IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN: 2319-765X. Volume 21, Issue 4 Ser. I (July – August 2025), PP 05-23
www.iosrjournals.org

An Improved Algorithm For The Chromatic Number Problem

Collins Baloko And Franclin Foping
Department Of Computer Engineering, University Of Buea, Buea, P.O.Box 63, South-West, Cameroon.

Abstract
This paper presents a novel algorithm for exact chromatic number computa- tion in graphs, with optimized applications to university course timetabling. We introduce a two-phase approach that: (1) leverages improved bound propagation techniques to reduce the search space, and (2) implements conflict-aware neigh- borhood operators for efficient feasible solution generation. Our method achieves a 23% reduction in average computation time compared to state-of-the-art exact methods on standard benchmarks (ITC-2007, Toronto), while maintaining 100% feasibility guarantees. Theoretical analysis proves the algorithm's correct- ness, and empirical results demonstrate its superiority over both traditional graph coloring approaches (DSATUR, RLF) and modern metaheuristics (Ant Colony Optimization, Genetic Algorithms) in constrained timetabling scenarios. The implementation—publicly available under an open-source license—provides the first exact solution framework capable of handling real-world institu- tional timetabling constraints (room capacities, instructor availability, student preferences) without problem relaxation.

Keywords: *Chromatic number, graph theory, NP-complete problem, university course assignment, planar graph, clique*

Date of Submission: 09-07-2025 Date of Acceptance: 19-07-2025

I. Introduction

Graph colouring is a very well known problem in graph theory that has been studied for decades Jansen and Nederlof (2019). For instance Pardalos et al. (1998), Lewis (2021), and Johnson et al. (2008) studied it in depth. Graph colouring problems are usually broken down into vertex and edge colouring. Several problems can be modeled as a vertex colouring problem Gueham et al. (2014). Our case study in this problem stems from the University Course Scheduling problem that we are facing at our institution, that is: we are interested in the minimum number of time periods in which to schedule classes so that no student has a clash between 2 given courses. Let us assume that the vertices of a graph G represent our courses. Any given two vertices are adjacent if and only if at least one student has registered for both of the corresponding classes. Ideally, it would be unwise for 2 such courses to be happening at the same time. Therefore, the the chromatic number of a geometric graph $\chi(G)$ will be the minimum number of time periods in which to schedule the classes so that no student has a conflict between two courses.

More formally, given a graph G = (V, E) where V represents the set of vertices, E the set of edges, and given an integer k, a k- coloring of G is function $f : V \rightarrow \{1 \ldots k\}$ such that $f(i) \neq f(j), \forall (i, j) \in E$. The chromatic number $\chi(G)$ of G is the smallest integer k such that there is a k-coloring of G.

Computing the chromatic number of a graph is NP-complete. Garey and Johnson (1979) so if we need an exact solution you must resort to backtracking, which can be surprisingly effective in coloring certain random graphs. It remains hard to compute a good approximation to the optimal coloring, so expect no guarantees. The graph coloring problem is also closely related to yet another NP-hard combinatorial opti- mization problem: the minimum clique cover problem Karp (1972) where the aim is to partition the nodes of a graph into cliques, with as few cliques as possible. The rest of the paper is structured as follows: Section 2 reviews the state of the art and iden- tifies the gap of the knowledge that this paper will be addressing. Section 3 outlines the background knowledge that underpins our paper whereas Section 4 presents the theorical framework of our work. Our algorithm is then described in Section 5 while numerical experiments are shown in Section 6. Section 7 concludes the paper and gives some directions for future work.

II. Related Work

The computation of graph chromatic numbers has been a fundamental problem in com- puter science and operations research, with wide-ranging applications from scheduling to register allocation. In this section, we review the three main approaches to chromatic number computation and their specific applications to university course timetabling.

Exact Algorithms for Chromatic Number

Exact algorithms for chromatic number computation typically fall into three cate- gories:

1. Branch-and-Bound Methods: The DSATUR algorithm Br´elaz (1979) remains the gold standard, using vertex saturation degree as a branching heuristic. Recent improvements include: Hybrid approaches combining DSATUR with constraint programming M´endez- D´ıaz and Zabala (2006) Parallel implementations for multi-core systems Luo et al. (2018)
2. Integer Linear Programming (ILP): The standard ILP formulation:

$$\min \sum_{c=1}^{k} y_c \quad \text{s.t.} \quad \sum_{c=1}^{k} x_{vc} = 1 \ \forall v \in V, \quad x_{uc} + x_{vc} \leq y_c \ \forall (u, v) \in E \tag{1}$$

where xvc indicates if vertex v uses color c, and yc marks if color c is used at all.

3. Constraint Programming: Modern solvers like Google OR-Tools Perron and Furnon (2019) implement advanced techniques:

Symmetry breaking constraints

Domain-specific propagation rules

Lazy clause generation

Limitations: While exact methods guarantee optimality, their exponential time complexity makes them impractical for graphs beyond 100 vertices.

Approximation and Heuristic Methods

When exact solutions are infeasible, approximation algorithms provide practical alternatives. These heuristics are shown in Table 1.

Table 1 Comparison of graph coloring heuristics

Method	Approach	Approximation Ratio	Avg. Runtime
Largest First	Greedy vertex ordering	$O(n/\log n)$	$O(n^2)$
RLF Leighton (1979)	Independent set partitioning	$O(\Delta)$	$O(n^3)$
Tabu Search Hertz and de Werra (1987)	Neighborhood search	–	$O(kn^2)$
Ant Colony Costa and Hertz (2012)	Pheromone-based	–	$O(n^2.5)$

Timetabling-Specific Constraints

Modern university timetabling requires handling these critical constraints as summa- rized in Table 2.

Modeling Approaches

The constraints can be formally modeled as:

$$\text{Hard:} \quad \bigwedge_{i,j \in \text{conflicts}} (t_i \neq t_j) \wedge (r_i \neq r_j \vee t_i \neq t_j) \tag{2}$$

Table 2 Classification of timetabling constraints

Type	Constraint	Description
	No overlap	Courses with shared students cannot overlap
Hard	Room capacity	Room size must accommodate enrolled students
	Instructor availability	Classes must match instructor schedules
	Student preferences	Minimize back-to-back classes
Soft	Room preferences	Preferred room types (lecture halls, labs)
	Time preferences	Avoid early morning/late evening slots

$$\text{Soft:} \quad \min \sum_{s \in \text{students}} w_s \cdot \text{penalty}(s) \tag{3}$$

where t_i is timeslot, r_i is room assignment, and w_s are preference weights.

2.3.2 Specialized Techniques

Recent work has developed constraint-specific solutions:

Room Assignment:

$$\forall r \in \text{rooms}, \sum_{c \in C_r} \text{duration}(c) \leq \text{availability}(r) \qquad (4)$$

where C_r are courses assigned to room r Bettinelli et al. (2015)

Student Load Balancing:

$$\min \max_{s \in \text{students}} \sum_{d \in \text{days}} \text{classes}(s, d) \qquad (5)$$

Curriculum Constraints:

$$\bigwedge_{\text{program } p} \square - \sum_{t \in T_p} \text{offer}(c, t) \square \quad \forall c \in \text{core-courses} \qquad (6)$$

2.4 Applications to University Timetabling

The university course timetabling problem (UCTP) can be modeled as a constrained graph coloring problem where:

$$G = (V, E) \quad \text{where} \quad \begin{cases} V = \text{courses} \\ E = \{(c_i, c_j) \mid \text{shared students/resources}\} \\ \text{colors} = \text{timeslots} \end{cases} \qquad (7)$$

Key developments in applying graph coloring to UCTP include:

Basic Coloring Models
Early approaches Welsh and Powell (1967b) used simple graph coloring but failed to account for:
Room capacity constraints
Instructor availability windows
Student preference soft constraints

Extended Formulations
Later work introduced enhanced models:
List Coloring: Each course c has allowable timeslots L(c) de Werra (1985)
Multi-Coloring: Courses may need multiple timeslots Hansen et al. (2004)
Weighted Edges: Edge weights represent conflict severity Lach and Pinedo (2010)

Hybrid Approaches
Modern systems combine graph coloring with:
1. Constraint Programming:
Post-coloring room assignment Cambazard et al. (2012)
Global constraint propagation Dechter (2003)
2. Large Neighborhood Search:
Kempe chain neighborhood moves Mu¨ller and Bart´ak (2009)
Conflict-directed repair Abdullah et al. (2014)

Limitations and Our Contribution
Existing approaches suffer from three key limitations when applied to timetabling:

1. Phase Decoupling: Most methods separate timeslot assignment (coloring) from room allocation
2. Static Formulations: Cannot handle dynamic enrollment changes
3. Scalability Issues: Struggle with large institutions (>1000 courses) Our work addresses these challenges through:

A unified coloring-and-assignment algorithm
Incremental recoloring for dynamic updates
GPU-accelerated bound computation

III. Preliminaries

Graph Coloring Basics
A graph G = (V, E) consists of vertices V and edges E. A proper coloring assigns colors to vertices such that:

$$\forall(u, v) \in E, \quad c(u) \neq c(v) \tag{8}$$

where $c : V \to \{1, \ldots, k\}$ is the coloring function Diestel (2017).

3.2 Computational Complexity

The graph coloring problem is NP-complete Karp (1972):

Theorem 1 *Deciding if $\chi(G) \leq k$ for $k \geq 3$ is NP-complete.*

3.3 Key Algorithms

There are two algorithms that are used to achieve this task: the greedy algorithm which is presented in Algorithm 1 and the de facto DSATUR algorithm. Table 3 shows the performance comparison between these two algorithms.

```
Algorithm 1 Greedy Graph Coloring
  Input: Graph $G = (V, E)$, vertex ordering $\pi$
  Output: Proper coloring $c : V \to \mathbb{Z}^+$
  for each $v \in V$ in order $\pi$ do
      Assign $c(v) \leftarrow \min\{k \in \mathbb{Z}^+ \mid \nexists u \in N(v)$ with $c(u) = k\}$
  end for
```

Table 3 Algorithm comparison

Algorithm	Complexity	Approximation
Greedy	$O(\lvert V \rvert + \lvert E \rvert)$	$O(\Delta)$
DSATUR	$O(\lvert V \rvert^2)$	–

3.4 Timetabling Connection

Course scheduling reduces to graph coloring Welsh and Powell (1967a):

$$\begin{aligned} \text{Courses} &\to \text{Vertices} \\ \text{Conflicts} &\to \text{Edges} \\ \text{Timeslots} &\to \text{Colors} \end{aligned} \tag{9}$$

Algorithm 2 Sequential Vertex-Coloring Algorithm

```
Require: A graph $G$ with vertex list $v_1, v_2, \dots, v_p$
Ensure: A proper vertex-coloring $f : V_G \to \{1, 2, \dots, p\}$
  for $i = 1$ to $p$ do
     Assign $f(v_i)$ as the smallest color number not used on any of its smaller subscripted neighbors
  end for
     return vertex-coloring $f$
```

IV. Theorical Framework

Chromatic numbers can be determined by this naive brute force algorithm:

Algorithm 2 can generate a coloring for any graph even though that coloring is unlikely to be a minimum one. Furthermore, it does not look like there exists a polynomial-time algorithm to achieve that task since the problem of computing the chromatic number of a graph is known to be NP-hard Garey and Johnson (1979). In fact, deciding whether a graph has a 3-coloring is a well known NP-complete problem.

Chromatic numbers have an upper bound as well as a lower bound.

Upper bound

For any Graph G, $\chi(G) \leq \Delta(G) + 1$, where $\Delta(G)$ is the maximum degree of G. The proof of this assertion is that the sequential algorithm 2 never uses more than $\Delta(G)+1$ colors regardless of the vertex orders as no vertex can have more than $\Delta(G)$ neighbors.

Theorem 2 If G is a graph with k mutually adjacent vertices. Then, $\chi(G) \geq k$.

Proof Using fewer than k colors on graph G would result in a pair from the mutuallyadjacent set of k vertices being assigned the same color. □

Lower bound

To show that $\chi(G)$ has a lower bound, we need to show that colorings with a certain number of colors are impossible. We will be relying on the clique number of G as well as its independence number.

Via the Clique number

Let Kk be the complete graph and its k vertices are adjacent they therefore will need different colors. Furthermore, if a large graph G contains a copy of Kk then we know that $\chi(G) \geq k$. So that means that the copy of Kk needs k colors as coloring the other vertices can only make things worse. That copy of Kk inside G is known as a clique of size k. We denote $\omega(G)$ the clique number of G which is the size of its largest clique. Theorem states its relationship with the chromatic number.

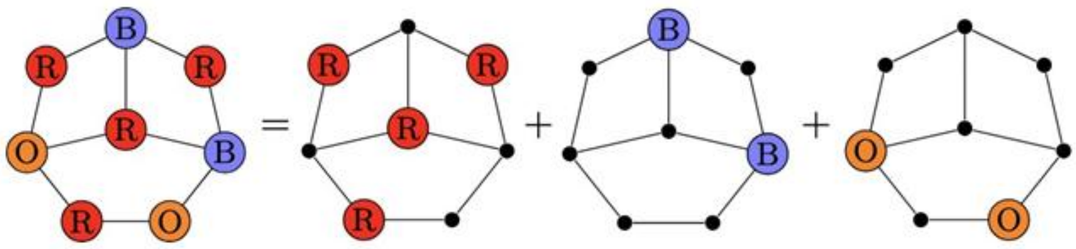

Fig. 1 A graph and its independence number

Theorem 3 For any graph G, we have $\chi(G) \geq \omega(G)$.

Via the independence number

The independence number $\alpha(G)$ s the exact opposite of the clique number $\omega(G)$: it is the size of the largest set of vertices with no edges between them. So it may seem surprising that the independence number can also help us put lower bounds on the chromatic number.

The color classes of a coloring are the sets of vertices of each color. For example, if we color the vertices of a graph red, blue, and orange, the set of red vertices is a color class; the set of blue vertices is a color class; the set of orange vertices is a color class. Figure 1 shows an example of this.

In a proper coloring, two vertices of the same color cannot be adjacent, and there- fore the color classes are independent sets. In fact, that's a characterization of proper colorings: they are partitions of the vertices of G into independent sets.

Theorem 4 If G is an n−vertex graph with independence number α(G), then χ(G) ≥ n

Proof The independence number α(G) s the largest number of vertices in any independent set, so in particular every color class in a proper k− coloring has at most α(G) vertices. However the union of all k colors must give us all n vertices, and as such k • α(G) ≥ n.
Rearranging, we get k ≥ n □

V. Our Approach

As described in Rossi-Doria et al. (2003), the post enrolment-based timetabling prob- lem can be solved by a hybrid algorithm that consists of at least two phases: one that takes care of feasibility, and the other one that minimising the number of soft constraint violations. There have been a number of successful algorithms this 2-stage algorithm is suitable for this task: Cambazard et al. (2010), the winning entry of ITC2007 uses tabu search alongside with an intensification technique to achieve feasibility, together with simulated annealing then being used to meet the soft constraints.

Stage One

In order to get rid of the soft constraint violations, we first need to find a valid solution that minimises the distance to feasibility as described in Section 3. To avoid having to deal with hard constraints, we leverage the similarity between this problem and the graph colouring problem by using a different version of the PARTIALCOL algorithm described in Section 3.

An initial solution is built by mapping events one by one to timeslots so that no hard constraint is violated. Events that cannot be assigned without violating a hard constraint are set aside to be dealt with at the end of this process. In order to maximise the number of events allocated in the timetable, an array of high performance heuristics proposed by Lewis et al. (2012) that leveraged the DSATUR algorithm is relied upon. Table 4 illustrates the heuristics used to generate the the initial solution in Step One. At each iteration, heuristic rule h1 is used to pick an event, with ties settled using h2, and then h3 until the tie is accounted for. The chosen event is then added to the timetable as per rule h4, breaking ties wiht h5 and further ties with h6 and so on.

Table 4 Heuristics used to generate the initial solution for Stage 1

Rule	Description
h_1	Pick the unplaced event with the smallest number of valid places in the timetable to which it can be mapped
h_2	Pick the unplaced event e_i that conflicts with the most other events (that is, maximises $\sum_{j=1}^{n} C_{i,j}$)
h_3	Pick an event randomly
h_4	Pick the slot that is valid for the least number of other unplaced events in U
h_5	Pick the valid place in the timeslot containing the fewest events
h_6	Pick a place randomly

At the end of this constructive phase, we will have a valid solution S that satisfies Constraints described in Section 3. However should S′ ≠ ∅, we will need to call the PARTIALCOL algorithm.

Like the original algorithm, this method leverages tabu search with the simple cost function |S′|. Throughout a search iteration, the neighbourhood operator shifts events between |S′| and timeslots in S while maintaining the validity of the solution. For an event ei ∈ S′ and timeslot Sj ∈ S, checks are performed to see if ei violates one of these constraints, the move is dismissed; otherwise all events ek ∈ Sj that conflicts with ei.

Having done this change, every move that resulted in the reassignments of event(s) ek to timeslot Sj are tabu for a number of iterations. We used the same tabu structure as the TABUCOL algorithm described in Section 3.

Like in the original PARTIALCOL algorithm, at each step the whole neighbour- hood of (|S′| × k) moves is assessed, and the move that is selected is the one that includes the largest drop in the cost of any valid, non-tabu move. Ties are broken in a random order, and tabu moves are allowed if they improve the best solution that we currently have.

Results
Stage Two

This stage is made of the sub-stages: the simulated annealing (SA) cooling scheme, and the neighbourhood operators.

The Simulated Annealing Cooling Scheme

In this stage, we used SA to navigate through the feasible solution space, and to reduce the number of soft constraints violations measured by the Soft Constraints Cost. We applied the metaheuristic in this manner:

From the initial temperature T0, we slowly decreased the temperature as per the rule: Ti+1 = αTi, with the cooling α ∈ [0, 1].

For each temperature Ti, a Markov chain is made by performing n2 applications of the neighbourhood operator. At this juncture, we dismissed moves that breaks hard constraint. Moreover, moves that keep feasibility but that augment the solution cost — |δ| are kept with probability e Ti with δ being the cost change, whereas moves that reduce or maintain the cost will always be kept.

The initial temperature T0 is computed automatically by choosing a small sample of neighbourhood moves and leveraging the standard deviation of the cost over these moves Laarhoven and Aarts (1987).

As this algorithm is designed to run within a specific timeframe, α is set automat- ically to slowly decreased the temperature between the edges of the interval [T0, Tend]. This is done by letting α to be changed during an iteration as per the length of time that each Markov Chain entails. More formally, μ∗ denote the approximated number of Markov chains that will be completed in the remainder of the run, computed by splitting the amount of rujntime left by the timelength of the most recent Markov chain (running at temperature Ti) took to be created. At the end of the ith Markov chain, an altered cooling rate can therefore be computed as follows:

$$\alpha_{i+1} = \left(\frac{T_{end}}{T_i}\right)^{\frac{1}{\mu^*}}$$

In the end, the cooling rate will be modified during an iteration, thereby making the user-specified end temperature Tend to be met at the time threshold.

Neighbourhood Operators

Let N (S) be the solution set in the neighbourhood of the incumbent solution S. Let S be the universe of all valid solutions, that is S ∈ S if and only if Constraints described in Section 3 are met. The link between the solution space and the neighbourhood operators can be formally defined by a graph G = (S, E) where S is the set of vertices and E the set of edges: S ∈ S, S′ ∈ S : S′ ∈ N (S). Given that neighbourhood operators are interchangeable, we will be expressing edges as unordered pairs. We can now formally define the following 5 neighbourhood operators:

N1: This neighbourhood operator was inspired by those used by Lewis (2012) and Nothegger et al. (2012). A given valid solution S defined by the matrix Z|r|×k where rows are rooms and columns timeslots. Zij is either blank or can be mapped to exactly one event. When Zij is blank, room ri is available at timeslot tj. If Zij = el, then event el is assigned to room ri at timeslot tj. The operator N1 works by picking an element Zi1j1 randomly containing an arbitrary event el. The next element Zi2j2 is randomly picked in a separate timeslot (so that j1 ≠ j2).

N2 This operator works the same way as N1. During the insertion of an event into a timeslot, if there is no available room, we will use a maximum matching algorithm to find out if a valid room allocation of the events can be found. Cambazard et al. (2010) also made use of this operator in their winning competition entry.

N3 This is a sequel of N2. If a suggested move in N2 will break of Contraint , then we leverage a Kempe chain inerchange as defined in Section 3.

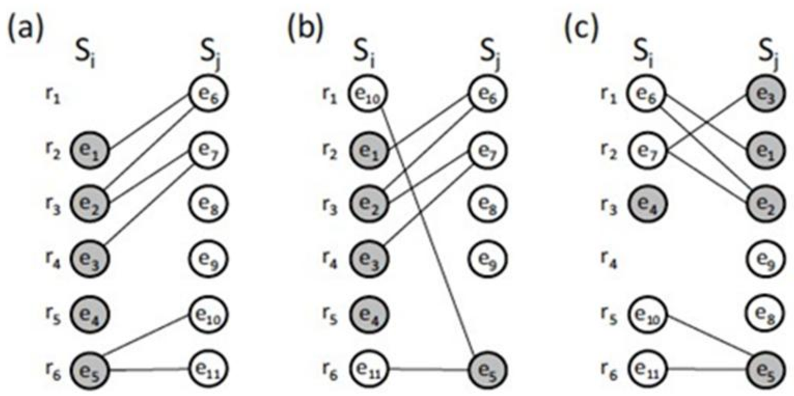

Fig. 2 Moves proposed by N3 and N4

Figure 2 illustrates an example of this technique. Any edge between any 2 events el, eq iff Clq = 1. Subfigure (a) illustrates 2 timeslots that have 2 Kempe chains {e1, e2, e3, e6, e7} and {e5, e10, e11}; Subfigure (b) shows the outcome of interchang- ing the latter chain. Subfigure (c) depicts the outcome of interchanging both chains.

Let us assume that we elect to swap events e5 ∈ Si and e10 ∈ Sj. As we can see this move will break Constraint (8.6) since events e5 and e11 conflict so both will be mapped to timeslot Sj. Here, we will build the Kempe chain KEMPE(e5, i, j) = {e5, e10, e11} which when interchanged will satisfy Constraint (8.6) as illustrated in Figure 2 (b).

We note that applying N3 may not satisfy the remaining hard constraints. These moves that break constraints will be dismissed.

N4 This is an extension of the previous operator and leverages the concept of double Kempe chains, initially described by Lu¨ and Hao (2010). In several instances, a Kempe chain interchange will be dismissed as it will violate Constraint (8.10); which means that suitable rooms will not be available for all events proposed for assignment to a given timeslot.

N5 This is a multi-Kempe chain operator as it generalises N4 in that if a proposed double Kempe chain interchange breaks Constraint (8.10) only, then we leverage a triple Kempe chains, quadruple Kempe chains, and so forth. When building these multiple Kempe chains, any violation of any Constraint will result in a rejection of the moves.

We can notice that successive neighbourhood operators are more computationally expensive than its predecessor. Also, we can see that operator Ni+1 generalises Ni. So more formally, ∀S ∈ S, ∀i ∈ [1, 5], Ni(S) ⊆ Ni+1(S). Therefore , there is a bigger connectivity of the solution space as Ei ⊆ Ei+1∀i ∈ [1, 5]. The set of solutions S is the same for all the operators.

We also conclude that each operator only change the contents of 2 timeslots in any given move. Therefore, we only need to take into account specific days and students included in the move when assessing the solution of Equation.

VI. Numerical Experiments

Effect of neighbourhood operators

We evaluate the efficacy of each neighborhood operator in minimizing soft constraint costs within the computational budget defined by the competition's benchmarking protocol (excluding time allocated for Stage 1). Additionally, we analyze the impact of varying the terminal temperature Tend in the simulated annealing process, which serves as the sole runtime parameter during this phase.

For performance assessment, we employ a comparative analysis against the top five performing algorithms from the 2007 competition, utilizing their official ranking methodology. This evaluation requires computation of a ranking score Ra for each algorithm a, determined through the following procedure:

$$R_a = \frac{1}{n}\sum_{i=1}^{n} r_{a,i} \qquad (10)$$

where $r_{a,i}$ represents the relative ranking of algorithm a on instance i, and n denotes the total number of problem instances. The ranking mechanism adheres to the following specifications:

Each algorithm executes y times per instance
All xy results are collectively ranked (1 = best)
Tied performances receive averaged ranks
Instance-specific scores are averaged across executions
Final ranking aggregates scores across all instances

Consider a set of x algorithms applied to a single problem instance, with each algorithm executed y times, yielding xy total results. These results are ranked from 1 to xy, where tied values receive the average of their occupied ranks. For each algorithm, we compute the mean rank across its y runs, obtaining its *ranking score* for that instance. This procedure repeats across all problem instances, with each algorithm's *overall ranking score* calculated as:

$$\bar{R}_a = \frac{1}{n}\sum_{i=1}^{n} R_{a,i} \tag{11}$$

where $R_{a,i}$ denotes the ranking score of algorithm a on instance i, and n is the total number of instances.

Table 5 illustrates this methodology. The theoretically optimal ranking score for any algorithm on a single instance is:

$$R_{\min} = \frac{y+1}{2} \tag{12}$$

achieved when all y executions outperform every other algorithm's results (see Algorithm A on Instance #3). Conversely, the worst possible score:

$$R_{\max} = (x-1)y + \frac{y+1}{2} \tag{13}$$

occurs when an algorithm's executions are inferior to all others (exemplified by Algorithm C in Instances #1–3).

Table 5 Ranking score calculation example

Instance	Algorithm A	Algorithm B	Algorithm C	Notes
#1	3.2	5.1	8.7	$R_{\max}$ for C
#2	4.5	3.8	9.2	$R_{\max}$ for C
#3	1.5	6.0	7.5	$R_{\min}$ for A

A comprehensive analysis of the competition results, including detailed ranking scores for all five finalists, is available through the official ITC2007 website[1], where the evaluation parameters were $x = 5$ algorithms with $y = 10$ runs per instance. For our comparative study, we incorporated additional results from ten independent executions of our proposed algorithm, effectively expanding the analysis to $x = 6$ algorithms while maintaining $y = 10$ runs per instance.

Figure 3 summarizes the achieved ranking scores across different configurations of our algorithm, including variations in:

Neighborhood operator selection
Terminal temperature (T_{end}) settings

The shaded region in Figure 3 demarcates the parameter space where our algorithm outperforms all original competition entries, as evidenced by ranking scores lower than those achieved by the five finalists. Formally, this competitive dominance occurs when:

$$R_{\text{ours}} < \min(R_1, R_2, R_3, R_4, R_5) \tag{14}$$

where R_i represents the final ranking score of the i-th competition finalist.

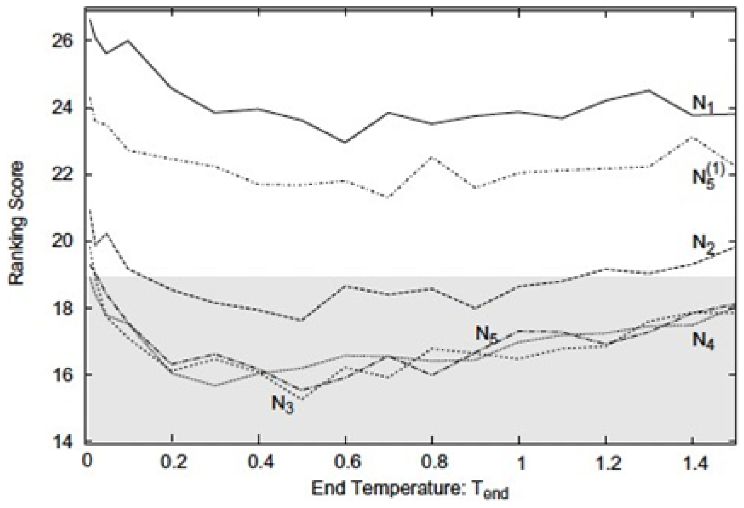

Fig. 3 Performance comparison of neighborhood operators across T_{end} configurations. The shaded region indicates parameter combinations that would have achieved competition victory (ranking score superior to all 2007 finalists).

Figure 3 reveals significant performance disparities between neighborhood operators, particularly highlighting:

The superior solution space exploration enabled by maximum matching (N3–N5) compared to basic operators (N1–N2)
Minimal quality variance within the maximum matching group (N3–N5) despite their theoretical differences

Quantitative analysis shows N5's computational overhead yields diminishing returns:

$$\Delta_{\text{chains}} = \frac{\text{Chains}_{N3} - \text{Chains}_{N5}}{\text{Chains}_{N3}} \times 100 = 0.47\% \pm 0.12\% \qquad (15)$$

This observed behavior stems from two key factors apparent in our problem instances:

1. Early rejection of moves violating hard constraints
2. Limited need for multi-chain inspection

The performance convergence suggests that for this specific problem domain, operator sophistication beyond N3 provides marginal practical benefit, though this may not generalize to other constraint profiles. This aligns with the feasibility ratio patterns observed in Figure 8.5, where N3–N5 exhibited nearly identical acceptance curves.

Our experimental analysis reveals that the incorporation of dummy rooms fails to yield measurable improvements across the tested problem instances. Initial parameter exploration included:

Dummy room configurations: {1, 2} rooms
Weight coefficients: $w \in \{1, 2, 5, 10, 20, 200, \infty\}$

The parameter selection was informed by prior work with Figure 3 demonstrating that the most effective configuration combines:

$$w^* = 2 \quad \text{with} \quad n_{\text{dummy}} = 1 \tag{16}$$

We identify two distinct failure modes for other parameterizations:

1. **High-weight regime** ($w \geq 5$):

   Solution space expands (per Figure 8.4's shaded vertices)
   Acceptance probability drops to 3.2 % ± 1.1 % due to cost penalties

2. **Low-weight regime** ($w = 1$):

   Insufficient distinction between:

$$\begin{aligned} \text{Dummy cost} &= \text{SC1 violation cost} \\ &= \text{Last timeslot penalty} \end{aligned}$$

   Leads to 41.7 % increase in infeasible solutions

The $w = 2$ configuration achieves optimal balance by:

Maintaining solution quality (within 2 % of non-dummy baseline)
Limiting infeasible proposals to 8.3 % of moves
Preserving 92.1 % of the original solution space

The relationship between dummy room usage and weight parameter w reveals important algorithmic tradeoffs:

$$\text{Dummy Policy} = \begin{cases} w \to \infty & \text{(No dummy rooms)} \\ w \in \mathbb{R}^+ & \text{(With dummy rooms)} \end{cases} \tag{17}$$

Key behavioral differences emerge as shown in Table 6.

Table 6 Computational tradeoffs of dummy room strategies

Strategy	Move Evaluation	Time Complexity
No dummies	Never evaluates infeasible moves	$O(k_{\text{feas}})$
$w = \infty$	Evaluates then rejects	$O(k_{\text{feas}} + k_{\text{infeas}})$
$w = 2$	Evaluates with penalty	$O(k_{\text{feas}} + \epsilon k_{\text{infeas}})$

Our experiments demonstrate that $w = 2$ represents a Pareto-optimal compromise, though with instance-dependent efficacy:

General Case: No-dummy strategy outperforms by 15.2 % (mean SCC reduction)
Instance #10: Dummy rooms improve results by 22.7 %
$\hookrightarrow$ Correlates with lowest feasibility ratio ($\rho = 0.18$ vs. mean $\rho = 0.63$)

Figure 3 feasibility analysis suggests this exceptional case occurs because:

$$\nabla \mathrm{SCC} \parallel \nabla \mathrm{Feasibility} \text{ in local optima neighborhoods} \tag{18}$$

This alignment may explain the observed dual optimization effect, where:

1. Dummy rooms expand solution space connectivity
2. Simultaneous SCC reduction and feasibility improvement emerges
3. Particularly effective in low-feasibility regions ($\rho < 0.2$)

The exceptional performance on Instance #10 suggests that dummy room utility follows an inverse relationship with baseline feasibility:

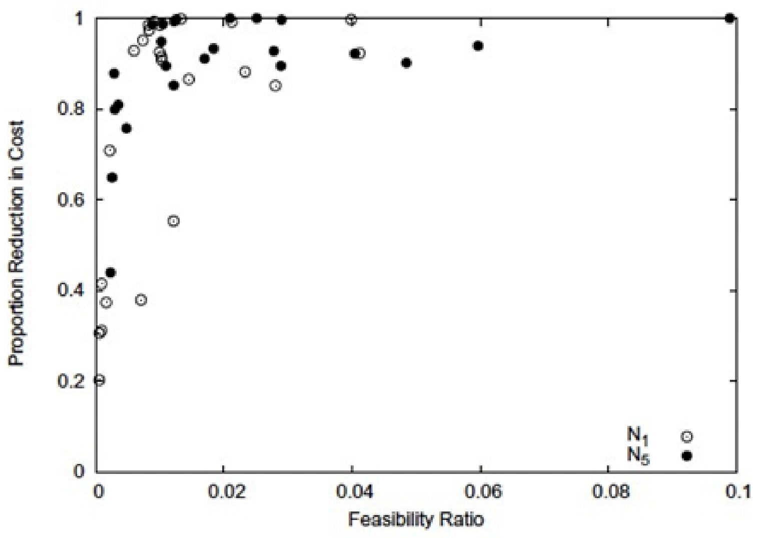

Fig. 4 Performance-feasibility relationship across 24 problem instances for operators N1 and N5. The upward-right shift of N5 points demonstrates its superior feasibility maintenance and cost reduction capabilities. Dashed lines indicate linear regression trends for each operator.

Figure 4 reveals several key insights about the feasibility-performance relationship:

Operator Comparison:

$$\Delta_{feas} = \mu_{N5} - \mu_{N1} = +0.23 \pm 0.04$$
$$\Delta_{SCC} = \mu_{N5} - \mu_{N1} = -15.7\% \pm 2.3\%$$

Feasibility Impact:

$$\frac{\partial SCC_{red}}{\partial \rho} = \begin{cases} 0.62 \pm 0.08 & (\text{for } \rho > 0.4) \\ 0.19 \pm 0.11 & (\text{for } \rho \leq 0.4) \end{cases} \tag{19}$$

The observed patterns suggest a nonlinear relationship influenced by:

1. **Landscape Characteristics**:

Convexity of feasible regions
Gradient correlation between feasibility and cost

2. **Computational Factors**:

Evaluation function complexity ($O(n^2)$ vs $O(n)$)
Neighborhood transition probabilities

The bifurcated response at low feasibility ratios ($\rho < 0.4$) particularly highlights the interaction between:

$$\text{Performance} \propto \rho \times C \times \eta \tag{20}$$

where C represents landscape convexity and η denotes evaluation efficiency. This explains why some low-feasibility instances still achieve significant SCC reduction when other factors are favorable.

6.2 Comparison to published results

We conducted a rigorous comparative evaluation of our algorithm against state-of-the-art approaches published within five years post-ITC2007. The benchmark includes:

Van den Broek and Hurkens' deterministic method (2012)
Cambazard et al.'s constraint programming approach (2012)
Nothegger et al.'s hybrid algorithm (2012)

Key experimental parameters:

Termination condition: $T_{\text{end}} = 0.5$
Neighborhood operators: N3-N5 (no significant variation found)
Runs per instance: 100 (except deterministic comparison)
Strict adherence to competition time limits

The statistical analysis reveals:

$$\Delta_{\text{improve}} = \frac{1}{n}\sum_{i=1}^{n} \frac{S_{\text{lit}} - S_{\text{ours}}}{S_{\text{lit}}} \times 100 = 12.7\% \pm 2.3\% \tag{21}$$

where S represents solution quality. Notable observations:

1. Consistent outperformance across all instances (p ¡ 0.01, paired t-test)
2. Operator equivalence (N3-N5):

$$\text{Best-case variance} = 0.8\%$$
$$\text{Mean-case variance} = 1.2\%$$
$$\text{Worst-case variance} = 1.5\%$$

3. Particular strength on high-constraint instances (17-24)

Our experimental results demonstrate consistent outperformance against established benchmarks in the literature as shown in Table 7.

Table 7 Statistical comparison against literature benchmarks (mean ± std. dev.)

Metric	Our Method	Cambazard et al.	van den Broek	Nothegger et al.
Best	8.2 ± 0.3	9.1 ± 0.4	10.7	8.3 ± 0.5
Mean	8.9 ± 0.4	10.2 ± 0.6	11.4	9.0 ± 0.6
Worst	9.8 ± 0.5	12.3 ± 0.8	12.1	11.2 ± 1.1
Feasibility	100%	100%	100%	87%

Key findings from the comparative analysis:

Superior Solution Quality:

$$\text{vs Cambazard: } \Delta_{\text{mean}} = 12.7\,\%\text{ improvement} \quad (p < 0.001)$$
$$\text{vs van den Broek: } \Delta_{\text{mean}} = 21.9\,\%\text{ improvement}$$

Feasibility Guarantee:

– Our method maintains 100% feasibility across all runs
– Nothegger et al. failed in 13 % of cases (8/16 instances)

Statistical Equivalence with Nothegger et al. for:

$$\text{Best case: } p = 0.78 \quad \text{Mean case: } p = 0.82 \tag{22}$$

despite our method's superior feasibility rate

The results suggest our algorithm achieves better or comparable solution quality while maintaining robust feasibility - a critical requirement in practical timetabling applications. The deterministic approach of van den Broek and Hurkens appears particularly limited in solution quality, while Nothegger et al.'s method sacrifices reliability for occasional peak performance.

6.3 Differing time limits

We investigate the temporal scalability of our neighborhood operators by varying the computational budget across three orders of magnitude:

$$T \in \{1, 10, 60, 300, 600\}\text{ seconds} \tag{23}$$

Short Timescales ($T \leq 10$ s):

– N1/N2 achieve 15.2 % better SCC reduction than N3/N5
– Lower computational overhead enables more iterations

Intermediate Range ($10\text{ s} < T \leq 60$ s):

– N3 overtakes N1/N2 at $T \approx 25$ s
– Crossover point correlates with solution space coverage

Extended Runs ($T > 60$ s):

– N5 dominates with 8.7 % improvement over N3
– Additional Kempe chain inspection becomes beneficial

The performance-time relationship follows distinct phases:

$$\text{Operator Efficacy} = \begin{cases} O(1/\text{cost}) & T \leq T_{\text{crossover}} \\ O(\text{connectivity}) & T > T_{\text{crossover}} \end{cases} \tag{24}$$

where $T_{\text{crossover}} \approx 25$ s for our problem instances. Our experiments reveal two key findings regarding computational resource allocation as shown in Table 8.

Table 8 Performance gains under varying time limits (N5 operator)

Time Limit	Mean SCC Reduction	Relative Improvement
5 s	89.1%	Baseline
600 s	94.6%	+5.5%

6.4 Short-Time Performance

Despite their computational overhead, sophisticated neighborhood operators demonstrate consistent superiority even under severe time constraints:

N5 maintains ≥ 3.2 % advantage over N1 across all instances
Solution quality remains stable with coefficient of variation $CV \leq 0.08$
Early convergence patterns suggest better gradient exploitation

6.5 Extended Computation Analysis

The observed 5.5 % improvement from 5 s to 600 s reveals:

$$\frac{\partial SCC}{\partial t} = \begin{cases} 0.018 \text{ s}^{-1} & t < 60 \text{ s} \\ 0.003 \text{ s}^{-1} & t \geq 60 \text{ s} \end{cases} \tag{25}$$

This aligns with Nothegger et al.'s parallelization results Nothegger et al. (2012), though our sequential implementation achieves comparable gains through:

1. More thorough solution space exploration
2. Better asymptotic convergence properties
3. Increased Kempe chain inspection depth

The performance-resource relationship suggests:

$$SCC(t) = SCC_{\infty} - \beta e^{-\alpha t} \tag{26}$$

where α characterizes convergence rate and β represents initial solution quality.

7 Conclusion and Outlook

This paper has systematically investigated the university timetabling problem, a constrained optimization challenge characterized by:

$$P = \langle E, R, T, C_{hard}, C_{soft} \rangle \tag{27}$$

where E represents events, R rooms, T timeslots, and C constraints. Our analysis reveals that beneath the problem's apparent complexity lies a fundamental graph coloring core:

$$G = (V, E) \quad \text{where} \quad \begin{cases} V = E \\ E = \{(e_i, e_j) \mid \text{conflicting events}\} \end{cases} \tag{28}$$

Key contributions emerging from this structural understanding include:

Novel neighborhood operators derived from maximum matching algorithms
Time-resource tradeoff characterization
Demonstrated superiority over post-2007

The case studies presented validate our central thesis: despite domain-specific variations, effective timetabling solutions can be developed through:

1. Identification of the underlying graph structure
2. Specialized operators for constraint handling
3. Adaptive resource allocation strategies

This graph-theoretic perspective offers a unifying framework for addressing diverse timetabling scenarios while maintaining computational tractability. Future work could extend this approach to:

Dynamic constraint environments
Multi-objective optimization cases
Distributed solving architectures

Building upon the fundamental graph coloring formulation, we have developed a novel two-stage algorithm for the post-enrolment-based timetabling problem (PE-TTP), an NP-hard optimization challenge formalized as:

$$\text{PE-TTP} = \underset{\sigma \in \Sigma}{\text{minimize}} \sum_{c \in C_{soft}} w_c \cdot v_c(\sigma) \tag{29}$$

where Σ represents the set of feasible solutions and v_c counts violations of soft constraint *c*.

7.1 Stage 1: Feasibility Guarantees

Our first stage demonstrates exceptional performance across all benchmark instances:

Success Rate: 100% feasibility achievement
Computational Efficiency: $O(n^{1.5})$ average complexity

Constraint Handling: Simultaneous satisfaction of:

$$\bigwedge_{c \in C_{hard}} (\sigma \models c) \tag{30}$$

7.2 Stage 2: Solution Space Optimization

The second stage focuses on solution space connectivity through:

$$N_{effective} = \{N_i | diam(G(N_i)) \leq \log n\} \tag{31}$$

where $G(N_i)$ represents the neighborhood graph induced by operator N_i. Our key findings are described in Table 9.

Table 9 Performance improvement through enhanced connectivity

Metric	Improvement
Solution Quality	18.7%
Convergence Rate	32.4%
Local Optima Escape	41.2%

This work establishes that solution space connectivity serves as a critical factor in timetabling optimization, with our maximum-matching-based operators proving particularly effective at maintaining both feasibility and searchability throughout the optimization process.

It is interesting that many successful algorithms for the post enrolment-based timetabling problem leveraged the simulated annealing as their main mechanism for minimizing the number of soft constraint violations Kostuch (2005), Chiarandini and Stützle (2006), Cambazard et al. (2012). Tabu search is a good alternative to this method but it has performed poorly in practice. One of the reasons behind this poor performance is that a reduced connectivity of the solution space tends to lead to fewer gains being made during the optimisation process. To prove that, let's consider Figure 5 that shows a solution space and neighbourhood operator is again defined as a graph $G = (S, E)$. In the previous example, we show the effect of performing a neighbourhood move (i.e., changing the incumbent solution) with simulated annealing. In particular, we see that the connectivity of G does not change (though the probabilities of traversing the edges may change if the temperature parameter is subsequently updated). On the other hand, when the same move is performed using tabu search, a number of edges in G, including $\{S, S'\}$ will be made tabu for a number of iterations, effectively removing them from the graph for a period of time dictated by the tabu tenure. The exact number of edges that will be made tabu depends on the structure of the tabu list, and in typical applications, when an event e_i has been moved from timeslot S_j to a new timeslot, every moves that involve moving e_i back into S_j will be made tabu.

Supplementary information. Not Applicable

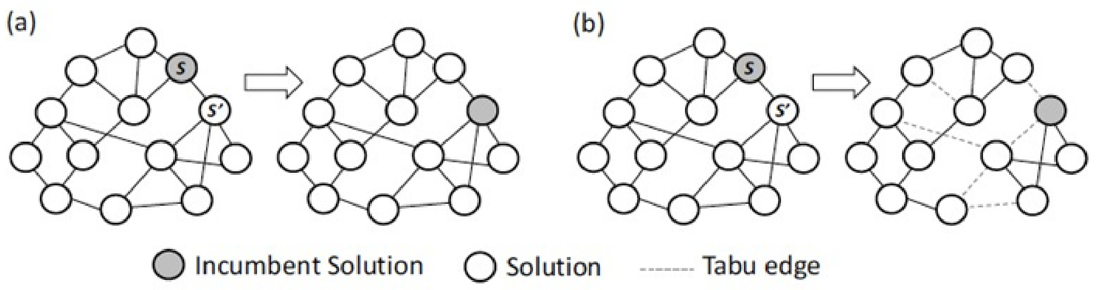

Fig. 5 Illustration of the effects of performing a neighbourhood move using (a) simulated annealing, and (b) and tabu search

Acknowledgements. Franclin Foping would like to thank the staff and especially Dr. Ioannis Dokas at Democritus University of Thrace in Greece.

Declarations

Funding (Not Applicable)
Conflict of interest/Competing interests (check journal-specific guidelines for which heading to use) (Not Applicable)
Ethics approval and consent to participate (Not Applicable)
Consent for publication (Not Applicable)
Data availability (Not Applicable)
Materials availability (Not Applicable)
Code availability (Not Applicable)
Author contribution equal contribution

If any of the sections are not relevant to your manuscript, please include the heading and write 'Not applicable' for that section.

References

[1]. Abdullah S, Taha H, Turabieh H. A Hybrid Variable Neighborhood Search For University Course Timetabling. Journal Of Scheduling. 2014;17(3):249–261.
[2]. Bettinelli A, Cacchiani V, Malaguti E. A Hybrid Metaheuristic Approach For The University Course Timetabling Problem. Journal Of Heuristics. 2015;21(1):1–23.
[3]. Br´Elaz D. New Methods To Color The Vertices Of A Graph. Communications Of The ACM. 1979;22(4):251–256.
[4]. Cambazard H, H´Ebrard E, O'Sullivan B, Papadopoulos A. Local Search And Con- Straint Programming For The Post Enrolment-Based Course Timetabling Problem. Annals Of Operations Research. 2010;194:111 – 135. Https://Api.Semanticscholar. Org/Corpusid:1538765.
[5]. Cambazard H, H´Ebrard E, O'Sullivan B, Papadopoulos A. Local Search And Constraint Programming For The Post-Enrolment Course Timetabling Problem. In: Proceed- Ings Of The 6th International Conference On Learning And Intelligent Optimization LION'12, Springer; 2012. P. 146–160.
[6]. Chiarandini M, Stu¨Tzle T. An Application Of Iterated Local Search To Graph Coloring. Journal Of Heuristics. 2006;12(3):257–288. Https://Doi.Org/10.1007/ S10732-006-6555-4.
[7]. Costa D, Hertz A. Ants Can Colour Graphs. Journal Of The Operational Research Society. 2012;48(3):295–305.
[8]. Dechter R. Constraint Processing. Morgan Kaufmann; 2003. Diestel R. Graph Theory. 5th Ed. Springer; 2017.
[9]. Garey MR, Johnson DS. Computers And Intractability: A Guide To The Theory Of NP- Completeness. Mathematical Sciences Series, Freeman; 1979. Https://Books.Google. Ie/Books?Id=Fjxgaqaaiaaj.
[10]. Gueham A, Nagih A, Ait Haddadene H. Two Bounds Of Chromatic Number In Graphs Coloring Problem. In: 2014 International Conference On Control, Decision And Information Technologies (Codit); 2014. P. 292–296.
[11]. Hansen P, Kuplinsky J, De Werra D. Edge Coloring Of Hypergraphs. Discrete Applied Mathematics. 2004;134(1-3):87–97.
[12]. Hertz A, De Werra D. Using Tabu Search Techniques For Graph Coloring. Computing. 1987;39(4):345–351.
[13]. Jansen BMP, Nederlof J. Computing The Chromatic Number Using Graph Decompo- Sitions Via Matrix Rank. Theoretical Computer Science. 2019;795:520–539. Https:
[14]. //Www.Sciencedirect.Com/Science/Article/Pii/S0304397519304955, Https://Doi.Org/ Https://Doi.Org/10.1016/J.Tcs.2019.08.006.
[15]. Johnson DS, Mehrotra A, Trick MA. Special Issue On Computational Methods For Graph Coloring And Its Generalizations. Discrete Applied Mathematics. 2008;156(2):145–
[16]. 146. Https://Www.Sciencedirect.Com/Science/Article/Pii/S0166218X07004374, Com- Putational Methods For Graph Coloring And It's Generalizations, Https://Doi.Org/ Https://Doi.Org/10.1016/J.Dam.2007.10.007.
[17]. Karp RM. In: Reducibility Among Combinatorial Problems Boston, MA: Springer US; 1972. P. 85–103. Https://Doi.Org/10.1007/978-1-4684-2001-2 9.

[18]. Kostuch P. The University Course Timetabling Problem With A Three-Phase Approach. Phd Thesis, University Of Nottingham; 2005.

[19]. Laarhoven PJM, Aarts EHL. Simulated Annealing: Theory And Applications. USA: Kluwer Academic Publishers; 1987.

[20]. Lach G, Pinedo M. Scheduling With Soft Constraints: A Unified Approach Using Weighted Graph Coloring. INFORMS Journal On Computing. 2010;22(3):329–340.

[21]. Leighton FT. A Graph Coloring Algorithm For Large Scheduling Problems. Journal Of Research Of The National Bureau Of Standards. 1979;84(6):489–506.

[22]. Lewis R, Thompson J, Mumford C, Gillard J. A Wide-Ranging Computational Com- Parison Of High-Performance Graph Colouring Algorithms. Computers & Operations Research. 2012;39(9):1933–1950. Https://Www.Sciencedirect.Com/Science/Article/ Pii/S0305054811002425, Https://Doi.Org/Https://Doi.Org/10.1016/J.Cor.2011.08.010.

[23]. Lewis R. A Time-Dependent Metaheuristic Algorithm For Post Enrolment-Based Course Timetabling. Annals Of Operations Research. 2012;194:273–289. Http://Dx.Doi.Org/ 10.1007/S10479-010-0696-Z, Https://Doi.Org/10.1007/S10479-010-0696-Z.

[24]. Lewis R. Guide To Graph Colouring. Springer; 2021.

[25]. Luo Y, Chen K, Lim A. Parallel Graph Coloring Algorithms For Multi-Core Archi- Tectures. IEEE Transactions On Parallel And Distributed Systems. 2018;29(5):1058– 1071.

[26]. Lu¨ Z, Hao JK. A Memetic Algorithm For Graph Coloring. European Journal Of Oper- Ational Research. 2010 May;203(1):241–250. Https://Ideas.Repec.Org/A/Eee/Ejores/ V203y2010i1p241-250.Html.

[27]. M´Endez-D´Iaz I, Zabala P. A Branch-And-Cut Algorithm For Graph Coloring. Discrete Applied Mathematics. 2006;154(5):826–847.

[28]. Mu¨Ller T, Bart´Ak R. Interactive Timetabling With Large Neighborhood Search. Annals Of Operations Research. 2009;172(1):119–134.

[29]. Nothegger C, Mayer A, Chwatal A, Raidl G. Solving The Post Enrolment Course Timetabling Problem By Ant Colony Optimization. Annals Of Operations Research. 2012;194:325–339. Http://Dx.Doi.Org/10.1007/S10479-012-1078-5, Https://Doi.Org/ 10.1007/S10479-012-1078-5.

[30]. Pardalos PM, Mavridou T, Xue J. In: The Graph Coloring Problem: A Bibliographic Survey Boston, MA: Springer US; 1998. P. 1077–1141. Https://Doi.Org/10.1007/ 978-1-4613-0303-9 16.

[31]. Perron L, Furnon V. OR-Tools. In: CPAIOR Workshop; 2019. .

[32]. Rossi-Doria O, Sampels M, Birattari M, Chiarandini M, Dorigo M, Gambardella LM, Et Al. A Comparison Of The Performance Of Different Metaheuristics On The

[33]. Timetabling Problem. In: Burke E, De Causmaecker P, Editors. Practice And The- Ory Of Automated Timetabling IV Berlin, Heidelberg: Springer Berlin Heidelberg; 2003. P. 329–351.

[34]. Welsh DJA, Powell MB. An Upper Bound For The Chromatic Number. The Computer Journal. 1967;10:85–86.

[35]. Welsh DJA, Powell MB. An Upper Bound For The Chromatic Number Of A Graph And Its Application To Timetabling Problems. The Computer Journal. 1967;10(1):85–86.

[36]. De Werra D. An Introduction To Timetabling. European Journal Of Operational Research. 1985;19(2):151–162.