in these final percents, solving some instances twice as fast as the others.
Table 6.4 shows that for a growing value of d_{BFS}, the solution size gets smaller in almost every instance. Those were this is not the case, only see the addition one color, making it plausible, that this happened by chance due to the edge insertion order. In many cases the best solution for all depths, was already found at a depth of 1, especially in smaller to medium size graphs. This makes sense, as with a large depth for a BFS-search in a small graph, not too much more can be discovered. In many cases, the whole graph is recolored multiple times. We would consider a BFS-depth of 1-3 a sweet spot for both runtime and solution quality. The other graph attributes we tracked, namely the ratio of fixated and mutable vertices and the average subgraph density, seem to have no impact on performance.

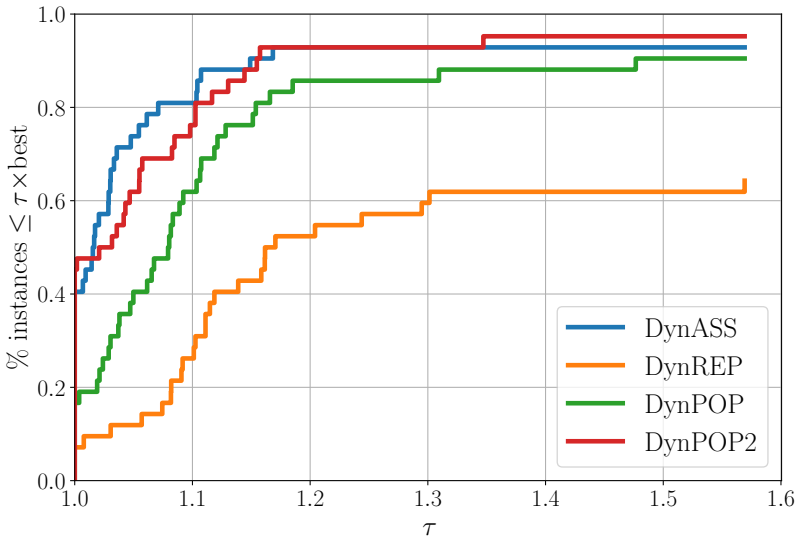
Finally, we want to compare our results with those of Jabrayilov and Mutzel [19]. We include the table of their findings as reference in Table 6.5. Here we can see that POP and POP2, the static versions of DynPOP and DynPOP2, outperform the other algorithms in every instance except one, where ASS is in the lead. This doesn't oppose our findings as much as one may think. As we have already discussed, the real strength of DynASS in our experiments, was its ability to prevent edge conflicts. In general, the results of Jabrayilov and Mutzel resemble what we observe with the dynamic versions, with some graph families like *le450* or *qg*. On the other hand some instances of the *FullIns* family were not solved by the dynamic ILPs for larger BFS-depths.



(a) Runtime



(b) Size



(c) # Conflicts

Figure 6.3: Performance profiles illustrating different metrics over all BFS-depth used.

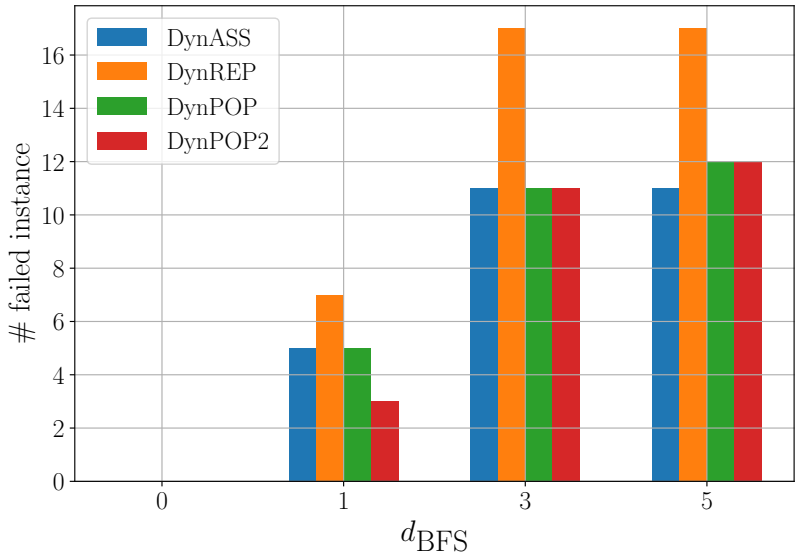


Figure 6.4: Number of failed instances per ILP per BFS-depth d_{BFS}.

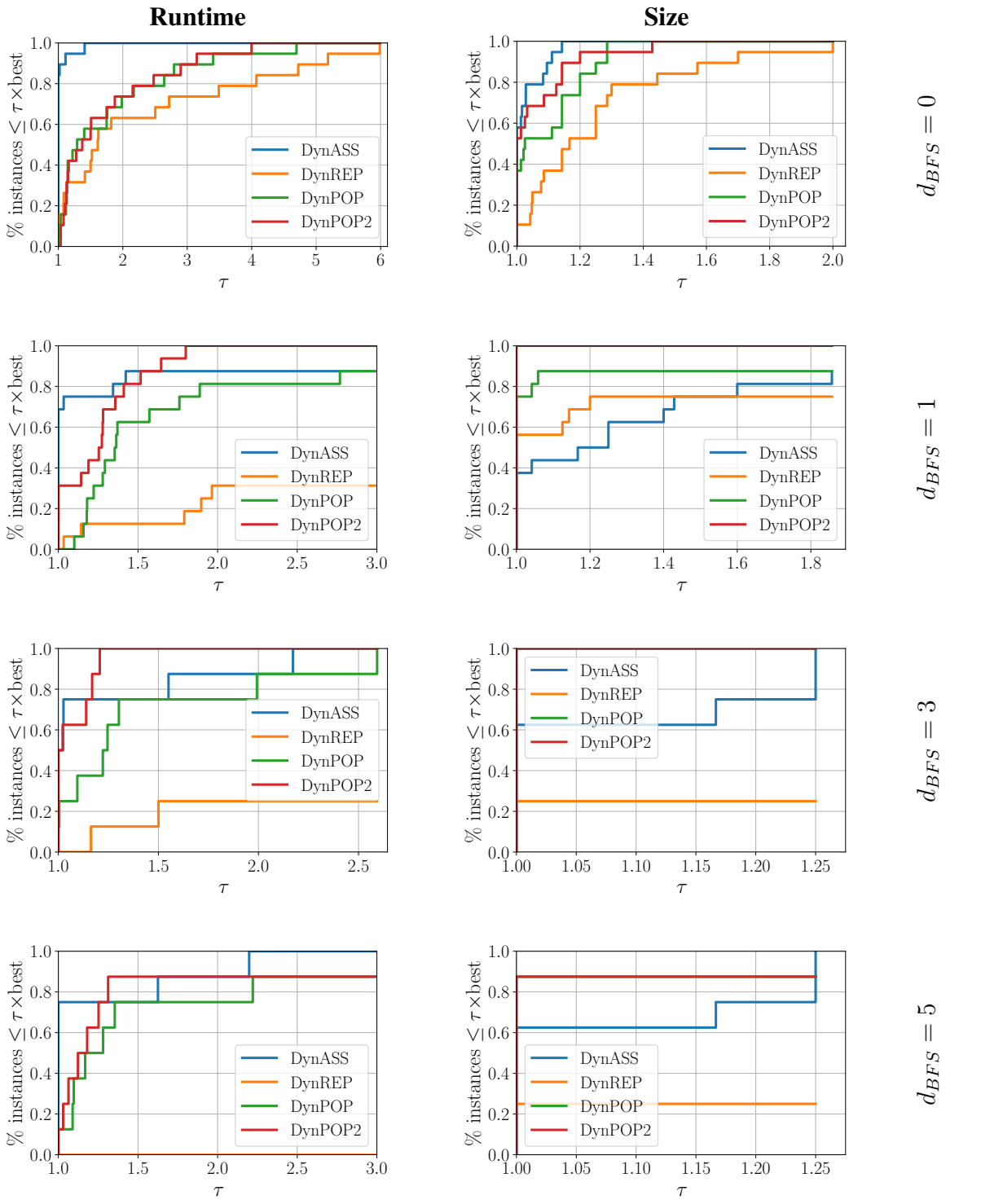


Figure 6.5: Performance profiles illustrating the runtime in the left column and the size of the solution (coloring) in the right column. The BFS-depth d_{BFS} is different in each row.

Table 6.1: Instances originating from finite element methods used for the experiments with a fixed number of edge conflict in Section 6.2.1.

Instance	n	m
333SP	3712815	11108633
AS365	3799275	11368076
M6	3501776	10501936
NACA0015	1039183	3114818
NLR	4163763	12487976

Table 6.2: Extensive runtime data display for finite element graphs for all dynamic ILPs at various BFS-depths. The lowest value for a given depth and instance is printed in bold. Hyphens mark data points, where the algorithm was not able to complete within the maximum compute time with a given ILP.

Inst.	d_{BFS}	**DynASS**		**DynREP**		**DynPOP**		**DynPOP2**	
		t_{100}	t_{1000}	t_{100}	t_{1000}	t_{100}	t_{1000}	t_{100}	t_{1000}
333SP	0	**4.84**	**11.77**	4.94	11.80	4.88	12.12	4.99	12.02
	1	**4.84**	**14.65**	5.49	17.36	5.20	15.66	5.29	15.44
	5	81.34	784.23	-	-	75.88	708.17	**75.34**	**689.72**
	10	81.34	784.23	-	-	75.88	708.17	**75.34**	**689.72**
	15	413.35	-	-	-	292.44	-	**271.44**	-
AS365	0	**7.77**	**16.89**	7.90	17.97	7.83	17.61	8.90	18.85
	1	**8.12**	**20.91**	8.89	27.05	8.31	22.94	9.10	23.34
	5	86.74	837.11	-	-	77.94	712.24	**76.96**	**694.08**
	10	86.74	837.11	-	-	77.94	712.24	**76.96**	**694.08**
	15	386.42	-	-	-	234.60	-	**228.84**	-
M6	0	8.22	15.92	**7.86**	**12.99**	8.16	16.97	8.99	17.50
	1	**7.58**	21.70	9.35	**19.69**	8.25	22.66	9.25	20.70
	5	88.14	841.74	-	-	80.13	715.34	**78.63**	**695.87**
	10	88.14	841.74	-	-	80.13	715.34	**78.63**	**695.87**
	15	368.59	-	-	-	245.30	-	**244.44**	-
NACA0015	0	**2.40**	7.82	2.71	7.02	2.73	7.60	2.77	7.75
	1	**2.98**	11.01	3.53	16.02	3.10	13.20	3.04	13.76
	5	78.50	733.02	-	-	71.97	662.74	**70.51**	659.76
	10	78.50	733.02	-	-	71.97	662.74	**70.51**	659.76
	15	347.81	-	-	-	230.05	-	**225.39**	-
NLR	0	8.81	16.52	6.47	14.40	**6.45**	14.96	9.30	17.12
	1	8.70	16.85	**7.11**	27.60	8.89	25.56	9.68	20.66
	5	85.24	812.44	-	-	**82.34**	734.98	84.23	753.24
	10	85.24	812.44	-	-	**82.34**	734.98	84.23	753.24
	15	453.47	-	-	-	**312.92**	-	316.02	-

Table 6.3: Instances used for the experiments on the DIMACS benchmark set in Section 6.2.2.

Instance	n	m
2-FullIns_4	212	1621
2-FullIns_5	852	12201
3-FullIns_3	80	346
3-FullIns_4	405	3524
4-FullIns_3	114	541
4-FullIns_4	690	6650
5-FullIns_3	154	792
ash608GPIA	1216	7844
ash958GPIA	1916	12506
DSJR500.5	500	58862
le450_5a	450	5714
le450_15a	450	8168
le450_15c	450	16680
le450_25c	450	17343
mug100_1	100	166
mug100_25	100	166
qg.order40	1600	62400
school1_nsh	352	14612
wap05a	905	43081

Table 6.4: Extensive data display for DIMACS benchmarks for all dynamic ILPs at various BFS-depths. The columns are as follows: t is the runtime of the algorithm, $|C|$ is the size of the solved coloring, $\#_c$ is the number of edge conflicts encountered, ρ is the average density of subgraphs in percent α is the ratio of number of fixated and mutable vertices in the subgraphs. If there is significance in the lowest value for a given depth and instance, it is printed in bold. Hyphens mark data points, where the algorithm was not able to complete within the maximum compute time with a given ILP.

Inst.	d_{BFS}	DynASS					DynREP					DynPOP					DynPOP2				
		t	$\|C\|$	$\#_c$	ρ	α	t	$\|C\|$	$\#_c$	ρ	α	t	$\|C\|$	$\#_c$	ρ	α	t	$\|C\|$	$\#_c$	ρ	α
2-FullIns_4	0	**0.68**	**7**	291	6.05	21.59	1.02	11	348	5.49	21.52	0.76	9	**289**	5.57	21.60	0.75	10	**289**	5.75	21.58
	1	**2.25**	7	**265**	2.65	10.40	10.34	**6**	307	2.64	10.03	3.96	**6**	347	2.83	9.79	4.05	**6**	357	2.77	9.56
	3	355.08	7	289	0.14	14.45	-	-	-	-	-	178.97	**6**	274	0.14	14.54	**163.47**	**6**	**261**	0.15	14.51
	5	**631.09**	7	273	0.00	14.99	-	-	-	-	-	692.32	**6**	266	0.00	15.02	671.30	**6**	**265**	0.00	15.04
2-FullIns_5	0	**3.13**	**10**	1193	13.20	11.87	10.92	13	1358	10.54	12.29	6.21	12	**1169**	11.50	11.94	6.74	12	1206	12.06	11.70
	1	55.07	13	1271	4.95	4.30	492.24	**7**	1300	5.09	4.33	50.01	**7**	1288	5.38	4.34	**38.70**	**7**	**1230**	5.20	4.39
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3-FullIns_3	0	**0.37**	**7**	109	2.83	34.85	0.42	8	120	2.37	35.80	0.42	8	112	3.18	33.98	0.40	8	**108**	2.90	34.84
	1	**0.56**	**6**	**108**	1.57	19.29	1.10	**6**	123	1.55	18.53	0.88	**6**	128	1.59	18.39	0.79	**6**	125	1.57	18.77
	3	6.40	**6**	110	0.21	20.79	-	-	-	-	-	16.18	**6**	107	0.22	20.65	**6.24**	**6**	**105**	0.22	20.63
	5	**16.56**	**6**	107	0.00	21.77	-	-	-	-	-	36.78	**6**	107	0.00	21.76	20.74	**6**	**104**	0.00	21.66
3-FullIns_4	0	**2.08**	10	554	6.44	19.73	3.37	13	633	5.90	20.08	2.38	10	**545**	6.08	20.06	2.39	**9**	546	6.08	20.07
	1	**5.13**	10	**514**	3.43	8.32	30.71	**7**	566	3.64	7.94	14.20	**7**	759	3.44	7.51	7.78	**7**	581	3.35	8.17
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4-FullIns_3	0	**0.50**	**7**	**146**	2.76	34.72	0.76	14	190	3.53	32.02	0.61	9	159	3.37	33.07	0.58	8	154	3.02	33.86
	1	**0.97**	**7**	**153**	1.82	16.80	1.84	8	**153**	1.85	16.91	1.33	**7**	163	1.82	17.13	1.24	**7**	168	1.76	16.94
	3	20.11	**7**	145	0.32	17.87	-	-	-	-	-	25.84	**7**	**140**	0.36	17.75	**12.97**	**7**	145	0.34	17.43
	5	59.00	**7**	151	0.00	19.03	-	-	-	-	-	39.55	**7**	154	0.00	18.94	**36.3**	**7**	**141**	0.00	18.92
4-FullIns_4	0	**3.40**	**10**	**921**	6.74	18.75	6.19	17	1078	6.46	18.79	4.39	12	949	6.47	18.79	4.34	**10**	964	6.50	18.64
	1	**12.63**	10	**808**	3.98	6.87	113.53	**8**	1005	4.20	6.48	17.10	**8**	906	4.03	6.87	20.78	**8**	933	3.80	6.69
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5-FullIns_3	0	0.76	**8**	212	3.74	31.70	0.87	10	233	2.71	33.56	**0.54**	10	214	3.33	32.50	0.81	9	**209**	3.22	33.00
	1	**1.72**	**8**	239	2.18	14.23	3.08	9	225	2.19	14.84	2.03	**8**	**208**	2.10	15.54	2.19	**8**	238	2.14	14.71
	3	**35.90**	**8**	**193**	0.38	15.84	-	-	-	-	-	36.05	**8**	206	0.38	15.46	36.68	**8**	197	0.38	15.63
	5	246.28	**8**	208	0.01	16.51	-	-	-	-	-	**112.04**	**8**	212	0.01	16.62	115.42	**8**	**196**	0.01	16.83
ash608GPIA	0	**6.45**	**7**	1856	5.24	18.29	6.99	8	**1831**	5.26	18.29	7.39	8	1876	5.26	18.31	7.28	**7**	1915	5.24	18.31
	1	**16.00**	8	**1701**	3.12	9.15	57.52	6	1921	3.19	9.03	20.46	**5**	1997	3.20	8.93	20.06	**5**	2000	3.20	8.98
	3	**234.85**	**4**	**1914**	0.57	8.69	-	-	-	-	-	305.88	**4**	1994	0.57	8.63	283.45	**4**	2043	0.57	8.45
	5	**940.69**	**4**	**1916**	0.25	7.19	-	-	-	-	-	1099.06	**4**	2012	0.26	7.13	1110.71	**4**	2042	0.25	7.09
ash958GPIA	0	**10.36**	8	2984	5.42	17.42	11.18	9	**2920**	5.42	17.43	11.61	8	2954	5.38	17.57	11.68	**7**	2989	5.41	17.48
	1	**26.85**	7	**2727**	3.23	8.32	103.53	**5**	3129	3.31	8.17	32.80	**5**	3088	3.31	8.23	31.93	**5**	3129	3.34	8.16
	3	**432.56**	**4**	**3139**	0.58	7.49	-	-	-	-	-	538.80	**4**	3513	0.59	7.48	505.58	**4**	3567	0.58	7.43
	5	**1736.7**	**4**	**3108**	0.26	6.04	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DSJR500.5	0	**84.23**	145	1781	71.21	3.99	118.95	147	**1614**	69.82	4.01	235.39	144	1742	70.05	4.03	336.77	**141**	2884	77.34	3.37
	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
le450_5a	0	4.37	13	**1092**	9.19	15.71	**4.29**	14	1191	9.71	14.82	6.01	**12**	1115	9.18	15.67	5.88	**12**	1139	9.37	15.48
	1	**42.76**	**10**	1187	7.05	6.50	597.45	**10**	**1154**	6.61	6.69	58.31	**10**	1225	6.81	6.44	58.05	**10**	1202	7.08	6.48
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
le450_15a	0	**6.96**	23	1277	14.61	12.05	18.92	22	1568	16.87	10.58	12.17	**21**	1257	15.41	11.89	10.49	**21**	**1211**	14.63	12.29
	1	**138.2**	**17**	**1218**	6.26	4.44	-	-	-	-	-	260.96	18	1420	6.14	4.13	177.03	**17**	1285	6.53	4.27
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Continued on next page

Table 6.4 – continued from previous page

Inst.	d_{BFS}	DynASS					DynREP					DynPOP					DynPOP2				
		t	$\|C\|$	$\#_c$	ρ	α	t	$\|C\|$	$\#_c$	ρ	α	t	$\|C\|$	$\#_c$	ρ	α	t	$\|C\|$	$\#_c$	ρ	α
le450_15c	0	**11.42**	**31**	1293	29.08	7.19	68.38	**31**	2330	32.10	5.51	24.78	**31**	1265	28.71	7.33	19.97	**31**	**1171**	28.13	7.62
	1	1200.07	25	1998	4.25	5.55	-	-	-	-	-	1034.09	25	1964	4.31	5.34	**893.32**	**24**	**1568**	4.35	5.84
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
le450_25c	0	**10.99**	36	1238	30.24	7.31	56.96	38	2168	33.94	5.65	29.05	**35**	1343	30.10	7.03	27.24	38	**1217**	29.92	7.38
	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**1494.80**	**29**	**1485**	4.05	6.52
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
mug100_1	0	0.30	**4**	132	1.03	56.44	**0.27**	5	131	1.03	55.52	0.28	**4**	**130**	1.03	55.85	0.28	**4**	**130**	1.03	55.85
	1	0.31	5	**117**	0.66	36.72	0.31	**4**	130	0.70	37.19	0.33	**4**	135	0.72	36.79	**0.30**	**4**	129	0.71	37.11
	3	0.44	5	**121**	0.48	20.08	0.50	**4**	130	0.47	20.07	0.43	**4**	134	0.48	19.82	**0.43**	**4**	131	0.48	20.01
	5	**0.57**	5	**122**	0.30	14.86	2.08	**4**	132	0.31	14.46	0.73	**4**	135	0.31	14.31	0.64	**4**	129	0.31	14.56
mug100_25	0	**0.28**	**4**	135	1.03	56.42	0.28	5	135	1.03	55.78	0.28	**4**	**131**	1.02	56.26	0.29	**4**	**131**	1.02	56.26
	1	**0.28**	**4**	**118**	0.67	37.25	0.32	**4**	132	0.70	37.77	0.33	**4**	132	0.70	38.00	0.32	**4**	128	0.68	38.48
	3	**0.36**	5	**117**	0.45	21.88	0.54	**4**	129	0.46	21.56	0.44	**4**	132	0.46	21.29	0.41	**4**	129	0.46	21.50
	5	**0.48**	5	**120**	0.29	16.29	1.92	**4**	131	0.30	15.81	0.65	**4**	130	0.30	15.91	0.63	**4**	134	0.29	15.69
qg_order40	0	**27.38**	78	**3931**	18.09	17.53	111.52	**77**	12571	15.70	13.10	128.56	78	5970	19.33	15.44	86.16	78	5404	15.82	16.83
	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
school1_nsh	0	**12.67**	**39**	1166	32.74	6.56	59.83	42	1566	36.23	5.82	22.27	40	1149	31.62	6.98	23.77	40	**998**	31.77	7.40
	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**843.45**	**19**	**1249**	3.28	8.83
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
wap05a	0	**33.64**	62	**3276**	30.02	14.73	84.22	64	3635	27.76	14.30	114.45	**61**	3430	28.16	15.43	97.49	63	3566	28.89	15.30
	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 6.5: Results by Jabrayilov and Mutzel for the hard instances from DIMACS benchmark set, [19, p. 9]

				REP			POP			POP2			ASS+(c)			ASS+(e)			[30]	[32]	[28]
instance	$\|V\|$	$\|E\|$	class	lb	ub	time	lb	ub	time	lb	ub	time	lb	ub	time	lb	ub	time	time	time	time
1-FullIns_4	93	593	NP-m	5	5	1.85	5	5	0.01	5	5	0.01	5	5	0.01	5	5	0.01	0.1		tl
1-FullIns_5	282	3247	NP-?	5	6	tl	6	6	6.01	6	6	1.54	6	6	1.82	6	6	2.12	tl		tl
2-FullIns_4	212	1621	NP-m	6	6	2.14	6	6	0.02	6	6	0.01	6	6	0.01	6	6	0.01	tl	4	tl
2-FullIns_5	852	12201	NP-?	5	7	tl	7	7	**5.02**	7	7	72.45	7	7	326.61	7	7	14.74	tl		tl
3-FullIns_3	80	346	NP-m	6	6	0.01	6	6	0.00	6	6	0.00	6	6	0.00	6	6	0.00	0.1		2.9
3-FullIns_4	405	3524	NP-?	7	7	3.92	7	7	0.03	7	7	0.02	7	7	0.02	7	7	0.03	tl		tl
3-FullIns_5	2030	33751	-	7	8	tl	8	8	**12.43**	8	8	32.90	8	8	3489.97	8	8	27.23	tl		tl
4-FullIns_3	114	541	NP-m	7	7	0.01	7	7	0.00	7	7	0.00	7	7	0.00	7	7	0.00	3		3.4
4-FullIns_4	690	6650	NP-?	8	8	4.61	8	8	0.05	8	8	0.02	8	8	0.03	8	8	0.02	tl		tl
4-FullIns_5	4146	77305	-	7	9	tl	9	9	**9.67**	9	9	16.03	8	9	tl	9	9	78.17	tl		tl
4-Insertions_3	79	156	NP-m	3	4	tl	4	4	**9.62**	4	4	15.75	4	4	141.48	4	4	54.04	4204		tl
5-FullIns_3	154	792	NP-m	8	8	0.01	8	8	0.00	8	8	0.00	8	8	0.00	8	8	0.00	20		4.6
5-FullIns_4	1085	11395	NP-?	9	9	12.36	9	9	0.05	9	9	0.05	9	9	0.04	9	9	0.04	tl		tl
ash608GPIA	1216	7844	NP-m	-∞	1215	tl	4	4	**34.84**	4	4	51.63	4	4	575.23	4	4	821.74	692		2814.8
ash958GPIA	1916	12506	NP-m	-∞	+∞	tl	4	4	**90.11**	4	4	105.77	4	6	tl	4	8	tl	tl	4236	tl
DSJC125.5	125	3891	NP-h	14	21	tl	11	20	tl	13	22	tl	13	21	tl	13	21	tl	tl		18050.8
DSJC125.9	125	6961	NP-h	44	44	**1.72**	36	50	tl	42	44	tl	42	45	tl	42	45	tl	tl		3896.9
DSJR500.1c	500	121275	NP-h	85	85	**0.33**	77	+∞	tl	83	86	tl	78	+∞	tl	78	+∞	tl	tl		288.5
DSJR500.5	500	58862	NP-h	122	497	tl	115	+∞	tl	122	122	**572.01**	122	122	1748.11	115	+∞	tl	tl		342.2
le450_15a	450	8168	NP-m	15	449	tl	15	16	tl	15	15	**598.55**	15	15	2439.49	15	15	801.94	tl		0.4
le450_15b	450	8169	NP-?	15	446	tl	15	15	2939.49	15	15	**700.50**	15	15	1393.29	15	15	1103.29	tl		0.2
le450_15c	450	16680	NP-?	-∞	450	tl	15	+∞	tl	15	+∞	tl	15	+∞	tl	15	25	tl	tl		3.1
le450_15d	450	16750	NP-?	-∞	450	tl	15	26	tl	15	26	tl	15	+∞	tl	15	+∞	tl	tl		3.8
le450_25c	450	17343	NP-?	25	450	tl	25	30	tl	25	31	tl	25	+∞	tl	25	+∞	tl	tl		1356.6
le450_25d	450	17425	NP-?	25	450	tl	25	30	tl	25	31	tl	25	+∞	tl	25	+∞	tl	tl		66.6
le450_5a	450	5714	NP-?	-∞	450	tl	5	9	tl	5	5	**21.17**	5	5	83.65	5	5	52.03	tl		0.3
le450_5b	450	5734	NP-?	-∞	450	tl	5	7	tl	5	5	**140.16**	5	5	503.29	5	5	168.67	tl		0.2
mug100_1	100	166	NP-m	4	4	1.14	4	4	0.24	4	4	0.09	4	4	0.39	4	4	0.13	60		14.4
mug100_25	100	166	NP-m	4	4	1.10	4	4	0.45	4	4	0.31	4	4	0.31	4	4	0.31	60		12
qg.order40	1600	62400	NP-m	-∞	+∞	tl	40	45	tl	40	40	**534.83**	40	46	tl	40	46	tl	tl		2.9
qg.order60	3600	212400	NP-?	-∞	+∞	tl	60	68	tl	60	62	tl	-∞	68	tl	-∞	68	tl	tl		3.8
queen10_10	100	1470	NP-h	10	12	tl	10	12	tl	10	12	tl	10	12	tl	10	12	tl	tl		686.9
queen11_11	121	1980	NP-h	11	13	tl	11	13	tl	11	13	tl	11	13	tl	11	13	tl	tl		1865.7
school1_nsh	352	14612	NP-m	14	14	981.45	14	14	22.39	14	14	**12.76**	14	14	31.23	14	14	28.22	0		17
wap05a	905	43081	NP-m	-∞	+∞	tl	40	+∞	tl	50	50	1308.37	50	50	**125.45**	41	+∞	tl	tl		293.2
wap06a	947	43571	NP-?	-∞	+∞	tl	40	+∞	tl	40	+∞	tl	40	+∞	tl	40	+∞	tl	tl		175
solved:						13			19			25			22			21	9	+2	25

CHAPTER 7

Discussion

7.1 Conclusion

This thesis investigates dynamic approaches, solving the Graph Coloring Problem (GCP) using Integer Linear Programming (ILP) on subgraphs. We propose dynamic versions of four ILPs considered by Jabrayilov and Mutzel [19] that handle the insertion of edges into a graph over time. We explore ILP specific pitfalls and optimization possibilities as well as preprocessing techniques. Furthermore, we introduce a color mapping that counteracts the burdens emerging from the problem of fixated neighborhoods. We examine the breadth-first search depth and the effect it has on runtime as well as solution quality.
The dynamic ILPs are evaluated using two experiments, to measure pure runtime performance in a more controlled environment, as well as on a large set of instances to measure their performance regarding size and behavior. Our key findings include that DynASS performs best on small to medium-sized graphs, while DynPOP and DynPOP two are advantageous on larger instances. DynREP demonstrates how symmetries in ILPs can vastly reduce effectiveness. This correlates to the data provided by Jabrayilov and Mutzel [19] under the premise that in the dynamic GCP the number of edge conflicts is of essential role, discovering that different ILPs are less likely to encounter these conflicts in their own colorings.

7.2 Future Work

Using ILPs to solve the GCP is not a new feat. Using it in a dynamic environment is not yet researched though. This thesis touches many areas where one can investigate further techniques. With the fixated neighborhood being of essence for the ILP, the impact of different mappings would be interesting. Lowering the assigned colors even further without increasing the upper bound could lead to an even better quality for small BFS-depths.

More sophisticated adaptions of preprocessing and ILPs or even completely new ones specialized for this purpose could increase performance. This is also true for possible hybrid approaches as a combination of ILPs and heuristics or metaheuristics. An adaptive model that chooses an ILP with a search depth when its most suitable could be an addition in a range of tools already in use.
We are deeply interested in the relation between the dynamic ILPs performance and certain graph attributes like density, diameter, vertex degrees as well as graph structures like cliques. Further research could try to develop adaptive ILPs that reacts to its environment and can dynamically adjust their strategy while the graph is evolving. More benchmarking with real world instances is of importance as well to illustrate the practical implications in e.g. social networks, roadmaps or communication networks.
Parallel and distributed computing could be investigated. Due to the nature of the problem the decomposition of the ILP into even smaller sub-problems that can be solved simultaneously might increase compute performance and solution quality.

Zusammenfassung

Das *Graph Coloring Problem (GCP)* besteht darin, alle Knoten eines Graphen mit so wenigen unterschiedlichen Farben wie möglich zu färben, ohne dass dabei zwei Knoten, die durch eine Kante verbunden sind, die gleiche Farbe erhalten. Das dynamische GCP betrifft Graphen, deren Struktur sich im Laufe der Zeit durch das Einfügen oder Löschen von Kanten und Knoten ändert. Da das GCP nicht in polynomialer Zeit lösbar ist, werden oft Heuristiken und Approximationen verwendet. Wir entwickeln eine Methode, um die Färbung eines Graphen nach dem Einfügen von Kanten effektiv zu aktualisieren, indem das GCP optimal auf einem durch eine Breitensuche erstellten Teilgraphen gelöst wird. Wir konvertieren bestehende *Integer Linear Programs (ILPs)*, die für das statische GCP von Jabrayilov and Mutzel [19] verwendet werden, um sie mit der dynamischen Umgebung kompatibel zu machen. Darüber hinaus behandeln wir Techniken zur Vorverarbeitung. Wir führen Experimente durch, um die Leistung unserer dynamischen ILPs hinsichtlich Laufzeit und Qualität der Lösung, sowie anderer Metriken, wie der Anzahl der auftretenden Farbkonflikte, zu bewerten.

Bibliography

[1] K. Appel and W. Haken. Special announcement. *Discret. Math.*, 16(2):179–180, 1976. doi: 10.1016/0012-365X(76)90147-3. URL https://doi.org/10.1016/0012-365X(76)90147-3.

[2] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors. *Graph Partitioning and Graph Clustering*, volume 588 of *Contemporary Mathematics*. American Mathematical Society and Center for Discrete Mathematics and Theoretical Computer Science, 2013. 10th DIMACS Implementation Challenge Workshop, February 13-14, 2012, Georgia Institute of Technology, Atlanta, GA.

[3] Nicolas Barnier and Pascal Brisset. Graph coloring for air traffic flow management. *Ann. Oper. Res.*, 130(1-4):163–178, 2004. doi: 10.1023/B:ANOR.0000032574.01332.98. URL https://doi.org/10.1023/B:ANOR.0000032574.01332.98.

[4] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. *CoRR*, abs/1711.04355, 2017. URL http://arxiv.org/abs/1711.04355.

[5] Daniel Brélaz. New methods to color vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979. doi: 10.1145/359094.359101. URL https://doi.org/10.1145/359094.359101.

[6] Manoel B. Campêlo, Ricardo C. Corrêa, and Yuri Frota. Cliques, holes and the vertex coloring polytope. *Inf. Process. Lett.*, 89(4):159–164, 2004. doi: 10.1016/J.IPL.2003.11.005. URL https://doi.org/10.1016/j.ipl.2003.11.005.

[7] Manoel B. Campêlo, Victor A. Campos, and Ricardo C. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discret. Appl. Math.*, 156(7):1097–1111, 2008. doi: 10.1016/J.DAM.2007.05.058. URL https://doi.org/10.1016/j.dam.2007.05.058.

[8] Siew Yin Chan, Teck Chaw Ling, and Eric Aubanel. The impact of heterogeneous multi-core clusters on graph partitioning: an empirical study. *Clust. Comput.*, 15(3):

281–302, 2012. doi: 10.1007/S10586-012-0229-4. URL https://doi.org/10.1007/s10586-012-0229-4.

[9] Fred C. Chow and John L. Hennessy. The priority-based coloring approach to register allocation. *ACM Trans. Program. Lang. Syst.*, 12(4):501–536, 1990. doi: 10.1145/88616.88621. URL https://doi.org/10.1145/88616.88621.

[10] Alane Marie de Lima and Renato Carmo. Exact algorithms for the graph coloring problem. *RITA*, 25(4):57–73, 2018. doi: 10.22456/2175-2745.80721. URL https://doi.org/10.22456/2175-2745.80721.

[11] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, 2002. doi: 10.1007/S101070100263. URL https://doi.org/10.1007/s101070100263.

[12] Michel Gamache, Alain Hertz, and Jérôme Olivier Ouellet. A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Comput. Oper. Res.*, 34(8):2384–2395, 2007. doi: 10.1016/J.COR.2005.09.010. URL https://doi.org/10.1016/j.cor.2005.09.010.

[13] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0-7167-1044-7.

[14] GNU Project. Glpk (gnu linear programming kit), 2024. URL https://www.gnu.org/software/glpk/. Accessed: 2024-05-26.

[15] LLC Gurobi Optimization. Gurobi optimizer, 2024. URL https://www.gurobi.com. Accessed: 2024-05-26.

[16] Monika Henzinger and Pan Peng. Constant-time dynamic (Δ +1)-coloring. *ACM Trans. Algorithms*, 18(2):16:1–16:21, 2022. doi: 10.1145/3501403. URL https://doi.org/10.1145/3501403.

[17] Thore Husfeldt. Graph colouring algorithms. *CoRR*, abs/1505.05825, 2015. URL http://arxiv.org/abs/1505.05825.

[18] IBM. IBM ILOG CPLEX Optimizer, 2024. URL https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer. Accessed: 2024-05-26.

[19] Adalat Jabrayilov and Petra Mutzel. New integer linear programming models for the vertex coloring problem. *CoRR*, abs/1706.10191, 2017. URL http://arxiv.org/abs/1706.10191.

[20] Adalat Jabrayilov and Petra Mutzel. Strengthened partial-ordering based ILP models for the vertex coloring problem. *CoRR*, abs/2206.13678, 2022. doi: 10.48550/ARXIV.2206.13678. URL https://doi.org/10.48550/arXiv.2206.13678.

[21] Yanjun Jiang and Xueyan Song. A weight-based graph coloring approach to airport gate assignment problem. In De-Shuang Huang, Vitoantonio Bevilacqua, Juan Carlos Figueroa García, and Prashan Premaratne, editors, *Intelligent Computing Theories - 9th International Conference, ICIC 2013, Nanning, China, July 28-31, 2013. Proceedings*, volume 7995 of *Lecture Notes in Computer Science*, pages 227–233. Springer, 2013. doi: 10.1007/978-3-642-39479-9_27. URL https://doi.org/10.1007/978-3-642-39479-9_27.

[22] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Oper. Res.*, 39(3):378–406, 1991. doi: 10.1287/OPRE.39.3.378. URL https://doi.org/10.1287/opre.39.3.378.

[23] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi: 10.1007/978-1-4684-2001-2_9. URL https://doi.org/10.1007/978-1-4684-2001-2_9.

[24] Eugene L. Lawler. A note on the complexity of the chromatic number problem. *Inf. Process. Lett.*, 5(3):66–67, 1976. doi: 10.1016/0020-0190(76)90065-X. URL https://doi.org/10.1016/0020-0190(76)90065-X.

[25] Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979. URL https://doi.org/10.6028/jres.084.024.

[26] Vahid Lotfi and Sanjiv Sarin. A graph coloring algorithm for large scale scheduling problems. *Comput. Oper. Res.*, 13(1):27–32, 1986. doi: 10.1016/0305-0548(86)90061-4. URL https://doi.org/10.1016/0305-0548(86)90061-4.

[27] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994. doi: 10.1145/185675.306789. URL https://doi.org/10.1145/185675.306789.

[28] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discret. Optim.*, 8(2):174–190, 2011. doi: 10.1016/J.DISOPT.

2010.07.005. URL https://doi.org/10.1016/j.disopt.2010.07.005.

[29] Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. *INFORMS J. Comput.*, 8(4):344–354, 1996. doi: 10.1287/IJOC.8.4.344. URL https://doi.org/10.1287/ijoc.8.4.344.

[30] Isabel Méndez-Díaz and Paula Zabala. A branch-and-cut algorithm for graph coloring. *Discret. Appl. Math.*, 154(5):826–847, 2006. doi: 10.1016/J.DAM.2005.05.022. URL https://doi.org/10.1016/j.dam.2005.05.022.

[31] Isabel Méndez-Díaz and Paula Zabala. A cutting plane algorithm for graph coloring. *Discret. Appl. Math.*, 156(2):159–179, 2008. doi: 10.1016/J.DAM.2006.07.010. URL https://doi.org/10.1016/j.dam.2006.07.010.

[32] Isabel Méndez-Díaz and Paula Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156(2):159–179, 2008. ISSN 0166-218X. doi: https://doi.org/10.1016/j.dam.2006.07.010. URL https://www.sciencedirect.com/science/article/pii/S0166218X0700100X. Computational Methods for Graph Coloring and it's Generalizations.

[33] Linda Ouerfelli and Hend Bouziri. Greedy algorithms for dynamic graph coloring. In *2011 International Conference on Communications, Computing and Control Applications (CCCA)*, pages 1–5, 2011. doi: 10.1109/CCCA.2011.6031437.

[34] Davy Preuveneers and Yolande Berbers. Acodygra: an agent algorithm for coloring dynamic graphs. 2004. URL https://api.semanticscholar.org/CorpusID:64368328.

[35] Pablo San Segundo. A new dsatur-based algorithm for exact vertex coloring. *Comput. Oper. Res.*, 39(7):1724–1733, 2012. doi: 10.1016/J.COR.2011.10.008. URL https://doi.org/10.1016/j.cor.2011.10.008.

[36] Edward C. Sewell. An improved algorithm for exact graph coloring. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 359–373. DIMACS/AMS, 1993. doi: 10.1090/DIMACS/026/17. URL https://doi.org/10.1090/dimacs/026/17.

[37] Matthieu Silard, Philippe Fabian, Georgios Z. Papadopoulos, and Patrick Savelli. Frequency reuse in iab-based 5g networks using graph coloring methods. In *Global Information Infrastructure and Networking Symposium, GIIS 2022, Argostoli, Greece, September 26-28, 2022*, pages 104–110. IEEE, 2022. doi: 10.1109/

GIIS56506.2022.9937005. URL https://doi.org/10.1109/GIIS56506.2022.9937005.

[38] Shay Solomon and Nicole Wein. Improved dynamic graph coloring. *ACM Trans. Algorithms*, 16(3):41:1–41:24, 2020. doi: 10.1145/3392724. URL https://doi.org/10.1145/3392724.

[39] Michael Trick. Dimacs graph coloring instances. https://mat.tepper.cmu.edu/COLOR02/, 2002. Accessed: 2024-05-26.

[40] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *Eur. J. Oper. Res.*, 242(3):693–709, 2015. doi: 10.1016/J.EJOR.2014.09.064. URL https://doi.org/10.1016/j.ejor.2014.09.064.