Chapter 2
The University Course Timetabling Problem

In this chapter, we study the University Course Timetabling Problem (UCTP), which is a very basic model of the course timetabling task. Despite its simplicity, the model captures the essence of many more complex models: for example, it serves as foundation of the course timetabling models studied in Chaps. 3 and 4. In the following, we will consider two problems based on the UCTP model, a search problem and a reconfiguration problem. The model considers as entities events, rooms, timeslots and event conflicts. A timetable assigns to each event a room and a timeslot. Additionally, a timetable should have two basic properties: First, no two lectures are assigned to the same room and the same timeslot, and second, no two lectures marked as conflicting are scheduled simultaneously, i.e., in the same timeslot. Given appropriate data, the search problem asks for a *witness timetable* that satisfies all required properties. Such a timetable is called feasible. Given two feasible timetables, the reconfiguration problem asks if one timetable can be transformed into the other timetable step-by-step using some elementary operation, such that all intermediate timetables are also feasible. We present a theoretical investigation of the reconfiguration problem and an experimental evaluation of a specialized heuristic for the search problem.

2.1 Problem Formulation

The University Course Timetabling Problem (UCTP) formalizes the task of creating a weekly course schedule for students and lecturers at a university. The UCTP is equivalent to the one given in [dW85, Section 3.4]. Our purpose here is to establish the general setting and to introduce the terminology. A discussion of related problems such as the school timetabling problem and the examination timetabling problem is postponed to the next section.

The UCTP is the following search problem:

Definition 2.1 (University Course Timetabling Problem (UCTP))
INSTANCE:

- a set of events $E = \{e_1, \ldots, e_n\}$
- a set of rooms $R = \{r_1, \ldots, r_\ell\}$
- a set of timeslots $P = \{p_1, \ldots, p_k\}$
- a graph $G = (E, L)$ with nodes E and edges $L \subseteq \{\{u, v\} \mid u, v \in E\}$

The graph G is referred to as the *conflict graph*. Two events are called *conflicting* if they are adjacent in G. The set $P \times R$ contains the *resources*. A *timetable* τ is an assignment $\tau : E \rightarrow P \times R$. Two events e, e' are *overlapping*, if $e \neq e'$ and $\tau(e) = \tau(e')$. A timetable is called *overlap-free* if no two events overlap. Two events e, e' are *clashing* in τ, if they are conflicting and they are assigned to the same timeslot. A timetable is *feasible*, if it is clash-free and overlap-free.
TASK: Find a feasible timetable.

Deciding if there is a clash-free timetable is NP-complete for $|P| \geq 3$. This can be shown by a straight-forward reduction from the (vertex-)k-coloring problem.

The clash-freeness requirement and its relation to the vertex coloring problem is the heart of the matter of automated timetabling in the academic context. It occurs in various formalizations of course and exam timetabling tasks, see e.g. [Sch99]. Depending on the context and the regulatory framework, numerous other requirements may arise in practice, mostly related to resources being available only under certain conditions. We will discuss two types of such requirements namely *availability requirements* and *precedence requirements*, which can be added to the basic UCTP formulation above. Such requirements often arise in practice, see e.g. [Car01, SH07]. Availability requirements dictate that only a certain subset of the resources can be assigned to an event. The availability function α determines for each event the set of available resources:

$$\alpha : E \rightarrow \mathcal{P}(P \times R) \ .$$

A timetable τ satisfies the availability requirements, if for each event $e \in E$, $\tau(e) \in \alpha(e)$. For convenience, the availability function is typically specified by marking rooms and timeslots as (un)available individually per event, see generally [Sch99]. For example, if a timeslot p is marked as unavailable for an event e, then $\alpha(e)$ contains no resources that refer to p. A timetable τ satisfies the timeslot availability requirements, if for each event $e \in E$, $\tau(e)$ does not refer to a timeslot that has been marked as unavailable for e. Similarly, if a room r is unavailable for e, then $\alpha(e)$ does not contain any resources that refer to r. A timetable τ satisfies all room availability requirements, if for each event $e \in E$, $\tau(e)$ does not refer to a room that has been marked as unavailable for e. Clearly, a timetable satisfying the room and timeslot availability requirements satisfies the availability requirements.

Precedence requirements mandates that certain events need to be take place before other events in the weekly schedule. Let the timeslots $(P, <)$ be ordered

and the resources $(P \times R, <)$ be ordered according to the timeslots, i.e., for two resource $(p, r), (p', r') \in P \times R$, $(p, r) < (p', r')$ iff $p < p'$. The precedence relation $\prec \subseteq E \times E$ on the events specifies the precedence requirements: For two events $e, e' \in E$, if $e \prec e'$ then e should precede e' in the timetable according to the given ordering of P. A timetable τ satisfies the precedence requirements, if for all events $e, e' \in E$, $e \prec e' \Rightarrow \tau(e) < \tau(e')$.

An important subproblem of the UCTP is the *room assignment problem*. Given a timeslot $p \in P$, a set $E' \subseteq E$ of events admits a room assignment, if there is an injective mapping $\rho : E' \to R$ such that $(p, \rho(e))$ is available for each $e \in E'$. If the choice of rooms is not restricted by availability requirements, then any set $E' \subseteq E$ of cardinality at most $|R|$ admits a room assignment. On the other hand, if availability requirements restrict the room choices then E' admits a room assignment if the following bipartite graph has a matching of cardinality $|E'|$:

$$H = (E' \cup R, F) \ ,$$

where $F \subseteq \{\{e, r\} \mid e \in E, r \in R\}$ and $\{e, r\} \in F$ if and only if (p, r) is available for the event e.

In addition to the basic entities such as rooms, timeslots, and events, there are typically some compound structures that represent groups of entities, see e.g. [DGMS07]. These compound structures do not add substantial content to the UCTP. However, depending on the setting, they may permit a more intuitive problem description. This is particularly useful for the specification of the objective function in optimization variants of the UCTP.

We will now give examples of compound structures, some of which will occur in the problems studied in Chaps. 3 and 4. Periods are typically grouped into *days* and *timeslots* in order to aid the specification of pattern and precedence constraints, see e.g. [MSP+10]. Furthermore, the conflict graph may be specified in terms of *courses* together with either *curricula* or an *attendance matrix* [MSP+10]. A *course* is a nonempty set of events and the courses are a partition of the events. The individual events of a course are sometimes referred to as *lectures*. Ordinarily, any two events of a course are assumed to be conflicting. A curriculum is a set of courses whose events must not be scheduled simultaneously. Thus, a curriculum induces a clique in the conflict graph. If students are required to explicitly enroll in a set of courses, an attendance matrix can be used to specify which student is enrolled in which course and from this information the conflict graph can be deduced. Additional conflicts may be due to a set of courses being taught by the same lecturer.

2.2 Related Problems

In the literature, numerous combinatorial problems can be found which formalize the task of creating timetables in the educational context. The educational timetabling problems are generally divided into three broad categories, school

timetabling, course timetabling, and examination timetabling [Sch99]. In addition to arranging courses and exams in a timetable, there is the task of providing students with an adequate and balanced workload during their course of study. This has been formalized as the Balanced Academic Curriculum Problem (BACP). Furthermore, there is the task of creating groups of students for tutorials, lab exercises, and the like. This problem often appears as a subproblem of the UCTP and is known as the Student Sectioning Problem (SSP). We will give a brief overview over these problems in the following sections.

We will focus on structural aspects of these problems and summarize the known complexity results. Our motivation for this is twofold: First, we intend to highlight relations between the various problems on a basic combinatorial level. Second, the analysis of the connectedness of the UCTP search space presented in Sect. 2.3.2 considers the reconfiguration variant of the graph coloring problem (see Sect. 2.3.1), which is closely related to the UCTP. The basic formulations of the problems mentioned above give some insights into which reconfiguration problems could be considered for a similar analysis of the search space for these problems. In fact, due to the similar structure, the search space analysis we present in Sect. 2.3.2 applies to the Examination Timetabling Problem (ETP) as well.

Most variants of the problems mentioned above are hard in the sense that the decision variants of the basic problem formulations are NP-complete. The two exceptions are the School Timetabling Problem (STP) and the SSP. Polynomial time algorithms are known for the decision and search variants of these two problems. However, by introducing additional constraints such as timeslot availability requirements, these problems also become NP-complete [dW85, DPS13]. The reductions given by Cooper and Kingston show that there are in fact several sources of NP-completeness for different flavors of the timetabling problem [CK95].

2.2.1 School Timetabling

The school timetabling problem captures the task of assigning teachers to classes in a school. Despite the similar educational context, the STP exhibits a combinatorial structure that is quite different from the UCTP. For the purpose of a brief discussion of the STP, we give a problem formulation that is equivalent to the one proposed in [dW85]. In this model we assume that for each class, the teachers as well as the required number of lectures taught by them are known in advance. For a comprehensive overview of different variants and extensions of the STP as well as solution approaches from the literature please refer to the survey articles [Sch99, Pil13].

Definition 2.2 (School Timetabling Problem (STP))
INSTANCE:

- a set of teachers $T = \{t_1, \ldots, t_n\}$
- a set of classes $C = \{c_1, \ldots, c_m\}$

- a set of timeslots $P = \{p_1, \ldots, p_k\}$
- a $m \times n$ requirement matrix $M = (m_{ij})$, where m_{ij} is the number of lectures taught by teacher t_j in class c_i

Let E be the set of lectures. The number of lectures is determined by M: there are r_{ij} distinct lectures taught by t_j for each class $c_1, \ldots, c_m$. A *meeting* is an element of $T \times C$. A timetable is an assignment $\tau : E \to P$. A timetable is *feasible* if each teacher and each class have at most one lecture in each timeslot.
TASK: Find a feasible timetable.

For a given STP instance $\mathcal{I}$, there is a feasible timetable if and only if each teacher and each class has at most k lectures [dW97]. This result is a consequence of Kőnig's theorem on the existence of k-edge colorings in bipartite multigraphs: every bipartite multigraph G has a k-edge coloring, where k is the maximum degree of the graph G [Kőn16]. Let G be the bipartite multigraph with nodes $T \cup C$ and r_{ij} multiedges between t_j and c_i for each $j \in \{1, \ldots, n\}$, $i \in \{1, \ldots, m\}$. By applying Kőnig's theorem the result is immediate. Equivalently, a feasible timetable can be constructed by finding a sequence of k edge-disjoint matchings in a k-regular bipartite multigraph that is obtained by adding appropriate dummy nodes and edges to G [EIS76]. Thus, in contrast to the UCTP, the search problem STP can be solved in polynomial time. However, if availability requirements are added to the model, which prohibit the use of certain timeslots for teachers, classes or individual meetings, then deciding the existence of a feasible timetable is NP-complete [EIS76, dW97].

In addition to availability requirements, examples of practical requirements that may be included as hard or soft constraints are preference, precedence, workload and lecture distribution requirements. For a systematic review of the different types of constraints that occur in the various STP formulations see e.g. [Pil13]. Due to the similar educational context there is a considerable congruence of UCTP and STP requirements. However, there are certainly differences in the administrative and legal requirements, which may impact, for example, the nature of workload and lecture distribution constraints. For example, leaving students unattended is more problematic in a school than in a university setting [Sch99].

2.2.2 Examination Timetabling

The Examination Timetabling Problem (ETP) and the UCTP are very similar on a structural level. In fact, the UCTP formulation given in Definition 2.1 can serve as a basic model of examination timetabling problems: Each event corresponds to an exam and the task is to schedule all exams such that clash and overlap freeness requirements are satisfied. According to Schaerf, "it is difficult to make a clear distinction between the two problems" [Sch99]. Assuming our UCTP model from

Sect. 2.1, the following three characteristics of the ETP from [Sch99] distinguish it from the UCTP:

- There are different kinds of constraints, in particular distribution constraints that limit the number of exams per day and require that exams are spread over the available timeslots.
- The number of timeslots may be variable.
- Several exams may share a room.

For an overview of different approaches to solving the ETP found in the literature, please refer to the recent survey articles [Lew08, QBM$^+$09].

2.2.3 Other Related Problems

In this section, we will discuss the SSP and the BACP, two combinatorial problems that, like the UCTP, appear in the academic setting. From the organizational perspective, the three problems are related in the following way: The academic workload determined by the BACP for a curriculum defines conflict constraints of the UCTP, and a feasible timetable for some UCTP instance serves as input of the student sectioning.

The Student Sectioning Problem

In order to deal with courses that are attended by a large number of students, universities may split up courses into several sections. Given a fixed timetable, the Student Sectioning Problem (SSP) captures the task of assigning students to sections of the courses such that the course structure is respected and conflicts are minimized [Sch99]. A task that often occurs in practice is the assignment of students to tutorial sessions with a limited capacity of each session. The SSP can be considered as a subtask in the course timetabling process. In recent course and examination timetabling models for benchmarking, such as the ones from the International Timetabling Competition 2007 (ITC2007) [MSP$^+$10], student sectioning is not a part of the problem model. Rudová et al. argue however, that for large-scale practical course timetabling, it is necessary to consider sectioning as a part of the overall course timetabling problem [RMM11].

Recently, several complexity results have been established for the SSP by Dostert et al. [DPS13]: Consider the following Basic Student Sectioning Problem (BSSP): Let $C = \{c_1, \ldots, c_n\}$ be a set of courses, $P = \{p_1, \ldots, p_k\}$ be a set of timeslots, and let $B = (b_{ij})$, be the capacity matrix, where $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, k\}$. Each entry b_{ij} of the matrix gives the maximum capacity of the section of course c_j in timeslot i. Assuming that each student is enrolled in all n courses of a given timetable, the task is to decide if there is a sectioning for m students such that no student is assigned to two simultaneous sections and all capacities requirements

are respected. The BSSP is equivalent to deciding the existence of m edge-disjoint matchings of cardinality n in a bipartite multigraph [DPS13]. The BSSP can be solved in polynomial time, but adding any combination of the following three kinds of requirements makes the problem NP-complete [DPS13]:

- Students can choose from several courses and their choices must be respected.
- Students have timeslot availability requirements.
- Sections have multiple events in different timeslots and conflicts between sections of different events must be respected.

The SSP and the STP have a similar combinatorial structure in the sense that both problems can be modeled as network flow problem with sources and sinks appropriately connected to the weighted complete bipartite graphs given by the capacity matrix and the requirements matrix, respectively. The main difference is the query: The SSP consists of deciding if the flow constraints are maintained for a particular number of students in the model. The STP asks if the arcs generated by the requirements matrix have left-over capacity for a given number of timeslots.

The Balanced Academic Curriculum Problem

Another problem related to course timetabling that occurs in the academic context is the BACP proposed in [CM01]. The general task is to assign courses to teaching periods (e. g. terms) such that the workload is evenly distributed over the teaching periods. In a sense, a solution to the BACP determines which courses need to be scheduled in a given semester, and thus a part of the input of the course timetabling task. An instance $\mathcal{I}$ of a very basic formulation consists of a set C of courses, a set P of teaching periods and a credit assignment $C \to \mathbb{N}$. By a reduction from the 3-partition problem [GJ79, Problem SP15], Chiarandini et al. showed that deciding whether $\mathcal{I}$ admits an assignment of courses to teaching periods such that the workload is perfectly balanced is strongly NP-complete [CDGGS12]. More realistic problem formulations include prerequisite requirements for courses, lower and upper limits for the workload (credits and courses) in each teaching period [CM01, HKW02].

We will revisit the BACP in Chap. 3 in the context of adding fairness criteria to optimization problems.

2.3   The Search Space

We provide a description of the structure of the search space of the UCTP with timeslot availability requirements in terms of graph theoretic concepts. Our motivation of studying the connectedness of the search space stems from the observation that a number of solutions approaches solve course timetabling optimization problems in two steps, see [Lew06, Section 2.3.2]: The first task is to solve the

search problem and the second task is to find among the feasible timetables an optimal one. We provide a brief overview of related results from the literature on the vertex coloring problem and its reconfiguration variant (*recoloring*, for short). We present some new results about the connectedness of so-called reconfiguration graphs which model the search space of clash-free timetables. We derive improved sufficient conditions for the connectedness of the clash-free timetable and show that for a range of instances the clash-free timetables are indeed connected.

2.3.1 Vertex Coloring and Recoloring

For the UCTP and the ETP, the chromatic number of the conflict graph provides a lower bound on the number of timeslots that are required for a feasible schedule. For $k \geq 3$, the problem of deciding whether a graph has a k-coloring is NP-complete [GJ79]. Computing the chromatic number of a graph G with n nodes is NP-hard and for any $\epsilon > 0$, the chromatic number is NP-hard to approximate within $O(n^{1-\epsilon})$ [Zuc07]. However, there are (polynomial-time computable) upper bounds on the chromatic number that prove to be useful in the context of timetabling. The bounds result from analyzing the *greedy coloring algorithm*.

The Greedy Coloring Algorithm

Given a graph G and an ordering of its vertices, the greedy coloring algorithm sequentially colors the vertices in the given order, such that a vertex v is assigned to the smallest color that does not currently appear in the neighborhood of v. For any vertex ordering, the greedy coloring algorithm uses at most $\Delta(G) + 1$ colors, since each vertex is adjacent to at most $\Delta(G)$ neighbors (each of which may have a different color). Better bounds can be obtained by carefully choosing a vertex ordering. Welsh and Powell proposed to sort the vertices in non-increasing order according to their degrees [WP67]. Greedily coloring the vertices of G in this order yields a coloring with at most

$$\mathrm{wp}(G) = \max_{i \in \{1,\dots,n\}} \min\{d_i, i\} \tag{2.1}$$

colors, where d_i is the degree of the ith vertex. They used this result to derive an improved bound on the number of timeslots required for creating clash-free schedules in the context of examination timetabling.

    A further improvement is due to the analysis of the greedy coloring algorithm by Szekeres and Wilf [SW68] and Matula [Mat68]. A graph G is called k-degenerate, if its vertices can be ordered such that each vertex has at most k neighbors preceding it. The smallest k for which G admits such an ordering is the *degeneracy* $\deg(G)$. Let $\sigma(G)$ be an ordering of the vertices of G that is a witness for the degeneracy of

G. The greedy coloring algorithm uses at most $\deg(G) + 1$ colors, if the vertices of G are processed sequentially according to $\sigma(G)$. The degeneracy is equivalent to the largest minimum degree of any subgraph, that is,

$$\deg(G) = \max_{H \subseteq G} \delta(H) \ . \tag{2.2}$$

A witness vertex ordering σ for $\deg(G)$ can be constructed efficiently by repeatedly removing vertices of minimal degree and then reversing the order of removal [Mat68, SW68]. By Using appropriate data structures, $\deg(G)$ can be computed along with a witness σ in time $O(\max\{|V(G)|, |E(G)|\})$ [BZ03].

Proposition 2.3 *The following inequalities hold for any graph G:*

$$\chi(G) \leq \deg(G) + 1 \leq \mathrm{wp}(G) + 1 \leq \Delta(G) + 1 \ .$$

Proof 1. $\chi(G) \leq \deg(G) + 1$. We essentially replicate the proof from [SW68]. Let $G' \subseteq G$ such that $\chi(G') = \chi(G)$, but for any subgraph $H \subset G'$ we have $\chi(H) < \chi(G)$. Thus, G' cannot contain a vertex with fewer than $\chi(G') - 1$ neighbors. Due to Eq. (2.2),

$$\chi(G') - 1 \leq \delta(G') \leq \deg(G) \ .$$

2. $\deg(G) \leq \mathrm{wp}(G)$. From Eq. (2.1) we can conclude that any graph G is $\mathrm{wp}(G)$-degenerate. Since $\deg(G)$ is smallest number k such that G is k-degenerate, we have $\mathrm{wp}(G) \leq \deg(G)$.
3. $\mathrm{wp}(G) \leq \Delta(G)$. The bound $\Delta(G) + 1$ on the number of colors used by the greedy coloring algorithm holds for any ordering of the vertices, including the ones that list the vertices of G in non-increasing order according to their degrees. Therefore, $\mathrm{wp}(G) \leq \Delta(G)$.

$\square$

In Sect. 2.4 we will discuss heuristic approaches to solving the UCTP that are based on the greedy coloring algorithm. Some of these heuristics determine the vertex ordering on-the-fly, which may lead to an improvement in practice but does not to lead to improved general bounds on the chromatic number. For an overview of sequential graph coloring heuristics with applications to timetabling, see [BdWK03].

Vertex Coloring Reconfiguration Graphs

In order to gain some insight into the structure of the search space of the UCTP, we consider the following problem: Given a graph G, is it possible to transform any coloring of G into any other coloring of G in a step-by-step manner, such that each intermediate coloring is proper. Problems of this kind are closely related to

reconfiguration problems. Given an instance $\mathcal{I}$ of a reconfiguration problem, the adjacency relation $\sim$ on $\mathcal{S}(\mathcal{I})$ gives rise to a *reconfiguration graph*, whose nodes are the feasible solutions $\mathcal{S}(\mathcal{I})$ and two nodes $u, v \in \mathcal{S}(\mathcal{I})$ are adjacent iff $u \sim v$. In graph-theoretic terms, a reconfiguration problem addresses the question whether two nodes of the reconfiguration graph are in the same connected component.

There has recently been quite some interest in the study of reconfiguration variants of classical combinatorial problems and their corresponding reconfiguration graphs. A central theme in this line of research is to determine the relation between the complexity class of a combinatorial problem and the complexity class of its reconfiguration problem. Many reconfiguration problems of NP-complete problems turn out to be PSPACE-complete, including for instance Boolean satisfiability (SAT) [GKMP09], the independent set, clique, vertex cover, set cover and integer programming reconfiguration problems [BC09, IDH$^+$11], as well as the graph k-coloring reconfiguration problem for $k \geq 4$ [BC09]. An exception is the reconfiguration variant of the 3-coloring problem, which is in P [CvdHJ11]. Most reconfiguration variants of problems in P studied in the literature have been shown to be in P as well. For instance, the reconfiguration variants of polynomial-time solvable SAT problems [GKMP09] and the matching problem [IDH$^+$11]. An exception is the shortest path reconfiguration problem, which is PSPACE-complete [Bon12].

In addition to attacking reconfiguration problems from the complexity theoretic point of view, there have been investigations of structural properties of reconfiguration graphs [Moh07, BJL$^+$11, BJL$^+$12]. The analysis of the UCTP search space will follow this line of research. Given a coloring c of a graph G, an *elementary recoloring* changes the color of a single vertex of G. Two k-colorings c_1 and c_2 of G are adjacent, $c_1 \sim_E c_2$, if there is an elementary recoloring that transforms c_1 into c_2. The Kempe-exchange is a generalization of the elementary recoloring operation. Recall that, given two colors a and b, a Kempe-exchange switches the colors of a Kempe-component, which is a connected component in $G(a, b)$. The result of this operation is a new coloring, such that, within the Kempe-component, all vertices of the of color a are assigned to color b and vice versa. Two colorings c_1 and c_2 of G are adjacent with respect to the Kempe-exchange, $c_1 \sim_K c_2$, if there is a Kempe-exchange that transforms c_1 into c_2. Each of the two adjacency relations $\sim_E$ and $\sim_K$ gives rise to a graph structure on the set of k-colorings of G.

Definition 2.4 ((Kempe-)k-Coloring Graph) For a graph $G = (V, E)$ and $k \in \mathbb{N}$ let

$$\mathcal{V} := \{c : V \to \{1, \ldots, k\} \mid c \text{ is a } k\text{-coloring of } G\}$$

$$\mathcal{E}_E := \{\{c_1, c_2\} \mid c_1, c_2 \in \mathcal{V} \text{ and } c_1 \sim_E c_2\}$$

$$\mathcal{E}_K := \{\{c_1, c_2\} \mid c_1, c_2 \in \mathcal{V} \text{ and } c_1 \sim_K c_2\} \ .$$

Then the k-coloring graph is the graph $\mathcal{C}_k(G) = (\mathcal{V}, \mathcal{E}_E)$. The Kempe-$k$-coloring graph is the graph $\mathcal{K}_k(G) = (\mathcal{V}, \mathcal{E}_K)$.

Fig. 2.1 The
(Kempe-)3-coloring graph of
K_2. *Solid edges* correspond
to elementary recolorings.
Dashed edges correspond to
Kempe-exchanges that are
not equivalent to an
elementary recoloring
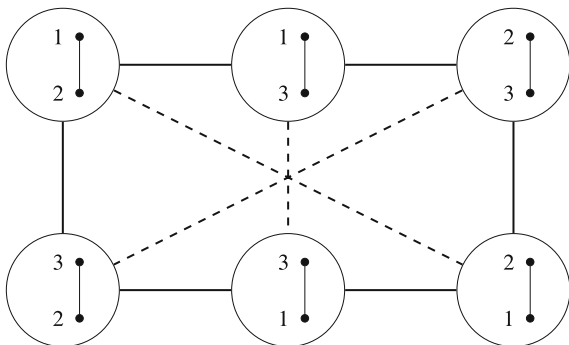
Since the Kempe-exchange is a generalization of the elementary recoloring operation, $\mathcal{C}_k(G) \subseteq \mathcal{K}_k(G)$ holds for any graph G and any $k \in \mathbb{N}$. Figure 2.1 shows the reconfiguration graphs $\mathcal{K}_3(K_2)$ and $\mathcal{C}_3(K_2)$. Any two nodes that are connected by a (thick) solid line are adjacent in $\mathcal{K}_3(K_2)$ and $\mathcal{C}_3(K_2)$. Any two nodes that are connected by a dashed line are adjacent in $\mathcal{K}_3(K_2)$ but not in $\mathcal{C}_3(K_2)$. To the best of our knowledge, the recognition problem for reconfiguration graphs in general and for vertex coloring reconfiguration graphs in particular has not been addressed so far in the literature. We state the following observation:

Proposition 2.5 *Let K_n be the clique on n nodes. Then*

1. *$\mathcal{C}_k(K_n)$ is $(n(k-n))$-regular for all $k \geq n$,*
2. *$\mathcal{C}_{n+1}(K_n)$ has girth six for all $n \geq 2$,*
3. *$\mathcal{C}_k(G)$ has girth three for any non-empty graph G and $k > \delta(G) + 2$.*

Proof Part 1: For each of the n nodes of K_n there are $k - n$ colors to choose from while all other colors remain fixed, so each node has $n(k - n)$ neighbors.
Part 2: There is no cycle in $\mathcal{C}_k(K_n)$ that involves a color change of only a single vertex since $k = n + 1$. Any cycle in $(\mathcal{C}_{n+1}(K_n))$ that changes the colors of at least three vertices of K_n has length at least six, since each vertex of G has to be assigned its original color eventually. Consider cycles of $\mathcal{C}_{n+1}(K_n)$ that recolor exactly two vertices. Such cycles are isomorphic to $\mathcal{C}_3(K_2)$, which is a cycle of length six, see Fig. 2.1. Thus, $\mathcal{C}_{n+1}(K_n)$ contains cycles of length six, and all other cycles have at least length six. Therefore, girth($\mathcal{C}_{n+1}(K_n)$) = 6.
Part 3: Pick any coloring c of G and a vertex $u \in V(G)$ of degree $\delta(G)$. Since $k > \delta(G) + 1$ there are at least two colors α and β that are not present in the neighborhood of u. Therefore, u can be recolored to α, then to β and back to $c(u)$, which implies that girth($\mathcal{C}_k(G)$) = 3. □

An inspection of the spectrum of $\mathcal{C}_{k+1}(K_k)$ for small values of k leads us to the following conjecture:

Conjecture 2.6 For $k \geq 3$, the k-coloring graph $\mathcal{C}_{k+1}(K_k)$ has eigenvalue spectrum $0, \pm 1, \dots, \pm k$.

The Kempe-exchange is a popular operation for exploring the search space of course and examination timetabling problems. For this purpose, we are in particular interested in whether a (Kempe-)k-coloring graph is connected, and to some extent, in upper bounds on the number of recoloring operations needed to reach any coloring from a given starting point. Clearly, if $C_k(G)$ is connected for some k and G, then $\mathcal{K}_k(G)$ is also connected. Bonsma and Cereceda showed in [BC09] that deciding if two colorings c, c' of a graph G are connected in $C_k(G)$ is PSPACE-complete for $k \geq 4$ by giving a reduction from the SLIDINGTOKENS problem from [HD05]. However, the connectedness of $C_k(G)$ has been established for sufficiently large k:

Theorem 2.7 *The k-coloring graph $C_k(G)$ is connected, if*

1. $k \geq \Delta(G) + 2$ *([Jer95]), or*
2. $k \geq \deg(G) + 2$ *([BC09]).*

Bonsma and Cereceda showed in [BC09] that two connected k-colorings may have superpolynomial (in the size of the graph) distance in the k-coloring graph. In the following cases however, the diameter of the reconfiguration graph is known to be quadratic:

Theorem 2.8 *The k-coloring graph $C_k(G)$ has diameter $O(|V(G)|^2)$ if any of the following conditions is satisfied:*

1. $k \geq 2\deg(G) + 1$ *[BC09]*
2. $\chi(G) = 3$ *and $C_3(G)$ is connected [CvdHJ11].*

Concerning Kempe-k-coloring graphs, Mohar has shown that:

Theorem 2.9 ([Moh07, Corollary 4.5]) *The Kempe-k-coloring graph $\mathcal{K}_k(G)$ is connected if G is planar and $k \geq \chi(G)$.*

The following theorem by Las Vergnas and Meyniel establishes sufficient conditions for the connectedness of Kempe-k-coloring graphs in a more general setting. We will give a proof which is essentially the same as the one given in [Moh07, Proposition 2.4]. However, we will present it in a slightly more algorithmic style. We encode a Kempe-exchange operation by a triple (a, b, u), where a and b are colors and u is a vertex of the Kempe-component. In order to conveniently modify colorings in our algorithms, we encode them as arrays indexed by the nodes of a graph. For example, by $c[u]$ we refer to the color of vertex u with respect to the coloring c.

Theorem 2.10 ([LVM81, Proposition 2.1]) *For any graph G, the Kempe-k-coloring graph $\mathcal{K}_k(G)$ is connected if $k \geq \deg(G) + 1$.*

Proof Let $\ell = \deg(G)$ and let c_1, c_2 be k-colorings of G. Without loss of generality, we assume that c_1 is the source coloring and c_2 is the target coloring. From a vertex ordering that is a witness of $\deg(G)$ we get a labeling $v_1, \ldots, v_n$ of the vertices of G such that for each $i \in \{1, \ldots, n\}$, v_i has at most ℓ neighbors in $G[v_1, \ldots, v_i]$. We prove that the algorithm KEMPERECOLOR with input G, c_1, c_2, and the labeling

---

Algorithm 1: KEMPERECOLOR

---

input : graph G, labeling $v_1, \ldots, v_n$ of the vertices, k-colorings c_1, c_2 of G
output: list of Kempe-exchanges transforming c_1 into c_2
data : array c of length n storing the current color of each vertex, list K of
    Kempe-exchanges

```
/* initialization                                              */
```
$K \longleftarrow$ empty list
for $i \longleftarrow 1$ **to** n:
  $\quad c[i] \longleftarrow c_1(v_i)$

```
/* recoloring                                                  */
```
for $i \longleftarrow 1$ **to** n:
  $\quad G' \longleftarrow G[v_1, \ldots, v_i]$
  ```
  /* Kempe-exchange κ encoded as κ = (a,b,u), where a,b are
     colors and u ∈ V(G')                                       */
  ```
  $\quad$**for** $\kappa = (a, b, u) \in K$:
  $\qquad$ let w.l.o.g. $c[i] \neq a$
1 $\qquad$ **if** $c[i] = b$ and v_i *has exactly one neighbor of color a in G'*:
  $\qquad\quad c[i] \longleftarrow a$
2 $\qquad$ **else if** $c[i] = b$ and v_i *has at least two neighbors of color a in G'*:
  $\qquad\quad$ *choose color $b' \neq b$, which is not used by any neighbor of v_i in G'*
  $\qquad\quad$ *insert Kempe-exchange (b, b', v_i) right before κ in K*
  $\qquad\quad c[i] \longleftarrow b'$
3 $\quad$ append Kempe-exchange $(c_2(v_i), c[i], v_i)$ to K
return K

$v_1, \ldots, v_n$ outputs a sequence of Kempe-exchanges that transforms c_1 into c_2. For this purpose, we show that for each iteration i of KEMPERECOLOR the following two properties hold: (i) the colors of $v_1, \ldots, v_{i-1}$ are not altered due to v_i, and (ii) v_i is assigned to its target color $c_2(v_i)$ at the end of iteration i.

The crucial steps needed to establish property (i) are performed in lines 1 and 2. If v_i is not affected by Kempe-exchange $\kappa = (a, b, u)$ then no special treatment of v_i is needed. If v_i is affected by κ, then there are two cases to consider. If v_i has color b and there is exactly one neighbor of v_i in $G' = G[v_1, \ldots, v_i]$ of color a (line 1), then κ either changes the color of v_i to a or it leaves the color of v_i unchanged. On the other hand, if v_i has color b and at least two neighbors of v_i in G' have color a (line 2), then at most $\ell - 1$ colors are present in the neighborhood of v_i in G'. Therefore, if $\ell + 1$ colors are available, then v_i can be recolored to a color b' that is not present in the neighborhood of v_i prior to performing the Kempe-exchange κ. As a consequence, v_i is then unaffected by κ and thus, v_i cannot interfere with the colors of $v_1, \ldots, v_{i-1}$. At the end of iteration i (line 3), v_i can safely be assigned to color $c_2(v_i)$, which establishes property (ii).

Thus, after iteration n, each vertex has been assigned to its target color. At most $\ell + 1$ colors are required in each step. $\qquad\qquad\square$

Note that the number of Kempe-exchanges added to the list K in Algorithm 1 may double in each iteration in the worst case. Thus, although no witness of two colorings at a superpolynomial has been mentioned in the literature, we cannot conclude that the output of KEMPERECOLOR has polynomial size.

In addition to these results several properties of (Kempe-)k-coloring graphs are known for graphs belonging to particular classes. A graph is *chordal*, if any cycle of length at least four has a chord. Every chordal graph is $(\chi - 1)$-degenerate. A *perfect elimination ordering* of a graph is an ordering of the vertices such that for each vertex v the neighbors that occur after v in the ordering induce a clique. A graph is chordal if and only if it admits a perfect elimination ordering. A perfect elimination ordering of the vertices of a chordal graph can be found efficiently by a variant of Lexicographic Breadth-first Search (LBFS) [RTL76]. LBFS is a particular class of Breadth-first Search (BFS) algorithms that is useful for efficiently recognizing various graph classes, including for instance chordal graphs, chordal-bipartite graphs, interval graphs and co-graphs [Cor05, Ueh02]. A graph is *chordal bipartite*, if it is bipartite and any cycle of length at least six has a chord.

Theorem 2.11 ([BJL$^+$11]) *If G is chordal or chordal-bipartite and $k \geq \chi(G) + 1$ then* $\mathrm{diam}(\mathcal{C}_k(G)) = O(|V(G)|^2)$.

Bonamy et al. further show that for each $k \geq 2$ there is a k-colorable chordal graph such that $\mathrm{diam}(\mathcal{C}_{k+1}(G)) = \Theta(|V(G)|^2)$ [BJL$^+$11]. We will show that any chordal graph G can be recolored with at most $|V(G)|$ Kempe-exchanges, i.e., $\mathrm{diam}(\mathcal{K}_k(G)) \leq |V(G)|$ (see Corollary 2.13). For this purpose consider the sequential recoloring algorithm SEQRECOLOR shown in Algorithm 2: The input is a graph G, labeling of the vertices $V(G)$ and two colorings c, c' of G. We assume that c is the source coloring and c' is the target coloring. The algorithm recolors the vertices one-by-one according to the labeling such that in each step i, one Kempe-exchange is performed that assigns to the current vertex v_i its target color $c'(v_i)$ if its current color and its target color differ. The output is a coloring of G. If the output is equivalent to the target coloring c' we say that SEQRECOLOR transforms c into c'. We investigate the conditions under which this is the case.

Algorithm 2: SEQRECOLOR

---

input : graph G, colorings c, c' of G, labeling $v_1, \ldots, v_{|V(G)|}$ of $V(G)$
output: the coloring b of G

```
/* initialization                                                    */
for i ⟵ 1 to |V(G)|:  b[v_i] ⟵ c(v_i)

/* recoloring                                                         */
for i ⟵ 1 to |V(G)|:
    if b[v_i] ≠ c'(v_i):
        perform Kempe-exchange (b(v_i), c'(v_i), v_i)

return b
```

Theorem 2.12 *For any chordal graph G there is an ordering of $V(G)$ such that for any two colorings c, c′ of G* SEQRECOLOR *transforms c into c′.*

Proof Without loss of generality, we can assume that G is connected, since if it is not, we can treat each connected component individually. Since G is chordal it admits a perfect elimination ordering. We show that applying SEQRECOLOR to the vertices of G in a reversed perfect elimination ordering transforms c into $c′$. We verify that in each iteration i, $1 \leq i \leq n$, the Kempe-exchange that recolors v_i from its current color to its target color $c′(v_i)$ leaves the color of any vertex $w > v_i$ unchanged.

We assume for a contradiction that in iteration i there is a vertex $w > v_i$ in the elimination ordering such that the Kempe-exchange $\kappa = (b[v_i], c′(v_i), v_i)$, which assigns v_i to its target color, changes the color of w as well. We pick w to be the smallest such vertex. If $b[v_i] = c′(v_i)$ then no recoloring is performed and thus no such w exists. If $b[v_i] \neq c′(v_i)$ then the Kempe-component corresponding to κ is a tree T, since $G(b[v_i], c′(v_i))$ is bipartite and every cycle of length at least four has a chord. Since the color of w is changed by κ it is a vertex of T and has either color $b[v_i]$ or $c′(v_i)$. Thus, there is a unique path $v_i = u_1 — \ldots — u_m — w$ in T. If w were adjacent to v_i then its color can neither be $b[v_i]$ nor $c′(v_i)$, so w is not adjacent to v_i and $m \geq 2$. The vertices on the path satisfy $u_m < \ldots < u_1 < w$ with respect to the elimination ordering, since $v_i = u_1 > u_2$ and for all $1 < i < m$, $u_i < u_{i-1} < u_{i+1}$ or $u_i < u_{i+1} < u_{i-1}$ implies that u_{i-1} and u_{i+1} are adjacent. From $u_m < u_{m-1} < w$ we can conclude that u_{m-1} is adjacent to w, which is a contradiction to T being a tree and thus, there is no vertex $w > v_i$ that is recolored by κ.   □

The next corollary follows directly from Theorem 2.12 and the fact that SEQRECOLOR performs at most one Kempe-exchange per vertex.

Corollary 2.13 *If G is chordal and $k \geq \chi(G)$ then* $\mathrm{diam}(\mathcal{K}_k(G)) \leq |V(G)|$.   □

Please note that the order in which the vertices are processed depends only on the structure of G, not on the given colorings. As a slight generalization of this, we introduce the class of sequentially recolorable graphs.

Definition 2.14 A graph G is *sequentially recolorable*, if for any two colorings c, $c′$ of G there is an labeling of the vertices such that SEQRECOLOR transforms c into $c′$.

Clearly, due to Theorem 2.12 any chordal graph is sequentially recolorable. So far, no other graph classes are known to be sequentially recolorable. We conjecture that:

Conjecture 2.15 Every chordal bipartite graph is sequentially recolorable.

In the following, we will present some results, mainly about which methods cannot be used to prove Conjecture 2.15. The following proposition is a direct consequence of [Moh07, Proposition 2.1], which establishes that $\mathcal{K}_k(G)$ is connected

for any $k \geq 2$ if G is bipartite. The proof is omitted in the reference, so we give a short proof here for completeness.

Proposition 2.16 *If G is bipartite and $k \geq 2$ then* $\mathrm{diam}(\mathcal{C}_k(G)) \leq 2|V(G)|$.

Proof Let $G = (A \cup B, E)$ be a bipartite graph, c, c' be two k-colorings of G, $k \geq 2$, and b be a 2-coloring of G. Without loss of generality, we assume that G is connected and $b(v) = 1$ if $v \in A$, and $b(v) = 2$ otherwise. We further assume that the vertices of G are ordered such that $v > w$ whenever $v \in A$ and $w \in B$ and ties are broken arbitrarily. We show that SEQRECOLOR transforms c into b if the vertices are processed according to such an ordering. For this purpose, we show that whenever a Kempe-exchange is performed on a vertex $v \in A$ then the number of vertices of color 1 in A strictly increases. Assume for a contradiction that this is not the case. Then performing the Kempe-exchange that recolors v from its current color $\alpha \neq 1$ to 1 must also recolor another vertex $w \in A$ from color 1 to α. Thus, there is a path $v - \ldots - w$ of even length in the subgraph induced by the vertices of color 1 or α, and since G is bipartite vertices of color α and 1 must be alternating on the path. However, since v is of color α and w is of color 1 this is a contradiction. The same argument holds whenever a vertex in B is assigned the color 2. Thus, SEQRECOLOR transforms c into b.

Reversing the order of the Kempe-exchanges performed by SEQRECOLOR for transforming c' into b yields a sequence of Kempe-exchanges that transforms b into c'. Thus we can transform c into c' via the 2-coloring b. Since SEQRECOLOR is used at most twice, $\mathrm{diam}(\mathcal{K}_k(G)) \leq 2|V(G)|$.   □

It is an open question if the bound on the diameter of the reconfiguration graph can be improved to $\mathrm{diam}(\mathcal{K}_k(G)) \leq |V(G)|$ if G is bipartite. Even for the class of chordal bipartite graphs the question is not settled (see Conjecture 2.15). In the spirit of the characterization of chordal graphs via a perfect elimination ordering, chordal bipartite graphs are characterized by the existence of a weak elimination ordering. A vertex u of a bipartite graph is called *weakly simplicial* if its neighbors can be labeled $v_1, \ldots, v_k$ such that $\mathrm{N}(v_i) \subseteq \mathrm{N}(v_j)$ whenever $i < j$ [BJL$^+$12]. An ordering $v_1, \ldots, v_n$ of the vertices of a bipartite graph G is a *weak elimination ordering* if for each $i \in \{1, \ldots, n\}$, v_i is weakly simplicial in the subgraph of G induced by $\{v_i, \ldots, v_n\}$.

Theorem 2.17 ([Ueh02, Theorem 1]) *A bipartite graph is chordal bipartite if and only if its vertices admit a Weak Elimination Ordering (WEO).*   □

A WEO of the vertices can be found in linear time by a variant of the LBFS algorithm proposed in [Ueh02]. Starting from a given initial vertex, generic BFS algorithm partitions the vertices of a graph into layers and processes the vertices layer by layer. Any two nodes in a layer have the same distance to the initial vertex. In contrast to the generic BFS, the LBFS algorithm visits the vertices of a layer in a specific order: In each step, the next vertex is selected from the vertices with the lexicographically largest label and the labels are updated after each iteration. The exact procedure is not relevant for our investigations. The details can be found

Fig. 2.2 A graph recoloring instance, consisting of a chordal bipartite graph G and two vertex colorings $c, c' : V(G) \rightarrow \{1, 2, 3\}$. The instance satisfies both parts of Proposition 2.18
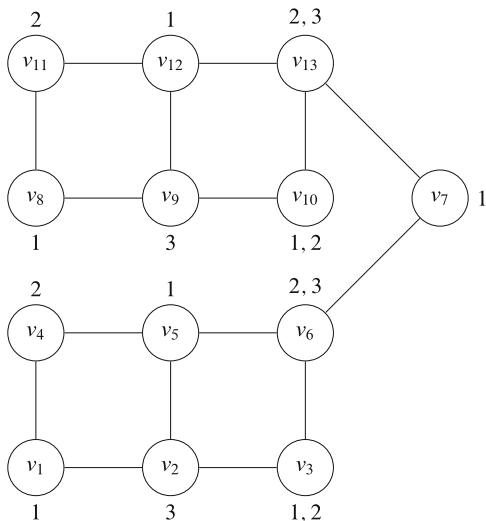
e.g. in [Cor05]. However, reversing the ordering in which the vertices are visited by the LBFS yields a WEO.

We present a graph coloring reconfiguration instance such that for none of the vertex orderings produced by a BFS, SEQRECOLOR transforms the source coloring into the target coloring. In particular, this includes all WEOs produced by a LBFS traversal of the graph. However, we also give a WEO of the vertices for which SEQRECOLOR correctly transforms the source into the target coloring. Therefore, our instance does not rule out WEOs for proving that chordal bipartite graphs are sequentially recolorable.

Figure 2.2 shows a bipartite chordal graph G consisting of two dominoes connected via the node v_7. There are two colorings $c, c' : V(G) \rightarrow \{1, 2, 3\}$ shown next to the nodes. The first color is the one assigned by c and the second color is the one assigned by c'. If only one color $\alpha \in \{1, 2, 3\}$ is shown next to a node v then $c(v) = c'(v) = \alpha$. For example, $c(v_1) = c'(v_1) = 1$, $c(v_3) = 1$, and $c'(v_3) = 2$.

Proposition 2.18 *There is a chordal bipartite graph G and two colorings c, c' of G such that*

1. SEQRECOLOR *does not transform c to c' for any vertex ordering that is the revers of a BFS vertex ordering, and*
2. *there is a WEO for which* SEQRECOLOR *transforms c to c'.*

Proof First, consider a BFS traversal of the lower domino $G[v_1, \ldots, v_6]$, which begins at vertex v_6. The BFS partitions the vertices into the layers

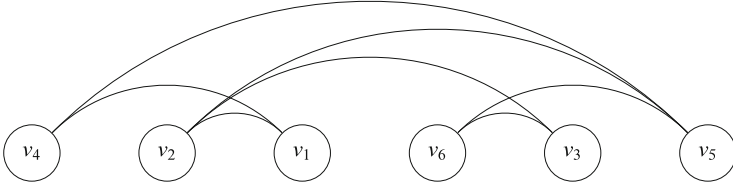$$\{v_6\}, \{v_3, v_5\}, \{v_2, v_4\}, \{v_1\} \ .$$

Fig. 2.3 A WEO of lower domino of the graph shown in Fig. 2.2

Breaking ties between the nodes within a layer is of no significance, thus the argument holds for any BFS specialization, including LBFS. After reversing the order of traversal, v_3 is processed by SEQRECOLOR after v_1, and therefore a Kempe-exchange assigning to v_3 its target color 2 also changes the color of v_5, which is in the previous layer. Thus, SEQRECOLOR will not produce the target coloring c'. By connecting the two dominoes via v_7, a BFS starting at any node either visits v_6 before v_3, or, equivalently, v_{13} before v_{10}. Therefore, SEQRECOLOR does not transform c into c' for any vertex ordering that is the reverse of a BFS vertex ordering, which concludes the proof of part 1.

Now, consider the vertex ordering $v_4, v_2, v_1, v_6, v_3, v_5$, shown in Fig. 2.3, which is a WEO of the lower domino. Similarly, $v_{11}, v_9, v_8, v_{13}, v_{10}, v_{12}$ is a WEO of the upper domino $G[v_8, \ldots, v_{13}]$. We combine the two vertex orderings such that the lower domino appears after the upper domino and insert v_7 in the ordering right after v_6. The resulting vertex ordering is a WEO. SEQRECOLOR transforms c into c', which proves part 2.   □

2.3.2 Connectedness

According to the classification of heuristic optimization algorithms for timetabling problems in [Lew06], many approaches in the literature fall in the category of *two-step optimization algorithms*. In the first step, the underlying search problem is solved and in the second step, the optimization takes place, during which only feasible solutions are considered and the result of the first step is used as a starting point. A recent example of a state-of-the-art two-step approach is [LH10], numerous other examples can be found in [Lew06]. During the optimization step, feasible timetables are modified using Kempe-exchanges or similar operations that preserve the feasibility. It is natural to ask whether any feasible timetable, in particular an optimal one, can be reached from an initial feasible timetable. We investigate conditions that establish the connectedness of feasible timetables with respect to the Kempe-exchange.

In our analysis, we consider the clash-freeness requirement and do not include any requirements related to assigning rooms. We apply the results from Sect. 2.3.1 to show that for a range of benchmark instances (which include computer-generated

and real-world instances) the search space of clash-free timetables is connected in the sense that the corresponding Kempe-p-coloring graph (see Definition 2.4) is connected. For other benchmark instances, the conditions for the connectedness are not satisfied, and further investigations are needed to settle whether the reconfiguration graphs are connected or not. We further show how timeslot availability requirements can be included in the analysis and derive improved conditions for the connectedness of the reconfiguration graphs in this setting.

In most applications, clash-freeness is not the only requirement a timetable needs to satisfy. Therefore, the overall goal is to include additional requirements such as timeslot availability, room availability and overlap-freeness in the analysis of the properties of the search space. Let $\mathcal{I}$ be a UCTP instance with p timeslots, a conflict graph G, and let V' be the set of colorings of G that satisfy the additional requirements. Then the search space of instance $\mathcal{I}$ is connected if $\mathcal{K}_p(G)[V']$ is connected. In particular, for the additional requirements above, the corresponding reconfiguration graphs are the subgraphs of $\mathcal{K}_p(G)$ induced by the following sets of nodes:

1. timeslot availability requirements:

$$V_\pi = \{c \in V(\mathcal{K}_p(G)) \mid \forall v \in V(G) : c(v) \text{ is available for event } v\}$$

2. overlap freeness and room availability requirements:

$$V_\rho = \{c \in V(\mathcal{K}_p(G)) \mid \forall i \in P : \text{ color class } i \text{ admits a room assignment}\}$$

In our analysis we will focus on conditions that establish the connectedness of $\mathcal{K}_p(G)[V_\pi]$. Regarding overlap freeness and room availability requirements, to the best of our knowledge, the properties of the corresponding reconfiguration graphs have not been studied so far. The *bounded vertex k-coloring problem* with bound $b \in \mathbb{N}$ is the problem of coloring a graph with k colors such that each color is used at most b times. The bounded vertex coloring problem has been studied, for example, by Lucarelli [Luc09], and Baker and Coffmann [BC96] in the setting of unit-time task scheduling on multiple processors and by de Werra in the timetabling context [dW97]. If overlap freeness is required and no particular room availability requirements are present, then the graph $\mathcal{K}_P(G)[V_\rho]$ is the reconfiguration graph of a bounded vertex coloring instance. The reconfiguration variant of the bounded vertex coloring problem seems to be an interesting problem which deserves further investigation. The situation gets more involved if room availability requirements are present. Checking if the k events in a color class admit a room assignment is equivalent to checking if a certain bipartite graph admits a matching of cardinality k. At the moment, a description of the corresponding reconfiguration graph that allows us to reason about its connectedness is out of reach.

Connectedness in the Presence of Period Unavailabilities

In the following, let $G = (V, E)$ be the conflict graph of a UCTP instance. The problem of deciding if there is a clash-free timetable that additionally satisfies timeslot availability requirements is equivalent to deciding if G admits a *list coloring*. Given a set $L(v)$ (called *list*) of available colors for each $v \in V$, a list coloring $c : V \to \bigcup_{v \in V} L(v)$ of G is a proper coloring of G such that $c(v) \in L(v)$ for each $v \in V$. Since in the UCTP context, the conflict graph is colored with the timeslots, the timeslot availability requirements for each event $v \in V$ can be expressed by letting $L(v)$ be the set of timeslots available for v. Graph coloring is a special case of list coloring, since all colors are available for each node. By using a standard technique [dW85, Proposition 3.2], list coloring can be reduced to graph coloring: Let the colors be labeled $1, \ldots, p$, where $p = \left| \bigcup_{v \in V} L(v) \right|$. Now, let the graph G' be a copy of G to which we add a clique C on p nodes $v_1, \ldots, v_p$. For each $v \in V(G)$, we add an edge $v \,\text{—}\, v_i$ to G', whenever $i \notin L(v)$. Clearly, G' admits a p-coloring if and only if G admits a list coloring.

Our goal is to show that $\mathcal{K}_p(G)[V_\pi]$ is connected if and only if $\mathcal{K}_p(G')$ is connected. However, there is no Kempe-exchange on $\mathcal{K}_p(G)[V_\pi]$ corresponding to a Kempe-exchange on G' involving any of the nodes $v_1, \ldots, v_p$. We construct a graph K, which is a copy of $\mathcal{K}_p(G)[V_\pi]$ with a self loop added to each node. Additionally, we add to K an edge between two colorings $u, v \in V(K)$, if there are two colors i and j such that u can be transformed into v by swapping the colors in all except a single connected component of $G(i, j)$. These additional edges are merely shortcuts for several individual Kempe-exchanges. Therefore, $\mathcal{K}_p(G)[V_\pi]$ is connected if and only if K is connected. Figure 2.4 shows the various graphs under consideration for

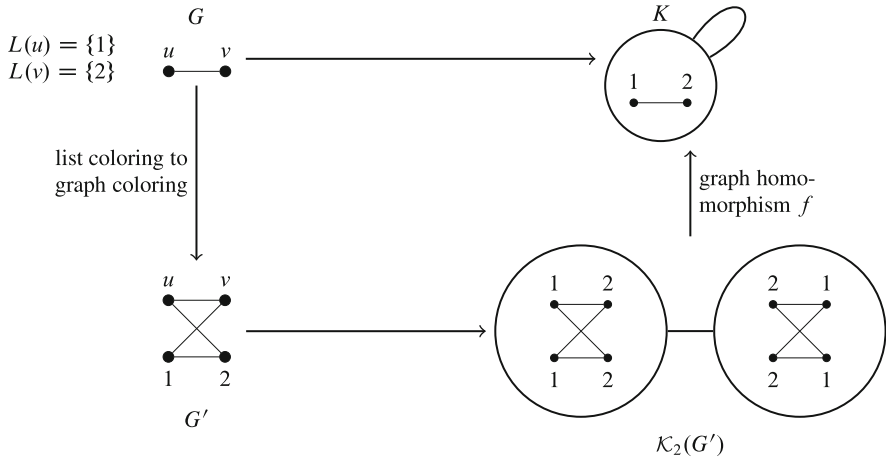**Fig. 2.4** Relations between the graphs G, G', K and $\mathcal{K}_p(G')$. The choice of G and the available colors determines the other graphs as described in the text. The existence of the graph homomorphism f is established by Lemma 2.19

a list-coloring instance consisting of a graph $G = (\{u, v\}, \{u — v\})$ and color lists $L(u) = \{1\}$ and $L(v) = \{2\}$. The nodes 1 and 2 of G' were added by the reduction from list to graph coloring.

Lemma 2.19 *There is a graph homomorphism* $f : \mathcal{K}_p(G') \to K$.

Proof We construct the mapping $f : V(\mathcal{K}_p(G')) \to V(K)$. Let $c \in V(\mathcal{K}_p(G'))$. First, we swap the colors of the p color classes such that v_i has color i for each $i \in \{1, \dots, p\}$. This can be achieved by applying a sequence of Kempe-exchanges to the coloring c: For each color $j \in \{1, \dots, p\}$, if the current color of v_j is $i \neq j$ we swap the colors in $G'(i, j)$. One Kempe-exchange is required for each connected component of $G'(i, j)$. Let c' be the resulting coloring. Except for the vertices $v_1, \dots, v_p$ and their incident edges, G' is just a copy of G. Now, pick $f(c) = \tilde{c}$, where $\tilde{c}$ is equivalent to c' restricted to the vertices $V(G) \subset V(G')$. Clearly, $\tilde{c}$ is a coloring of G. We show that it is also a list coloring of G: For a node $v \in V(G)$, assume that $\tilde{c}(v) = i$ and color $i \notin L(v)$. Then c' cannot be a proper coloring of G' since there is an edge $v — v_i$ in G' and $c(v_i) = i$, a contradiction. Since $\tilde{c}$ is a list coloring of G we have $\tilde{c} \in V(K)$.

We show that f is a graph homomorphism as required. Let $\{a, b\} \in E(\mathcal{K}_p(G'))$ and let κ be a Kempe-exchange that is a witness of $a \sim_K b$. There are two cases to consider:

1. The Kempe-exchange κ does not involve any of the nodes $v_1, \dots, v_p$. Then f renames the color classes of the colorings a and b if required and there is a Kempe-exchange corresponding to κ that establishes $f(a) — f(b)$ in K.
2. The Kempe-exchange κ involves two nodes $u, v \in \{v_1, \dots, v_p\}$. We need to consider following two subcases. If $G'(a(u), a(v))$ is connected then $f(a) = f(b)$ and thus, $f(a) — f(b)$ since each node of K has a self-loop. If it is not connected then $f(a)$ and $f(b)$ differ with respect to the color classes $a(u)$ and $a(v)$. We show that $f(a)$ and $f(b)$ are connected by a sequence of Kempe-exchanges that swaps the colors in all except a single connected component of $G'(a(u), a(v))$ and thus $f(a) — f(b)$ by the construction of K above. To obtain $f(b)$, we first apply κ to a on G' and then apply f to the resulting coloring. The Kempe-exchange κ swaps the colors of the connected component of $G'(a(u), a(v))$ containing u and v, and then f swaps the colors of the color classes u and v. As a result, $f(b)$ can be constructed from $f(a)$ by swapping the colors in all connected components of $G'(a(u), a(v))$ except the one containing u and v in the preimage $f^{-1}(V(G(a(u), a(v))))$.

   In summary, for all $a, b \in V(\mathcal{K}_p(G')) : a — b$ implies $f(a) — f(b)$. □

The graph homomorphism f defines the following equivalence relation $\sim_f$ on $V(\mathcal{K}_p(G'))$: for $a, b \in V(\mathcal{K}_p(G')) : a \sim_f b$ if $f(a) = f(b)$.

Theorem 2.20 $\mathcal{K}_p(G)[V_\pi]$ *is connected if and only if* $\mathcal{K}_p(G')$ *is connected.*

Proof We noted above that $\mathcal{K}_p(G)[V_\pi]$ is connected if and only if K is connected. Let $f : \mathcal{K}_p(G') \to K$ be the graph homomorphism from Lemma 2.19.

"Only if" part: Let $\mathcal{K}_p(G')$ be connected. Then K is connected since there is a graph homomorphism $\mathcal{K}_p(G') \to K$ and graph homomorphisms preserve connectedness. Thus, $\mathcal{K}_p(G)[V_\pi]$ is connected.

"If" part: Let $\mathcal{K}_p(G)[V_\pi]$ be connected. Then K is connected. Due to the first isomorphism theorem, $K \cong \mathcal{K}_p(G')_{/\sim_f}$ and thus, $\mathcal{K}_p(G')_{/\sim_f}$ is also connected. Any two colorings u, v of G' such that $u \sim_f v$ are connected by Kempe-exchanges since one can be obtained from the other by permuting the colors of the color classes.
□

For general graphs, not much is known about the diameter of the corresponding Kempe-k-coloring graphs. Using the graph homomorphism from Lemma 2.19, we show that the reduction from list to graph coloring increases the (possibly unknown) diameter only moderately:

Theorem 2.21 $\mathrm{diam}(\mathcal{K}_p(G)[V_\pi]) \leq \lfloor \frac{|V(G)|-1}{2} \rfloor \cdot \mathrm{diam}(\mathcal{K}_p(G'))$.

Proof For any adjacent nodes u, v in $\mathcal{K}_p(G')$, we count how many Kempe-exchanges are required to get from $f(u)$ to $f(v)$ in $\mathcal{K}_p(G)[V_\pi]$. Let κ be the Kempe-exchange that is a witness of $u \text{---} v$, and let i and j be the involved color classes. If $u \sim_f v$ then, in the worst case, all except one connected component of $G(i, j)$ need to be switched to get from u to v for the reasons stated cases 1 and 2 in the proof of Lemma 2.19. There are at most $\lfloor (|V(G)| - 1)/2 \rfloor$ components and one Kempe-exchange is required for each of them. If $u \nsim_f v$ then there is a single Kempe-exchange on G that establishes $f(u) \text{---} f(v)$. Thus, a shortest path of maximum length t in $\mathcal{K}_p(G')$ corresponds to a path of length at most $t \cdot \lfloor (|V(G)| - 1)/2 \rfloor$ in $\mathcal{K}_p(G)[V_\pi]$.
□

Given two colorings c, c' of G', we can use the algorithm KEMPERECOLOR (see Algorithm 1) to transform c into c' as long as there is a sufficient number of colors available. However, since G' contains K_p as a subgraph it has degeneracy at least p. According to Theorem 2.10, at least $p + 1$ colors are needed by KEMPERECOLOR and thus, we cannot simply use Theorem 2.10 to prove the connectedness of the search space in the presence of availability requirements. To improve the situation, we assume that the vertices $v_1, \ldots, v_p$ of G' have colors $1, \ldots, p$ for all colorings of interest. Under this assumption, we need to determine a vertex ordering such that KEMPERECOLOR requires fewer colors than the naïve approach. The idea is to use KEMPERECOLOR to generate paths in $\mathcal{K}_p(G')$ such that the algorithm never needs to pick alternative colors for the clique vertices.

In the following, let C, $D \subseteq V(G)$ such that $C \cap D = \emptyset$ and $C \cup D = V(G)$. Further, let $S(G)$ be the set of vertex orderings of G and let $S' \subset S(G)$ be the vertex orderings satisfying

$$\forall u, v \in D, w \in C : u < v \wedge u \text{---} v \wedge v \text{---} w \Rightarrow w < v \ . \tag{2.3}$$
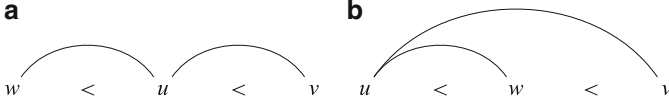
Fig. 2.5 Two vertex orderings of the graph P_3. (**a**) Ordering compatible with Eq. (2.3). (**b**) Ordering incompatible with Eq. (2.3)

That is, if v is a successor of u and they are adjacent, then all neighbors of v in C must precede v. Figure 2.5 shows two examples of vertex orderings of the graph $P_3 = u \,\text{---}\, v \,\text{---}\, w$. For $D = \{u, v\}$, $C = \{w\}$, ordering Fig. 2.5a satisfies the condition in Eq. (2.3) and Fig. 2.5b does not.

For $\sigma \in S$, $v \in V(G)$, let $\text{pred}(v, \sigma)$ be the number of predecessors of v adjacent to v. Let the *subdegeneracy* $\lambda(C, G)$ of G relative to C be

$$\lambda(C, G) = \min_{\sigma \in S'} \max_{v \in D} \text{pred}(v, \sigma) \ .$$

Note that $\lambda(C, G) = \deg(G)$ if C is empty. If C is not empty then $\lambda(C, G) \leq \deg(G)$.

Theorem 2.22 *Let c, c' be k-colorings of G that agree on C. KEMPERECOLOR transforms c into c' such that all intermediate colorings also agree on C, if the vertices are processed according to $\sigma \in S'$ and $k \geq \lambda(C, G) + 1$.*

Proof We first show that the colors of the vertices C are not changed by KEMPE-RECOLOR. Assume for a contradiction that in some intermediate coloring a vertex $w \in C$ has a color different from $c(w)$. Then w has been recolored because a neighbor u of w preceding it in σ received color $c(w)$. There are two possible reasons: Either u was recolored to $c(w)$ because $c'(u) = c(w)$, but then $c'(w) \neq c(w)$, a contradiction. If this is not the case, then u was recolored in case 1 or 2 of KEMPERECOLOR, because of a neighbor v preceding it. But this is a contradiction to $\sigma \in S'$.

We now show that $\lambda(C, G) + 1$ colors are sufficient. Since the vertices in C are never recolored, we consider only the vertices D. An unused color may be picked for a vertex $v \in D$ in case 2 of Algorithm 1. There are at most $\lambda(C, G)$ neighbors of v preceding it, and there are at most $\lambda(C, G) - 1$ colors different from the color of v present among these vertices. Thus, there is at least one other color available for v. □

We propose a heuristic approach to finding a witness vertex ordering of $\lambda(C, G)$. Let $\tilde{S} \subset S'$ be the vertex orderings such that the vertices C precede all other vertices. Recall that for any graph G a witness vertex ordering of the degeneracy $\deg(G)$ can be found by repeatedly removing vertices of minimal degree [Mat68, SW68]. In a similar fashion, we can determine an ordering of the vertices D that minimizes $\max_{v \in D} \text{pred}(v, \sigma)$ over all vertex orderings $\sigma \in \tilde{S}$.

Algorithm 3: VERTEXELIMINATION

---

input : graph G, vertices $C \subseteq V(G)$
output: ordering $v_1, \ldots, v_{|D|}$ of the vertices $D = V(G) \setminus C$

$G_{|D|} \longleftarrow G$
for $i \longleftarrow |D|$ **downto** 1:
$\quad$ choose v_i from $\mathrm{argmin}_{v \in D}\{\delta(v, G_i)\}$
$\quad$ $G_{i-1} \longleftarrow G_i - v_i$.
return $v_1, \ldots, v_{|D|}$

Theorem 2.23 *The output of* VERTEXELIMINATION *is a vertex ordering which minimizes* $\max_{v \in D} \mathrm{pred}(v, \sigma)$ *over all vertex orderings* $\sigma \in \tilde{S}$.

Proof The proof is based on the remark on the optimality of the vertex elimination algorithm in [Mat68]. Let $\ell = |D|$ and for an ordering $v_1, \ldots, v_\ell$ of D let $G_i = G[C \cup \{v_1, \ldots, v_\ell\}]$. Further, let

$$\hat{\delta} := \max_{G[C] \subseteq H \subseteq G} \min_{v \in V(H) \setminus C} \{d(v, H)\} .$$

Intuitively, $\hat{\delta}$ is analogous to the degeneracy of G, but the vertices C are irrelevant. If an ordering $\sigma = v_1, \ldots, v_\ell$ of D is an output of VERTEXELIMINATION then

$$\max_{1 \leq i \leq \ell} \mathrm{pred}(v_i, \sigma) = \max_{1 \leq i \leq \ell} \{d(v_i, G_i)\}$$

$$= \max_{1 \leq i \leq \ell} \min_{v \in V(G_i) \setminus C} \{d(v, G_i)\} \leq \hat{\delta} .$$

The graphs G_i coincide with those in Algorithm 3.

Now let H^* be a graph such that $G[C] \subseteq H^* \subseteq G$ and

$$\min_{v \in V(H^*) \setminus C} \{d(v, H)\} = \hat{\delta} .$$

Let $v_1, \ldots, v_\ell$ be any ordering of D and let i be the smallest index such that $H^* \subseteq G_i$. Then v_i must be a vertex of H^* and $d(v_i, G_i) \geq \hat{\delta}$. Therefore, for any ordering $v_1, \ldots, v_\ell$ of D, $\max_{1 \leq j \leq \ell}\{d(v_j, G_j)\} \geq \hat{\delta}$, with equality if the vertex ordering is an output of VERTEXELIMINATION. $\qquad\square$

As a potential improvement, the following post-processing step can be performed: Let $v_1, \ldots, v_{|D|}$ be an output of VERTEXELIMINATION and let k be the largest number such that $v_1, \ldots, v_k$ are independent. Then the vertices $v_1, \ldots, v_k$ can be moved before the vertices C in the ordering without violating condition (2.3). The resulting ordering $\sigma' \in S'$ is not in $\tilde{S}$ and can thus not be generated by VERTEXELIMINATION. There is a potential advantage because the construction guarantees that $\max_{v \in D} \mathrm{pred}(v, \sigma') \leq \max_{v \in D} \mathrm{pred}(v, \sigma)$.

In summary, the heuristic for computing a vertex ordering σ of G such that $\max_{v \in D} \text{pred}(v, \sigma)$ is close to $\lambda(C, G)$ performs the following two steps:

1. Run VERTEXELIMINATION to generate an ordering $v_1, \ldots, v_{|D|}$ of the vertices D.
2. Let $k \in \mathbb{N}$ be the largest number such that $v_1, \ldots, v_k$ are independent in G. Move the vertices $v_1, \ldots, v_k$ before the vertices C in the ordering.

We apply this heuristic to prove the connectedness of the clash-free timetables that satisfy the availability constraint for a number of benchmark instances. First, we use the reduction from list to graph coloring described above to construct from a conflict graph G the graph G', which contains a clique $v_1, \ldots, v_p$. Then we chose $C \subseteq V(G')$ to include every vertex with $p - 1$ neighbors in $\{v_1, \ldots, v_p\}$, that is

$$C = \{v \in V(G') \mid |\Gamma(v) \cap \{v_1, \ldots, v_p\}| = p - 1\} \ .$$

Now we can apply the heuristic to obtain an upper bound $\lambda' \geq \lambda(C, G')$ and check if $p \geq \lambda' + 1$. If this is the case, then Theorem 2.22 implies that the search space is connected.

Results

Table 2.1 shows some results of applying the search space analysis to a number of UCTP benchmarking instances. All instances can be obtained from the SaTT group website, University of Udine [DGS]. The instances comp01,...,comp21 are from the Curriculum-based Course Timetabling (CB-CTT) track of the ITC2007 competition. The instances ITC2_i01,...,ITC2_i24 are from the Post-enrollment Course Timetabling (PE-CTT) track of the same competition. The erlangen instances are from the school of engineering at the University of Erlangen-Nürnberg. The toy instance is a small example instance from the website [DGS]. The table contains for each instance the number $|P|$ of timeslots, the degeneracy $\deg(G)$ of the conflict graph G, and an upper bound $\lambda'(C, G')$ on $\lambda(C, G')$ produced by the heuristic, where G' is the graph obtained from G by the list coloring reduction and C is set as described above. Entries marked in bold face indicate the connectedness of the corresponding reconfiguration graphs, which is established by Theorems 2.10 (no availability requirements) and 2.22 (with availability requirements).

By Theorem 2.10, reconfiguration graphs of clash-free timetables are connected if $p > \deg(G)$ and by Theorem 2.22, the reconfiguration graphs of clash-free timetables satisfying availability requirements are connected if $p > \lambda(C, G')$ for a suitably chosen $C \subseteq V(G')$. The corresponding values of $\deg(G)$ and $\lambda(C, G')$ in Table 2.1 are marked in bold face in these cases. The data indicates that the clash-free timetables for all CB-CTT and erlangen instances are connected, while the requirements of Theorem 2.10 are not satisfied for any of the PE-CTT instances. For eight CB-CTT instances, the upper bound on $\lambda(C, G')$ is sufficient to show that the reconfiguration graphs are connected in the presence of availability

Table 2.1 The number of timeslots, deg(G) and upper bounds $\lambda'(C, G') \geq \lambda(C, G')$ produced by the heuristic are given for each instance

Instance	$|P|$	deg(G)	$\lambda'(C, G')$	Instance	$P|$	deg(G)	$\lambda'(C, G')$
comp01	30	**23**	**24**	ITC2_i01	45	91	109
comp02	25	**23**	30	ITC2_i02	45	99	119
comp03	25	**22**	27	ITC2_i03	45	73	92
comp04	25	**17**	25	ITC2_i04	45	78	100
comp05	36	**26**	43	ITC2_i05	45	81	99
comp06	25	**17**	28	ITC2_i06	45	80	100
comp07	25	**20**	**24**	ITC2_i07	45	80	106
comp08	25	**20**	**24**	ITC2_i08	45	69	97
comp09	25	**22**	25	ITC2_i09	45	89	108
comp10	25	**18**	27	ITC2_i10	45	97	116
comp11	45	**27**	**27**	ITC2_i11	45	75	93
comp12	36	**22**	40	ITC2_i12	45	91	109
comp13	25	**17**	**22**	ITC2_i13	45	87	106
comp14	25	**17**	**23**	ITC2_i14	45	87	107
comp15	25	**22**	27	ITC2_i15	45	79	106
comp16	25	**18**	25	ITC2_i16	45	55	83
comp17	25	**17**	25	ITC2_i17	45	50	71
comp18	36	**14**	**32**	ITC2_i18	45	91	112
comp19	25	**23**	27	ITC2_i19	45	101	120
comp20	25	**19**	**23**	ITC2_i20	45	73	92
comp21	25	**23**	28	ITC2_i21	45	72	90
erl.2011-2	30	**22**	32	ITC2_i22	45	98	118
erl.2012-1	30	**14**	31	ITC2_i23	45	117	128
erl.2012-2	30	**20**	32	ITC2_i24	45	77	97
erl.2013-1	30	**16**	30	toy	20	**10**	**11**

For each instance, G' is obtained from the conflict graph G by the reduction from graph to list coloring. The instances are the CB-CTT, PE-CTT and Erlangen benchmarking instances from [DGS]. Entries marked in bold face indicate the connectedness of the corresponding reconfiguration graphs

constraints. The situation is quite different for the PE-CTT instances, since neither deg(G) nor $\lambda'(C, G')$ is sufficient to show the connectedness of the reconfiguration graphs. Getting better bounds on $\lambda(C, G')$ is of no use since $\lambda(C, G') \geq$ deg(G). Thus, a new approach for proving the connectedness or disconnectedness of the reconfiguration graphs is likely to be required for these instances. In Appendix A.1, the degeneracy values of the conflict graphs are documented for other instance sets without timeslot availability requirements.

We will show that for the instance toy, the proposed heuristic yields a vertex ordering that is a witness for the value $\lambda(C, G')$. Let G be the conflict graph for this instance and let G' be the graph that results from the reduction from list to graph coloring. In the CB-CTT formulation the events are grouped into courses and
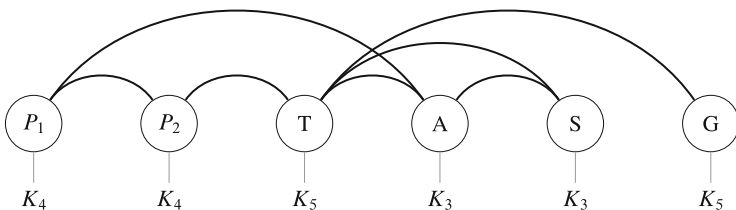
Fig. 2.6 Succinct representation of an optimal vertex ordering of the graph G' obtained from the conflict graph of the instance `toy` by the reduction to graph coloring. All nodes represent cliques as denoted below the nodes

for each course, events of the course are a clique in G and G'. Similarly, if two courses are in conflict, then the events of both courses are a clique in G and G'. If certain timeslots are unavailable for a course, then the events of the course and the timeslots form a clique in G'. In the `toy` instance, there are four courses which consist of 16 events in total. Figure 2.6 shows a succinct representation of an optimal vertex ordering of the graph G'. Each node of the shown graph is a clique, as noted below the nodes, and the cliques are ordered from left to right. Two nodes of the shown graph are connected if all nodes of the corresponding cliques are connected. The nodes T, A, S and G correspond to the courses labeled SceCosC, ArcTec, TecCos and Geotec, respectively. The node P_1 represents to the timeslots marked unavailable for course ArcTec and the node P_2 represents the timeslots unavailable for SceCosC. Any two conflicting courses are connected. Let $C = V(P_1) \cup V(P_2)$.

Let σ be a vertex ordering of G' such that the cliques are arranged in the order P_1, P_2, T, A, S, G with some arbitrary choice of the relative ordering of the vertices within each clique. This ordering is a possible output of the algorithm VERTEX-ELIMINATION. From

$$\max_{v \in V(G') \setminus C} \text{pred}(v, \sigma) = 11 \ ,$$

we can conclude that $\lambda(C, G') \leq 11$.

Proposition 2.24 *For the instance* `toy`, $\lambda(C, G') = 11$.

Proof Let σ be an ordering of $V(G')$ and $V' \subseteq V(G') \setminus C$. The maximum number of predecessors adjacent to any vertex of V' in G' is denoted by

$$p(V', \sigma) = \max_{v \in V'} \text{pred}(v, \sigma) \ .$$

Note that for a clique $K \in \{T, A, S, G\}$, the value $p(K, \sigma)$ is determined by the last vertex of K in σ. Thus, the value of $p(K, \sigma)$ depends only on the relative order of the last vertices of the cliques $\{T, A, S, G\}$ in σ. Let $\tilde{S}$ be the vertex orderings of G' such the vertices C precede all other vertices of G' and let $\hat{S}$ be the total orderings of $\{T, A, S, G\}$. For each ordering $\sigma' \in \hat{S}$ we can pick an ordering $\ell(\sigma')$ of G' that

Table 2.2 Enumeration of the maximum number $p(V, \sigma)$ of adjacent predecessors of the vertices V with respect to the ordering σ for all relevant cases in the proof of Proposition 2.24

Clique ordering $\sigma' \in \hat{S}$	$p(T, \ell(\sigma'))$	$p(A, \ell(\sigma'))$	$p(S, \ell(\sigma'))$	$p(G, \ell(\sigma'))$
T, A, S, G	8	11	10	9
T, S, A, G	8	14	7	9
A, T, S, G	11	6	10	9
S, T, A, G	11	11	2	9
A, S, T, G	14	6	5	9
S, A, T, G	14	9	2	9

is compatible with σ' in the sense that the relative ordering of the last vertices of the cliques is in accordance with σ'. We have,

$$\lambda(C, G') = \min_{\sigma \in \tilde{S}} \max_{K \in \{T,A,S,G\}} p(K, \sigma) = \min_{\sigma' \in \hat{S}} \max_{K \in \{T,A,S,G\}} p(K, \ell(\sigma')) \ .$$

For any ordering $\sigma' \in \hat{S}$ such that $G < T$, we have $p(T, \ell(\sigma')) \geq 13$, because the last vertex of T has at least 13 adjacent predecessors in G'. Thus, we only need to consider orderings such that $G > T$. Furthermore, since no vertex of G is adjacent to any vertex of A or S, changing the relative order of A and G or S and G does not change the number of adjacent predecessors. Hence, we can assume G is a maximum in any ordering of interest and therefore only the six cases remain, which we deal with by enumeration. In Table 2.2 we give the values of $p(K, \ell(\sigma'))$ all for $K \in \{T, A, S, G\}$ and all permutations of $\{T, A, S\}$. From these values we get

$$\lambda(C, G') = \min_{\sigma' \in \hat{S}} \max_{K \in \{T,A,S,G\}} p(K, \ell(\sigma')) = 11 \ ,$$

which concludes the proof.   □

From Proposition 2.24 we can conclude that the proposed heuristic produces a witness of $\lambda(C, G') = 11$ on the instance toy.

2.4 Solution Approaches

In this section we will classify and review algorithms for solving the UCTP search problem. A solution to the search problem is a by-product of solving the optimization problem, so this is in itself not a restriction. However, we are particularly interested in cases in which the search problem itself is challenging and thus receives special attention. After reviewing different approaches from the literature, we will discuss the Kempe Insertion Heuristic (KIH) proposed in [MW10b*]. Then, we will give a SAT formulation of the UCTP, which we will use to provide a point

of reference in our experimental evaluation of the KIH in addition to the results generated by the heuristic state-of-the-art approaches.

Many of the heuristic algorithms for the UCTP take at some point a constructive approach, that is, the events are inserted into the timetable one-by-one such that clash and overlap freeness as well as other requirements are maintained. Thus, at some point during the construction, a subset $E' \subseteq E$ has been assigned to resources while the remaining events are yet unassigned. In these cases, the timetable $\tau : E \rightarrow P \times R$ is a partial function and to make clear the distinction we call it a *partial timetable*. We refer to the number of unscheduled events in a partial timetable as the *distance to feasibility*. The distance to feasibility of a feasible timetable is zero.

2.4.1 Existing Approaches

In this section, we provide an overview of the approaches in the literature targeted at solving the search version of the UCTP. Many techniques such as the sequential heuristics have been studied extensively in the context of examination timetabling [CLL96,QBM+09]. Due to the similar structure of the UCTP and examination timetabling problems, the techniques developed for examination timetabling problems generally carry over to the UCTP and vice versa. Lewis and Paechter argued that finding feasible timetables is too easy on existing benchmark sets, because they were intended not as search problems but as optimization problems with a focus on minimizing soft constraint violations [LP07]. Thus, they proposed a set of 60 artificially generated instances which are challenging with respect to finding feasible timetables. We will restrict our literature review to four categories of approaches that are relevant for this instance set, namely sequential heuristics, Local Search (LS), population-based heuristics, and hyper-heuristics. For a complete overview of the vast body of literature on algorithms for the UCTP, please refer to the very good surveys [CL96, Sch99, BdWK03, QBM+09].

Sequential Heuristics

Sequential heuristics assign events to resources one by one and are in essence adaptations of the greedy coloring algorithm to the UCTP. The algorithmic scheme works as follows: The input is a UCTP instance, and there are two policies: policy P_1 that picks the next event to be scheduled, and a policy P_2 that selects a resource for a particular event. In each iteration, the sequential heuristic picks an event e according to P_1 and assigns an available resource to e according to P_2. If P_1 depends on the events that have already been scheduled, the sequential heuristic is called *dynamic*. Otherwise, the sequential heuristic is called *static*, and the order in which events are scheduled is determined a priori. The policy P_2 typically chooses the first suitable resource from a sequence of candidate resources. For instance, in [TBM07, MW10b*], the resources are arranged in random order and

the first suitable resource is chosen. We will refer to this resource selection policy as Random Ordering (RO) policy. Additional constraints may be considered by P_2 when choosing one of several available resources [CLL96]. If at some point there are no available resources for the current event, it is left unassigned and the resulting timetable will not be feasible. Choices of P_1 from the literature [CL96, BdWK03] include:

- *Largest Degree (LD) (static)*: The events are arranged in non-increasing order according to their degrees in the conflict graph, see [WP67]. The next event is selected according to this ordering.
- *Largest Weighted Degree (LWD) (static)*: For each event, the total number of students involved in its conflicts is determined and the events are arranged in non-increasing order according to these values. The next event is selected according to this ordering.
- *Random (static)*: The next event is picked at random from the remaining events.
- *Least Saturation Degree (LSD) (dynamic)*: An event with the least number of available resources is selected as the next event to be scheduled.
- *Largest Color Degree (LCD) (dynamic)*: An event with the largest number of conflicts in the current timetable is selected as the next event to be scheduled.

Since sequential heuristics are computationally cheap, they can provide a starting point for computationally more involved heuristics [MW10b*, TBM07].

Local Search-Based Heuristics

Local Search (LS) is a class of heuristics for solving combinatorial search and optimization problems. Popular classes of LS-based heuristics are

- Simulated Annealing (SA) [KGV83],
- Tabu Search (TS) [GL97],
- Hill Climbing (HC) [BB12], and
- Great Deluge (GD) [Due93].

The general scheme fits quite well in the formal framework of reconfiguration problems from Sect. 2.3.1. Given some adjacency relation on the set of solutions and an initial solution, the search space is traversed by iteratively moving to neighboring solutions. LS-based heuristics differ with respect to the adjacency relation and the policy that determines under which conditions the current solution is replaced by a neighbor, and which of the neighbors is chosen to replace the current solution. Several adjacency relations may be used individually or in combination by a LS-heuristic [DGS06].

Virtually all neighborhood structures that appear in LS-based algorithms for the UCTP and related problems are based on the Kempe-exchange, which is a popular operation used in various solvers for timetabling problems over the last two decades [TD98, MBHS02, LH10, TBM07]. We will present a unified view on the neighborhood structures in terms of which subsets of Kempe-exchanges are

considered. In addition to choosing which Kempe-exchanges may be performed to generate a neighborhood, there are a variety of options how to deal with availability requirements and the room assignment when moving events between timeslots.

- Move Event Neighborhood (ME): For a given event e the neighborhood contains all timetables that result from moving e to a different unoccupied resource. Moving an event e in this is equivalent to a Kempe-exchange that operates on the Kempe-component $(\{e\}, \emptyset)$.
- Swap Event Neighborhood (SE): Contains all timetables that result from swapping the timeslots of exactly two events. A swap of two events e_1 and e_2 is equivalent to a single Kempe-exchange that operates on the Kempe-component $(\{e_1, e_2\}, \{e_1 - e_2\})$ if the two events are conflicting, or two consecutive steps in the ME neighborhood otherwise.
- Kempe-exchange Neighborhood (KX): For a given event e, the neighborhood contains all timetables that result from applying a Kempe-exchange such that e is contained in the Kempe-component. This neighborhood has been used in the context of examination timetabling [MBHS02, BEM^{+}10].

Typically, only such neighbors are considered that respect overlap-freeness and availability requirements [TBM07, CDGS12, MBHS02, BEM^{+}10]. Generating neighbors in the KX neighborhood while respecting room availability requirements is computationally expensive in comparison to the simpler ME or SE neighborhoods: the former requires solving two room assignment subproblems, i.e. two maximum cardinality matchings, to determine a feasible room assignment for all events.

We will discuss two local search approaches that are considered state-of-the art in more detail. The general idea that is common to both is that the number of events that cannot be scheduled in the timetable without violating overlap freeness and availability requirements is minimized. In both approaches, there seemed to be some gain in minimizing the number of students attending the events that were not properly scheduled instead of just counting the number of such events.

Hybrid Simulated Annealing [TBM07] The hybrid simulated annealing algorithm uses three neighborhood structures, ME, SE, and a neighborhood based on a special variant of the Kempe-exchange, called the Pair-wise Kempe Chain (PKC) neighborhood. A move to a neighbor solution in the SE neighborhood exchanges the resource assignments of two events. The pair-wise Kempe-exchange works as follows: Let G be the conflict graph of a UCTP instance, τ be a partial timetable and p_1 and p_2 be two distinct timeslots. Add an edge between two nodes of G whenever the corresponding events are assigned to the same room according to τ. The pairwise Kempe-exchange exchanges the resources assigned to any two events in a Kempe-component of $G(p_1, p_2)$ that are scheduled in the same room. Any pairwise Kempe-exchange operates on a union of ordinary Kempe-components and is therefore equivalent to performing one or more Kempe-exchanges. However, the benefit of the pair-wise Kempe-exchange is that assigning rooms for all events in a

Kempe-component is straight-forward due to pairing events in the same room in the two timeslots.

In a preprocessing step, the algorithm uses a variant of the sequential heuristics: In each step, the heuristic chooses the next event to be scheduled according to the least saturation degree and ties are broken according to the largest degree. Each event is assigned to a random suitable resource. Events that have no suitable resources left are accommodated in artificial timeslots, such that conflict and overlap requirements are satisfied. The whole preprocessing step consists of 500 trials of this sequential heuristic. After the preprocessing, the number of artificial timeslots is minimized using an SA-based approach. For each temperature level, the algorithm first searches the neighborhood ME followed by the neighborhood SE, accepting neighbors according to the usual probabilistic SA acceptance criterion, see [KGV83]. If no improvement has been made for a fixed number of trials in these neighborhoods, the algorithm switches to the PKC neighborhood. Again, if no improvement has been made within a fixed number of iterations, the algorithm switches back to the other two neighborhoods after adjusting the temperature. Several steps in the PKC neighborhood may be performed before the resulting timetable is evaluated and compared to the prior timetable. The cooling schedule decreases the temperature by the factor $T/(1 + T\beta)$ when the temperature is adjusted, where β is an algorithm parameter that is picked from the interval $[0.0005, 0.001]$.

Tuned Simulated Annealing [CDGS12] This approach includes the ME and SE neighborhoods in a fairly standard SA setting that is tuned to the UCTP search problem (and several related optimization problems) using statistical methods. Room and timeslot availability requirements as well as overlap-freeness requirements are satisfied by design. In contrast to the hybrid simulated annealing proposed in [TBM07], conflicts are tolerated and minimizing the number of event conflicts is part of the objective. The room assignment possibilities in the SE neighborhood are more relaxed in comparison to the SE neighborhood in [TBM07]: whenever two events in different timeslots are swapped, any available, unoccupied in the respective target timeslot can be assigned. Similar to [TBM07], there is an artificial timeslot that accommodates the events not yet scheduled. There is an additional restricted neighborhood, ME^-, that prohibits moving events to the artificial timeslot.

Some constraint propagation is performed in a preprocessing step. For instance if two events can be scheduled only is the same single room they cannot be scheduled in the same timeslot and are thus marked as conflicting. After this step, one of two possible construction procedures is performed to insert events into the timetable. The first procedure, referred to as I_0 picks for each event e a random timeslot and schedules the event in this timeslot if an unoccupied room is available. If none is available, the event is scheduled in the artificial timeslot. The second procedure, I_1 tries for each event to find an unoccupied room in a number of available timeslots. After the constructive procedure, the simulated annealing is performed. The cooling scheduled is geometric, i.e., in each temperature update, the temperature is multiplied by a decay factor $\beta \in (0, 1)$. The objective is to minimize

the sum of the event conflicts and the number of students in the events scheduled in the artificial timeslot.

In order to tune the general algorithmic scheme to the instances at hand, three different combinations of neighborhoods and constructive procedures were evaluated and for these three, the parameter space was sampled using Nearly Orthogonal Latin Hypercubes (NOLH) [CL07]. The best configurations were then determined by the F-race algorithm [BYBS10, Bir], see also section "Preliminaries: Algorithm Configuration". The tuning process was performed separately for the various sets of instances, since the problem formulations differ between different instance sets. For a comparison with the KIH from Sect. 2.4.2, the most relevant instances are those generated by Lewis and Paechter [LP07, LPb]. The best configuration on these instances was found to a combination of initialization I_0 followed by the SA procedure using the union of the neighborhoods ME and SE. The initial temperature was set to 31.62 on the medium instances and set slightly higher, to 36.30, on the big instances. The minimum temperature was determined to be approximately 0.12 for both instance sets. The number of neighborhood searches per iteration was configures such that the minimum temperature was reached after a given number of iterations with $\beta = 0.9999$.

Population-Based Heuristics

Population-based heuristics include Genetic Algorithms (GA), Evolutionary Algorithm (EA), Particle Swarm Optimization (PSO), and other nature-inspired optimization techniques [BFM97, Ken10, DS04]. In contrast to LS-based methods, a population of candidate solutions is maintained, which are modified iteratively to identify and explore promising regions of the search space. A key difference to other approaches is that there is some mechanism for exchanging information between different candidate solutions in the population. A brief overview over the EA is given in Sect. A.2.

Grouping Genetic Algorithm (GGA) [LP07] Lewis and Paechter apply a GGA to the UCTP. Following the work of Falkenauer [Fal94], their motivation is that traditional solution representations in genetic algorithms include too much redundancy and thus the traditional mutation and crossover operators do not perform well. A grouping problem captures the task of gathering groups of objects such that a certain objective function is minimized. Among other combinatorial problems, the UCTP can be described as a grouping problem: events need to be grouped in timeslots such that clash and overlap freeness requirements are satisfied. Similar to the local search approach in [TBM07], artificial timeslot are introduced to accommodate events that cannot be scheduled in any of the regular timeslots without violating conflict or overlap freeness requirements.

The initial population is generated using a constructive mechanism that combines the saturation degree, largest degree and random sequential heuristics with three different policies for choosing rooms. Artificial timeslots are added to a partial

solution as needed. The mutation operator deletes groups (timeslots) from a candidate timetable and inserts all events that were assigned to these timeslots using the aforementioned constructive mechanism. The crossover operator follows the approach for grouping problems proposed by Falkenauer [Fal94, Fal98]. Given two parents s_1 and s_2 an offspring is generated by the crossover operator in the following way: A set of consecutive group is chosen from s_1 and is injected after a randomly chosen groups in s_2. The standard approach [Fal94] suggests to remove events that occur twice from the groups they were part of originally in s_2. Lewis and Paechter argue that this does not work well for the UCTP since the number of timeslots is likely to be increased by the crossover operation in comparison to the parents and suggest to remove all groups of s_2 that contain events injected by the crossover. The events that were removed from the timetable altogether are then inserted using the constructive mechanism in the same way as the mutation operator. In addition, a local search procedure is added in order to improve the overall performance of the GGA. During the local search, two tasks are performed: First, for each of the unoccupied resources, the algorithm checks if any of the unplaced events can be assigned to them. If this is the case, the event is assigned to the resource. In the second step, a number of moves are performed in the ME neighborhood.

The impact of several objective functions on the performance of the GGA (with and without local search) has been investigated in [LP07]. The overall results shown on the website [LPa] were generated using the objective function

$$f(s) = \frac{\sum_{i=1}^{g} C_i^2}{g} \quad ,$$

proposed by Erben in [Erb01], where s is a timetable in the GGA representation containing groups $1, \ldots, g$ of events and C_i is the number of conflicts of the events in group i with other events in the timetable. This objective function is to be maximized and rewards grouping events of high degree.

Hyper Heuristics

A hyper heuristic solves search and optimization problems by selecting heuristics from a pool of *low-level heuristics* applicable to a concrete problem domain. The aim of a hyper heuristic is maximal generality in the sense that it can work on any problem domain as long as it is provided with a set of suitable low-level heuristics, which encapsulate all problem-dependent knowledge. A crucial component of a hyper heuristic approach is the *high-level heuristic* which identifies the most successful low-level heuristics for a given problem either during the optimization process or offline [BLSS05].

Hybrid Graph-Based Hyper Heuristic (HGHH) [BMM+07] The HGHH uses TS to find a schedule of the sequential heuristics discussed above. The schedule is a list of sequential heuristics that determines which heuristic is applied to insert an

event in the timetable. The neighborhood consists of all heuristic schedules that can be generated from the current schedule by exchanging two heuristics. In addition to the tabu list the algorithm keeps track of prefixes of schedules that produce infeasible timetables. If a new schedule matches any of the prefixes it is discarded immediately. Furthermore, each entry in the heuristic schedule is responsible for inserting two events in the timetable, which considerably reduces the size of the search space. For various Examination Timetabling (ETT) instances, Burke et al. report an improvement over using the sequential heuristics individually, even if additional backtracking mechanisms are employed, as in [CL96]. Hybridisation schemes that combine various alternative local search variants as high-level heuristics with the sequential low-level heuristics have been investigated in [QB08]. The results indicate that Iterated Local Search (ILS) and Variable Neighborhood Search (VNS) are to be preferred over TS.

Other Approaches

There are certainly approaches that do not clearly follow any of the schemes discussed above, for example Ant Colony Optimization (ACO) [NMCR12]. The following rather recent approach is roughly based on the construction versus perturbation dichotomy, similar to the Kempe Insertion Heuristic discussed in Sect. 2.4.2:

Clique-Based Construction [LZC11] At the heart of the clique-based, or rather independent set-based, algorithm lies the task of finding large independent sets in the conflict graph G of a UCTP instance. Liu, Zhang, and Chin rely on a heuristic for maximum clique problem proposed in [Öst02], and apply the algorithm on the complement $\overline{G}$ of the conflict graph. For performance reasons, the backtracking part of the heuristic is left out and what remains is the following procedure: Let $c \subseteq V(\overline{G})$ be a clique, possibly $c = \emptyset$. Let $U = V(G)$ if c is empty and $U = N_{\overline{G}}(c)$ otherwise. While U is not empty, add a vertex v of maximum degree in $\overline{G}$ to c and replace U by $U \cap N(v)$. The algorithm terminates when U is empty and at this point c contains a clique that is potentially larger than the original clique. Dually, this algorithm extends a given independent set c in G to a maximal independent set.

The algorithm proceeds in three steps, the initialization phase, the recombination phase and the perturbation phase. After the initialization, the algorithm alternates between recombination and perturbation until a feasible timetable has been found or the timeout has been hit. During the initialization, for each timeslot a maximal independent set is generated, then the room assignment subproblem is solved, see Sect. 2.2. In each iteration of the recombination phase, each event in the timetable is removed with probability ρ, a parameter which deteriorates exponentially over time. Then, for each timeslot p, the maximum independent set heuristic above is applied to extend the current independent events in p to a maximal independent set of events. For a timeslot p, the new independent set is accepted only if it is an improvement in terms of the (i) number of events scheduled in p (ii) number

of students in the events scheduled in p (iii) sum of the degrees of the events in p. The rationale behind (iii) is that if the sum of the degrees of the events increases, then more of the "difficult" events could be placed in p, which is desirable because no effort has to be spent on placing these events in other timeslots. The recombination phase terminates whenever ρ is below a given minimum probability or a feasible timetable has been found. The perturbation phase exchanges events between different timeslots. The exchange of events is similar to the Kempe-exchange. The main difference is that only a subset of a Kempe-component is processed. Let e be an event that should be moved from timeslot p_1 to p_2. When e is moved to p_2 all events in p_2 conflicting with e are simultaneously moved to p_1. If any of the latter events are in conflict with any event in p_1 then the move is not performed. Similarly, if there is no feasible room assignment after swapping the events, the move is also not performed. The perturbation phase is finished after a fixed number of iterations.

2.4.2 The Kempe Insertion Heuristic

It can be readily observed that the purely constructive approaches, which insert events in the timetable one-by-one while respecting clash and overlap freeness, are rather unsuccessful on many of the UCTP and ETT standard instances. Typically, at some point during the construction of the timetable there are no suitable resources left for a set of events, which are therefore left "unscheduled". In order to avoid such a situation, backtracking [CLL96] or look-ahead [BN99, BMM$^+$07] strategies have been applied. More recent state-of-the-art approaches relax the requirements in some way, for example, by allowing unscheduled events [Kos05, LZC11, MW10b*], adding extra timeslots [TBM07, CDGS12], or tolerating event conflicts [CDGS12]. The goal is then to minimize the violations of the original requirements. The algorithm KEMPEINSERTION proposed in [MW10b*] relaxes the requirement that all events need to be scheduled, similar to the approaches [Kos05, LZC11]. In order to create opportunity for inserting additional events and ultimately find a feasible timetable, KEMPEINSERTION iteratively modifies partial timetables using Kempe-exchanges. In comparison to the other approaches, there is a significant difference with respect to how opportunity to insert events is generated.

The algorithm KEMPEINSERTION is composed of a preprocessing phase followed by a construction and a perturbation. The latter two are performed alternatingly until a feasible timetable has been found or the time limit has been reached. The purpose of the preprocessing is to find a good starting point for the rest of the KIH with a reasonable computational effort. This is accomplished by a variant of the sequential heuristics discussed in Sect. 2.4.1. During the construction phase, additional events are inserted in the timetable by clearing timeslots from events conflicting with the yet unscheduled events. In contrast to other approaches, we use Kempe-exchanges to deliberately move these conflicting events to other timeslots. Similar to the sequential heuristics, the constructive part of the KIH

may get stuck and thus, a perturbation mechanism is needed in order to generate feasible timetables on many benchmark instances. The perturbation inserts yet unscheduled events in the timetable and removes other events from the timetable to ensure that conflict and overlap requirements are respected. A high level view of KEMPEINSERTION is shown in Algorithm 5. We will first discuss the construction and the perturbation phase individually and then show how construction and perturbation were balanced for good performance on the benchmark instances.

Preprocessing

The preprocessing runs a sequential heuristic for a fixed number of iterations and keeps a timetable with the lowest distance to feasibility. Multiple runs of the sequential heuristic are performed in order to compensate for the variance in the distance to feasibility of the generated timetables. The behavior of a sequential heuristic is determined by the event selection policy and the resource selection policy, see Sect. 2.4.1. In [MW10b*], we used as sequential heuristic the one that was advocated in [TBM07] as preprocessing for the Hybrid Simulated Annealing algorithm: The next event is selected according to the LSD policy and ties are broken by the LD policy (see p. 40). We will refer to this event selection policy as LSDLD. Suitable resources are selected at random (policy RO). In our experimental evaluation presented in Sect. 2.5, we will investigate the distance to feasibility attained by different sequential heuristics including the LSDLD/RO heuristic on a number of benchmark instances. As a possible improvement to the RO policy we propose the following resource ordering policy, which we refer to as Saturation Ordering (SO). The SO selection policy arranges the timeslots that do not contain events conflicting with e in non-decreasing order according to how many of the remaining events can be scheduled in the timeslot without violating clash or overlap freeness requirements. Then, for each timeslot p in the given order, the room assignment is performed for all events already in p and e. If there is a room assignment for these events, then e can be inserted in the timetable. Please note that rooms may be re-assigned for the other events in timeslot p. If e cannot be inserted in any of the suitable timeslots in this fashion it is left unassigned.

The Construction Phase

Consider inserting events into a partial timetable using a sequential heuristic. At some point the sequential heuristic may not be able to insert the next event e into the timetable, because for each timeslot p suitable for e, at least one of the following obstructions is present:

1. a conflicting event e' of e is in timeslot p.
2. there is no feasible room assignment for the events in timeslot p when e is scheduled in p.

Instead of using backtracking or lookahead techniques as a remedy, our key observation is that the event e can still be inserted in the timetable, if we manage to clear the obstructions for any of the suitable timeslots by using Kempe-exchanges. For this purpose we consider the following problem:

Definition 2.25 (Event Insertion Problem (EIP))
INSTANCE: A UCTP instance $\mathcal{I}$, a partial timetable τ, an event e and a timeslot p.
TASK: Find a sequence of Kempe-exchanges that clears obstructions 1 and 2 for e and timeslot p.

Clearly, if we manage to solve a suitable EIP instance, we can insert an additional event into the timetable and are one step closer to a feasible solution.

We will first have a look at this problem from the reconfiguration problem perspective. Let $G[\tau]$ denote the subgraph of the conflict graph induced by the events in the partial timetable τ. We are looking for a path in $\mathcal{K}_p(G[\tau])$ from the coloring given by τ to a coloring that does not exhibit obstructions 1 and 2. We cannot assume in general that such a path exists: Let $\ell \geq 3$, $k = \ell + 1$ and let $G \times H$ denote the categorial product of two graphs G and H. We can pick a UCTP instance $\mathcal{I}$ and a partial timetable τ for $\mathcal{I}$ such that $G[\tau] = K_\ell \times K_k$. Assume that τ induces a k-coloring of $G[\tau]$. By [Moh07, Proposition 2.1], $G[\tau]$ is ℓ-colorable and the unique ℓ-coloring is not connected to any k-coloring. Thus, it is impossible to clear obstruction 1 for an event that is to be scheduled in timeslot k. To the best of our knowledge, so far no hardness results are known for the problem of deciding the connectedness of Kempe-k-coloring graphs. Due to these considerations, it is reasonable to make only a limited effort to check if the obstructions can be cleared easily.

The procedure EVENTINSERTION is shown in Algorithm 4. It is used repeatedly in the construction phase of KEMPEINSERTION. The loop in line 1 deals with obstruction 1, while loop 2 deals with obstruction 2. If EVENTINSERTION indicates success, then after performing the Kempe-exchanges returned by the algorithm timeslot p does not contain any events conflicting with e and there is at least one unoccupied room in p. The algorithm attempts to clear events conflicting with e by moving all such events to other timeslots using Kempe-exchanges. The alternative timeslots for the conflicting events are tried in random order. However, such Kempe-exchanges can introduce additional conflicting events to timeslot p. To keep things simple and computationally cheap, only such Kempe-exchanges are considered admissible that do not introduce any events conflicting with e. In order to clear obstruction 2, EVENTINSERTION searches for a Kempe-exchange that does not introduce conflicts of e to p and frees up at least one room. In the presence of room availability requirements, EVENTINSERTION can be modified in a straightforward manner such that a room suitable for e is freed up. In total, EVENTINSERTION performs $O(|R| \cdot |P|)$ Kempe-exchanges in order to clear obstructions 1 and 2.

The loop in line 1 of Algorithm 5 contains the constructive part of the KEMPE-INSERTION, which essentially uses EVENTINSERTION to solve the EIP repeatedly for events that are yet to be scheduled. In each iteration, an event e from the list of unscheduled events is picked uniformly at random (uar). Then, the

---

Algorithm 4: EVENTINSERTION

---

input : UCTP instance $\mathcal{I}$, partial timetable τ for $\mathcal{I}$, event e, timeslot p
output: a sequence of Kempe-exchanges, indication of success

$K \longleftarrow$ EmptyList
1 **foreach** *conflict e' of e in p*:
     find timeslot $p' \neq p$ such that Kempe-exchange (e', p, p') does not introduce events
     conflicting with e in p
     if *suitable p' was found*:
        append (e', p, p') to K
     else:
        return K, no success

   if *there is a free room for e in p*:
     return K, success
2 **foreach** *event e' in timeslot p*:
     find timeslot $p' \neq p$ such that Kempe-exchange (e', p, p') decreases number of events
     in p
     if *suitable p' was found*:
        append (e', p, p') to K
        return K, success

   return K, no success

---

timeslots available for e are processed in random order, and for each timeslot p, EVENTINSERTION is employed to solve the EIP instance $(\mathcal{I}, \tau, e, p)$. The Kempe-exchanges returned by EVENTINSERTION are applied to the timetable τ and the room assignment subproblem is solved for the events in timeslot p and e. If room availability requirements are present, it is possible at this point that there is no feasible room assignment, in which case e is not scheduled and we continue with the next timeslot. If the attempt to scheduled e in p was successful and the resulting partial timetable τ has fewer unscheduled events than the current best timetable τ_{best}, the latter is replaced by τ. As shown in line 2 of Algorithm 5, if EVENTINSERTION is unsuccessful for a particular timeslot p, the events conflicting with the event e are collected in p to increase the chances of successfully solving the EIP for other timeslots. This operation basically does the opposite of what EVENTINSERTION does: We try to increase the number of events conflicting with e by performing Kempe-exchanges between p and all other timeslots that are yet to be processed in the current iteration.

The constructive part has been described in [MW10b*] in terms of a neighborhood search that repeatedly samples timetables from a set $\mathcal{N}_{e,p}(\tau)$ of *neighbors*, for a given event e and a timeslot p:

$$\mathcal{N}_{e,p}(\tau) = \{\tau' \mid \tau' \text{ is obtained from } \tau \text{ by solving the EIP } (\mathcal{I}, \tau, e, p)\} \ .$$

In contrast to other local search approaches to the UCTP and related problems that sample a number of solutions from a neighborhood [DGS01, TBM07], the difficulty here is to generate a single neighbor. Also, there is no immediate use in sampling multiple neighbors and taking the best one, as it is common, for example, in the tabu search metaheuristic [Glo89], since all of them have the same distance to feasibility.

The Perturbation Phase

The purpose of the perturbation phase is to introduce some changes to the list of unscheduled events whenever the constructive phase gets stuck. Thus, there will be different unscheduled events in the next construction phase which can potentially be inserted into the timetable. In essence, the perturbation can undo decisions that were taken earlier in the search process by removing events from the timetable. A perturbation operation consists of picking an unscheduled event e uar. Then the suitable timeslots for e are arranged in non-decreasing order by the number of events they contain that are in conflict with e. A timeslot is then picked from this sequence according to random variable that is exponentially distributed with parameter λ. Due to the characteristics of the distribution, it is more likely to pick a timeslot with few events conflicting with e. Once a timeslot has been picked, all events conflicting with e are removed from p and e is scheduled in the timeslot. In [MW10b*], the perturbation mechanism was termed *forced event insertion.* Typically, when KIH has made some progress, several conflicting events are removed from the timetable during the perturbation, so the distance to feasibility is increased. The aim is to use the perturbation just often enough so that the construction phase can insert additional events. In the original work [MW10b*], we used a scheme based on counting the number of successful attempts to insert events into the timetable to balance the usage of the constructive phase and the perturbation phase.

The forced event insertion is similar to the *blow-up* operation from [Kos05]. The two main differences are that the suitable timeslots are picked according to an exponentially distributed variable and that there is no attempt to place additional events in p after scheduling e in p. According to our experiments there is no benefit in trying to place additional events in p in order to fill up the rooms.

Combining Construction and Perturbation

Algorithm 5 gives an overview of the algorithm KEMPEINSERTION as proposed in [MW10b*], with a focus on how the construction and the perturbation are combined. The algorithm terminates when a feasible timetable has been found or, depending on the experimental setup, when a timeout or a maximum number of iterations has been reached. The loop in line 1 contains the constructive part of the algorithm, which essentially uses EVENTINSERTION to solve the EIP for events picked uniformly at random (uar) from the list of unscheduled events. The perturbation phase of KEMPEINSERTION starts in line 3, if at most m of the

---

Algorithm 5: KEMPEINSERTION

---

input : UCTP instance $\mathcal{I}$, integers k, ℓ, m, exponential distribution parameter λ
in/out : partial timetable τ

$\tau_{best} \longleftarrow \tau$
$\mu \longleftarrow$ list of unscheduled events
while *stopping criterion not met*:
     /* construction phase   */
1   **for** $\min\{k, |\mu|\}$ *iterations*:
       pick event e from μ uar
       foreach *timeslot p available for e*:
         solve EIP $(\mathcal{I}, \tau, e, p)$ using EVENTINSERTION
         apply Kempe-exchanges found by EVENTINSERTION to τ
         if EVENTINSERTION *indicated success*:
           schedule e in p, remove e from μ if successful
           update τ_{best} if necessary
           break
2   **else:** gather conflicts of e in p

     /* perturcation phase   */
3   **if** EVENTINSERTION *was successful at most m times*:
       for ℓ *iterations*:
         pick event e from μ uar
         pick available timeslot p for e (involves λ, see text)
         remove all events conflicting with e in p from τ
         schedule e in p
         update μ

return τ_{best}

$\max\{k, |\mu|\}$ attempts to solving the EIP has been successful in the construction phase, where k is an algorithm parameter and μ is a list of unscheduled events.

Clearly, finding a good balance between construction and perturbation is crucial. The parameter k determines how many attempts to inserting events in the timetable are performed before considering the perturbation. Furthermore, the parameter m can be used to put more focus on the perturbation, that is, increasing m makes the perturbation more likely. The two parameters ℓ and λ set the strength of the perturbation. The parameter ℓ sets the number of perturbation operations that are performed. If ℓ is too small, then KEMPEINSERTION may get stuck in local optima and if ℓ is too large then too many events are removed from the timetable, which hinders progress. During the perturbation, the exponential distribution parameter λ determines how many events are likely to be removed from the timetable. In some sense, it determines the strength of a single perturbation operation. In the original work [MW10b*], λ and ℓ were set to 1.0 and 1, respectively. In Sect. 2.5 we will present the results of an automatic tuning of the parameter choices and the resulting impact on the performance of KEMPEINSERTION.

2.4.3   SAT Encoding of the UCTP

In order to motivate the use of certain specialized heuristics for a given problem domain, it is beneficial to compare their performance to that of standard tools such as SAT solvers and Integer Linear Programming (ILP) solvers. In this section, we will provide a formalization of the UCTP problem in terms of a Boolean formula so we can compare the performance of the KIH and other heuristics for the UCTP to that of a SAT solver. Generally, SAT solvers are very good at solving search problems.

We will first briefly introduce the necessary formalisms and then provide a reduction from the UCTP to the SAT problem.

Boolean Satisfiability

The Boolean satisfiability (SAT) problem is a classical NP-complete problem, see e.g. [GJ79, Section 2.6]. Given a Boolean formula φ in Conjunctive Normal Form (CNF), the task is to decide if there is an assignment of the variables to the truth values {TRUE, FALSE} that satisfies φ.

Boolean Formulas   Let $\mathcal{A}$ be a finite set of propositional variables. A *literal* is either a variable $x \in \mathcal{A}$ or its negation $\neg x$. A *clause* is a disjunction ($\vee$) of literals. A Boolean formula in Conjunctive Normal Form (CNF) is a conjunction ($\wedge$) of clauses. An *assignment* of the variables is a mapping $a : \mathcal{A} \rightarrow \{\text{TRUE}, \text{FALSE}\}$. By applying the usual semantics of conjunction, disjunction and negation, an assignment determines the truth value of a CNF formula φ over $\mathcal{A}$: A literal $\ell = x$ is true iff $a(x) = \text{TRUE}$ and $\ell = \neg x$ is true iff $a(x) = \text{FALSE}$. A clause is true, iff at least one of its literals is true and a CNF formula is true iff each clause is true. For convenience, we will denote by $a(\varphi)$ the truth value of φ resulting from the assignment a of the variables. The formula φ is satisfiable if there is an assignment a such that $a(\varphi) = \text{TRUE}$. For any Boolean formula φ constructed from the variables $\mathcal{A}$, parentheses and the logical connectives $\{\wedge, \vee, \neg\}$ a formula φ' in CNF can be given that is logically equivalent. That is, any assignment that satisfies φ also satisfies φ' and vice versa. However, the transformation of φ into φ' may result in an exponential blowup of the number of connectives. We represent a CNF formula as a set of its clauses, so we can create a CNF $\varphi \wedge \varphi'$ from two CNF formulas φ and φ' over $\mathcal{A}$ by taking the set union $\varphi \cup \varphi'$. An assignment satisfies $\varphi \wedge \varphi'$ if and only if it satisfies both φ and φ'.

Unit Propagation   A *unit clause* is a clause that contains a single literal. An inference rule used by many SAT solvers is *unit clause propagation*, or *unit propagation* for short. In particular, unit propagation is the main inference rule of all SAT solvers based on the Davis–Putnam–Logemann–Loveland (DPLL) procedure [DLL62], including, for example, MiniSAT [ES04] and (z)Chaff [MMZ+01]. Let φ be a CNF formula with a unit clause that contains the literal $\ell \in \{x, \neg x\}$, $x \in \mathcal{A}$.

Then the value $a(x)$ is the same for any assignment a that satisfies φ: If $\ell = x$ then $a(x) = $ TRUE since otherwise $a(\varphi) = $ FALSE. Similarly, if $\ell = \neg x$ then necessarily $a(x) = $ FALSE. Now, all clauses that contain literals in $\{x, \neg x\}$ can be simplified by substituting the value of x. This process can be performed repeatedly if new unit clauses appear due to the substitution. The advantage of unit propagation is a SAT solver can use it to efficiently prune the search space.

Cardinality Constraints It is a common requirement in SAT encodings of combinatorial problems that at least, at most, or exactly k variables from a given set $X \subseteq \mathcal{A}$ are true. Such requirements are called *cardinality constraints*. The main challenge is to find economical encodings of these constraints with respect to the number of clauses and auxiliary variables required. For our purpose of modeling the UCTP we need to formulate the requirement that among n variables $x_1, \ldots, x_n \in \mathcal{A}$ exactly one is true, which is denoted by $=_1(x_1, \ldots, x_n)$. We will express this constraint by a conjunction of the constraints $\leq_1(x_1, \ldots, x_n)$ (at most one of the variables is true) and $\geq_1(x_1, \ldots, x_n)$ (at least one of the variables is true). The latter can be realized in a straight-forward fashion by the following clause:

$$\geq_1(x_1, \ldots, x_n) = (x_1 \vee \ldots \vee x_n) \quad .$$

The $\leq_1$ constraint can be modeled by prohibiting for any choice of two variables that both are true:

$$\leq_1(x_1, \ldots, x_n) = \bigwedge_{\substack{i,j \in \{1,\ldots,n\} \\ i > j}} \left(\neg x_i \vee \neg x_j \right) \quad .$$

This CNF encoding of $\leq_1$, referred to as the pairwise encoding, can be improved by various techniques that have been developed for the $\leq_k$ constraint, which is true iff at most k of the given variables are true [Sin05, AANORC09, FG10, Che10]. However, for the SAT encoding of the UCTP we use the pairwise encoding given above with good results.

Reduction of UCTP to SAT

There are many different ways to model the UCTP as a CNF. The differences are manifest mainly in the number of variables, the number of clauses and the lengths of the clauses required to describe the problem, and certainly how well a SAT solver can handle the resulting instances. Depending on the problem encoding, SAT solvers may show quite different performance. See for example [VG08] for an investigation of the impact of different encodings of the graph coloring problem on the performance of various SAT solvers. Our objective is to give a problem encoding of the UCTP that is succinct and enables the SAT solver to make good use of unit propagation. The succinctness is mainly important to ensure that the

SAT instances generated from large UCTP instances fit in the main memory on commodity hardware, which can be a problem with other encodings.

Let $\mathcal{I}$ be a UCTP instance with room and timeslot availability requirements. In the following, let $e_1, \ldots, e_{|E|}$ be the events, $r_1, \ldots, r_{|R|}$ be the rooms, and $p_1, \ldots, p_{|P|}$ be the timeslots given by the instance. The choice of the ordering is irrelevant since any two different orderings will just result in different variable names. We give a CNF $\varphi(\mathcal{I})$ such that $\varphi(\mathcal{I})$ is satisfiable if and only of $\mathcal{I}$ has a feasible solution.

The SAT encoding $\varphi(\mathcal{I})$ has the following variables:

- event to room mapping: $\{er_{e,r} \mid e \in E, r \in R\}$

$$er_{e,r} = \begin{cases} \text{TRUE} & \text{if } e \text{ is scheduled in room } r \\ \text{FALSE} & \text{otherwise} \end{cases},$$

- event to timeslot mapping: $\{ep_{e,p} \mid e \in E, p \in P\}$

$$ep_{e,p} = \begin{cases} \text{TRUE} & \text{if } e \text{ is scheduled in timeslot } p \\ \text{FALSE} & \text{otherwise} \end{cases},$$

- event conflicts: $\{conf_{e_i,e_j} \mid i, j \in \{1, \ldots, |E|\}, i > j\}$

$$conf_{e_i,e_j} = \begin{cases} \text{TRUE} & \text{if } e_i \text{ and } e_j \text{ are conflicting} \\ \text{FALSE} & \text{otherwise} \end{cases},$$

- room availability: $\{ra_{e,r} \mid e \in E, r \in R\}$

$$ra_{e,r} = \begin{cases} \text{TRUE} & \text{if } e \text{ can be scheduled in room } r \\ \text{FALSE} & \text{otherwise} \end{cases},$$

- timeslot availability: $\{pa_{e,p} \mid e \in E, p \in P\}$

$$pa_{e,p} = \begin{cases} \text{TRUE} & \text{if } e \text{ can be scheduled in timeslot } p \\ \text{FALSE} & \text{otherwise} \end{cases},$$

We also introduce the following auxiliary variables.

- "same room"-indicators $\{sr_{e_i,e_j} \mid i, j \in \{1, \ldots, |E|\}, i > j\}$
- "same timeslot"-indicators $\{sp_{e_i,e_j} \mid i, j \in \{1, \ldots, |E|\}, i > j\}$

All relations between two events, i. e., the event conflicts and the "same room" and "same timeslot" properties, are symmetric. The ordering of the events lets us exploit this symmetry and reduce the number of variables required to encode these

relations. The total number of variables in $\varphi(\mathcal{I})$ is given by:

$$\text{var}(\varphi(\mathcal{I})) = 2|E| \cdot (|R| + |P|) + 3 \cdot \binom{|E|}{2} . \tag{2.4}$$

In the following, we will characterize the UCTP by a CNF formula. First of all, we make sure that the auxiliary variables have the following properties: A "same room"-indicator variable should be set to TRUE whenever the two corresponding events are scheduled in the same room, irrespective of their timeslot assignments. That is, for a timeslot $p \in P$ and any two events $e_i, e_j, i > j$,

$$ep_{e_i,p} \wedge ep_{e_j,p} \longrightarrow sp_{e_i,e_j} .$$

This property is realized by the following clauses:

$$\bigwedge_{\substack{r \in R \\ i,j \in \{1,\dots,|E|\} \\ i > j}} \left(\neg er_{e_i,r} \vee \neg er_{e_j,r} \vee sr_{e_i,e_j} \right) . \tag{2.5}$$

Similarly, a "same timeslot"-indicators variable should be TRUE if the two corresponding events are scheduled in the same timeslot. This is realized by the following clauses:

$$\bigwedge_{\substack{p \in P \\ i,j \in \{1,\dots,|E|\} \\ i > j}} \left(\neg ep_{e_i,p} \vee \neg ep_{e_j,p} \vee sp_{e_i,e_j} \right) . \tag{2.6}$$

Now we can capture all properties of the UCTP in a straight forward fashion:

- Each event is scheduled in exactly one room:

$$\bigwedge_{e \in E} \left(\leq_1 (er_{e,r_1}, \dots, er_{e,r_{|R|}}) \wedge \geq_1 (er_{e,r_1}, \dots, er_{e,r_{|R|}}) \right) \tag{2.7}$$

- Each event is scheduled in exactly one timeslot:

$$\bigwedge_{e \in E} \left(\leq_1 (ep_{e,p_1}, \dots, ep_{e,p_{|R|}}) \wedge \geq_1 (ep_{e,p_1}, \dots, ep_{e,p_{|R|}}) \right) \tag{2.8}$$

- No two events are in the same room and in the same timeslot:

$$\bigwedge_{\substack{r \in R \\ i,j \in \{1,\dots,|E|\} \\ i > j}} \left(\neg sr_{e_i,e_j} \vee \neg ep_{e_i,e_j} \right) \tag{2.9}$$

- No two conflicting events are in the same timeslot:

$$\bigwedge_{\substack{r \in R \\ i,j \in \{1,...,|E|\} \\ i > j}} \left(\neg sp_{e_i,e_j} \vee \neg conf_{e_i,e_j} \right) \tag{2.10}$$

- Room availability requirements are respected:

$$\bigwedge_{\substack{r \in R \\ e \in E}} (ra_{e,r} \vee \neg er_{e,r}) \tag{2.11}$$

- Period availability requirements are respected:

$$\bigwedge_{\substack{p \in P \\ e \in E}} (pa_{e,p} \vee \neg ep_{e,p}) \tag{2.12}$$

The following unit clauses encode the room and timeslot availability for each event, as well as the event conflicts:

$$\bigwedge_{\substack{p \in P \\ e \in E}} \begin{cases} pa_{e,p} & \text{if } p \text{ is available for } e \\ \neg pa_{e,p} & \text{otherwise} \end{cases}, \tag{2.13}$$

$$\bigwedge_{r \in R,\, e \in E} \begin{cases} ra_{e,r} & \text{if } r \text{ is available for } e \\ \neg ra_{e,r} & \text{otherwise} \end{cases}, \tag{2.14}$$

$$\bigwedge_{\substack{i,j \in \{1,...,|E|\} \\ i > j}} \begin{cases} conf_{e_i,e_j} & \text{if } e_i \text{ is conflicting with } e_j \\ \neg conf_{e_i,e_j} & \text{otherwise} \end{cases}, \tag{2.15}$$

Combining the CNF formulas (2.5)–(2.15) into a single CNF formula (technically by taking the set union) yields the CNF formula $\varphi(\mathcal{I})$. All constraints need to be satisfied simultaneously. It is evident from the construction that $\mathcal{I}$ admits a feasible timetable if and only if $\varphi(\mathcal{I})$ is satisfiable. We will briefly discuss a subtlety in the role of the auxiliary variables. Note that in Eqs. (2.6) and (2.5) the auxiliary variables are necessarily true whenever two events are in the same timeslot and room, respectively. If the events are not in the same timeslot or room, the auxiliary variables may still be set to true. This however, adds restrictions on the problem, see Eqs. (2.9) and (2.10). Therefore, no matter how the values of the auxiliary variables are chosen in these cases, a feasible assignment of the variables $er_{e,r}$ and $ep_{e,p}$, $e \in E, r \in R, p \in P$ encodes an assignment of the events to resources that satisfies all constraints, i.e. a feasible timetable.

The number of clauses of $\varphi(\mathcal{I})$ is given by:

$$\text{size}(\varphi(\mathcal{I})) = 2 + 2 \cdot |E| \cdot (|P| + |R|) + (1 + 3 \cdot |R| + |P|) \cdot \binom{|E|}{2}$$

$$+ |E| \cdot \left(\binom{|R|}{2} + \binom{|P|}{2} \right) \ . \tag{2.16}$$

Since we can assume that $|E| = |R| = O(|\mathcal{I}|)$ we have $\text{size}(\varphi(\mathcal{I})) = O(|\mathcal{I}|^3)$. The proposed CNF encoding of the UCTP is very unit propagation friendly: Eqs. (2.13)–(2.15) can be eliminated immediately and all other clauses except the $2 \cdot |E|$ $\geq_1$-constraints have length at most three, so there is potential for eliminating more clauses quickly.

2.5 Experimental Evaluation of the Kempe Insertion Heuristic

The overall goal of this section is a comparison of the performance of the KIH against other approaches from the literature that focus on finding feasible timetables. Before this comparison, we will investigate the following two questions:

1. What is the impact of the preprocessing phase in the KIH?
2. What is the performance of the KIH compared to other approaches from the literature on suitable benchmark instances?

Most state-of-the-art approaches that deal with finding feasible timetables use some form of preprocessing, see for example [TBM07,MW10b*,LZC11,CDGS12]. The intention is to quickly create an initial timetable, which is used as a starting point for a more elaborate algorithm that deals with the remaining hard constraint violations. The first question is motivated by the fact that despite this widely used decomposition into a preprocessing and an optimization phase, there is no rigorous investigation of the quality of the timetables produced by various preprocessing algorithms. That is, it is unclear with how many constraint violations the elaborate optimization approaches have to deal for different preprocessing algorithms. We will provide experimental data on the distance to feasibility of the timetables produced by five different preprocessing heuristics, including the one used in [TBM07,MW10b*].

To investigate the second question, we first perform an automated tuning of the KIH parameter settings. Note that the performance of any of the popular meta-heuristic approaches depends on the parameter settings. For example, the performance of SA depends on choosing good temperature levels for the problem at hand [KGV83]. Similarly, EA performance naturally depends on mutation and crossover rates. The KIH has a number of parameters and for our investigation of

the second question we are interested in tuning these parameters in a systematic fashion to a set of UCTP benchmark instances. In order to determine good parameter settings, we first check if there are significant differences between a number of tuned parameter settings. We include in this comparison the parameter choice from the original work [MW10b*], which has been determined by trial and error without using statistical methods such as hypothesis testing.

We will rely on the parameter settings determined by the systematic tuning in our overall comparison of the KIH against other approaches from the literature. As noted by [TBM07], there are only few approaches that focus on finding feasible timetables. One reason for this is certainly, that on many popular benchmark instances, for example, the instances from the timetabling competition 2002 [PGRD], it is rather easy to find feasible timetables [LP07] and the actual challenge is to minimize the soft constraint violations. As a remedy, Lewis and Paechter introduced three sets of artificially generated UCTP instances, referred to as small, medium and big, with a focus on finding feasible timetables rather than soft constraint satisfaction [LP07, LPb]. The small instances have between 200 and 225 events and 5 or 6 rooms, the medium instances have between 390 and 425 events and 10 or 11 rooms, and the big instances have between 1,000 and 1,075 events and 25 to 28 rooms. The instances sets were created by selecting from a set of randomly generated instances 20 for each set on which the algorithms from [AL05, LP04] performed poorly [LP07]. These instances are indeed challenging, and as of today, none of the approaches from the literature is able to produce feasible timetables on all instances. We perform our two experiments on these instances. In our overall comparison of approaches for creating feasible timetables, we will focus on approaches that have been evaluated on the Lewis and Paechter instance sets.

Preliminaries: Algorithm Configuration

The goal of automated algorithm tuning is to select parameters of a given algorithm such that it performs best on a given set of training instances. This problem has been formalized in terms of the Algorithm Configuration Problem (ACP) [BYBS10]. A choice of values for the parameters of an algorithm is referred to as *configuration*. The task is to find a configuration such that the expected value of a cost measure— in our setting, for example, the distance to feasibility—is minimal over a set of benchmark instances. We use the I-RACE algorithm, from [BYBS10], to perform the tuning task. An implementation of this algorithm is available as a package for the R statistics software [LIDLSB11, R D08] and a brief outline of the approach is given below. We will use statistical hypothesis testing in order to determine if there is a significant benefit from tuning the algorithm parameters in Experiment 2.

The Iterated F-RACE (I-RACE) Algorithm [BYBS10] F-RACE evaluates a given set of candidate configurations on a set of benchmark instances in order to determine the best configurations according to some quality measure. For minimization

problems, the measure is typically the expected cost generated by a configuration. The naïve approach is to just evaluate all given configurations on all instances sufficiently often. The motivation of F-RACE is to make better use of computational resources by discarding configurations as soon as there is enough statistical evidence indicating that they perform worse than the other configurations. Hence, if a number of configurations can be discarded, more time can be spent on evaluating the performance of the well-performing configurations. In each step of the algorithm, the configurations which have not been discarded yet are evaluated on a particular instance. The null hypothesis is that all remaining configurations show equal performance. If the null hypothesis is rejected according to Friedman's two-way analysis of variance by ranks [Fri37, Con06] (i. e., the Friedman test) with a significance level $\alpha = 0.05$, then appropriate post-hoc tests are performed in order to decide which configurations are discarded. If there is not enough statistical evidence to reject the null hypothesis, all configurations are kept for the next iteration. While F-RACE evaluates a fixed set of configurations, I-RACE iteratively executes F-RACE and uses the results in order to bias the sampling of new configurations and thus offers arguably better exploration of the configuration space.

2.5.1 Experiment 1: The Impact of Preprocessing

Algorithms for the UCTP typically use some computationally cheap constructive preprocessing step that generates an initial timetable used as a starting point for more elaborate techniques that deal with the remaining hard constraint violations. Examples of such approaches can be found in [Kos05, TBM07, MW10b*, LZC11, CDGS12]. The motivation for this decomposition is to save computation for the sophisticated techniques. To the best of our knowledge, no detailed data has been published on the constraint violations that remain after the preprocessing phase on the Lewis and Paechter instances from [LPb]. Thus, it is not clear, how much work is left to be done by the elaborate techniques using different preprocessing approaches. Here, we will provide empirical data on the distance to feasibility that can be attained by five different sequential heuristics including the one used in [TBM07, MW10b*].

Experiment Design The behavior of a sequential heuristic is determined by an event selection policy and a resource selection policy. See Sect. 2.4.1 for a list of policies from the literature. We denote by A/B the sequential heuristic that uses event selection policy A and resource selection policy B. Our comparison includes the heuristics LSD/RO, LSDLD/RO, LD/SO, LSD/SO and LSDLD/SO. The two resource selection policies RO and SO were discussed in Sects. 2.4.1 and 2.4.2, respectively. A run of a sequential heuristic is evaluated with respect to the distance to feasibility of the resulting timetable. We do not consider the runtimes of the sequential heuristics, because the differences are negligible due to the overall very similar algorithmic scheme. We collect statistical information over 500 independent

runs per heuristic and per instance, namely the attained minimum, average, and maximum distance to feasibility. Our intention for this is twofold: First, we want to get a sufficient sample size to allow for conclusions about the expected distance to feasibility for each algorithm and instance, and second, we intend to independently verify the following observation by Tuga et al. in [TBM07]: According to their findings, running the LSDLD/RO heuristic 500 times per instance produces feasible timetables on 13 out of the 60 benchmark instances.

Data Table 2.3 shows for each of the 60 benchmark instances from [LPb] the minimum, average, and maximum number of events left unscheduled by the five different sequential heuristics. The last row of Table 2.3 summarizes for each heuristic how often it performed at least as good as any of the other heuristics with respect to the minimum/average/maximum number of events that were left unscheduled.

Results We can conclude from the data shown in Table 2.3 that there is quite some variation in the distance to feasibility, even for different runs of the same heuristic. Thus, it is reasonable to run the sequential heuristics multiple times in the preprocessing phase in order to reduce the variance of the minimum distance to feasibility attained. According to Tuga et al. [TBM07], feasible solutions to 13 out of 60 instance can be found by running the LSDLD/RO heuristic 500 times per instance. We were able to confirm this result: According to the data in Table 2.3, feasible timetables were found on 11 instances by this sequential heuristic and on several other instances the resulting timetables were close to feasibility. Overall, the SO resource selection policy seems to generate the best results in combination with the LSD event selection policy, with or without using LD for breaking ties.

2.5.2   Experiment 2: Algorithm Configuration and Evaluation

The KIH proposed in [MW10b*] has a number of parameters that have some influence on its behavior, for example, the balance between construction and perturbation. Naturally, we are interested in picking a configuration such that the algorithm performs best on the instances of interest. In [MW10b*], this task has been performed by hand with satisfactory results. In order to check in how far the performance of the algorithm can be improved by altering the configuration and how much the quality of the results depends on the configuration, we perform an automated parameter tuning of the KIH using the I-RACE algorithm (see).

Fixed and Varied Parameters

Recall from Sect. 2.4.2, that the algorithm KEMPEINSERTION has parameters k, ℓ, m, and λ. The parameters k and m determine the relative importance of the construction phase and the perturbation phase, and ℓ is the perturbation strength.

Table 2.3 Results of Experiment 1: minimum/average/maximum distance to feasibility attained by five different sequential heuristics on the Lewis and Paechter instances from [LPb]

	LSD/RO	LSDLD/RO	LD/SO	LSD/SO	LSDLD/SO
small_1	**0**/0.4/4	**0**/0.4/**3**	**0**/**0.3**/4	0/7.4/15	4/6.7/8
small_2	**0/0.0/0**	**0/0.0/0**	**0/0.0/0**	**0/0.0/0**	**0/0.0/0**
small_3	6/13.3/23	7/13.6/23	10/21.7/33	**0**/2.2/13	1/**1.0/1**
small_4	**0**/5.8/11	**0**/6.0/13	**0**/5.1/12	1/6.9/19	**0**/**3.1/8**
small_5	10/17.8/28	10/17.0/24	14/22.2/30	**1/8.3**/17	7/8.7/**12**
small_6	**0/0.0/0**	**0/0.0/0**	**0/0.0/0**	0/2.4/7	1/3.2/6
small_7	3/7.6/14	2/6.9/14	2/7.9/17	**0**/0.5/7	**0/0.0/0**
small_8	23/32.1/42	23/31.6/39	23/34.2/44	**16/25.5**/36	23/26.9/**30**
small_9	32/52.1/69	44/59.2/72	35/49.6/68	**8**/23.4/43	18/**23.0/28**
small_10	14/28.9/57	10/18.1/23	34/45.0/59	**0**/7.3/32	**0**/0.2/2
small_11	**0**/0.0/4	**0/0.0/2**	**0**/0.4/5	**0**/0.8/4	**0**/1.6/5
small_12	**0/0.2/4**	**0**/0.6/5	2/5.5/10	3/10.6/15	6/10.7/14
small_13	26/48.7/75	33/45.4/53	30/44.4/62	**2**/22.0/58	7/**12.1/14**
small_14	31/39.5/52	28/41.0/52	29/41.3/54	**16/27.7**/42	20/30.5/**35**
small_15	**0**/5.7/14	2/7.1/17	**0**/4.3/13	**0/0.0/0**	**0/0.0/0**
small_16	12/22.7/39	12/21.8/43	14/28.5/40	**0**/2.3/16	**0/0.0/0**
small_17	31/42.3/53	26/44.9/53	25/38.1/51	**0**/9.8/29	3/**3.6/5**
small_18	9/19.5/30	14/22.4/29	**4/8.4/15**	12/22.8/30	14/18.4/23
small_19	47/60.4/75	47/60.4/75	37/50.8/64	**1/21.6**/55	24/27.0/**30**
small_20	**0**/0.4/5	**0**/**0.4**/5	**0**/1.1/**4**	1/4.4/14	**0**/2.5/5
med_1	**0**/5.8/15	**0/5.6/12**	1/7.6/16	5/12.7/25	4/11.4/18
med_2	**0**/5.7/18	**0/4.6/12**	1/9.4/19	4/15.3/29	6/10.4/17
med_3	**1/10.0/21**	4/12.9/22	11/21.1/33	8/17.8/30	19/21.1/25
med_4	**5/13.0**/24	7/15.1/24	13/21.1/31	12/23.9/40	6/13.9/**22**
med_5	22/38.2/49	26/38.8/49	22/36.6/47	**19**/29.2/44	20/**26.2/33**
med_6	29/47.7/61	38/50.5/67	30/48.1/63	**15**/31.8/43	28/**30.9/33**
med_7	57/71.7/90	62/74.8/91	81/99.5/114	**44**/59.3/76	49/**58.1/68**
med_8	49/61.9/77	48/63.5/77	50/65.5/79	**23**/36.3/52	33/**34.9/39**
med_9	52/73.6/96	65/79.5/97	73/90.6/110	**40**/56.8/81	59/63.2/69
med_10	**0**/0.4/5	**0/0.4/4**	**0**/1.1/6	1/11.8/29	5/10.2/18
med_11	27/41.6/56	28/40.6/56	28/41.2/54	**0**/10.3/37	1/**5.2/12**
med_12	3/10.8/19	3/10.5/19	5/16.1/33	**0**/**3.0**/7	2/4.3/**7**
med_13	47/63.2/78	53/65.5/78	58/71.9/87	**13**/26.1/41	32/34.4/**38**
med_14	16/28.8/44	15/29.6/42	20/32.8/47	**0**/5.7/23	2/**3.0/4**
med_15	21/30.9/42	18/30.2/39	24/34.5/44	**6**/22.9/46	8/**14.4/23**
med_16	97/116.9/133	99/117.3/137	99/120.5/145	**27/62.4**/104	63/65.6/**68**
med_17	58/75.0/91	60/74.7/88	48/70.4/95	**0**/20.3/52	13/**14.5/24**
med_18	126/147.6/168	126/150.2/173	112/137.0/164	**21/79.3**/131	80/86.8/**94**
med_19	114/133.4/155	110/132.3/150	109/135.6/154	**31/79.9**/118	80/80.2/**81**
med_20	78/93.0/113	78/94.3/113	83/103.1/121	**1**/43.0/79	33/**33.0/33**

<div align="right">(continued)</div>

Table 2.3 (continued)

	LSD/RO	LSDLD/RO	LD/SO	LSD/SO	LSDLD/SO
big_1	**0**/1.8/6	**0**/1.4/8	**0/0.5/4**	2/16.7/39	8/18.1/30
big_2	**9/21.9/36**	**9**/26.0/40	28/42.2/60	34/49.0/67	35/47.8/61
big_3	**14/28.4/41**	19/34.2/46	21/32.7/46	31/45.4/60	31/42.4/55
big_4	44/**58.7**/84	**39**/58.8/**72**	68/88.3/107	55/79.2/111	57/67.7/77
big_5	**53/74.8/89**	65/78.2/93	93/111.8/129	78/92.9/110	80/93.2/107
big_6	95/117.4/137	98/115.8/137	158/182.6/209	90/113.5/138	**88/101.9/119**
big_7	170/196.0/230	183/209.4/237	256/279.5/306	**159/187.5**/223	181/192.4/**212**
big_8	**52/70.3**/90	53/70.5/**89**	87/109.3/126	59/75.5/93	63/79.7/93
big_9	**39**/55.6/76	41/55.5/71	66/82.4/99	44/62.8/88	43/**54.0**/66
big_10	**60**/75.4/93	64/**75.0/89**	101/117.7/137	68/93.2/129	68/79.6/93
big_11	**55/71.4/85**	57/72.9/91	100/117.4/137	74/94.2/114	70/83.3/96
big_12	41/59.1/80	47/62.4/78	29/42.1/59	28/44.3/66	**27/38.5/51**
big_13	52/73.5/90	60/71.3/84	51/73.6/89	**41**/59.9/78	43/**52.1/60**
big_14	47/73.9/93	58/74.8/91	37/56.3/71	**36**/52.7/79	38/**50.3/68**
big_15	146/178.5/209	155/182.5/203	214/242.1/275	**97/128.4/165**	148/152.8/166
big_16	133/157.2/180	136/158.9/180	127/155.4/184	36/64.9/96	**27/38.2/47**
big_17	339/364.5/394	341/368.9/393	345/388.3/427	**189/257.2**/318	254/272.3/**293**
big_18	221/246.0/275	225/254.5/285	255/293.2/329	**107/153.0**/217	154/163.8/**177**
big_19	284/306.6/334	287/313.4/333	331/353.7/378	**194/239.3**/298	228/245.3/**257**
big_20	233/269.4/294	249/279.2/310	248/287.1/331	**120**/164.5/211	136/**160.2/174**
∑	21/11/8	13/8/10	10/5/5	**39**/17/4	11/**26/41**

Entries in bold face mark the best performing heuristic in each category

The parameter λ is the decay rate of an exponentially distributed random variable that is used in order to select which event is selected to be inserted in the timetable in the perturbation phase. The unscheduled events are arranged in non-decreasing order by the number of conflicts in a certain timeslot and an event is selected to be scheduled in this timeslot according to the exponential distribution. We vary the four parameters k, ℓ, m, and λ in the tuning process. We also vary the event selection policy P_1 and resource selection policy P_2 of the preprocessing in order to check, which of the sequential heuristics performs best in combination with KEMPEINSERTION. For P_1, we consider the LSD and the LSDLD policies, and for P_2 the RO and SO policies.

We concluded from Experiment 1 that sequential heuristics should be run several times in the preprocessing phase in order to compensate for that rather high variance of the distance to feasibility. In the current experiment we set the number of trial runs of the sequential heuristics to 15, which is considerably lower than what Tuga et al. used in [TBM07]. To give some justification for this choice, we iteratively performed the following experiment: We measured the difference between the minimum distances to feasibility attained by 15 and 500 runs of a sequential heuristic in percent relative to the total number of events. Table 2.4 shows the average outcome and the standard deviation over 50 runs per instance for each

Table 2.4 Mean and standard deviation of the difference in the distance to feasibility between running the sequential heuristics 15 and 500 times in percent relative to the total number of events

Instance set	LSD/RO	LSDLD/RO	LSD/SO	LSDLD/SO
Small	1.36 ± 1.50	1.39 ± 1.50	0.35 ± 0.62	0.92 ± 1.22
Med	0.60 ± 0.78	1.40 ± 0.99	0.20 ± 0.31	1.80 ± 2.30
Big	0.74 ± 0.51	0.84 ± 0.56	0.40 ± 0.33	1.00 ± 0.85

instance set and each sequential heuristic. We consider the differences between 15 and 500 runs to be sufficiently small to cut down the number of iterations of the sequential heuristic to 15.

Performance Measures

We measure the performance of a run of the KIH (preprocessing followed by KEMPEINSERTION) according to two criteria, the runtime and the distance to feasibility. For a comparison of different configurations of the KIH we combine these two criteria in the following way: Let (t_1, d_1) and (t_2, d_2) be the measured performance of two runs r_1 and r_2, where t_1 and t_2 are the runtimes given in seconds and d_1 and d_2 are the distances to feasibility. Then r_1 showed better performance than r_2 if and only if $(t_1, d_1) \prec_{lex} (t_2, d_2)$. Intuitively, a run that produces a feasible timetable is always better than one producing an infeasible timetable. Among two runs that produce feasible timetables, one that takes less runtime is to be preferred and among two runs that produce infeasible timetables, one that leaves fewer events unscheduled is to be preferred. The performance of the KIH is compared to the performance of various algorithms from the literature according to the minimum and average number distance to feasibility over a number of runs as well as the average runtime used.

Experiment Design

We will first discuss the experiment designs that were used for evaluating different approaches from the literature in order to motivate the design of our experiment. Lewis and Paechter evaluate their GGA algorithms using a timeout of 30, 200, and 800 s for the small, medium and big instance sets, respectively [LP07]. Liu et al. evaluate their CB algorithm using the same timeouts. However, also set a fixed maximum number of iterations and on some instances their algorithm apparently terminates before the timeout, even if no feasible timetable has been found [LZC11]. Tuga et al. use timeouts of 200, 400, and 1,000 s for the small, medium and big instance sets, respectively [TBM07]. Ceschia et al. use a fixed number of 10^8 iterations of their SA-based approach and no timeout [CDGS12]. Due to the different experiment design, it is quite a challenge to make a fair comparison between the different approaches.

There are also some more fundamental issues with the different experiment designs that require some consideration, most of which have been discussed in [Joh02]. Using a timeout can be quite misleading because of ever-increasing CPU clock speeds and architectural improvements of the hardware. Certainly, 3 s of CPU time in 2013 is different from 30 s of CPU time in 2007, or in 1990, which impairs reproducibility in particular of older results. Additionally, in the presence of a timeout the efficiency of the implementation has a significant influence on the results. So, in this experiment design, there is no satisfactory answer to the question whether an improved solution quality should be attributed to a proposed algorithm, the implementation or the underlying hardware. Setting a maximum number of iterations or evaluations of the cost function avoids these pitfalls but introduces a new one: different heuristic approaches may perform a vastly different amount of work per iteration. For example, performing a single iteration of KEMPEINSERTION is quite costly in terms of CPU time, in particular when the EIP cannot be solved, because a number of Kempe-exchanges and maximum cardinality matchings are computed. In comparison, performing an iteration of SA with a simple neighborhood structure is much quicker. Furthermore, a comparison between heuristics and exact approaches such as SAT or ILP solvers based on a maximum number of iterations is non-sensical. In conclusion, neither a timeout nor a fixed iteration count is the silver bullet for conducting comparable computer experiments.

For the parameter tuning and the overall comparison of KIH with the other approaches we will use the timeouts suggested by Lewis and Paechter [LP07] in our experiments, which allows for a somewhat fair comparison of our results and the results in [LP07, TBM07, LZC11], and to some extent the ones in [CDGS12]. Furthermore, we will compare the results of these heuristics to those produced by a SAT solver using the formulation from Sect. 2.4.3. All experiments were performed on a single core of a 3.0 GHz i7 machine. Certainly, in our comparison of the heuristic approaches, we cannot cure the flaws that are inherent in using timeouts, however, this machine should show comparable performance with the machines used in [LP07, TBM07, LZC11].

Following [CDGS12], we perform the parameter tuning separately on each set of instances from [LPb]. For each run of the KIH with a given configuration, we use the timeouts suggested by [LP07], that is, 30 s for the small instances, 200 s for the medium instances and 800 s for the big instances. After some preliminary screening, the parameter domains used by I-RACE were set to the values shown in Table 2.5.

In total, the total computational budget of I-RACE was set to 1,000 runs of the KIH on each instance set. The instances that present no challenge to the KIH were removed from the instance sets. Thus, the tuning is focused on getting good performance on the challenging instances, which offer more potential for improvements. The assumption is that a configuration tuned to a restricted instance set also performs well on the remaining instances. We will comment on this issue later when we discuss the results. The instances shown in Table 2.6 were considered "challenging" and were therefore used as training instances for the automated tuning.

Table 2.5 Parameter domains considered in the automated tuning of the KIH

Parameter	Domain
P_1	{LSD,LSDLD}
P_2	{RO,SO}
ℓ	[10, 60]
m	[1, 5]
k	[1, 5]
λ	[0.5, 2.0]

Table 2.6 Training instance sets used in the automated tuning process

Instance set	Training instances
Small	8, 9, 12, 13, 14, 18, 19, 20
Medium	7, 8, 9, 12, 18, 19, 20
Big	4, 5, 6, 7, 8, 9, 10, 11, 15, 17, 19, 20

Table 2.7 KIH configurations for the small, medium and big instances

Configuration	P_1	P_2	ℓ	m	k	λ
(a) Configurations for the instances `small_1,...,small_20`						
A_{small}	LSD	SO	32	2	3	1.20
B_{small}	LSD	RO	17	2	1	1.72
C_{small}	LSD	SO	22	2	4	1.30
D_{small}	LSD	SO	15	2	1	1.85
E [MW10b*]	LSDLD	RO	20	1	1	1.00
(b) Configurations for the instances `med_1,...,med_20`						
A_{med}	LSDLD	SO	13	3	2	1.65
B_{med}	LSDLD	SO	16	3	4	1.15
C_{med}	LSDLD	SO	17	3	2	1.63
D_{med}	LSDLD	SO	11	3	2	1.64
E [MW10b*]	LSDLD	RO	20	1	1	1.00
(c) Configurations for the instances `big_1,...,big_20`						
A_{big}	LSDLD	SO	17	2	5	1.53
B_{big}	LSDLD	SO	23	2	5	1.46
C_{big}	LSDLD	SO	18	2	5	1.67
D_{big}	LSDLD	SO	22	2	2	1.68
E [MW10b*]	LSDLD	RO	20	1	1	1.00

Configurations A–D have been determined by I-RACE for each instance set and configuration E is from [MW10b*]

The output of I-RACE is a set of the best four configurations found for each instance set. We perform 20 independent runs for each of these configurations on each instance of the corresponding (complete) instance set. Additionally, we perform 20 independent runs for each of the 60 instances with the KIH configuration given in [MW10b*], referred to as configuration E. This configuration is also shown in Table 2.7. This configuration has been determined by manual experimentation. Next, we will check if the differences in performance between any two

configurations are significant according to the one-tailed Wilcoxon-Mann-Whitney
(WMW) test [Wil45, Con06]. That is, for any two configurations X and Y, the null-
hypothesis being tested is that the KIH does not show a different performance with
configuration X compared to Y. The alternative hypothesis is that X produces better
results than Y. We perform the test on samples of 20 independent runs for each
configuration on each instance of the corresponding set.

Finally, we compare the performance of the KIH to other approaches from the
literature and the SAT-solver `minisat` [ES04] with respect to the minimum and
average distance to feasibility and, if available, the average processing time used.
We use for each instance set the best configuration among the tuned configurations
and the original configuration from [MW10b*]. Thus, we provide an update to
the experimental results in [MW10b*] in a slightly modified experimental setup.
Please note that in this comparison, a lower distance to feasibility on the difficult
instances is more desirable than spending less CPU time on the easier instances. This
preference is not present in the performance measure, but rather by the selection
of instances that are used for the configuration tuning. The SAT instances were
created according to the SAT formulation from Sect. 2.4.3. Overall, we include in
our comparison the SAT solver and the data from the literature shown in Table 2.8.

These algorithms were summarized briefly in Sects. 2.4.1 and 2.4.2. This com-
parison gives a complete picture of the relative performance of published approaches
to solving the Lewis and Paechter instances.

In addition to the distance to feasibility and the processing time, we measure the
average number of room assignments performed by KEMPEINSERTION per second
on each instance. Our motivation is to be able to identify potential scaling issues of
the algorithm. Since the number of rooms increases between small, medium, and
large instances, we expect the number of room assignments performed per unit time
to be significantly lower on the big instances compared to the small and medium
instances.

Table 2.8 Algorithms from the literature which are compared to the KIH, along with the
corresponding publications and a rough classification of the used techniques

Identifier	Publication	Technique
GGA	Lewis and Paechter [LP07, LPa]	Grouping Genetic Algorithm
LS	Lewis and Paechter [LP07, LPa]	Local Search
KIH	M. and Wanka [MW10b*] (see text)	Kempe Insertion Heuristic
HSA	Tuga et al. [TBM07]	Simulated Annealing
CB	Liu et al. [LZC11]	Clique-based Construction
TSA	Ceschia et al. [CDGS12]	Simulated Annealing

Data

Table 2.7 shows the four best KIH configurations determined by I-RACE for each instance set along with the configuration E from [MW10b*]. The configurations A_{small}–D_{small}, A_{med}–D_{med}, and A_{big}–D_{big} are the best configurations found by I-RACE for the small, medium, and big instances, respectively. Table 2.9 shows the results of the WMW-tests performed to detect performance differences between the configurations. For each pair (X, Y) of configurations, the corresponding cell shows the number instances on which configuration X is significantly better than Y according to the one-tailed WMW test. For example, the configuration B_{small} performs better than A_{small} on 9 instances according to the WMW test. Tables 2.11, 2.12 and 2.13 show the results obtained by the KIH algorithm, the SAT solver, and the various approaches from the literature. The columns of the tables are arranged such that the results are presented in chronological order, from the two algorithms GGA and LS by Lewis and Paechter [LP07] to the most recent algorithm TSA by Ceschia et al. [CDGS12]. For

Table 2.9 Results of Experiment 2: relative performance of the KIH configurations from Table 2.7

(a) Relative performance of the configurations A_{small}, ..., D_{small}, E on instances small_1,...,small_20

	A_{small}	B_{small}	C_{small}	D_{small}	E
A_{small}	–	5	6	7	5
B_{small}	9	–	11	10	5
C_{small}	4	5	–	6	6
D_{small}	1	4	5	–	6
E	0	1	10	11	–

(b) Relative performance of the configurations A_{med}, ..., D_{med}, E on instances med_1,...,med_20

	A_{med}	B_{med}	C_{med}	D_{med}	E
A_{med}	–	6	4	2	11
B_{med}	2	–	3	3	7
C_{med}	1	0	–	1	8
D_{med}	0	1	1	–	7
E	5	6	5	5	–

(c) Relative performance of the configurations A_{big}, ..., D_{big}, E on instances big_1,...,big_20

	A_{big}	B_{big}	C_{big}	D_{big}	E
A_{big}	–	4	0	0	9
B_{big}	1	–	0	0	7
C_{big}	4	2	–	0	6
D_{big}	7	5	6	–	10
E	6	4	4	3	–

For each pair (X, Y) of configurations, the corresponding table entry indicates the number of instances on which configuration X is superior to configuration Y

each solver from the literature we give the best and average distance to feasibility and, if available, the average runtime in seconds. The SAT solver either found a feasible solution or hit the timeout, i.e., produced no result. On the small, medium, and large instances, we ran the KIH with configurations B_{small}, A_{med}, and D_{big}, respectively. The mean and standard deviation of the average number of room assignments performed per second were determined to be 36150 ± 18856, 24332 ± 18216 and 6333.3 ± 3155.5, on the small, medium and big instances, respectively.

Results

According to the data shown in Table 2.7, the best KIH configurations for the small instances found by I-RACE use the LSD/SO or LSD/RO preprocessing heuristics. This is consistent with our findings from Experiment 1 shown in Table 2.3 which indicate that these heuristics leave the fewest events unscheduled among the preprocessing heuristics under consideration. On the medium and big instances, all tuned configurations use the LSDLD/SO preprocessing. On these instance sets, the results shown in Table 2.3 do not clearly indicate that LSDLD/SO is a superior choice. WMW tests however reveal this is indeed the case: For example, LSDLD/SO is better than LSD/SO on 13 medium instances and 12 big instances. On the other hand, LSD/SO is better than LSDLD/SO on 6 medium and 7 big instances. A similar comparison of LSDLD/SO and LSD/RO is also in favor of the LSDLD/SO heuristic. Considering this, we can conclude that the best overall performance of the KIH is produced with the best preprocessing heuristic.

The tuned configurations indicate that for the remaining parameters ℓ, m, k, and λ, the best configurations can be found within tighter intervals than the ones explored by I-RACE. The parameters in the 16 best configurations identified by irace are shown in Table 2.10.

From the relative performance of the different configuration shown in Table 2.9 we cannot conclude that the tuned configurations found by I-RACE generally show significantly better performance than configuration E from [MW10b*]. However, configurations B_{small}, A_{med}, and D_{big} are clearly superior choices compared to configuration E on the respective instance sets. We further observed that the

Table 2.10 KIH parameter domains of the best performing configurations found by I-RACE

Parameter	Values
ℓ	[11, 32]
m	[2, 3]
k	[1, 5]
λ	[1.15, 1.72]

tuned configurations outperform configuration E on similar sets of instances and vice versa: Each of the configurations A_{med}–D_{med} outperforms configuration E on the six instances `med_6`, `med_8`, `med_11`, `med_14`, `med_17`, and `med_19`, and E outperforms every tuned configuration on `med_1`–`med_4`, and `med_10`. Similarly, configurations A_{big}–D_{big} all outperform E on the instances `big_6`, `big_7`, `big_14`, `big_15`, `big_17`, and `big_19`. In turn, E performs better than all tuned configurations on `big_1` and `big_3`. All instances on which significant performance differences were found are part of the reduced instance set used by I-RACE. Therefore, the fact that the configuration E consistently outperforms the tuned configurations on some instances cannot be attributed to the exclusion of instances from the tuning.

Compared to other approaches published in the literature, the KIH performs best on the small and medium instances with respect to the total average distance to feasibility. The KIH was the first published approach that produced feasible timetables on every of the small and medium instances. Overall, on these instances, the KIH uses less CPU time than the HSA, and, with several exceptions, also less than the clique-based approach. However, small differences in runtime should not be considered significant since they may be due to platform and implementation differences. On the big instances, the performance relative to the other approaches is not as good. This indicates scaling issues, which are partly caused by the excessive use of the bipartite matching algorithm: On the big instances, on average only a quarter of the room assignments can be performed per second compared to the medium instances. Thus, the extra time of 600 s for solving the big instances is completely devoted to solving room assignment subproblem instances. Thus, there is no time left for dealing with other sources of the increasing complexity. Since the results shown in Tables 2.11, 2.12 and 2.13 were obtained using the tuned configurations, we cannot expect further substantial improvements without modifying the algorithm.

The SAT solver `minisat` has been employed in order to check if the use of specialized heuristics is justified for finding feasible timetables on the Lewis and Paechter instances. The SAT solver was able to find feasible timetables on 16 small instances, 12 medium instances and 6 big instances using the CNF encoding from Sect. 2.4.3. Whenever the SAT solver was able to solve an instance, the result was produced consistently within a very competitive amount of CPU time. The results indicate that the small instances have become too easy, and benchmarking should focus on the medium and big instances, as advocated by [CDGS12]. On the latter instance sets, the use of specialized heuristics is well justified according to our experiments. A drawback of the SAT-based approach is that if the SAT solver fails to find a satisfying assignment then we get no solution at all. As a remedy, a (partial) MAX-SAT-based approach can be used, which has been demonstrated to produce some of the known best results on the CB-CTT problem by [AAN12].

Table 2.11 Results obtained by the KIH, the SAT solver, and various approaches from the literature on the small instance set from [LPb]

Instance	GGA Best (avg)	LS Best (avg)	HSA Best (avg)	CPU (s)	KIH Best (avg)	CPU (s)	CB Best (avg)	CPU (s)	Minisat CPU (s)
small_1	0 (0)	0 (0)	0	0	0 (0.00)	0.00	0	0.05	0.44
small_2	0 (0)	0 (0)	0	0	0 (0.00)	0.00	0	0.02	0.50
small_3	0 (0)	0 (0)	0	9	0 (0.00)	0.16	0	1.25	0.47
small_4	0 (0)	0 (0)	0	0	0 (0.00)	0.08	0	0.05	0.46
small_5	0 (1.05)	0 (0)	0	5	0 (0.00)	0.34	0	4.20	0.57
small_6	0 (0)	0 (0)	0	0	0 (0.00)	0.00	0	0.02	0.46
small_7	0 (0)	0 (0)	0	0	0 (0.00)	0.11	0	0.02	0.50
small_8	0 (6.45)	0 (1)	0 (1.9)	79	0 (0.75)	25.01	0 (0.2)	0.05	Timeout
small_9	4 (2.5)	0 (0.15)	0 (3.85)	84	0 (0.00)	3.92	0 (0.3)	0.02	Timeout
small_10	0 (0.1)	0 (0)	0	15	0 (0.00)	0.61	0 (0.15)	1.25	0.79
small_11	0 (0)	0 (0)	0	0	0 (0.00)	0.00	0	0.02	0.50
small_12	0 (0)	0 (0)	0	0	0 (0.00)	0.00	0	2.75	0.71
small_13	0 (1.25)	0 (0.35)	0 (1)	15	0 (0.05)	4.68	0	1.00	0.78
small_14	3 (10.5)	0 (2.75)	3 (5.95)	136	0 (0.65)	24.10	0 (0.7)	70.55	Timeout
small_15	0 (0)	0 (0)	0	0	0 (0.00)	0.12	0	0.55	0.43
small_16	0 (0)	0 (0)	0	13	0 (0.00)	0.13	0	0.55	0.48
small_17	0 (0.25)	0 (0)	0	13	0 (0.00)	0.29	0	2.00	0.62
small_18	0 (0.7)	0 (0.2)	0 (0.45)	36	0 (0.00)	0.20	0 (0.7)	51.65	Timeout
small_19	0 (0.15)	0 (0)	0 (1.2)	25	0 (0.00)	0.87	0	1.00	0.66
small_20	0 (0)	0 (0)	0	0	0 (0.00)	0.00	0 (0.15)	70.85	104.6

For each approach except the SAT solver we give the minimum and average distance to feasibility and, if available, the average CPU time in seconds. For the SAT solver minisat, we give the time taken to find a feasible timetable

Table 2.12 Results obtained by the KIH, the SAT solver, and various approaches from the literature on the medium instance set from [LPb]

| Instance | GGA | LS | HSA | | KIH | | CB | | TSA | Minisat |
	Best (avg)	Best (avg)	Best (avg)	CPU (s)	Best (avg)	CPU (s)	Best (avg)	CPU (s)	Best (avg)	CPU (s)
med_1	0 (0)	0 (0)	0 (0.00)	0	0 (0.00)	1.23	0 (0.00)	5.85	0 (0.00)	2.00
med_2	0 (0)	0 (0)	0 (0.00)	0	0 (0.00)	1.07	0 (0.00)	1.5	0 (0.00)	1.93
med_3	0 (0)	0 (0)	0 (0.00)	8	0 (0.00)	1.78	0 (0.00)	5.05	0 (0.00)	3.02
med_4	0 (0)	0 (0)	0 (0.00)	3	0 (0.00)	1.18	0 (0.00)	2.05	0 (0.00)	4.03
med_5	0 (3.95)	0 (0)	0 (0.00)	85	0 (0.00)	13.74	0 (0.00)	59.75	0 (0.00)	Timeout
med_6	0 (6.2)	0 (0)	0 (0.00)	20	0 (0.00)	4.25	0 (0.00)	7.95	0 (0.90)	Timeout
med_7	34 (41.65)	14 (18.05)	1 (4.15)	440	0 (0.00)	119.58	0 (3.55)	134.45	0 (0.00)	Timeout
med_8	9 (15.95)	0 (0)	0 (0.00)	12	0 (0.00)	6.46	0 (0.00)	11.35	0 (0.30)	Timeout
med_9	17 (24.55)	2 (9.7)	0 (4.90)	269	0 (0.35)	143.04	0 (2.15)	123.2	0 (0.35)	Timeout
med_10	0 (0)	0 (0)	0 (0.00)	0	0 (0.00)	1.038	0 (0.00)	0.35	0 (0.00)	1.86
med_11	0 (3.2)	0 (0)	0 (0.00)	25	0 (0.00)	0.84	0 (0.00)	3.4	0 (0.00)	2.55
med_12	0 (0)	0 (0)	0 (0.00)	54	0 (0.25)	50.00	0 (0.00)	6.5	0 (0.60)	2.09
med_13	3 (13.35)	0 (0.5)	0 (0.50)	172	0 (0.00)	2.00	0 (0.00)	9.2	0 (0.05)	Timeout
med_14	0 (0.25)	0 (0)	0 (0)	59	0 (0.00)	0.25	0 (0.00)	10.9	0 (0.00)	2.26
med_15	0 (4.85)	0 (0)	0 (0.05)	72	0 (0.00)	1.48	0 (0.00)	7	0 (0.00)	25.05
med_16	30 (43.15)	1 (6.4)	1 (5.15)	733	0 (0.00)	7.38	0 (0.30)	21.65	0 (0.00)	Timeout
med_17	0 (3.55)	0 (0)	0 (0.0)	39	0 (0.00)	0.95	0 (0.00)	1.8	0 (0.15)	3.02
med_18	0 (8.2)	0 (3.1)	0 (6.05)	529	0 (0.00)	5.10	0 (0.00)	8.65	0 (0.30)	1.88
med_19	0 (9.25)	0 (3.15)	0 (5.45)	511	0 (0.00)	10.20	0 (0.00)	16.46	0 (0.50)	Timeout
med_20	0 (2.1)	3 (11.45)	2 (10.6)	457	0 (0.00)	8.06	0 (0.65)	24.55	0 (0.55)	10.1

For each approach except the SAT solver we give the minimum and average distance to feasibility and, if available, the average CPU time in seconds. For the SAT solver minisat, we give the time taken to find a feasible timetable

Table 2.13 Results obtained by the KIH, the SAT solver, and various approaches from the literature on the big instance set from [LPb]

Instance	GGA	LS	HSA		KIH		CB		TSA	Minisat
	Best (avg)	Best (avg)	Best (avg)	CPU (s)	Best (avg)	CPU (s)	Best (avg)	CPU (s)	Best (avg)	CPU (s)
big_1	0 (0)	0 (0)	0 (0.00)	0	0 (0.05)	49.30	0 (0.00)	68.4	0 (0.15)	13.93
big_2	0 (0.7)	0 (0)	0 (0.00)	283	0 (0.00)	43.33	0 (0.00)	94.2	0 (0.60)	20.63
big_3	0 (0)	0 (0)	0 (0.00)	447	0 (0.00)	37.21	0 (0.00)	55.4	0 (1.45)	32.96
big_4	30 (32.2)	8 (20.5)	0 (0.00)	406	0 (0.45)	427.17	0 (0.00)	133.5	0 (0.00)	Timeout
big_5	24 (29.15)	30 (38.15)	0 (1.10)	743	3 (9.70)	800.00	1 (3.20)	353.1	0 (0.00)	Timeout
big_6	71 (88.9)	77 (92.3)	5 (8.45)	893	24 (35.15)	800.00	10 (15.40)	319.25	1 (2.85)	Timeout
big_7	145 (157.3)	150 (168.5)	47 (58.30)	966	103 (108.60)	800.00	39 (46.65)	383.85	21 (29.25)	Timeout
big_8	30 (37.8)	5 (20.75)	0 (0.00)	210	0 (1.30)	701.88	0 (0.00)	136.75	0 (0.00)	Timeout
big_9	18 (25)	3 (17.5)	0 (0.05)	419	0 (0.75)	585.97	0 (0.00)	122.6	0 (0.00)	Timeout
big_10	32 (38)	24 (39.95)	0 (1.25)	660	4 (7.60)	800.00	0 (1.95)	319.6	0 (0.00)	Timeout
big_11	37 (42.35)	22 (26.05)	0 (0.35)	444	9 (12.65)	800.00	0 (2.35)	271.7	0 (0.00)	Timeout
big_12	0 (0.85)	0 (0)	0 (0.00)	240	0 (0.00)	23.75	0 (0.00)	54.55	0 (1.15)	20.26
big_13	10 (19.9)	0 (2.55)	0 (0.00)	274	0 (0.00)	83.17	0 (0.00)	84.15	0 (1.15)	146.70
big_14	0 (7.25)	0 (0)	0 (0.00)	271	0 (0.00)	29.40	0 (0.00)	67.8	0 (1.20)	21.68
big_15	98 (113.95)	0 (10)	0 (0.00)	255	0 (3.35)	756.12	0 (0.00)	83.95	1 (3.50)	Timeout
big_16	100 (116.3)	19 (42)	0 (2.00)	755	0 (0.00)	72.74	0 (0.00)	46.85	0 (0.00)	Timeout
big_17	243 (266.55)	163 (174.9)	76 (89.90)	998	0 (10.80)	798.61	0 (2.05)	554.35	12 (22.00)	Timeout
big_18	173 (194.75)	164 (179.25)	53 (62.60)	764	0 (0.00)	280.12	0 (1.70)	437.95	8 (13.55)	Timeout
big_19	253 (266.65)	232 (247.35)	109 (127.00)	998	124 (137.80)	800.00	40 (53.20)	410.45	37 (52.85)	Timeout
big_20	165 (183.15)	149 (164.15)	40 (46.70)	827	6 (9.95)	800.00	9 (14.05)	370.5	11 (15.05)	Timeout

For each approach except the SAT solver we give the minimum and average distance to feasibility and, if available, the average CPU time in seconds. For the SAT solver minisat, we give the time taken to find a feasible timetable

Discussion

Experiment 2 demonstrated the very good performance of the KIH on the small and medium instance sets. As a weakness of the approach, we identified scaling issues of the KIH algorithm on the big instances. These scaling issues result, to some extent, from the algorithm solving room assignment problems almost exclusively. In order to resolve these issues, the use of the bipartite matching algorithm needs to be restricted. This can possibly be achieved by the following two modifications, which can be considered in the future. First, other neighborhood structures could be added, which are not based on the Kempe-exchange. This would help to limit the excessive usage of the bipartite matching algorithm. Second, following a conflict-minimization approach with extra timeslots similar to the one from [CDGS12] seems to be a promising direction. The KIH could be used in order to move events from the extra timeslots to the regular timeslots.