Algorithms for the Restricted Linear Coloring Arrangement Problem

Bachelor's Thesis in Informatics Engineering

Department of Computer Science

Author: Joan Llop Palao
Director: María José Serna Iglesias

Defence date: Thursday 04th July 2019

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Abstract

The aim of this project is to develop efficient algorithms for solving or approximating the Minimum Restricted Linear Coloring Arrangement Problem. It is the first approach to its algorithms, and we will face the problem from different perspectives: constraint programming, backtracking, greedy, and genetic algorithms. As a second goal we are interested in providing theoretical results for particular graphs.

Acknowledgements

I would like to thank Professor Maria Serna from the Department of Computer Science of UPC for having proposed the project and for having directed my Bachelor's Thesis. This project would not have been possible without her contribution and guidance.
I would also like to thank Professor Pierre Flener from the Department of Information Technology of Uppsala University for his great lectures in the Combinatorial Optimisation and Constraint Programming course. Also, executing the algorithms for the project might have been impossible without using the Uppsala University computers. Therefore, I would like to express my deep gratitude to Professor Pierre Flener and to the Uppsala University for having helped me.

I would like to offer my special thanks to my parents and sister for their understanding.

Contents

List of Figures

List of Tables

Chapter 1

Project Definition

1.1 Introduction

1.1.1 The Problem

Layout problems are combinatorial optimization graph problems based on finding a labeling of the vertices such that satisfy certain conditions. Labeling problems are an important field inside graph theory with a very wide range of applications, which include optimization of networks for parallel computer architectures, VLSI circuit design, information retrieval, numerical analysis, computational biology, graph theory, scheduling and archaeology [5]. The problem of finding an optimal assignment of numbers to vertices (or linear arrangement) was first introduced by Harper, L. 1964 [12]. A minimum linear arrangement (minLA) can be defined as a one-to-one mapping between the vertices of a graph and the positions of a line such that minimizes the sum over all edge lengths. In the words of M. Brufau [28]: minLA can be defined as finding the best way of labelling each vertex of a graph in such a manner that the sum of the induced distances between two adjacent vertices is minimum.

The Minimum Linear Arrangement Problem (minLA) has been widely studied by the computer science community and one version of minLA is the Minimum Linear Coloring Arrangement Problem (minLCA). In this version, instead of labelling each vertex with a different integer, we group them with the condition that two adjacent vertices cannot be in the same group, or equivalently, we constraint the vertex labelling to be a proper colouring of the graph. This problem has been studied by [1] and [28].
In this project we will study a new version of a linear arrangement: The Minimum Restricted Linear Coloring Arrangement Problem (minRLCA) which can be described as finding the best way of labelling each vertex of the graph into a restricted number of groups of limited size, where the subgraph, induced by the vertices of each group, can be restricted to be connected or to be an independent set. Let's see an example with the following graph:

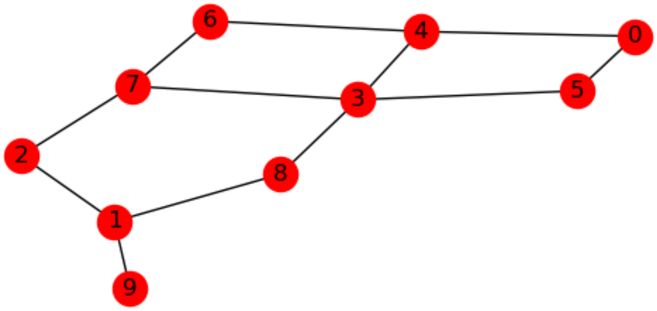


Figure 1.1: Simple graph of 10 vertices and 12 edges

Example 1.1.1. Imagine that we ask to label each vertex into 5 groups of size 2, and we ask for each group to be connected. As always each group will be aligned and the cost will be calculated as the sum of the distances between adjacent vertices.

As we can see the vertices 0 and 5 has been assigned to the first group, the vertices 4 and 6 has been assigned to the second group, etc. The cost is the sum of the distances of the edges, for

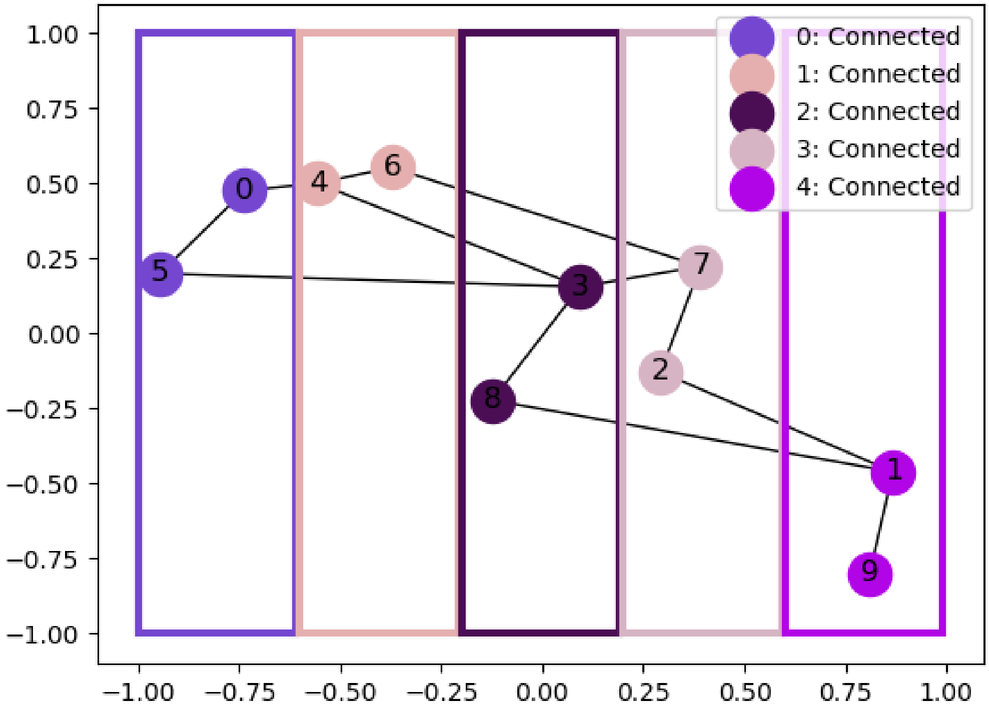


Figure 1.2: Solution to the problem of the Example 1.1.1 on the graph 1.1

instance the cost of this solution is 10: the sum of the distances of {0, 5}: 0, {0, 4}: 1, {5, 3}: 2, {4, 6}: 0, {4, 3}: 1, {6, 7}: 2, {3, 8}: 0, {3, 7}: 1, {7, 2}: 0, {8, 1}: 2, {2, 1}: 1 and {1, 9}: 0. These solution has been proven to have the minimum cost, this does not mean that is the only solution, in fact there are more solutions even with the same cost, for example if we consider the mirror of figure 1.2.

Insted of asking for each group to be connected we could have asked to be independent sets or we could have ask for no restrictions on the form of the group.

1.1.2 Goals of the project and scope

The main goal of this project is the study of The Minimum Restricted Linear Coloring Arrangement problem (minRLCA), a new generalization of the Minimum Linear Coloring Arrangement problem (minLCA) and of the well-known Minimum Linear Arrangement problem (minLA). Initially, we want to establish the problem complexity in a classical way, and from there on continue with some theoretical and practical results on certain types of graphs. We want to develop some exact algorithms, which can be done by using constraint programming or backtracking algorithms. We want to develop some greedy algorithms. Finally, we will try to develop a genetic algorithm and evaluate their performance on different classes of graphs such as bipartite graphs and random graphs generated using a binomial distribution for the edges. To ensure that this goals will be accomplished there will be a weekly meeting with the director of the project.

1.1.3 Interest of the problem

The minimum Restricted Linear Coloring Arrangement problem (minRLCA) can be easily seen 2.2 as a general case of the minimum Linear Arrangement problem (minLA). The minLA problem is a well-known problem studied among others by Jordi Petit [19] [20], and it has been shown NP-Hard and its decisional version has been proven NP-Complete [9]. So, we can expect that minRLCA will be computationally difficult and if so, theoretical and practical results would be useful for the whole computer science community.
Another problem that can be a particular case of the minRLCA is the minimum Linear Coloring Arrangement problem but other possibilities arise from limiting the number of groups (this can be seen as limiting the number of colours in minLCA) or limiting the size of each group. For example if we limit the number of groups to 2, the size of each group to the half of the number of vertices, and for both groups we restrict that the subgraph formed by the vertices of each group to be connected, we have a version of the balanced connected partition where we minimize the number of edges between partitions, which is a version of the maximally balanced connected partition problem that has been widely studied [7].
The main reason to study this problem is because the Algorithmics, Bioinformatics, Complexity and formal Methods(ALBCOM) research group from the Computer Science department (CS) of UPC · BarcelonaTech is interested in this problem and the proposal of its study came directly from them. In addition, we have seen this problem is connected to a wide range of well-known NP-hard problems and therefore we hope that this project will benefit the computer science community.

1.1.4 State of the art

The minimum Linear Arrangement problem (minLA) has been largely studied. Josep Díaz, Jordi Petit, and Maria J. Serna gave a great historical perspective in [5]:

> The Minimum Linear Arrangement problem (MinLA) was first stated in [Harper 1964, [12]]. Harper's aim was to design error-correcting codes with minimal averages absolute errors on certain classes of graphs. Latter, this problem was also considered in [Mitchison and Durbin 1986, [18]] as an over-simplified model of some nervous activity in the cortex. MinLA also has applications in single machine job scheduling [Adolphson 1977, [14], Ravi et al. 1991, [21]] and in graph drawing [Shahrokhi et al. 2001, [23]].

In [19], Jordi Petit stated:

> The extensive experimentation and the benchmarking we have presented suggests that the best heuristic to approximate the MinLA problem on sparse graphs is Simulated Annealing when measuring the quality of the solutions. However, this heuristic is extremely slow whereas Spectral Sequencing gives results not too far from those obtained by Simulated Annealing, but in much less time. In the case that a good approximation suffices, Spectral Sequencing would clearly be the method of choice.

Then Koren and Harel published [13], and their solutions were an improvement in the execution time of Petit's.
More recently, in [20] (2011) Jordi Petit concluded:

> With regard to the experimental results for the MinLA problem, one can remark that the current research on lower bounding techniques based on linear programming is very promising, having established in some cases that heuristics yield solutions close to the optima. This may suggest that further development of heuristics for this problem is not worth, whereas the current lower bounding methods should be improved to be applied to larger graphs.

We can see that the minLA problem has been largely studied and many of the research possibilities have been already explored.
In [1] I. Sánchez did an initial study of a new problem: the minimum linear coloring arrangement problem (minLCA), which is a modification of the minLA problem. Instead of having a bijective mapping between the set of vertices and a set of integers, it groups the vertices forming a proper coloring. It is because of these similarities that some of the algorithms developed for minLA could give ideas for developing the algorithms for minLCA. I. Sánchez established the problem complexity and developed three types of algorithms: exact, greedy and local search algorithms. For the exact algorithms he developed an integer programming model. Which concluded that the algorithm took too much time to execute and did not get to an optimal solution except for very few cases. The greedy algorithms are based on breadth-first search and for the local search algorithms the simulated annealing algorithm was used.
In [28] M. Brufau continued the study of minLCA. From a theoretical point of view, it broadened the results for particular graphs families such as k-trees, complete balanced k-partite graphs and bounded treewidth graphs. M. Brufau concluded:

> ... we have been able to prove that the MinLCA problem with a fixed number of colours and bounded treewidth is in the class FPT. We also have been able to disprove the conjecture stated in [1] that the optimal value for the MinLCA problem can be obtained using the chromatic number of the graph. We have done so by using a counterexample with two complete balanced 3-partite graphs. ... We have devised an exact algorithm based on backtracking and three similar heuristic algorithms based on a maximal independent set approach. ... We have been able to determine the best approach for the maximal independent set algorithm and for which families of graphs our algorithms have a better behaviour.

The minimum restricted linear coloring arrangement problem (minRLCA) is a new variation on minLCA, we introduce restrictions in the size and the number of the groups and the characteristics of the subgraphs induced by these groups. For this reason some of the algorithms developed for minLCA can give ideas for developing the algorithms for minRLCA, but due to their differences, the algorithms need to be different.

In the previous sections we have seen that a specific case of this problem has a similar, well studied, problem. The Maximally Balanced Connected Partition problem has been studied from different points of view: [2]. And has many applications in real world such as scheduling [15]. One approach used in [7] to solve the problem is Genetic Algorithms which makes interesting for us to try to solve the minRLCA problem using genetic algorithms as well.

1.2 Project planning

1.2.1 Schedule

The project is planned to take four and a half months, from the 18^{th} of February 2019 until the 5^{th} of July 2019. Having that in mind, the deadline for the follow-up milestone will be set around the 5^{th} of June, 21 days before the project deadline and one month before the presentations (this deadline is not definitive).
Writing the report after doing all the experiments is not an option because of the size of the project. The method chosen in this project is to write the report in every phase. That is usually indicated as ”Formalisation and Analysis”. The project planning, explained in the following sections, is also available as a Gantt chart at Figure 1.3.

1.2.2 Resources

The resources included in this project are:

1. **Computer cluster** The use of computer cluster is shown in the gantt chart as execution time. This time is clearly above the real execution time, this way we can be sure that each phase of the project can be finished before the deadlines.

2. **Personal computer** The writing of this project, research, coding and testing before using the computer cluster will be done with a personal computer MSI with a processor 4x Intel(R) Core(TM) i7-7500 CPU @ 2.70GHz, and 8084MB of memory, with Ubuntu 16.04 LTS.

The software used for this project will be different for each phase:

1. **Exact algorithms** here several models will be made with different tools. First of all we will develop a model using minizinc an open source constraint modeling language. Then a Gecode (open source Contraint programming language) model will be created (implementing a custom propagator which makes it interesting to compare with the minizinc model). Then a backtracking algorithms (a problem-specific algorithm) will be done using c++ and gcc as a compiler, we will use make as well.

2. **Greedy algorithms** This phase will be done using gcc. The design of this part will be based on the Backtracking techniques (the propagation part).

3. **Genetic algorithms** This phase will be done using gcc. And we find interesting to study it because has been used successfully to solve the maximally balanced connected partition problem.

1.2.3 Project planning description

There are six big phases of this project. Project management course, project preparation, exact algorithms, greedy algorithms, genetic algorithms and the comparison of results and writing of conclusions. Each of these phases are explained in the following sections. The hours spend in each phase can be consulted in table 1.1

Project management course

The course is worth 3 ECTS credits, so it should take 75 hours (25 for each ECTS credit). This part consists in 7 steps, and each step takes about 10 hours:

1. Formulation of the problem
2. Goals of the project and scope
3. Interest of the problem
4. State of the art
5. Project planning
6. Budget, sustainability and regulations
7. Final presentation and document

Project preparation

There are some basic definitions and terminology that should be clarified from the beginning of the project. The study of the problem from a classic complexity point of view will be done in this part. The instances to test the algorithms will be acquired from 2 sources: our own generator of random graphs using a binomial distribution for the edges and some real world graphs from public DB.
The cluster is managed by RDLab and it will require some experimentation to familiarize with it, although it is similar with another cluster used in another subject.

Exact algorithms

There are a very wide range of ways to approach a problem like minRLCA. The first attempt will be using constraint programming in order to see how it performs, and to have some initial results. Another way to solve the problem is with a backtracking algorithm. There is a possibility of finding exact algorithms that solve the problem efficiently in some kinds of graphs, but in general we don't expect to find the optimum in large instances with exact algorithms, so we will explore greedy and genetic algorithms.

Greedy algorithms

These algorithms can be seen as simple and intuitive. Therefore they will probably not find the best solution. But we hope to find good approximations. Our aim is to use the propagation in the backtracking algorithms as a base to develop the greedy algorithms.

Genetic algorithms

Another way to approach optimization problems is using genetic algorithms. This optimization technique has been used in similar problems, for example in [7].

Follow-up milestone

This will be the point where all the work should be done. And only the final conclusions and the comparison with the previous studies should be pending.

1.2.4 Alternative plans

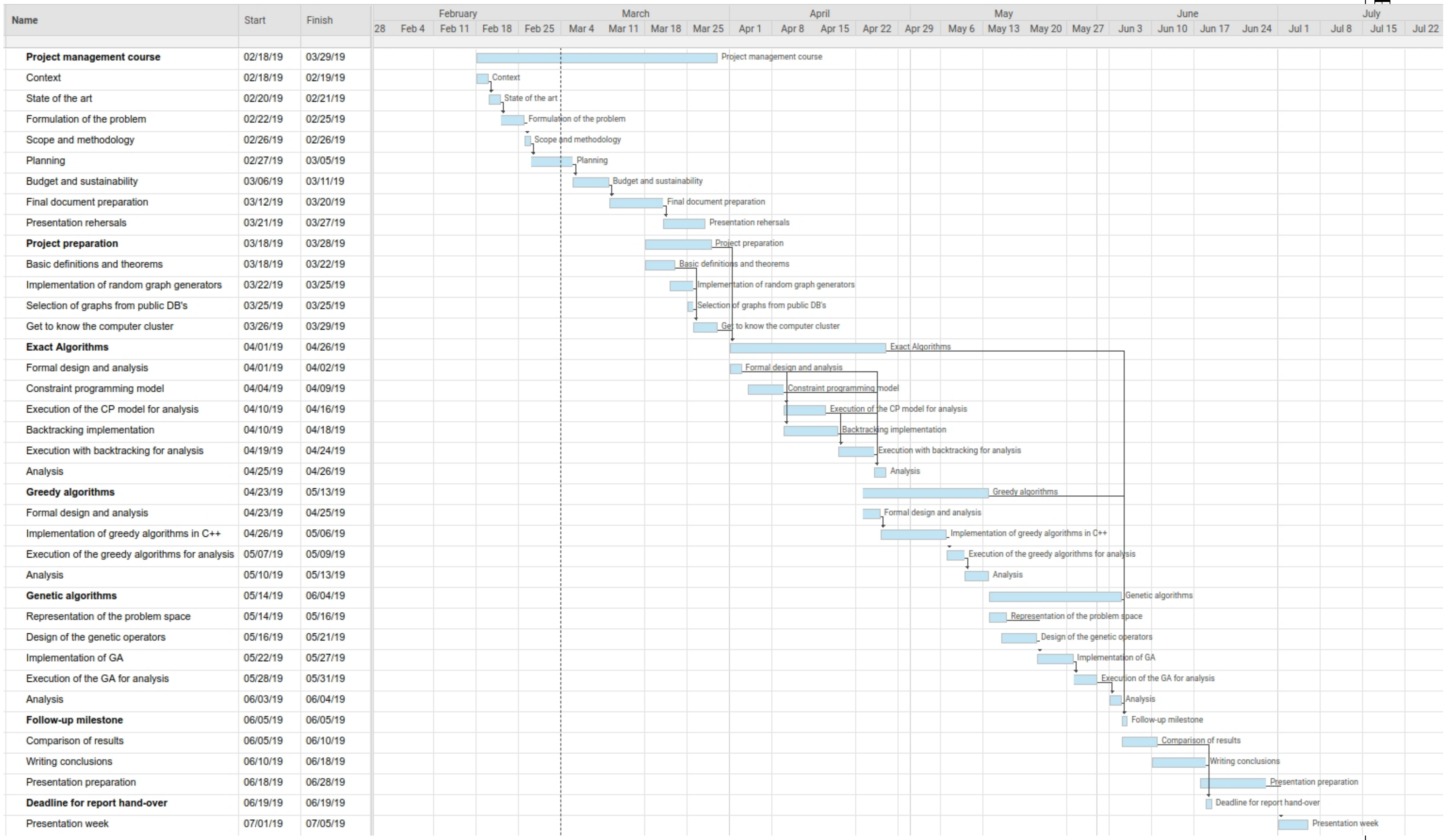
Although the plan is feasible in the proposed time, some modifications can occur due to future decisions related to design problems or executing problems. These modifications will be decided and approved by the director of the project in the weekly meetings that we will have. All changes and modifications must ensure that we reach the deadlines. This is why it is so important to have a follow-up milestone a month before the dateline.

1.2.5 Activities by days and hours

In the following table 1.1 the total amount of days exceeds the real number of days, this is because some activities overlap, this can be seen in the following Gantt chart 1.3. The amount of hours spend in each activity has been calculated considering that the amount of work will be 3 h/day and when 2 activities overlap these hours are split in a way that the amount of hours do not exceed 3 h/day.

Activity	days	hours
Project management course	45d	70h
Project preparation	10d	30h
Exact algorithms	28d	80h
Greedy algorithms	20d	70h
Genetic algorithms	20d	70h
Comparison of results and writing of conclusions	17d	60h
Meetings with the director	20d	40h
Total	160	420

Table 1.1: Hours and days for each activity



Name	Start	Finish
Project management course	02/18/19	03/29/19
Context	02/18/19	02/19/19
State of the art	02/20/19	02/21/19
Formulation of the problem	02/22/19	02/25/19
Scope and methodology	02/26/19	02/26/19
Planning	02/27/19	03/05/19
Budget and sustainability	03/06/19	03/11/19
Final document preparation	03/12/19	03/20/19
Presentation rehersals	03/21/19	03/27/19
Project preparation	03/18/19	03/28/19
Basic definitions and theorems	03/18/19	03/22/19
Implementation of random graph generators	03/22/19	03/25/19
Selection of graphs from public DB's	03/25/19	03/25/19
Get to know the computer cluster	03/26/19	03/29/19
Exact Algorithms	04/01/19	04/26/19
Formal design and analysis	04/01/19	04/02/19
Constraint programming model	04/04/19	04/09/19
Execution of the CP model for analysis	04/10/19	04/16/19
Backtracking implementation	04/10/19	04/18/19
Execution with backtracking for analysis	04/19/19	04/24/19
Analysis	04/25/19	04/26/19
Greedy algorithms	04/23/19	05/13/19
Formal design and analysis	04/23/19	04/25/19
Implementation of greedy algorithms in C++	04/26/19	05/06/19
Execution of the greedy algorithms for analysis	05/07/19	05/09/19
Analysis	05/10/19	05/13/19
Genetic algorithms	05/14/19	06/04/19
Representation of the problem space	05/14/19	05/16/19
Design of the genetic operators	05/16/19	05/21/19
Implementation of GA	05/22/19	05/27/19
Execution of the GA for analysis	05/28/19	05/31/19
Analysis	06/03/19	06/04/19
Follow-up milestone	06/05/19	06/05/19
Comparison of results	06/05/19	06/10/19
Writing conclusions	06/10/19	06/18/19
Presentation preparation	06/18/19	06/28/19
Deadline for report hand-over	06/19/19	06/19/19
Presentation week	07/01/19	07/05/19

Figure 1.3: Initial gantt chart

1.3 Budget, sustainability and regulations

1.3.1 Budget

Considerations

To develop this project we will take into account all the elements with a cost used in the planning. We will have human associated costs, hardware costs, software costs and indirect costs. In the previous sections we can find the time needed to develop each part.
In order not to exceed the total budget planned here, we will overbudget each phase. The final updates with the real costs will be shown in the final version of the project.

Human resources budget

This kind of project is usually done by a researcher in training and it is usually coordinated by a director of research. Regarding the 2018 budget of UPC [26] we can see that the annual salary of a fellow researcher in training is €16,422 per year. If we consider that it is a 40 hours-per-week job, that makes a total of 1,920h, then we have a salary of 8.55 €/h. But in this simplification we have not taken into account any free days nor holidays, so to make it easy we will round it to 9 €/h. According to the 2018 budget of UPC [26] a director of research has a salary of €35,958.66 per year. As before, we can consider it a 40 hours-per-week job, which makes 18.73 €/h. Same as before, we have not taken into account any free days nor holidays so we can round up to 20 €/h.
The amount of hours spent by the researcher is about 3 hours per day, 5 days a week during 5 months, which makes 300 hours in total. We must add to the total the 70 hours previously calculated for the project management course, a 2 hours-per-week meeting with the director of the project and 10 extra hours based on the time spent in unplanned events. This makes a total of 420 hours of work. The director will spent only the meeting hours which are 40 hours in total.

Role	Est. salary	Est. hours	Est. cost
Researcher in training	9 €/h	420 h	3,780 €
Director	20 €/h	40 h	800 €
Total cost			4,580 €

Table 1.2: Human resources budget

Hardware budget

Although the execution of the algorithms will take place in a computer cluster, their development and the writing of this project will be done in a laptop. The laptop allows to work in different places and is usually more efficient than a desktop computer. This is why we choose a laptop over a desktop computer.
In order to choose a suitable laptop we must take into account that good graphics are not necessary but, as it will need to compile big projects it would be interesting to have at least 8 GB of RAM. The laptop will have a 4x Intel(R) Core(TM) i7-7500 CPU @ 2.70GHz processor, and 8084MB of memory, with Ubuntu 16.04 LTS. The elctric consumption of this laptop is about 90 watts, we will use this laptop for almost all the work hours which has been previously calculated to 420h. The market value of this laptop is about 800 €. Considering a lifespan of 4 years for a project of 5

months, you will find the estimated cost in Table 1.3.

Product	Price	Units	Lifespan	Est. depreciation
Laptop	800 €	1	4 years	66.66 €
Electricity	0.20 €/kwh	37.8 Kwh		7.56 €
Total estimated				74.22 €

Table 1.3: Laptop budget

The computer cluster is owned by ALBCOM and other research groups, and RDLab offers them the maintenance. The cost of the hardware and maintenance is not included but we can approximate it using the data from other projects [1] that have used this cluster. The cluster works in packs. Each pack gives 2 GB of RAM + CPU time of 1h. The price of one pack is 0.4 €. We estimates that we will need 2 days to get to know the cluster by sending a few programs in the different types of queues; seven more days to execute the Constraint programming models; seven more to execute the backtracking model; three more to execute the greedy algorithms; and four more to execute the Genetic Algorithm. This makes a total of 552 h.

Product	hours	price per hour	Est. cost
Cluster	552 h	0.4 €/h	220.8 €
Total estimated			220.8 €

Table 1.4: Cluster budget

Product	Est. cost
Laptop	74.22 €
Cluster	220.8 €
Total estimated	295.02 €

Table 1.5: Hardware budget

Software budget

As we have set before, we will use an Ubuntu from the family of GNU/Linux operating systems. Git is an open source tool to maintain different versions of the code and to work in group projects. In order to store the code in the cloud we will use GitLab, which provides free private repositories. To write the report we will use Latex (distributed under the LaTeX Project Public License) on the platform Overleaf, which is free of charge. Different tools will be used on each part of the project:

1. **Constrain programming:** We will work in minizinc, which is a free and open source constraint modeling language. It is translated into flatzinc and this is understood by many constraint programming languages. We will use Gecode as our main solver but we will use Gurobi as well. Although Gecode is open source, Gurobi is a proprietary software which costs approximately 20 €/h. We will need to generate data and to process this data to adapt the input and the output. This will be done using python 3 which is open source.

2. **Backtracking, greedy and genetic algorithms:** We will use c++ programming language and to compile it we will use the GNU Compiler collection [11] and GNU Make tools. To create the source code we will use sublime-text which costs 71.1 €. Finally, the source code documentation will be generated using Doxygen, which apart from their licences, they are offered free of charge.

Product	Price	Est. time	Est. cost
GitLab	0	5 moths	0
Overleaf	0	5 moths	0
Latex	0	5 months	0
Minizinc	0	3 weeks	0
Gecode	0	2 weeks	0
Gurobi	20 €/h	1 weeks	3,360 €
python 3	0	3 weeks	0
GNU compiler collection	0	4 months	0
Sublime-text	71.1 €	5 months	71.1 €
Doxygen	0	4 months	0
Total estimated			3,431.7 €

Table 1.6: Software budget

Total Cost

Adding all previous costs, the estimated budget for this project is available in Table 1.7.

Concept	Est. cost
Human resources	4,580 €
Hardware	295.02 €
Software licences	3,431.7 €
Total estimated	8,306.72 €

Table 1.7: Total budget

Thanks to the cooperation of Uppsala University the software expenses regarding Gurobi licence will be taken on by them.

Deviation control

Like we have said in the previous sections, we plan to do a weekly meeting with the director. This will allow us to do an agile development, and will help us do the changes in such a way that the costs can be minimally modified. Furthermore, the calculated value for all the consulted items exceeds the market value, the only cost that could not be accurate is the amount of computation time, this is why in each phase the calculation time will be corrected.

1.3.2 Sustainability

From an economic point of view we can say that this budget is clearly too high for a 4 months university project, but the real costs will be much less because of the computer cluster and licence costs (and there will be no researcher in training). The majority of the tools are open source so there are few tools that costs money, so Gurobi is the only problem in the software budget.

The social implications of this project are related to the licence used. almost all the tools used in this project are open source or free software, so this project and all the code developed will be publish under a free software licence and they will be public in an open benchmarking platform. Another social implication more related to the content of the work is that there will be an implementation of a layout problem with a constraint programming model, which has never been tried, if this leads to positive results the models could be used to minimize costs in real-life problems related to the graph layout problems.

If the experiments give negative results this project will only be usefull as a first approach to the problem so the computation costs will be in part wasted. Even though nowadays computer clusters are more efficient than they were before, they usually consumes a lots of energy since they are very well refrigerated. Another important environmental problem is the materials of the computers used in this project, many of the minerals needed to build the computers come from areas under civil wars to control of the area of the material, these social tragedies can give us for a whole bachelor thesis.

1.3.3 Laws and regulations

This project has to comply with some legal and ethical requirements. The written work must respect the intellectual property of the authors of the used references, to meet this requirement, we have included all references using a standard style at the end of the document.

Regarding all the models, algorithms and software developed, we will use open source libraries and proprietary software such as Gurobi. When open source libraries will be used there will be the adequate recognition to the authors. Some of the libraries may not accept modifications so we will not change these libraries but work on the top of them. For the non open source software, we will use an academic licence because all our results will be solely for research purposes.

Other regulations for which our project will have to take care will be those related to the use of the computer cluster. The use of the cluster for any purpose that is not the execution of the algorithms of this project is not allowed, and although it has its own way to control the abuse, common sense and fair use must prevail when using it. If some other universities collaborate providing a computation time or other resources, they must be recognised and the common sense and fair use must prevail when using their resources.

As for the licenses and intellectual property of the software developed during the project, we have to comply with the University's rules [25]. According to the Title 2, Article 3, Section 3 of the University rules UPC is the owner of the exploitation rights but it is not the exclusive author, and since the authors are by default owners of the exploitation rights according to the Spanish law, we will release it under a free software licence.

Chapter 2

Formal definition and preliminaries

2.1 Preliminaries

We would like to make this project self-contained. In order to do so, the following section will introduce several definitions and properties that will help us understand our problem. Most of them are standard in graph theory and we will follow the notations of [6].

2.1.1 Basic definitions

Definition 2.1.1. Graph. A simple undirected graph G is a pair $G = (V, E)$ satisfying $E \subseteq \{\{u, v\}|u, v \in V, u \neq v\}$. The elements of V are the vertices of the graph G and the elements of E are the edges. The vertex set of the graph G is referred to as $V(G)$ and its edge set as $E(G)$.

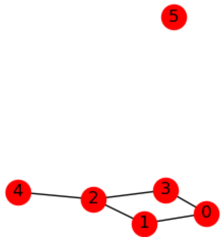


Figure 2.1: Graph on V = {0,...,5}, E = {{0,1}, {0,3}, {1,2}, {2,3}, {2,4}}

The number of vertices of a graph (called order of a graph) is denoted by $|V(G)|$, and the number of edges is denoted by $|E(G)|$. A vertex v is *incident* with an edge e if $v \in e$; then e is an edge at v. The set of all the edges in E at a vertex v are denoted by $E(v)$. If two vertices are incident to an edge, we call them ends (or endvertices), and the edge *joins* its ends. An edge $\{u, v\}$ can be referred to as uv (or vu). Two vertices u and v of G are *adjacent* or *neighbours* if uv is an edge of G. If this edge does not exist, the two vertices are called *independent*. More formally, a set of vertices are called independent if non of the pairs are neighbours. If all pairs of vertices of G are adjacent then G is *complete*. A complete graph $G = (V, E)$ with $n = |V|$ is a K_n.

Definition 2.1.2. Subgraph, induced subgraph and spanning graph. Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We call G' subgraph of G, and G supergraph of G' if $V' \subseteq V$ and $E' \subseteq E$. When G' is a subgraph of G it is written as $G' \subseteq G$. If $G' \subseteq G$ and G' contains all the edges $uv \in E$ such that $u, v \in V'$, then G' is an induced subgraph of G. We call G' a spanning subgraph of G when $V' = V$ and $E' \subseteq E$.

Definition 2.1.3. Multigraph. Graph that can have more than one edge between two vertices, and allows loops that are edges with the same end vertices.

Definition 2.1.4. The *union* of two graphs (not to be confused with *disjoint union*), $G = (V, E)$ and $G' = (V', E')$, denoted with $G \cup G'$, it is defined as the graph resulting from the union of the vertex sets and the edge sets:

$$G \cup G' = (V \cup V', E \cup E')$$

Definition 2.1.5. Neighbourhood. Let $G = (V, E)$, the set of neighbours of a vertex v in G is called neighbourhood and is denoted by $N_G(v)$:

$$N_G(v) = \{u|uv \in E(G)\}$$

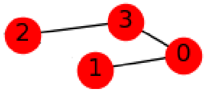


Figure 2.2: Subgraph of 2.1

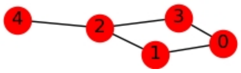


Figure 2.3: Induced subgraph of 2.1

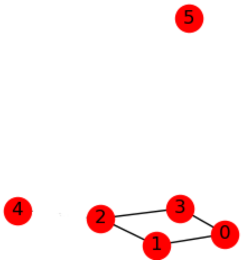


Figure 2.4: Spanning subgraph of 2.1

More generally, for $U \subseteq V$, the neighbours in $V \setminus U$ of vertices in U are called *neighbours* of U:

$$N_G(U) = \{u | uv \in E(G), v \in U, u \notin U\}$$

Definition 2.1.6. Degree of a vertex. Given $G = (V, E)$, the degree of a vertex $d_G(v)$ can be defined as the number of edges at that vertex, $|E(v)|$. By our definition of graph, this is equal to the number of neighbours of v:

$$d_G(v) = |N_G(v)|$$

More generally, for $U \subseteq V$, the degree of U can be defined as the number of neighbours of U:

$$d_G(U) = |N_G(U)|$$

Definition 2.1.7. Path. Given a graph $G = (V, E)$ satisfying $V = \{u_0, ..., u_n\}$ and $E = \{u_0u_1, u_1u_2, ...u_{n-1}u_n\}$ it is called a *path* of length n and is denoted by P_n.

Theorem 2.1.1. All graphs can be rewritten as the union of paths.

Proof. Let $G = (V, E)$, for each vertex $v \in V(G)$ we define a path with only one vertex: v, we will call the set of all these paths $Paths_1$. Note that the number of paths in P_1 will be $|V|$. Now, for each edge $(v, u) \in E$ we define a path with the vertices u, v and one edge: (u, v), we will call the set of all these paths $Paths_2$. Then, the result of the union of all paths in $Paths_1$ will be a graph with vertex set V, and edge set $\varnothing$, and the result of the union of all paths in $Paths_2$ will be a graph with edge set E, and a vertex set $O \subseteq V$ (the vertices with degree greater than 0). If we do the union of all paths in $Paths_1$ and $Paths_2$ we will have a graph with vertex set $V \cup O$ and edge set $\varnothing \cup E$, since $O \subseteq V$, this graph will have as a vertex set V and as edge set E and that is, by definition, G. □

Definition 2.1.8. Cycle. It can be defined as a graph $G = (V, E)$ which satisfies $V = u_0, ..., u_{n-1}$ and $E = u_0u_1, u_1u_2, ...u_{n-2}u_{n-1}, u_{n-1}u_0$, and it can be denoted by C_n. $C_n = P_{n-1} \cup \{\{u_0, u_{n-1}\}, \{\{u_{n-1}, u_0\}\}\}$. Note that the degree of each vertex is 2 and its neighbours are:

$$N_{Cn}(v_i) = \{v_{(i-1)\%n}, v_{(i+1)\%n}\}$$

Definition 2.1.9. Connectivity. Let $G = (V, E)$, then G is *connected*, if for every pair of vertices in V(G) are linked by a path. Examples of connected graphs are all cycles and paths. A maximal connected subgraph of G is called a *component* of G. Therefore, each graph can be defined as the union of all of its components (the empty graph has 0 components).

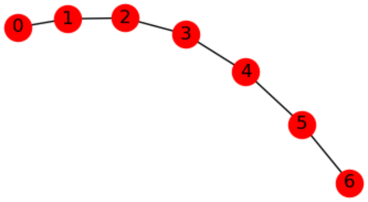


Figure 2.5: P_6

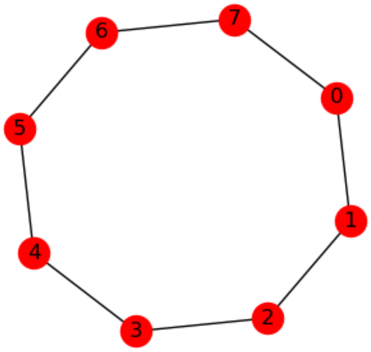


Figure 2.6: C_8

Definition 2.1.10. Forest and Trees. A graph without cycles is called a $forest$. A connected forest is called a *tree*, we usually denote a tree by T. The following assertions are equivalent for a graph T:

- T is a tree.
- Between any 2 vertices in T, there is exactly one path.
- T is connected, but $T - xy$ is disconnected for every $xy \in E(T)$.
- T is acyclic, but $T + xy$ contains a cycle as long as $x, y \in V(T)$ and $xy \notin E(T)$.

Theorem 2.1.2. All connected graphs contain a spanning Tree.

Proof. Let $G = (V, E)$. Induction on the number of vertices of G.

- $|V(G)| = 1$. $T = G$.
- $|V(G)| = n$. Suppose that for each graph with $|V| < n$ exists a tree $T = (V_t, E_t)$ with $|V_t| = n - 1$ (spanning Tree). If we remove one vertex u of G, no matter which one, we will end up with one or more components of G - u. Since each of this components are of size $n - 1$ or less, there is a spanning tree of each component. Since *component* is defined as a maximally connected subgraph, there is no path between the different trees of the components. Therefore, we can always build a graph without cycles by uniting the trees of each component using the vertex u and one of the edges connecting u with the different trees. This graph will have n vertices and no cycles, therefore it will be a spanning Tree of G.

□

Definition 2.1.11. Partition. Given a set A and another set $B = \{A_1, ..., A_k\}$ of disjoint subsets of A. B is a partition of A if $A = \bigcup_{i=1}^{k} A_i$ and $A_i \neq \emptyset$ for every i.

Definition 2.1.12. k-partite graphs. Let $G = (V, E)$ and let $k \geq 2$ be an integer. G is $k - partite$ if V admits a partition into k classes such that every edge has its ends in different classes and there are no adjacent vertices inside a class. A $2 - partite$ graph is called *bipartite*.

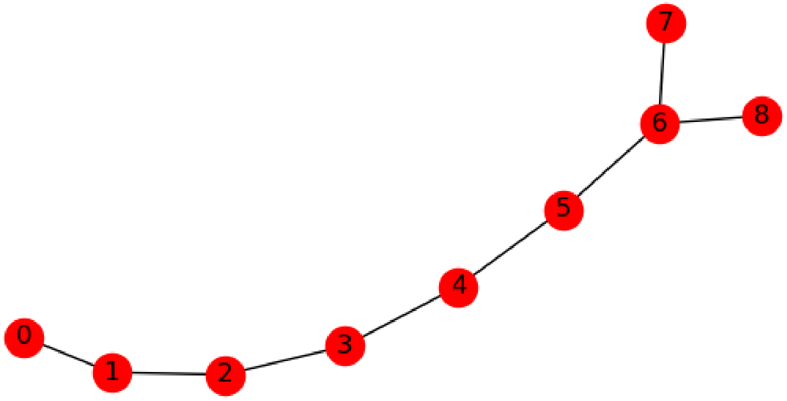


Figure 2.7: Tree with 9 vertices

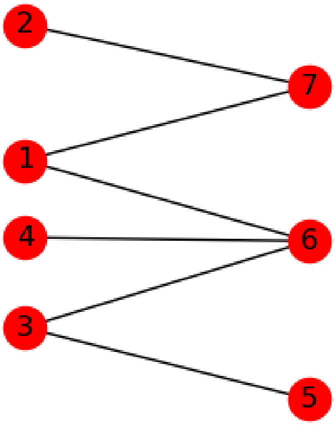


Figure 2.8: Bipartite graph

2.2 Minimum Restricted Linear Coloring Arrangement Problem

The Minimum Restricted Linear Coloring Arrangement Problem is a combinatorial optimisation problem where each vertex gets a label from $\{0, \dots, k\}$. Vertices with the same label form a group. Each group has two restrictions, the size of the group must be at most r and each group can be restricted to be an independent set or to be connected. Note that we have said can, that is because we can choose to not restrict the groups to be neither of both.

More formally, given a graph $G = (V, E)$, $r \in \mathbb{N}$, $k \in \mathbb{N}$ and $C : \{1, \dots, k\} \mapsto \{\varnothing, 0, 1\}$, a Restricted Linear Coloring Arrangement is a function $\varphi : V(G) \mapsto \{1, \dots, k\}$ such that for all $i \in \{1, \dots, k\}$

1. $|\{u \mid \varphi(u) = i\}| \leq r$
2. $G[\{u \ \mid \varphi(u) = i\}] \begin{cases} \text{is connected} & \text{if } C[i] = 1 \\ \text{is an independent set} & \text{if } C[i] = 0 \end{cases}$

The cost of a solution will be called *linear distance* $L[\varphi]$.

$$L[\varphi] = \sum_{(u,v) \in E(G)} |\varphi(u) - \varphi(v)|$$

The set of all Restricted Linear Coloring Arrangements that satisfy the conditions for the given parameters $< G, k, r, C >$ will be denoted F(G, k, r, C).

$$F(G, k, r, C) = \{ \quad \varphi \quad | \quad \forall \varphi \text{ meeting the restrictions 1 and 2}\}$$

The goal of our problem will be to find whether there is a Restricted Linear Coloring Arrangement φ that meets the restrictions and if that is the case, to find the linear arrangement with minimum cost:

$$MinRLCA(G, k, r, C) = min_{\varphi \in F(G,k,r,C)} L(\varphi)$$

Definition 2.2.1. Given a graph $G = (V, E)$ and $n = |V|$, a Linear Arrangement without restrictions can be defined as follows:

$$\varphi : V(G) -> \{1, \dots, n\} \text{ such that LA is a bijective function}$$

The set of all Linear Arrangements of a Graph G will be called Linear Arrangements of G, LA(G):

$$LA(G) = \{ \quad \varphi \quad | \quad \varphi : V(G) -> \{1, \dots, n\} \text{ such that } \varphi \text{ is a bijective function}\}$$

The cost of a linear arrangement is its Linear Distance. Given a graph $G = (V, E)$, the MinLA can be defined as follows:

$$MinLA(G) = min_{\varphi \in LA(G)} L[\varphi]$$

Definition 2.2.2. Decisional version of MinLA consists in finding a Linear Arrangement $LA \in LA(G)$, such that, the cost is less than W.

$$MinLA - dec(G, W) = \varphi \in LA(G) \text{ such that } L[\varphi] < W.$$

Our problem can be seen as a generalization of the Minimum Linear Arrangement Problem. Let $G = (V, E)$, $r = 1$, $k = |V(G)|$ and that C does not matter because each group will only contain one vertex; with this parameters we will have a one-to-one mapping of the vertices minimizing the sum of the induced distances among all the pairs of adjacent vertices, which is one definition of the Minimum Linear Arrangement Problem.

Example 2.2.1. Given the following graph $G = (V, E)$ and setting the number of groups equal to the number of vertices $k = |V(G)| = 7$, the size of each group equal to 1 ($r = 1$), we can see that the solution of figure 2.10 is a Minimum Linear Arrangement.

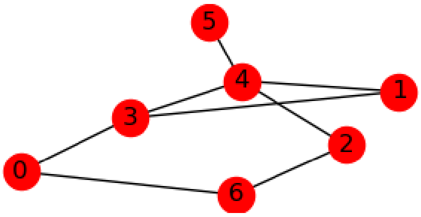


Figure 2.9: Graph with 7 vertices and 8 edges

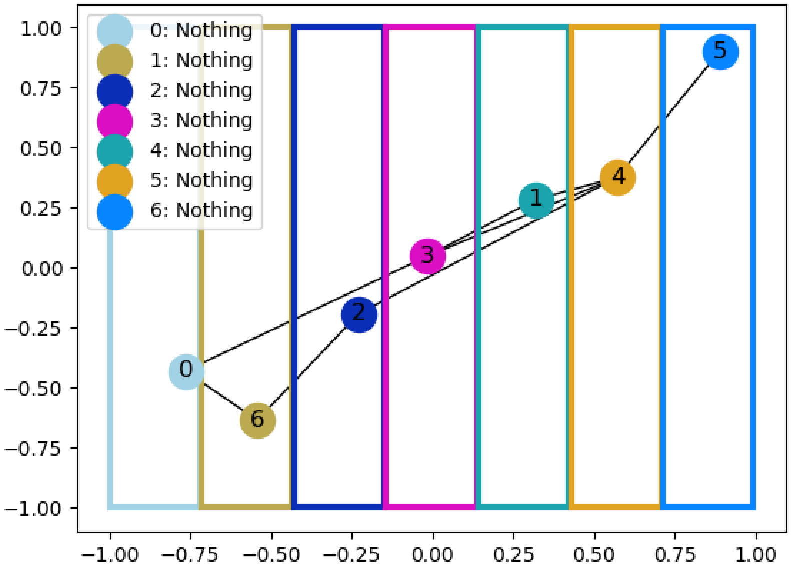


Figure 2.10: Optimal solution for the problem 2.2 with cost 13

Our problem can also be seen as a generalization of the Minimum Linear Coloring Arrangement Problem (MinLCA). MinLCA can be defined as follows: given a graph $G = (V, E)$, find a Linear Coloring Arrangement $\varphi : V(G)- > \{1, \ldots, |V(G)|\}$ such that for all $i \in \{1, \ldots, |V(G)|\}$

- $G[\{u \mid \varphi(u) = i\}]$ is an independent set

The 2 differences between the MinLCA and the Minimum Linear Arrangement are:

- Instead of a one-to-one mapping we group the vertices without limit on the size of the groups.
- We restrict each group to be an independent set.

Thus, if we consider the following instance of our problem: $< G, k, r, C >$ and we set $k = r = |V(G)|$ and $\forall i \in \{1, \ldots, k\} \quad C[i] = 0$, which means that there are no limits on the number of groups nor the size of each group and all groups are independent sets, we will obtain a solution that will solve the minLCA problem.

Since all groups are independent sets, the solution can also be seen as a coloration of the graph. Note that the cost that we minimize does not ensure that the result will be a coloration with minimum colors (minimum number of groups). An example can be seen in the solution of figure 2.11 where vertex 4 could have been assigned to group 0 instead of group 3 reducing the number of groups but increasing the cost of the solution.

Example 2.2.2. Given the graph $G = (V, E)$ of figure 2.9 and setting the restriction as mentioned before, we obtain the solution of figure 2.11.

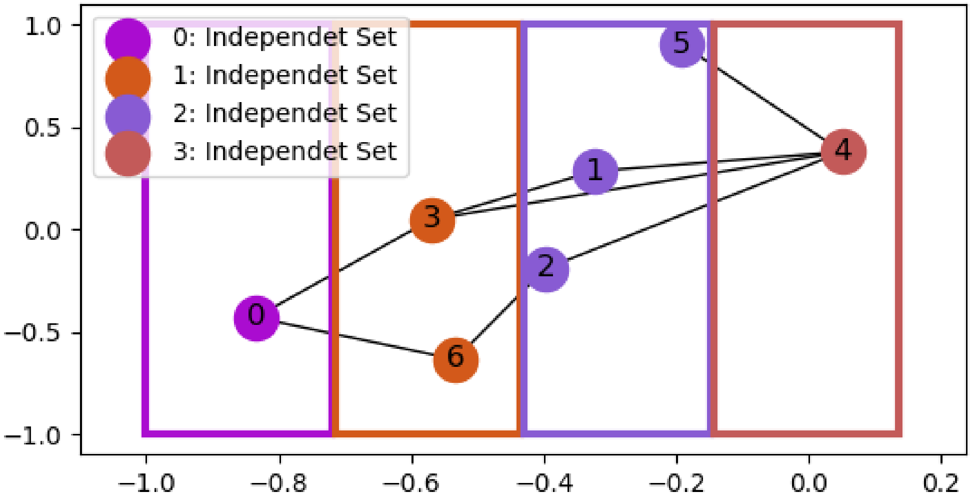


Figure 2.11: Optimal solution for the problem 2.2.2 with cost 9

Chapter 3

Complexity and properties

3.1 Complexity of the problem

Lets define a decisional version of MinRLCA. Given a graph $G = (V, E)$, $r \in \mathbb{N}$, $k \in \mathbb{N}$ and $C : \{1, \ldots, k\} \mapsto \{\varnothing, 0, 1\}$, instead of finding a φ such that minimize the Linear distance ($L[\varphi]$), we introduce a new parameter W and we try to find a solution such that $L[\varphi] \leq W$. Now we have erased the minimization factor from the definition of the problem.

Theorem 3.1.1. The decisional version of MinRLCA is NP-complete.

Proof. :

- The decisional version of MinRLCA is NP. To prove this, we need to see that, given an instance and its solution, we can verify it in polynomial time. To prove it, we have created an algorithm [28] that runs in polynomial time.
  This algorithm can be divided in three parts:

  1. The first postcondition [line 3]. In this part we check that the number of vertices assigned in each group is r at most. The cost is $O(|V|)$.
  2. The second postcondition [line 7]. For each group i we have to check whether it is an independent set or a connected subgraph or non of them. To check whether it is an independent set we have to ensure that for each u and v such that $\varphi(u) = \varphi(v) = i$, $(u, v) \notin E$. The cost is $O(|V| * (|V| - 1)/2)$. To check whether the group i is connected, we will use a classic Breadth-first-search algorithm (BFS). The cost of a BFS is $O(|V| + |E|)$ ([4]:p.597).
  3. The third postcondition [line 27]. The cost can be calculated by the difference in φ between the two ends of each edge. This part has a cost of $O(|E|)$.

  The cost of the verifier [28] is $O(|V|^2 + |E|)$, hence the decisional version of MinRLCA is NP.

- The decisional version of MinRLCA is NP-hard. To prove it, we can reduce an NP-hard problem to the MinRLCA problem. In fact, we show that the problem is a generalization of an NP-hard problem. We will reduce the decisional version of the Minimum Linear Arrangement Problem (MinLA-dec) 2.2.2, which has been proved NP-complete in [9], to the decisional version of MinRLCA.
  Let the graph $G = (V, E)$ and $W \in \mathbb{N}$ be the input for the decisional version of the Minimum Linear Arrangement Problem, the corresponding input for the decisional version of the Minimum Restricted Linear Arrangement Problem is the graph G', $k \in \mathbb{N}$, $r \in \mathbb{N}$, $C : \{1, \ldots, k\} \mapsto \{\varnothing, 0, 1\}$ and $W' \in \mathbb{N}$ defined as follows:

  * $G' = G$
  * $k = |V(G)|$
  * $r = 1$
  * $\forall_{i \in \{1,\ldots,k\}} \quad C[i] = \varnothing$
  * $W' = W$
  * A Restricted Linear Coloring Arrangement will be defined as $\varphi : V(G) \mapsto \{1, \ldots, k\}$

Notice that C does not matter since the maximum size of each group is one, hence the restriction that says that the subgraphs induced by the groups have to be independent sets or connected does not matter (restriction 2 on the definition of the problem).
Since the size of each group will be at most one and the number of groups will be exactly the same as the number of vertices, the solution φ will be a one-to-one mapping. Furthermore, we can rewrite the restriction $|\{u \mid \varphi(u) = i\}| \leq r$ [1] as $\forall_{u,v \in V(G)} phi(u) \neq phi(v)$ and we can say that the set of all Restricted Linear Coloring Arrangements:

$$F(G', |V(G)|, 1, C) = \text{ all permutations of } \{1, \ldots, |V(G)|\}$$

The subset of permutations P in $F(G', |V(G)|, 1, C)$ such that its cost will be less or equal to W' will be defined as:

$$P = \forall_{\varphi = F(G', |V(G)|, 1, C)} L[\varphi] \leq W'$$

By definition, it will also be the set of all the possible solutions for the MinRLCA problem, and since $W' = W$ and $F(G', |V(G)|, 1, C)$ contain all the possible permutations of $\{1, \ldots, |V(G)|\}$, then P is the set of all the possible solutions for the decisional version of MinLA. Considering that all the possible solutions of MinRLCA are the same as all the possible solutions of MinLA, the decisional version of MinRLCA is NP-hard and NP, which means that the decisional version of MinRLCA is NP-complete.

□

One of the properties of the MinRLCA that make this problem so hard is that for each input graph $G = (V, E)$ and $n = |V|$ we can define several instances. Let's take a look to the parameters of the problem:

- k: the number of groups, can take values from 1 to n
- r: the maximum size for each group, can take values from 1 to n
- C: for each group $i \in \{1, \ldots, k\}$, $C[i]$ can take 3 different values.

Since the number of groups multiplied by the size of each group must be grater or equal than n (to be a valid instance), we can define the number of possible instances given a graph $G = (V, E)$ and assuming $n = |V|$:

$$\text{Number of instances } = \sum_{k=1}^{n} \sum_{r=n-k+1}^{n} r * k * 3^k$$

The high number of possible instances for each problem combined with the fact that there is no symmetry in the majority of these instances, makes the MinRLCA a very difficult problem.

Algorithm 1 Verifier for the decisional version of minRLCA

Precondition Let $G = (V, E)$ be a graph, represented as an adjacency matrix.
$r \in \mathbb{N}$, $k \in \mathbb{N}$, $C : [k]- > \{\varnothing, 0, 1\}$.
$\varphi : V- > [k]$ a possible solution (witness).
$W \in \mathbb{N}$.

Postcondition Verifies $\forall i \in [k]$:

1. $|\{u \mid \varphi(u) = i\}| \leq r$
2. $G[\{u \ \mid \varphi(u) = i\}] \begin{cases} \text{is connected} & \text{if } C[i] = 1 \\ \text{is an independent set} & \text{if } C[i] = 0 \end{cases}$
3. $L[\varphi] = \sum_{(u,v) \in E(G)} |\varphi(u) - \varphi(v)| \leq W$

```
function VERIFIER($G, r, k, C, \varphi, W$)
    $Groups$ is a vector of lists of size k
    for $u = 1$ to $|V|$ do                                    ▷ check postcondition 1
        $Groups[\varphi(u)].push(u)$
        if $Groups[\varphi(u)].size > r$ then
            return $False$
    for $i = 1$ to $k$ do                                      ▷ check postcondition 2
        if $C[i] = 0$ then
            for $u = Groups[i].first$ to $Groups[i].last$ do
                for $v$ =next of $u$ to $Groups[i].last$ do
                    if $G[u][v] = 1$ then
                        return $False$;
        else if $C[i] = 1$ then
            Q is an empty Queue
            Visit is a boolean vector of size $|V|$ initialized to $False$
            if not $Groups[i].empty$ then
                $fst = Groups[i].first$
                $Q.push(fst)$
                $Visit[fst] = True$
                while $not\ Q.empty()$ do
                    $u = Q.pop()$
                    for $v = Groups[i].first$ to $Groups[i].last$ do
                        if $G[u][v] = 1$ and $Visit[v] = False$ then
                            $Q.push(v)$
                            $Visit[v] = True$
                for $u = Groups[i].first$ to $Groups[i].last$ do
                    if $Visit[u] == False$ then return $False$
    $cost = \sum_{(u,v) \in E} |\varphi[u] - \varphi[v]|$                  ▷ check postcondition 3
    return $cost \leq W$
```

3.2 Lower and upper bounds for the MinRLCA

Theorem 3.2.1. Upper bound on the Restricted Linear Coloring Arrangement. Let $G = (V, E)$, $k \in \mathbb{N}$, $r \in \mathbb{N}$ and $C : \{1, \dots, k\} \mapsto \{\varnothing, 0, 1\}$

$$\forall_{\varphi \in F(G,k,r,C)} \quad L[\varphi] \leq |E|(k-1)$$

Proof. Since a Restricted Linear Coloring Arrangement is defined as $\varphi : V(G) \mapsto \{1, \dots, k\}$, the maximum value of $\varphi(u)$ is k for any $\varphi \in F(G, k, r, C)$ and $u, v \in V$, and the minimum is 1. Then $max_{(u,v)\in E}|\varphi(u) - \varphi(v)| = k - 1$,

$$\forall_{\varphi \in F(G,k,r,C)} \quad L[\varphi] \quad = \sum_{(u,v)\in E(G)} |\varphi(u) - \varphi(v)| \quad = \sum_{(u,v)\in E(G)} (k-1) \quad = \quad |E|(k-1)$$

□

Theorem 3.2.2. Lower bound on the Restricted Linear Coloring Arrangement. Let $G = (V, E)$, $k \in \mathbb{N}$, $r \in \mathbb{N}$ and $C : \{1, \dots, k\} \mapsto \{\varnothing, 0, 1\}$

$$\forall_{\varphi \in F(G,k,r,C)} \quad L[\varphi] \geq 0$$

Proof. $L[\varphi]$ is defined as a sum of absolute values, therefore its lower bound must be equal or higher than zero. □

Note that we can easily produce an example where the cost is 0: take any graph $G = (V, E)$ and set $k = 1$, $r = |V|$, $C = \{1\} \mapsto \{\varnothing\}$, then the only possible solution will be: $\forall_{u \in V(G)} \quad \varphi(u) = 1$, thus:

$$L[\varphi] \quad = \sum_{(u,v)\in E(G)} |\varphi(u) - \varphi(v)| \quad = \sum_{(u,v)\in E(G)} 0 \quad = \quad 0$$

3.3 Closed results for complete graphs

A complete graph $G = (V, E)$ is a K_n with $n = |V|$ and an edge between each pair of different vertices $\forall_{u,v \in V, u \neq v} \quad (u, v) \in E$. Since each vertex is adjacent to all the other vertices the number of edges can be calculated as follows:

$$|E(K_n)| = \frac{n(n-1)}{2}$$

In a complete graph all groups of more than one vertex are connected, so there is no point in differentiate between connected groups, and groups without restrictions.

Now, let's define an input graph $G = (V, E) = K_n$ and a problem where some of the groups are restricted to be connected (or are not restricted). There can be more than one solution to this problem, but the cost of a solution φ is, as always $L[\varphi] = \sum_{(u,v)\in E(G)} |\varphi(u) - \varphi(v)|$.

3.3.1 Simplification of the cost for complete graphs

Let's define a multigraph $M = (V', E')$ such that each node will be a group of the graph $V' = 1, \ldots, k$ and for all $(u, v) \in E(G)$ there will be an edge $(\varphi(u), \varphi(v))$ in E(M). Let's see an example:

Example 3.3.1. Given the graph $G = (V, E)$ of figure 3.1 and $k = 4$, $r = 2$ and $C = [1, 0, 0, 1]$ a possible solution could be $\varphi = [1, 1, 2, 3, 4, 4]$, and its multigraph can be found in figure 3.2.

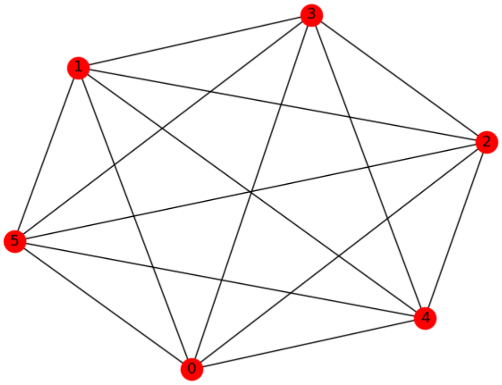


Figure 3.1: K_6

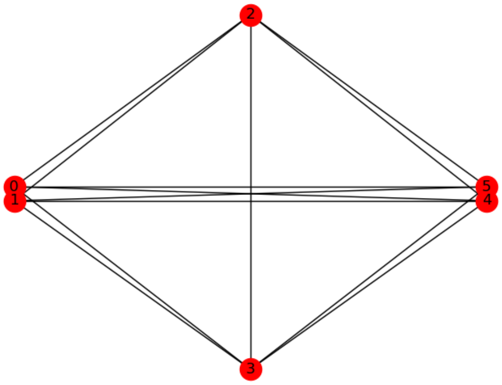


Figure 3.2: The multigraph of the graph of figure 3.1 and solution $\varphi = [1, 1, 2, 3, 4, 4]$, the node 0,1 should be only 1 and the node 4,5 should be only 4, but this way the process is clearer

Note that the costs of the solution φ in the multigraph M and the graph G is the same, since all edges crossing two groups still exist. The difference lies in the name of the node, which indicates the group:

$$L[\varphi] = \sum_{(u,v)\in E(G)} |\varphi(u) - \varphi(v)| = \sum_{(c1,c2)\in E(M)} |c1 - c2|$$

Observe that, since G is a complete graph, the number of edges between two nodes is the number of vertices of one node multiplied by the number of vertices of the second node. Therefore, we can convert the multigraph M into a simple graph $R = K_k$ where each node is a vertex of this new graph, and all the edges connecting the same two nodes are reduced to only one edge. The cost of this edge will be the number of edges in the multigraph crossing these two nodes, which can be calculated as the number of vertices in one node multiplied by the number of vertices in the second node, and multiplied by the cost of one of these edges in the multigraph (the cost of all the edges in the multigraph between the same two nodes is the same).
If we define the size of a node c as $s(c)$:

$$\forall_{c\in V(R)} \quad s(c) = |\{u|\varphi(u) = c\}|$$

then, the total cost can be calculated as:

$$L[\varphi] \quad = \sum_{(c1,c2)\in E(R)} |c1 - c2| * s(c1) * s(c2)$$

Note that since R is a complete graph K_k with as many nodes as groups in G, its cost can be calculated as:

$$L[\varphi] = \sum_{c1=1}^{k-1} \sum_{c2=c1+1}^{k} (c2 - c1) * s(c1) * s(c2)$$

We can see the inner loop $(c2 - c1)$ as a sum from 1 to $k - c1$:

$$L[\varphi] = \sum_{c1=1}^{k-1} \sum_{c2=1}^{k-c1} c2 * s(c1) * s(c2)$$

Note that the inner loop in the first iteration of the outside loop will be a sum from 1 to $k - 1$ and the last iteration will be from 1 to $k - (k - 1) = 1$, then, we can invert the inner loop (instead of starting by $k - 1$ and ending with 1, we can start by 1 and end with $k - 1$):

$$L[\varphi] = \sum_{c1=1}^{k-1} \sum_{c2=1}^{c1} c2 * s(c1) * s(c2)$$

Now we can forget about the multigraph (which was only a tool to understand the previous result) and think about the cost of a solution in terms of the number of vertices assigned to each group and the distances between groups.

In the following, we analyze different cases for complete graphs.

3.3.2 Independent sets

Let's study the case where all the groups are restricted to be independent sets. In this case all groups have size one or zero. If more than one vertex were assigned to the same group, there would be an edge crossing two vertices in the same group, and that would not be an independent set. Given any complete graph and a problem were all groups are required to be independent sets, we can imagine a solution as the line in figure 3.3. Each circle represents a group and each group can or cannot contain a vertex.

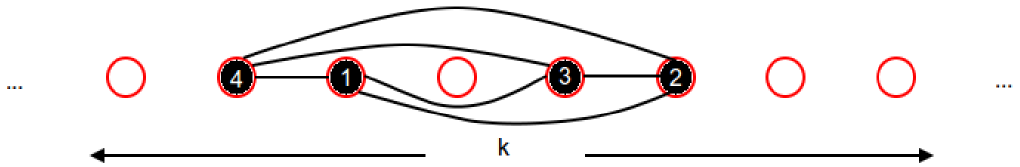


Figure 3.3: Representation of a possible solution for a complete graph of 4 vertices (K_4) and at least 8 groups

Let's assume that the number of groups is equal or greater than the number of vertices (if we have less groups than vertices there are no possible solutions); if the number of groups is equal to the number of vertices, there is only one possible solution, a one to one mapping where it does not matter which vertex is assigned to which group. This happens because of the symmetry of a complete graph.
Now let's study the case where there are more groups than vertices. We have to decide which groups end without a vertex assigned to them, in the example of figure 3.3 we can intuitively see that there is no point in allowing spaces between the groups with vertices, so a possible better solution would be figure 3.4.

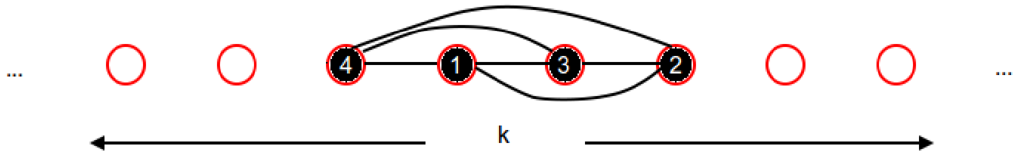


Figure 3.4: K_4 and 8 or more groups without spaces between groups with vertices

Lemma 3.3.1. In complete graphs when all the groups are restricted to be independent sets, solutions where all labels can be ordered to be consecutive numbers have a better cost than solutions with gaps between groups.

Proof. Let's start by defining the cost of a solution without gaps. Given a graph $G = (V, E)$, we assume $n = |V|$, $k > n$ (r and C do not matter since all groups have at most size one). Let's suppose a solution φ where the first group with vertex is the group $x1 \geq 1$ and $x1 \leq k - n$; and the last group with vertex is the group $x2 = x1 + n - 1$, therefore there are no gaps and there are n consecutive groups with vertices, for convenience, and without loss of generality let's assume

$x1 = 1$ and $x2 = n$. The cost will be defined as:

$$L[\varphi] = \sum_{c1=1}^{k-1} \sum_{c2=1}^{c1} c2 * s(c1) * s(c2)$$

Since all groups from n to k have size 0, and all the others have cost 1, we can simplify the cost:

$$\begin{aligned} L[\varphi] &= \sum_{c1=1}^{n-1} \sum_{c2=1}^{c1} c2 = \sum_{c1=1}^{n-1} \frac{c1(c1+1)}{2} \\ &= \frac{1}{2} \sum_{c1=1}^{n-1} c1^2 + c1 = \\ &= \frac{1}{2} \sum_{c1=1}^{n-1} c1^2 + \sum_{c1=1}^{n-1} c1 = \\ &= \frac{1}{2} \left(\frac{(n-1)*n*(2n-1)}{6} + \frac{(n-1)*n}{2} \right) = \\ &= \frac{1}{2} \frac{(n^2-n)(2n+2)}{6} = \\ &= \frac{n^3-n}{6} \end{aligned}$$

Now let's consider that there are d gaps among the groups. To calculate the cost we can assume the gaps as groups with imaginary vertices and edges. Then, the total cost will be the cost of a complete graph including the gaps minus the cost of the imaginary edges for each gap. The cost of the edges $H(p)$ of a gap at position p can be defined as the sum from the 1 until $p-1$ and from 1 to $f-p$ where $f = n + d$ is the total number of vertices plus the number of gaps.

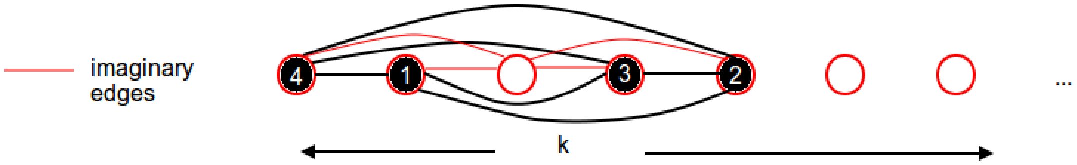


Figure 3.5: Solution with added costs from the gap at position 3

$$\begin{aligned} H(p) &= \sum_{i=1}^{p-1} i + \sum_{j=1}^{f-p} j = \\ &= \frac{(p-1)*p}{2} + \frac{(f-p)*(f-p+1)}{2} \end{aligned}$$

Then the cost of a solution with a gap at position p will be defined as:

$$L[\varphi] = \frac{(f)^3 - (f)}{6} - H(p)$$

Now let's find which position p of a gap, makes the cost of a solution φ lower. Since the position of the gap only appears when we are subtracting the imaginary edges, we can find the relatives extremes of the cost of the edges of a gap at position p: $H(p)$ and since we subtract $H(p)$ to $L[\varphi]$, whenever we have the maximum value of $H(p)$, we will have the minimum value of $L[\varphi]$.

$$\begin{aligned} H'(p) &= \left[\frac{(p^2-p)}{2}\right]' + \left[\frac{(f-p)*(f-p+1)}{2}\right]' = \\ &= \frac{(2p-1)}{2} + \frac{1}{2}\left[(f^2+p^2-(2f+1)p+f)\right]' = \\ &= \frac{2p-1+2p-(2f+1)}{2} = 2p-f-1 \end{aligned}$$

Therefore, will have a relative extrema at:

$$\begin{aligned} H'(p) &= 0 \\ p &= \frac{f+1}{2} \end{aligned}$$

Since its second derivative is positive:

$$H''(p) = 2$$

we can see that $H(p)$ is a parabola with minimum value at $p = \left\lfloor \frac{f+1}{2} \right\rceil$. Therefore its local maximums will be at the extremes of the domain of p: $p = 1$ and $p = f$.

$$\begin{aligned} H(1) &= \frac{(1-1)*1}{2} + \frac{(f-1)*(f-1+1)}{2} = \frac{f(f-1)}{2} \\ H(f) &= \frac{(f-1)f}{2} + \frac{(f-f)*(f-f+1)}{2} = \frac{f(f-1)}{2} \end{aligned}$$

Note that the maximum value of $H(p)$ implies that the cost of the solution $L[\varphi]$ will be minimum.

Let's consider 2 solutions: one solution φ_1 where we have n vertices and d gaps and another solution φ_2 where we have n vertices and $d+1$ gaps. Without loss of generality we can consider that the extra gap of the solution φ_2 is in the position 1, since the cost of the gap will be maximum in this position and the total cost of φ_2 will be minimum. Then, if we can prove that the minimum cost of φ_2 is always higher than the cost of φ_1 we will prove that solutions with one gap less (regardless of its position) are always as good as the one with one gap more.

Let's assume that e is the cost of all the gaps but the extra gap at position $p = 1$ of φ_2, note that $f = n + d$ but the number of gaps in φ_2 is d+1, therefore, when dealing with φ_2 we will consider $f + 1$, instead.

$$\begin{aligned} L[\varphi_1] &= \frac{(f)^3-(f)}{6} - e \\ L[\varphi_2] &= \frac{(f+1)^3-(f+1)}{6} - e - H(1) = \\ &= \frac{(f+1)^3-(f+1)}{6} - e - \frac{(f+1)f}{2} \end{aligned}$$

Now let's prove that the cost of φ_2 is always higher than the cost of φ_1

$$\begin{aligned}
L[\varphi_1] &\leq L[\varphi_2] \\
\frac{(f)^3 - (f)}{6} - e &\leq \frac{(f+1)^3 - (f+1)}{6} - e - \frac{(f+1)f}{2} \\
\frac{(f)^3 - (f)}{6} &\leq \frac{(f+1)^3 - (f+1)}{6} - \frac{3(f+1)f}{6} \\
\frac{(f)^3 - (f)}{6} &\leq \frac{(f+1)((f+1)^2 - 1 - 3f)}{6} \\
\frac{(f)^3 - (f)}{6} &\leq \frac{(f)^3 - (f)}{6} \\
0 &\leq 0
\end{aligned}$$

□

Theorem 3.3.2. When building a solution on any complete graph, with all groups restricted to be independent sets, we can always build a solution without gaps and we know that the cost of this solution will be as good as the best solution. The cost of this solution will be:

$$L[\varphi] = \frac{n^3 - n}{6}$$

Proof. From lemma 3.3.1. □

Corollary 3.3.2.1. When building a solution on any complete graph, with all groups restricted to be at most one, regardless if there are required to be connected or independent sets, we can always build a solution without gaps and we know that the cost of this solution will be as good as the best solution. The cost of this solution will be:

$$L[\varphi] = \frac{n^3 - n}{6}$$

Proof. There is no difference between a group restricted to be an independent set and a group with size one at most. Therefore, we can extend the theorem 3.3.2 to any problem on complete graphs that restricts to one the size of each group. □

3.3.3 One connected group

Let's now consider the case in complete graphs where only one group is restricted to be connected with an arbitrary size limit. As we have seen before we don't differentiate between groups restricted to be connected and groups not restricted, because in complete graphs all groups will be always connected.

Example 3.3.2. Given a complete graph with 6 vertices, $G = K_6$, and an instance with at least 8 groups where each group is restricted to be and independent set except one group which is restricted to be connected (or nothing, since there is no difference in complete graphs). Each group can have a limit size of 2.

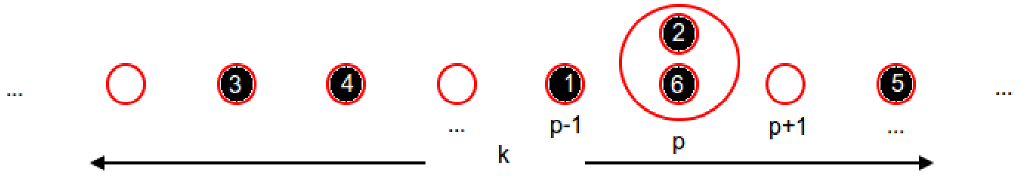


Figure 3.6: Possible solution for a complete graph K_6, with 8 groups and one of them connected. The limit size for each group is two

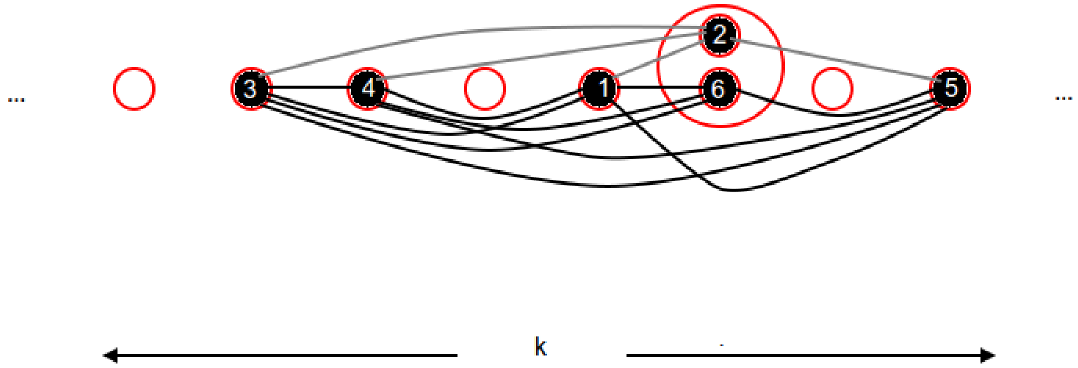


Figure 3.7: Solution of figure 3.6 with all the edges with cost greater than zero.

In the case of figure 3.7 we can see that the cost of vertex 2 its exactly the same of verex 6, actually if we take a look to the general formula for complete graphs:

$$L[\varphi] = \sum_{c1=1}^{k-1} \sum_{c2=1}^{c1} c2 * s(c1) * s(c2)$$

we will see that the cost regarding the connected group will be calculated as the size of this group multiplied to the distances to all the other groups with at least one vertex in them, that is this way because the size of all the other groups will be or 1 or 0. Let's explore the right part of the connected group, considering the representation made in figure 3.6 and 3.7. All the vertices assigned to the groups at the right of the connected group can be considered a particular case where all the groups are required to be independent sets. In this scenario we can use the theorem 3.3.2 to prove that better solutions don't have gaps. We can repeat the reasoning for the groups in the left of the connected group.

Once we know that better solutions hold when there are no gaps, the solution of figure 3.6 can be transformed to the solution of figure 3.8. Let's calculate the cost of a solution without gaps in the independent sets. The cost of this solution will be the sum of distances of all the edges, the edges inside the same group are not significant because its cost is 0. Now, imagine that the size of the connected group is 1 (the same as the groups required to be independent sets), as we have

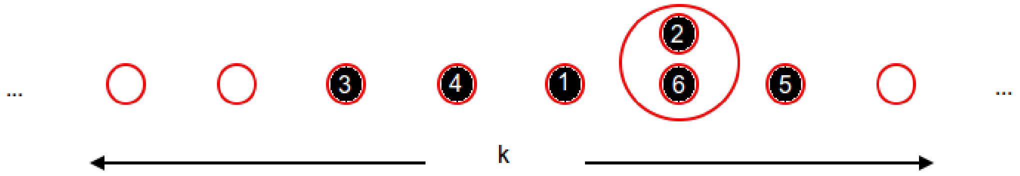


Figure 3.8: Rearrangement of the solution of figure 3.6

said before there will be no difference between that instance and one instance where all groups are independent sets, therefore, assuming $n =$ number of vertices, its cost will be:

$$L[\varphi] = \frac{n^3 - n}{6}$$

If the size of the connected group is not one we have to add the cost for each vertex in the connected group except for the one that we have already take in account in the previous calculation. Assuming n equals the total number of vertices, d equals the number of vertices assigned to the connected group, f equals the number of groups with at least one vertex in it and p is the position of the connected group regarding the first non empty group, we can observe that the cost of the layout φ will be:

$$L[\varphi] = \frac{f^3 - f}{6} + (d - 1) * H(p)$$

To understand this cost we have to imagine that one of the vertices of the connected group and all the vertices in the independent set groups form the same case than where all groups where independent sets, and all the other vertices assigned to the connected group $d - 1$ have the same cost as the vertex in the position p since all have the same adjacent vertices and the inner edges does not matter, we have already seen in 3.3.1 that this cost is $H(p)$ because its the sum of the distances to all the other groups but the one at the position p.

Lemma 3.3.3. In complete graphs with only one connected group, better solutions appear when the connected group is full, regardless of the position of the connected group.

Proof. First of all let's consider the case where there are no vertices in the connected group. There are two options:

1. The solution has all the groups assigned in the right or the left part of the connected group. Since all groups have size 1 or 0 and by theorem 3.3.2 we know that the groups with size 1 will be consecutive, we can move the beginning of the consecutive groups in order to include the connected group in the line and assign a vertex in this group, and the cost will remain the same.

2. The connected group is in the middle of the solution. In this case we are in the same situation than if all groups were independent sets. By corollary 3.3.2.1 we know that a better solution will be to assign one vertex in the connected group.

Therefore, it has no sense to consider that the connected group has no vertices, because we can rearrange the vertices to form a solution with at least the same cost.

We have just seen that at least there will be one vertex assigned to the connected group, and we want to prove that the cost of assigning a vertex to a group restricted to be an independent set, is worse than assigning the vertex to the only connected group. Let's consider two possible arrangements, assuming n vertices, and only one connected group at position p.

- φ_1, with d vertices assigned to the connected group and f groups with size at least 1.
- φ_2, with $d-1$ vertices assigned to the connected group and $f+1$ groups with size at least 1.

The linear distances of this to possible arrangements will be:

$$\begin{aligned} L[\varphi_1] &= \frac{f^3-f}{6} + (d-1)*H(p) \\ L[\varphi_2] &= \frac{(f+1)^3-(f+1)}{6} + (d-2)*H(p) \end{aligned}$$

We want to prove that the cost of φ_1 is always equal or less than the cost of φ_2

$$\begin{aligned} L[\varphi_1] &\leq L[\varphi_2] \\ \frac{f^3-f}{6} + (d-1)*H(p) &\leq \frac{(f+1)^3-(f+1)}{6} + (d-2)*H(p) \\ (d-1)*H(p) - (d-2)*H(p) &\leq \frac{(f+1)^3-(f+1)}{6} - \frac{f^3-f}{6} \\ H(p) &\leq \frac{f^3+3f^2+3f+1-f-1-f^3+f}{6} \\ H(p) &\leq \frac{f^2+f}{2} \end{aligned}$$

We have seen in 3.3.1 that H(p) is a parabola with maximum value when $p = 1$ and when $p = f$ and its value is $\dfrac{f(f-1)}{2}$

$$\begin{aligned} \frac{f(f-1)}{2} &\leq \frac{f^2+f}{2} \\ f(f-1) &\leq f(f+1) \\ 0 &\leq 2 \end{aligned}$$

□

As we have seen in complete graphs with only one group restricted to be connected and all the others being independent sets, better solutions appear when the connected group is full.
Now, the only possible movement to be done is to displace the solution horizontally maintaining the connected group full, we can see an example in figure 3.9.
Let's G be the complete graph K_n, n the number of vertices, k the total number of groups, r the maximum size of the groups, $f = n-r+1$ the number of non-empty groups, x the first non-empty group and p the position of the connected group.

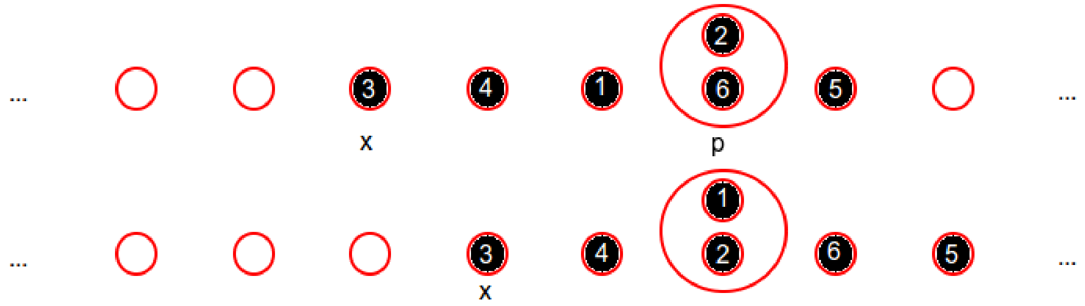


Figure 3.9: Displacement of a solution keeping the connected group full, x stands for the first non-empty group, p is the position of the connected group and in this case the number of occupied groups f would be 5

Theorem 3.3.4. When we require all groups except one to be independent sets, the best solution on this configuration comes when the position x of the first non-empty group is:

$$x = \min\left(max\left(p - \left\lceil \frac{f+1}{2} \right\rceil, 1\right), k - f\right)$$

and the cost of the solution is:

$$L[\varphi] = \frac{f^3 - f}{6} + (r-1) * H\left(p - x\right)$$

Proof. Lemma 3.3.3 and theorem 3.3.2 implies that the cost of a solution can be calculated as::

$$L[\varphi] = \frac{f^3 - f}{6} + (r-1) * H(p)$$

In lemma 3.3.3 we did not care about the position x where the first non-empty group started. Now, we need to understand that this position x is the key to the best solution. The $\frac{f^3 - f}{6}$ is constant regardless the position of the connected group, then the variable part is $(r-1)H(p)$. We have seen in 3.3.1 that $H(p)$ is a parabola with minimum value at $p = \frac{f+1}{2}$, considering that the number of non-empty groups f starts at the first non-empty group x, the best place to set x will be:

$$p = x + \left\lceil \frac{f+1}{2} \right\rceil$$

$$x = p - \left\lceil \frac{f+1}{2} \right\rceil$$

And of course, we need x to be inside the boundaries defined by 1, and $k - f$. □

3.3.4 The general case

Let's now consider the general case for complete graphs, where it can be an arbitrary number of connected groups. Due to the lack of symmetry of the general problem we have not been able to find general formulas.

Conjecture 3.3.1. Given a complete graph $G = K_n$ and an instance of the MinRLCA problem, the best solution φ (the one with the minimum linear distance) is the one where all labels can be ordered to be consecutive numbers. In this case the number of non-empty groups in φ is minimum.

If we were able to prove this conjecture we would be able to find the optimum solution for complete graphs in polynomial time. Actually, if we were able to prove that in the best solution there are no gaps, we could just look in linear time through all possible solutions without gaps, since the number of possible solutions without gaps is $O(k)$. As we have seen in 3.3.1, and assuming that $s(c)$ is the size of the group c in the solution φ, The cost for this solution would be:

$$L[\varphi] = \sum_{c1=1}^{k-1} \sum_{c2=1}^{c1} c2 * s(c1) * s(c2)$$

Chapter 4

Constraint programming models

4.1 Introduction to combinatorial optimization

A Combinatorial optimization problem is an optimization problem in which the aim is to find an optimal object from a finite set of objects. There are several techniques used to find the optimum solution, for instance, exact techniques, such as general purpose solvers or backtracking algorithms, usually uses exhaustive search combined with branch and bounds techniques. In others techniques, such as the heuristic methods, randomized search is used. We have seen in section 3.1 that the MinRLCA problem is a hard problem, therefore we have started our journey using a tool that allow us to search for the best solution using lots of knowledge packed behind several solvers.

The problem, defined as in section 2.2, has two types of restrictions, restrictions on the size of each group and restrictions on the form of the groups. The objective function is its linear distance. Our first approach to solve the problem is by using different solver-technologies. A solver is a piece of code that *solves* a generic form of problems. In general, solvers take as input a problem in a general form, called model, and returns the solution, if any.

4.2 MiniZinc

MiniZinc is a constraint modeling language. According to The MiniZinc Handbook [17], we can use MiniZinc to model constraint satisfaction and optimization problems. A model is then compiled into FlatZinc, a solver input language that is understood by a wide range of solvers. One of the reasons to pick MiniZinc as our first approach to solve the MinRLCA problem is because of its simplicity. MiniZinc takes advantage of a large library of pre-defined constraints, these constraints make it easy for us to model a very wide range of problems. A MiniZinc model its structured as follows:

- Parameters: the input of the program. This kind of variables behaves in the same way than variables in the imperative paradigm.

- Decision Variables: the variables whose value must be searched in execution time. A variable x in constraint programming has a domain $D(x)$, which is usually a finite set of values that represent the possible assignments of x, in this project we will always work with finite domains. The difference between these variables and the ones in an imperative paradigm relies on the fact that these ones don't need an assigned value, actually, the value is assigned by the solver when trying to determine if there is any value that satisfies the constraints.

- Constraints. Restrictions on the decision variables. In words of the MiniZinc Handbook [17], The constraints specify the Boolean expressions that the decision variables must satisfy to be a valid solution to the model. Let $X = \{x_1, x_2, \ldots, x_m\}$ be a set of variables with its respective domains $\{D(x_1), D(x_2), \ldots, D(x_m)\}$, the Cartesian product of the domains represent all possible assignments when there are no constraints, then, a constraint Y on the set of variables X, can be seen as a subset of its Cartesian product. The constraints can take the form of Boolean relations, such as $<, \leq, >, \geq, =, \neq$ on integer variables, or predefined predicates, such as the at_most constraint explained below.

- Objective function. Usually it is a decisional variable, whose value must be minimized or maximized. There is also the option to solve satisfy, which does not minimize or maximize an objective function, instead it tries to find an assignation that satisfy the constraints.

Once the model is complete, it is translated into FlatZinc, which is understood by some general purpose solvers. We will use three different technologies:

- Constraint programming. In this case we will use the Gecode solver [10]. We have seen that it usually has very good results. We have developed an entire model using the specific language of Gecode (see section 4.3). This provides a way to compare between the results of the MiniZinc modeling language translated to FlatZinc interpreted by Gecode and the result of a pure Gecode model.
- Mixed Integer Programming. We will use Gurobi, a proprietary solver widely used in the industry.
- Satisfability solver. In this case we will use the picatSAT solver [29], a SAT compiler implemented in picat. It has won some of the categories of the MiniZinc Challenge [16].

4.2.1 The MiniZinc model

Parameters

The number of vertices will be an integer. The representation of the graph $G = (V, E)$ has some problems in MiniZinc. We know that a graph can be represented as an adjacency matrix or an adjacency list and for this model we would rather use an adjacency list, but MiniZinc does not allow to have arrays of different lengths, therefore we have had to choose to represent the graph as an adjacency matrix.

	v_1	...	v_n
v_1	0	...	1
...	...	...	
v_n	1	...	0

Table 4.1: Adjacency matrix, when there is a 1 in the cell (v_1, v_n) means there is an edge between vertice v_1 and vertice v_n

The limit number of groups k and the limit size for each group r, will be integers. The function $C : \{1, \ldots, k\} \mapsto \{\varnothing, 0, 1\}$ will be an array C, where, $C[i] = 0$ means that that the group i must be an independent set, $C[i] = 1$ means that the group i must be connected and $C[i] = 2$ means that there are no restrictions on the form of the group.

Decisional variables

A solution of the problem has been represented by the function $\varphi : V(G) \mapsto \{1, \ldots, k\}$ and its called Restricted Linear Arrangement. In this MiniZinc model a Restricted Linear Arrangement will be an array φ of variables whose length will be the number of vertices and the domain of each variables will be the set $\{1, \ldots, k\}$. Then, if $\varphi(v) = i$ that means that the vertex v is assigned to the group i.

Constraints

The size limit constraint The first constraint is the limit on the size of the groups. In order to do so, we will use a predefined constraint in MiniZinc called the at_most constraint, which can be defined as follows:

- Given an integer r, an array of integer variables (whose domain is a subset of the integers) φ and another integer v, it requires at most r variables in φ to take the value v.

We will need k at_most constraints, one for each group. There is a special case where all the groups are restricted to have size at most 1, that can be modeled by a "alldifferent" constraint which allows the solvers to prune more values from the domains of the array, because it can be seen as a special case of bipartite matching allowing to prune all the impossible combination of values in polynomial time, for further explanation we recommend [27].

Independent set groups To restrict the vertices assigned to some groups to be independent sets, we can use a simple $\neq$ relation. Let i be a group restricted to be an independent set, then the following restriction will imply that the group i is an independent set:

$$\forall (u,v) \in E(G) \quad (\varphi(u) \neq i \vee \varphi(v) \neq i)$$

This constraint will work because, if the two ends of an edge are assigned to the same group, this group is not an independent set.

Connected groups The connectivity of graphs in constraint programming using global constraints has been very challenging. Actually, we even believed that there was no way to do it (we saw an answer from one of the creators of Gecode [22] saying that the global constraints did not support that operation and that we should implement our own propagator, which is not possible in MiniZinc). First, we realize that using the minimization factor, we could know if a subset of the vertices were connected, but we could not enforce them to be connected. The idea was to create an auxiliary matrix of decision variables, between 0 and 1. The rows would be the groups and the columns would be the vertices. Now, we would constrain for each group, all pairs of vertices with equal groups (when the group is required to be connected) to have the same value, and ensure that at least one of the vertex assigned to each connected group has the auxiliary matrix element (the one corresponding to the vertex and the group) set to 0. Then, we would maximize the sum of the elements in the auxiliary matrix and if, after finding the optimum assignment, all the vertices assigned to a connected group have its analogous auxiliary matrix elements set to 0, that would mean that the group is connected.

	v_1	...	v_n
1	1	...	1
...	...	...	
k	1	...	0

Table 4.2: Auxiliary matrix to ensure connectivity through the minimization factor

There are several problems with this definition, the first one is that modifying the objective function value in order to include the sum of the auxiliary matrix could led the wrong optimum values. The

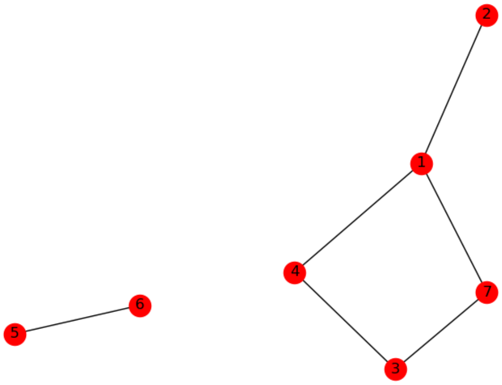


Figure 4.1: Graph with 7 vertices and 6 edges

other problem is that a restriction is always more important than a minimization factor, therefore, if we ensure that all vertices assigned to a connected group must have their analogous value of the auxiliary matrix set to zero, that would make the minimization factor irrelevant and therefore we could not ensure connectivity.

The solution that finally worked for us, was to create a three dimension array. The first dimension is the groups, the second one is the vertices and the third one is the ”steps”. In constraint programming there is no concept of state, we are just expressing properties of the decision variables. We are trying to extend, step by step, the connected groups in order to enforce connectivity. So we need to fake the ”step by step” part. Imagine tat we want express the restriction of connectivity on the graph of figure 4.1.

For each group we can imagine that we have a matrix like the one of the table 4.3.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
1	1	2	3	4	5	6	7
2							
...							
n							

Table 4.3: Matrix to ensure connectivity by expanding the minimum group to the adjacent vertices

At the first row of the matrix (the first step) we will assign to each vertex its own number, in figure 4.2 that is represented by colors, each colors is a number.

In the second row of the table 4.4 (the second step), we will assign for each pair of adjacent vertices, the value of the minimum of both. In figure 4.3, we can see that the adjacent vertices have taken the colors of the vertices with minimum label.

In the last row (final step), we will have all connected components with the same values among the connected vertices and different values between vertices of different components. We can see the result in table 4.4 and in figure 4.4

We are missing a final constraint requiring all the vertices of the connected group to have the same value. We will repeat this process for each group. Since this method uses a lot of decisional

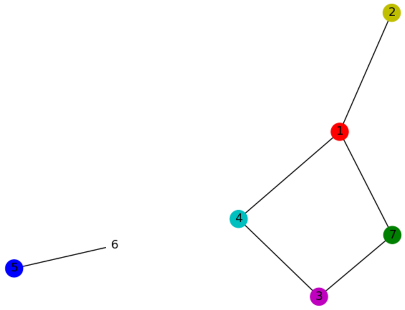


Figure 4.2: First step to ensure connectivity

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
1	1	2	3	4	5	6	7
2	1	1	3	1	5	5	1
...							
n	1	1	1	1	5	5	1

Table 4.4: Matrix to ensure connectivity. The first row is equivalent to the graph of figure 4.2. The second row is equivalent to the graph of figure 4.3, and the row n is equivalent to the graph of figure 4.4

variables, we can expect that the models will be too complicated and slow, but what we must take in account is that we have managed to create a model to enforce connectivity in constraint programming using only predefined constraints.

The objective function As defined in 2.2 the cost to be minimized is the linear distance of the solution φ, we will represent the objective function in the same way:

$$L[\varphi] = \sum_{(u,v)\in E(G)} |\varphi(u) - \varphi(v)| \tag{4.1}$$

Symmetry breaking constraints We haven't been able to find any symmetry on the general problem, but when developing the Gecode model we realized that there were some cases where C (the function that indicates the form of the groups) is symmetric, then we could exchange all the vertices assigned to each of the symmetric parts of C and the result would still be the same. We did not found a way to implement this constraint because, as we can see in the section 4.3.1, we needed some predicates that do not exist in MiniZinc. We will leave this part as future work.

The model has been updated with the name "model_bfs_expan.mzn'

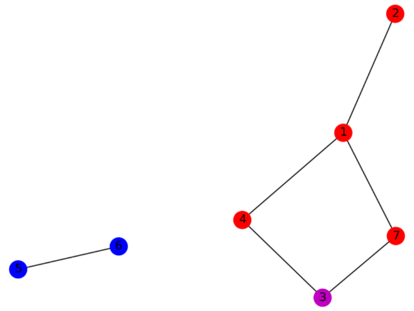


Figure 4.3: Second step to ensure connectivity

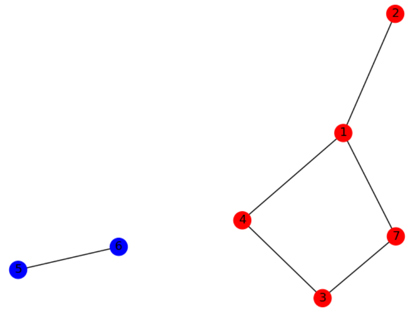


Figure 4.4: Final step to ensure connectivity

4.2.2 Experiment design

In order to create an instance of the problem $MinRLCA(G, k, r, C)$ we need to choose the graph G, the maximum number of groups k, the size limit r and the form of the groups C. Assuming $n = |V(G)|$

- k and r have been chosen as random integers between 1 and n.
- The graph. All the graphs have been generated using a binomial distribution. If a graph G is a binomial random graph generated with n vertices and a probablity of p, the resulting graph will have n vertices and every edge will exist with probability p. We have chosen this method because if we use a probability $p = 0.5$ then all graphs with n vertices appears with the same probability. The final instances have been generated using a random value of p for each instance, between 0 and 0.5. The number of vertices has been chosen as 2^i for $i \in \{5..10\}$.
- C is an array of k values, and each of them can be $\varnothing$, 0 or 1. We have repeated the process of creating graphs with 2^i vertices for $i \in \{5..10\}$ three times, one time setting all the groups to be independent sets, another setting all the groups to be connected and another mixing all the groups and including some groups without restrictions on its form.

Finally we have added some small instances in order to see the behavior on very small graphs. All the instances are named in the following way: $graph\_|V(G)|\_|E(G)|\_k\_r\_groups - form$, where $group - form$ can be ”connected”, ”IndepSet” or ”mix”.

The Uppsala University has let us run all the experiments on their computers, allowing us to use all their software. This includes all the MiniZinc solvers, such us Gecode, Gurobi and picatSAT. All experiments were run under Linux Ubuntu 18.04.1 LTS (64 bit) on an Intel(R) Xeon(R) of 2.27GHz with an 8 MB L2 cache and a 74 GB RAM (Uppsala University computers).

4.2.3 Results

Our model has shown very poor results in all the solvers and almost all the instances, but it has shown us that its possible to build a connectivity constraint using the predefined predicates in MiniZinc. Lets study the case solver by solver:

- The Gecode solver has been the best solver in terms of feasibility (the ability to find some solutions, even if the optimum has not been found). We can observe in table 4.5 that the maximum number of vertices for which Gecode has found some solutions is 32, which means that we can only solve very small instances of the problem. We can see that the instances where all groups are restricted to be independent sets are the fastest ones, actually, the connectivity seems the problem. This is what we were expecting, because the extra decisional variables used to enforce connectivity have a huge impact on the size of the problem. Although a high number of decisional variables does not make a model slow per se, usually its a good indicator that the model is too complicated and in this case the number of decisional variables needed to enforce connectivity are at least $O(k * n^2)$ where $n = |V(G)|$ and k is the number of groups required to be connected.

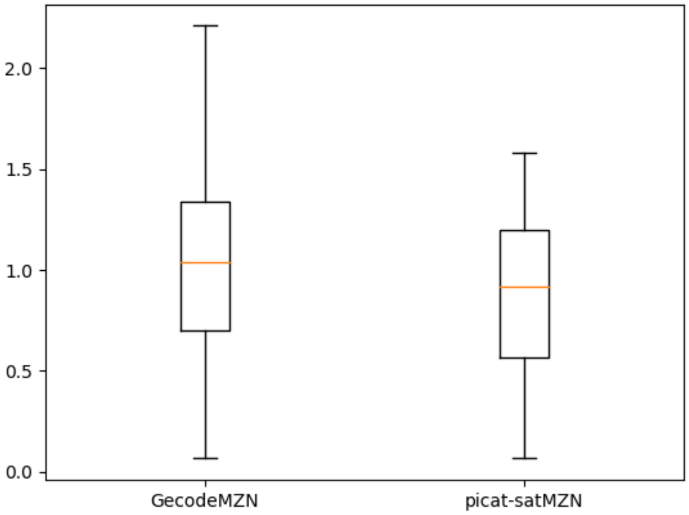
- The picatSAT solver is able to solve almost the same instances than the Gecode solver, but the big difference is the time 4.5. It is interesting to see that when there is one or more groups required to be connected, the picatSAT solver increases the time needed to solve the instances compared to the Gecode solver. But, when all groups are restricted to be independent sets, the solvers usually runs equal or even faster than the Gecode solver.

- The Gurobi solver. We were not expecting such bad results to occur using a professional solver. In every instance where at least one group is required to be connected, the Gurobi solver throws an error, which indicates that the model is too complicated. We can see that in the small instances where all the groups are required to be independent sets, good solutions are found. Also, when the number of vertices increases, the error appears even when all groups are required to be independent sets.

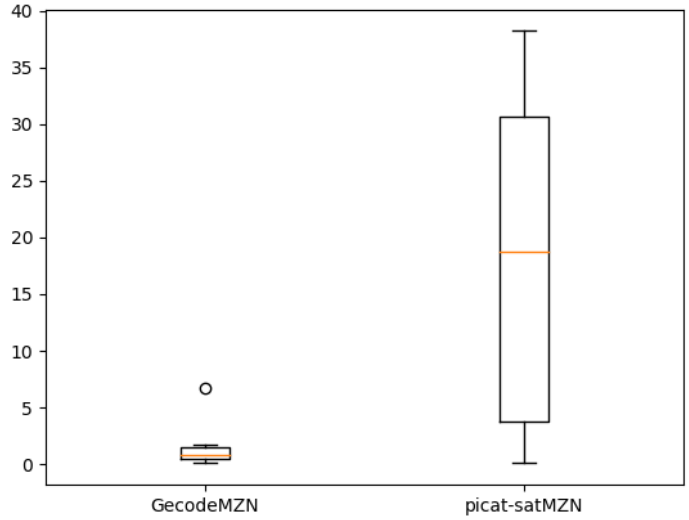
In the Gecode solver and in the picatSAT solver, the time required to compile the models, when the number of vertices is equal or greater than 128, is superior than the timeout (5 hours)

Technology	CP		MIP		SAT	
Solver	Gecode		Gurobi		Picat	
Backend	Gecode (CP)		Gurobi (MIP)		Picat (SAT)	
instance	`Linear distance`	time	`Linear distance`	time	`Linear distance`	time
graph_10_12_5_2_connected	10	9s	ERR	–	10	4m34s
graph_10_12_5_2_mix	11	10s	ERR	–	11	2m56s
graph_10_12_5_2_IndepSet	14	1s	14	1s	14	1s
graph_16_30_5_6_mix	2	50s	ERR	–	2	16m39s
graph_16_43_4_8_connected	13	4m48s	ERR	–	13	27m23s
graph_16_30_5_6_IndepSet	37	9s	37	7s	37	4s
graph_32_100_11_7_IndepSet	166	t/o	160	t/o	158	t/o
graph_32_119_11_7_mix	263	t/o	ERR	–	–	t/o
graph_32_100_11_7_connected	–	t/o	ERR	–	–	t/o
graph_32_339_11_7_mix	–	t/o	ERR	–	–	t/o
graph_64_690_5_18_mix	–	t/o	ERR	–	–	t/o
graph_64_819_6_17_connected	–	t/o	ERR	–	–	t/o
graph_64_718_7_16_IndepSet	–	t/o	ERR	–	–	t/o
graph_128_390_6_31_mix	–	t/o	ERR	–	–	t/o
graph_128_190_18_15_connected	–	t/o	ERR	–	–	t/o
graph_128_1002_55_12_IndepSet	–	t/o	ERR	–	–	t/o
graph_256_2660_177_23_mix	–	t/o	ERR	–	–	t/o
graph_256_1489_10_230_connected	–	t/o	ERR	–	–	t/o
graph_256_1789_29_41_IndepSet	–	t/o	ERR	–	–	t/o
graph_512_18934_78_31_mix	–	t/o	ERR	–	–	t/o
graph_512_15291_68_61_connected	–	t/o	ERR	–	–	t/o
graph_512_12019_103_23_IndepSet	–	t/o	ERR	–	–	t/o
graph_1024_70404_640_2_mix	–	t/o	ERR	–	–	t/o
graph_1024_120404_523_5_connected	–	t/o	ERR	–	–	t/o
graph_1024_12002_520_8_IndepSet	–	t/o	ERR	–	–	t/o

Table 4.5: Results for the MiniZinc model on the chosen instances. In the 'time' column, if the reported time is less than the time-out (5 hours here), then the reported objective value in the '`Linear distance`' column was *proven* optimal. In the 'Linear distance' column, if the reported value is 'ERR' then an error occur when executing the solver;



(a) Boxplot of the cost for edge of the Gecode and picatSAT solvers in MiniZinc



(b) Boxplot of the time for edge of the Gecode and picatSAT solvers in MiniZinc

Figure 4.5: Boxplots of the results for the Gecode and the picatSAT solvers

4.3 Gecode

Our aim is to develop algorithms capable of finding solutions for real world instances. Until now, we haven't been able to do so. One of the conclusions of the experimentation on MiniZinc is that the constraint to enforce connectivity is one of the reasons why the model is so complicated. We have seen that the Gecode solver has been the best solver so far. For this reason we have chosen Gecode to implement a model with a custom propagator to ensure connectivity.

Gecode [10] can be described as an open source C++ toolkit for developing constraint-based systems and applications. The Gecode architecture can be found in figure 4.6 taken from the Modeling and Programming with Gecode (MPG) [3], compulsory reading for anyone who wants to understand Gecode.

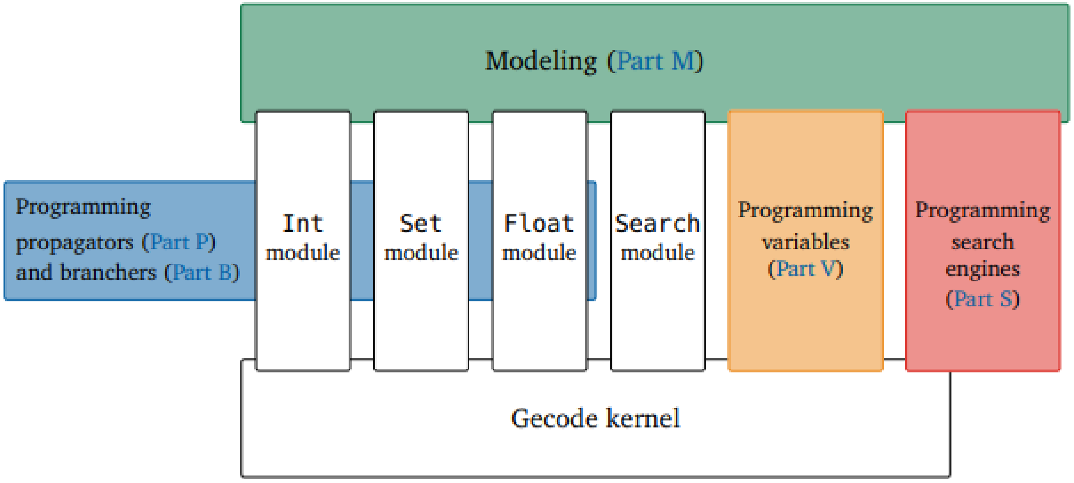

Modeling (Part M)
Programming propagators (Part P) and branchers (Part B)
Int module
Set module
Float module
Search module
Programming variables (Part V)
Programming search engines (Part S)
Gecode kernel

Figure 4.6: Gecode Architecture extracted from the MPG [3]

The Gecode environment provides C++ with all the tools that we could need to develop a constraint programming model, also provides C++ with the search engine and tools to customize almost everything. According to the MPG [3], in an object-oriented language such as C ++ , an elegant approach to programming a model is by inheritance. We can see a model as a class, making the symbiosis between c++ and the Gecode libraries as easy as possible.

Gecode provides the users with modules (Int, Set and Float in figure 4.6) to implement the decision variables as we have seen in the MiniZinc section. There are a big amount of predefined constraints (as in MiniZinc) and we have the power to create our own ones.

Let's now do a brief explanation on how Gecode works. A model has one or more decision variables which behaves exactly the same as explained in the MiniZinc section, the whole set of decisional variables is called store. A Gecode model has constraints (in the same sense that in MiniZinc). Each constraint has a propagator function which will prune impossible values of the decision variables, these propagators can achieve different levels of consistency:

- Value consistency can be defined in words of Guido Tack (one of the creators of Gecode) as ”for all variables with singleton domains, there exist values in the domains of the other variables that are consistent with the constraint”, in practice, that means that the propagator will be executed when a value is assigned to the decision variable.

- Bounds consistency checks whether the bounds of the domain are feasible. In practice that means that the propagator will be executed whenever the bounds of the domain do not meet the constraint.
- Domain consistency checks whether all values meet the restrictions. The propagator will be executed whenever a value of the domain does not meet the constraint.

When a propagator is executed, it is necessary to prune all the impossible values according to its consistency. When a propagator cannot prune more values under the current store, we call that the propagator is at fix point. When a propagator cannot prune any more values independently of the store, then the propagator is subsumed. The Gecode kernel its the director of the orchestra (using the words of the professor Pierre Flener [8]) it is the module responsible to execute the propagators and to perform the search. If a problem can be solved using only propagation (by propagation we mean polynomial time propagation), that means that the problem can be considered easy (in other words, polynomial time solvable), otherwise we need to search. The search space can be modeled as a tree, Gecode provides us with GIST, a tool to represent the search space, and to explore each node in running time, we can see an example in figure 4.7.

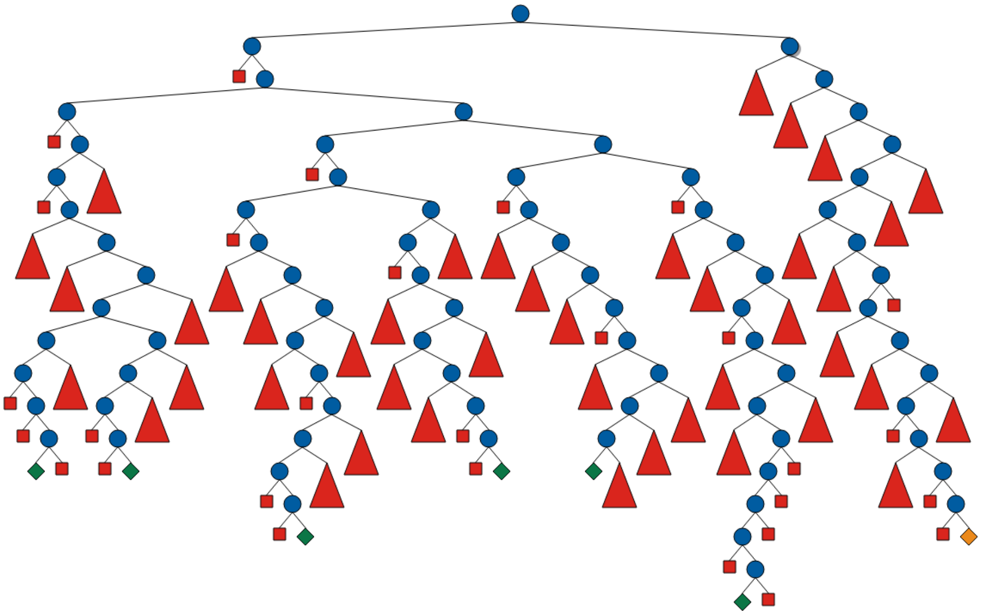


Figure 4.7: Search space of the instance graph_10_12_5_2_mix, generated with GIST

Each node represent an store, the red nodes are incorrect solutions, the triabgles means that all the sub-tree lead to incorrect solutions, green nodes are solutions, and the orange node is the best solution. The way of proceed is by executing the propagators as long as their are not at fix point or subsumed, then the search phase is executed in the way that the user has specified. For example, the user can decide the order of the decisional variables when assigning the values in search time, and which values have to be assigned first. In order to find the optimum solution we can use an already implemented strategy: the branch and bounds, the branch only is explored if the cost of a the best possible solutions is less than the best solution found.

This has been a very brief introduction to Gecode, for more information we refer to [3].

4.3.1 The Gecode model

Parameters

One of the advantages of working with C++ is that we have all the imaginable data structures available to us, that means that we have been able to represent the input graph as an adjacency list (instead of an adjacency matrix, as in MiniZinc). This, we belief will reduce the cost of reading the input (it was $O(n^2)$ now it is $O(n + m)$, understanding n as the number of vertices and m as the number of edges). The rest of the parameters have been represented in the same way as the MiniZinc model.

Decision variables

The φ decision variable is an array of Int variables, indicating the group for each vertex. The other decision variable is the cost.

Constraints

The size limit constraint We have used the same idea as in the MiniZinc model: the atmost constraint for the general model and the alldifferent constraint when the size of groups is set to one. We have set each constraint to be domain consistency, because we want to prune all intermediate values.

Connected groups To meet this constraint we have created our own propagator. Our propagator works as follows:

Let $G = (V, E)$ be the graph of the figure 4.8. If the vertex 6 is assigned to the connected group i, then, the propagator will perform a simple Breadth-first-search algorithm and will prune the group i from the domains of all the disconnected vertices, in this case, from the vertices 1, 2, 9 and 10. We have to repeat this process for all the connected groups. The cost of performing the propagation will be $O(k * (n+m))$ where n is the number of vertices and m is the number of edges. This propagator is activated only when a vertex is assigned to a group, therefore, this propagator achieve value consistency.

Independent set groups In the MiniZinc model we used a simple $\neq$ relations between variables, but there are at least two improvements:

- We are posting a lot of constraints affecting only two variables at each time. This is very inefficient and we are posting more constraints than the needed ones.

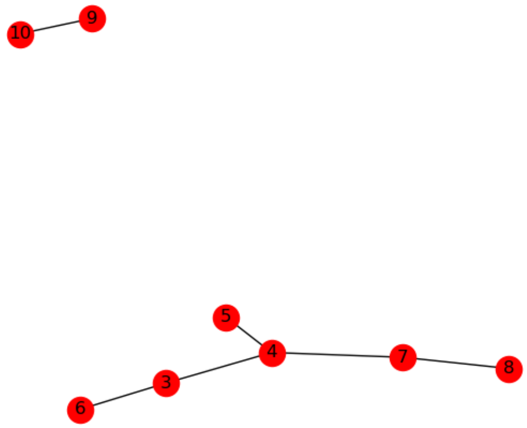


Figure 4.8: Graph with 10 vertices and 3 connected components

- we are using an or-constraint this is usually very difficult to treat for a solver, because there are to many dependencies between constraints.

For these reasons, we have decided to implement a custom propagator as well. Our propagator will use a simple idea. For each group restricted to be an independent set, we look for the set of vertices adjacent to some of the vertices already assigned to the group, and prune the group from their domain. One example could be: given the graph of figure 4.8 and a group i restricted to be an independent set, if we assign the vertices 1 and 4 to the group i, then group i will be pruned from the domains of the vertices 3, 5, 7, and 2. We can see it in the figure 4.9, where the dark-orange vertices are the ones assigned to group i and the cyan vertices are the ones that will have the group i pruned from their domain.

Symmetry breaking constraint When the function C is symmetric (the line that holds the groups is symmetric, for example when all groups are required to be the same), we can exchange the values between groups and we still have the same result. We have created a constraint requiring the minimum vertex of the first group to be less than the minimum vertex of the last group.

The objective function The first idea was to make the objective variable equal to the cost and then try to reduce the cost, but in Gecode, the objective function is treated equally than any other decisional variable, and during experimentation we were not able to find any good solutions because the search tree got stuck trying to find a solution with the same cost as the decisional variable. At the end, we realized that making the objective variable equal or greater than the cost allowed the solver to find solutions with lower costs and solved our problem. The branching strategy search first on the φ decision variable and then on the objective cost variable. The groups has been assigned in ascendant order because our experiments has shown slightly better results with this branching strategy (we tried randomized assignments and descendant order, as well).

The final model has been uploaded with the name 'model-propagators.cpp', the propagator to enforce connectivity has been uploaded with the name 'subGraphConnectivity.cpp' and the propa-

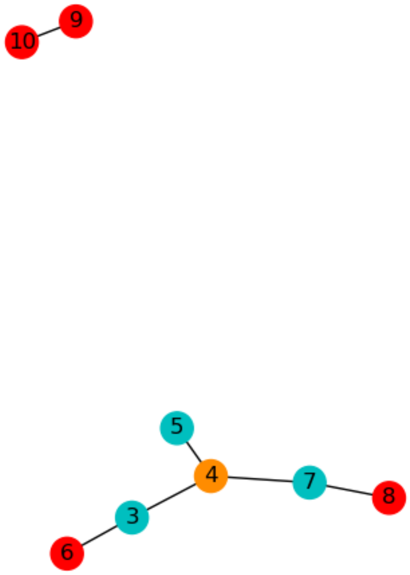


Figure 4.9: Representation of an execution of the independent set propagator

gator to enforce a group to be an independent set has been uploaded with the name 'subGraphIndependentSet.cpp'. We have created a Makefile to simplify the compilation process.

4.3.2 Experiment design

The set of instances is explained in section 4.2.2.

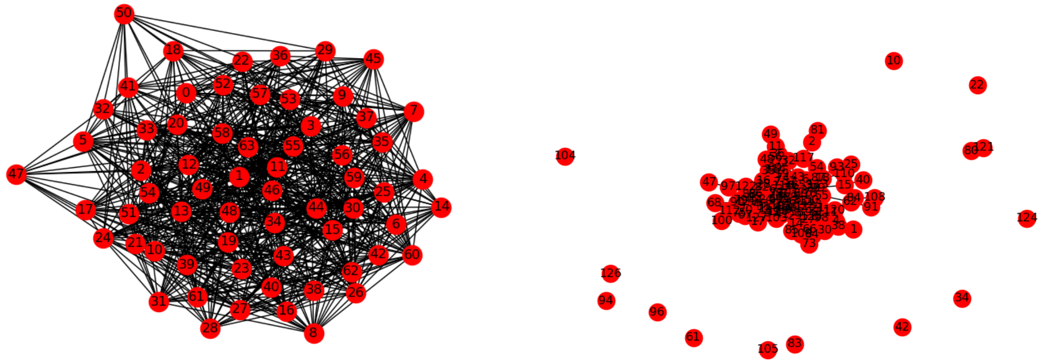
The Uppsala University has let us run all the experiments on their computers allowing us to use all their software. In this case, the g++ compiler with the libraries that conform Gecode. All experiments were run under Linux Ubuntu 18.04.1 LTS (64 bit) on an Intel(R) Xeon(R) of 2.27GHz with an 8 MB L2 cache and a 74 GB RAM (Uppsala University computers).

4.3.3 Results

The Gecode model has feasable solutions up to 1024 vertices. This has been a huge improvement over the MiniZinc model. We can observe in table 4.6, that there are 5 instances without solution. We will see that other approaches have been able to solve the instance with 32 vertices, the instance with 256 vertices and the instance with 1024 vertices. Lets take a look to the instances with 64 and 128 vertices that do not have any solution:

As we can see the graph with 64 vertices is quite dense, and the instance restricts the number of groups to 5 and the maximum size for each group to 18. If we take a look to its groups:

$$C = [0, 0, 1, 0, \varnothing]$$



(a) Graph with 64 vertices without solution

(b) Graph with 128 vertices without solution

Figure 4.10: Graphs without solution

we can see that there are only 2 groups were the vertices can be connected, while the other 3 groups must be independent sets. We are going to see that its very likely to not have any solution although we do not have a formal prove. The fact that all our methods have failed to find a solution and that we need to organize a minimum of 34 vertices into 3 independent set groups, knowing that we can not ensure that there is a solution (its not a planar graph and the number of independent sets is 3 not 4), makes us thing that this instance is a problem without solution.

The other case is clearer, we have to assign 128 vertices into 18 groups of 15 vertices each and all the groups have to be connected. There are 14 connected components and 13 of them sum 14 vertices while the other has 114 vertices, this means that we would have to split 114 vertices into 5 groups of 15 vertices each, a simple sum will tell us that this is impossible. Therefore, the instance with 128 vertices were all groups are restricted to be connected is impossible to solve.

We can now study the three instances that this model has not found solutions for, but other models have found solutions (we will see that in the following chapters). We can see that the three of them require at least some groups to be connected (two of them have all groups required to be connected while the other is a mix between connected groups, independent sets and nothing), none of the instances where all groups were required to be independent sets have end without solution. This fact makes us thing that we can keep improving the propagator to ensure connectivity.

Technology	CP	
Solver	Gecode	
model	Gecode (CP)	
instance	`Linear distance`	time
graph_10_12_5_2_connected	10	1s
graph_10_12_5_2_mix	11	5s
graph_10_12_5_2_IndepSet	14	1s
graph_16_30_5_6_mix	2	1s
graph_16_43_4_8_connected	13	1s
graph_16_30_5_6_IndepSet	37	12s
graph_32_100_11_7_IndepSet	157	t/o
graph_32_119_11_7_mix	172	t/o
graph_32_100_11_7_connected	–	t/o
graph_32_339_11_7_mix	628	t/o
graph_64_690_5_18_mix	–	t/o
graph_64_819_6_17_connected	988	t/o
graph_64_718_7_16_IndepSet	1296	t/o
graph_128_390_6_31_mix	517	t/o
graph_128_190_18_15_connected	–	t/o
graph_128_1002_55_12_IndepSet	3719	t/o
graph_256_2660_177_23_mix	–	t/o
graph_256_1489_10_230_connected	743	t/o
graph_256_1789_29_41_IndepSet	4700	t/o
graph_512_18934_78_31_mix	115365	t/o
graph_512_15291_68_61_connected	42617	t/o
graph_512_12019_103_23_IndepSet	90966	t/o
graph_1024_70404_640_2_mix	12002144	t/o
graph_1024_120404_523_5_connected	–	t/o
graph_1024_12002_520_8_IndepSet	510476	t/o

Table 4.6: Results for the Gecode model on the chosen instances. In the 'time' column, if the reported time is less than the time-out (5 hours here), then the reported objective value in the '`Linear distance`' column was *proven* optimal;

Chapter 5

Backtracking algorithm

5.1 Why backtracking

We have seen several constraint programming models, and we have been improving the results by gaining control over the execution of the algorithms. Now, we would like to implement a backtracking solver from zero. One of the reasons to do so, is to be able to introduce specific knowledge of the problem into the search engine. Another reason to implement it, is to take advantage of the concept of state, which did not exist in Constraint programming.

5.2 The Backtracking model

The model that we have created recall the Gecode idea of using the object oriented paradigm to perform the search. Each partial solution is an object, and the search engine creates all possible variations of each partial solution, an example can be found in figure 5.1

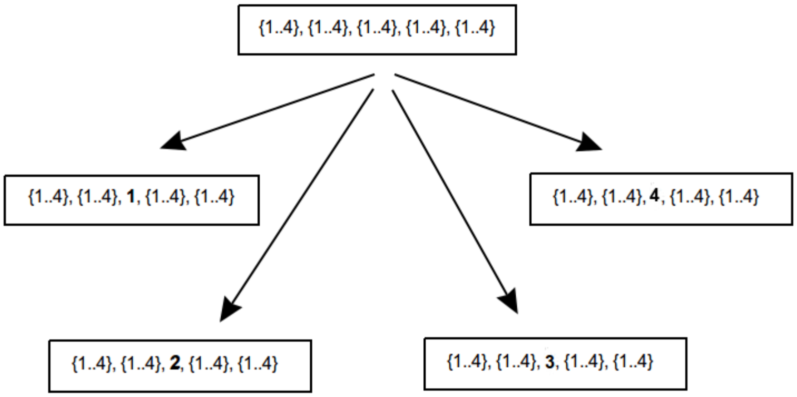


Figure 5.1: Representation of a possible assignments of a partial solution on a certain position

5.2.1 Parameters

The input parameters have no difference with the Gecode model.

5.2.2 Decision variables

The decision variable φ, can be represented as in figure 5.1, where each vertex has a domain or an assigned value.

5.2.3 Constraints

The size limit constraint We have developed a propagator (in this context propagator means a function that prunes values from the domains) which erases the complete groups from the domains of all the other variables.

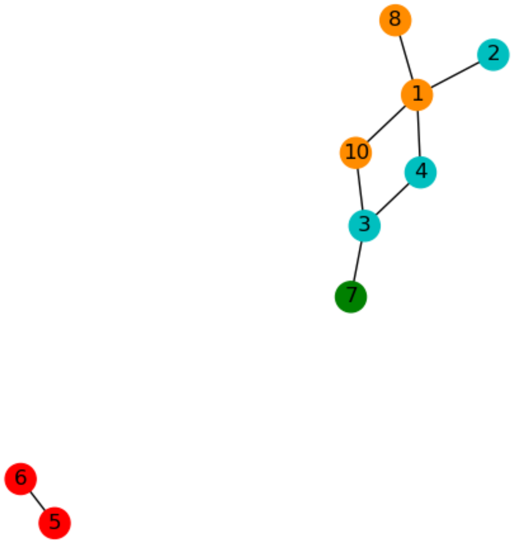


Figure 5.2: Example of the connectivity propagator

Independent set groups For this constraint we have reused the same idea as the Gecode propagator. For all the details see section 4.3.1.

Connected groups - Nearest propagator One of the reasons to create this model is to improve the constraint that enforces certain groups to be connected. The idea behind this propagator comes with the concept of state. Let's imagine that we have the graph of figure 5.2, and the vertices with color 'dark orange' are assigned to a group required to be connected. The Gecode propagator would erase the group from the domains of the red vertices, but, what it does not consider is, that if we assign the green vertex to the group, we will not have a connected group until another node (in this case the node 3) is assigned to the same group too. Then, it makes more sense, to erase the group from the domain of the green node, and add the group to the domain of the green node, again, when the vertex 3 is assigned to the group. In other words, we will establish what we call a "frontline" (blue vertices in figure 5.2), which are the vertices that we can assign to the group in the next step. This frontline will be recalculated every time we assign a vertex to the group. Luckily, we will reduce the search space. This propagator could not be developed in the Gecode model, because of there is not such a thing as state, and because when a value of a domain of a variable is erased it cannot be added again to the domain.

5.2.4 The search

The search engine is the part of the backtracking model that decides which variables are assigned first, and which groups are tested first. In the first try, we thought that any order of assignation would be correct. It turned out that, since the propagator of a connected group depends on its frontline, the vertices must be assigned to this group in the same order, because if not, we are not exploring all the search tree, and this could led to not finding the best solutions.

5.2.5 The cost calculation

Another reason to implement this model, is to implement the branch and bound algorithm using some specific knowledge of the problem, specially the cost calculation of a partial solution. As we have seen in the Geocde model, the cost of a partial solution can be calculated as the linear distance of the vertices already assigned. But, we can improve the accuracy of the cost of a partial solution a lot.

The main idea is to calculate the minimum cost for each edge, even if it is not assigned. Let's see all possible cases:

- Both ends of the edge has assigned groups. Then the cost is the absolute value of the distance.
- One of the ends is assigned, and the other not. In this case we can do a binary search in the domain to find out which is the closest group to the one already assigned.
- Neither of both is assigned. In this case we can navigate both domains in linear time $O(k)$, in order to get the minimum cost. See figure 5.3.

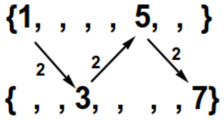


Figure 5.3: Calculating the cost of an edge whose ends are not assigned

Then, we can calculate the minimum cost of a partial solution φ, in $O(|E(G)| * k)$. This implies that can discard partial solutions, when its minimum cost is greater than the best solution found.

The final model has been uploaded with the name 'backtracking-model-nearest.cpp'. We have created a makefile to simplify the compilation process.

5.3 Experiment design

The set of instances is explained in section 4.2.2.

Again, The Uppsala University has let us run all the experiments on their computers allowing us to use all their software. All experiments were run under Linux Ubuntu 18.04.1 LTS (64 bit) on an Intel(R) Xeon(R) of 2.27GHz with an 8 MB L2 cache and a 74 GB RAM (Uppsala University computers).

5.4 Results

There has been a general improvement of the results compared with the previous models. We can see in table 5.2 that we have found solutions for the instances with 32, 256 and 1024 vertices (the ones that the Gecode model did not find). The only instance that the Gecode model has solutions for, and the Backtracking model does not, is the instance with 256 vertices and all groups restricted to be connected.

Let's study the results separating the instances where there is, at least, one group required to be connected, and the instances where all groups are restricted to be independent sets.

- Connected groups. A comparative table can be found in 5.3. Here, we can observe the biggest improvement compared with the Gecode model. There are three instances that did not have any solution in the Gecode model, that now, have feasible solutions. There is also one instance that had results under the Gecode model and now it does not. Our explanation for this instance, is that, since we are asking to form 10 connected groups of 230 vertices (the graph has 256 vertices) and the backtracking propagator to enforce connectivity tries to fill the connected groups first, this can lead to have a big connected group and up to 26 independent components, which is impossible to assign to the 9 remaining connected groups. At the same time the Gecode model does not try to fill the connected groups first, which can lead to smaller groups allowing the 10 groups to be filled in a more balanced manner.
  The cost of all the instances, but one, has improved (the instance with 512 vertices where all groups are restricted to be connected). This is another special case where the Gecode model perform better (we have not found a logical explanation to this case). Excluding these outliers, most of the instance improve its results. We think that this can be due to the new way to implement the connectivity propagator, making it better than the constraint programming one.

- Independent set groups. A comparative table can be found in 5.4. We can see that most of the instances improve a little. There are two instance which do not improve, but the general rule can be that that the Backtracking model perform better than the Gecode model in the instances where all groups are restricted to be independent sets. The propagators for independent sets of both models have been designed under the same idea, therefore the variation must be due to the cost calculation, and the search order.

Overall, most instances have improved results compared to the previous models and we have found solutions to instances that did not have them.

Technology	C++	
Solver	Backtracking	
model	Backtracking model	
instance	`Linear distance`	time
graph_10_12_5_2_connected	10	1s
graph_10_12_5_2_mix	11	1s
graph_10_12_5_2_IndepSet	14	1s
graph_16_30_5_6_mix	2	1s
graph_16_43_4_8_connected	13	1s
graph_16_30_5_6_IndepSet	37	5s
graph_32_100_11_7_IndepSet	165	t/o
graph_32_119_11_7_mix	167	t/o
graph_32_100_11_7_connected	87	t/o
graph_32_339_11_7_mix	614	t/o
graph_64_690_5_18_mix	–	t/o
graph_64_819_6_17_connected	926	t/o
graph_64_718_7_16_IndepSet	1173	t/o
graph_128_390_6_31_mix	483	t/o
graph_128_190_18_15_connected	–	t/o
graph_128_1002_55_12_IndepSet	3713	t/o
graph_256_2660_177_23_mix	9706	t/o
graph_256_1489_10_230_connected	–	t/o
graph_256_1789_29_41_IndepSet	4725	t/o
graph_512_18934_78_31_mix	114126	t/o
graph_512_15291_68_61_connected	44417	t/o
graph_512_12019_103_23_IndepSet	90864	t/o
graph_1024_70404_640_2_mix	11899045	t/o
graph_1024_120404_523_5_connected	8181938	t/o
graph_1024_12002_520_8_IndepSet	510402	t/o

Table 5.1: Results for the Backtracking model on the chosen instances. In the 'time' column, if the reported time is less than the time-out (5 hours here), then the reported objective value in the '`Linear distance`' column was *proven* optimal;

Technology	CP		C++	
Solver	Gecode		Backtracking	
model	Gecode (CP)		Backtracking model (C++)	
instance	`Linear distance`	time	`Linear distance`	time
graph_10_12_5_2_connected	10	1s	10	1s
graph_10_12_5_2_mix	11	5s	11	1s
graph_10_12_5_2_IndepSet	14	1s	14	1s
graph_16_30_5_6_mix	2	1s	2	1s
graph_16_43_4_8_connected	13	1s	13	1s
graph_16_30_5_6_IndepSet	37	12s	37	5s
graph_32_100_11_7_IndepSet	157	t/o	165	t/o
graph_32_119_11_7_mix	172	t/o	167	t/o
graph_32_100_11_7_connected	–	t/o	87	t/o
graph_32_339_11_7_mix	628	t/o	614	t/o
graph_64_690_5_18_mix	–	t/o	–	t/o
graph_64_819_6_17_connected	988	t/o	926	t/o
graph_64_718_7_16_IndepSet	1296	t/o	1173	t/o
graph_128_390_6_31_mix	517	t/o	483	t/o
graph_128_190_18_15_connected	–	t/o	–	t/o
graph_128_1002_55_12_IndepSet	3719	t/o	3713	t/o
graph_256_2660_177_23_mix	–	t/o	9706	t/o
graph_256_1489_10_230_connected	743	t/o	–	t/o
graph_256_1789_29_41_IndepSet	4700	t/o	4725	t/o
graph_512_18934_78_31_mix	115365	t/o	114126	t/o
graph_512_15291_68_61_connected	42617	t/o	44417	t/o
graph_512_12019_103_23_IndepSet	90966	t/o	90864	t/o
graph_1024_70404_640_2_mix	12002144	t/o	11899045	t/o
graph_1024_120404_523_5_connected	–	t/o	8181938	t/o
graph_1024_12002_520_8_IndepSet	510476	t/o	510402	t/o

Table 5.2: Results for the Gecode model and the Backtracking model side by side. In the 'time' column, if the reported time is less than the time-out (5 hours here), then the reported objective value in the '`Linear distance`' column was *proven* optimal.;

Technology	CP		C++	
Solver	Gecode		Backtracking	
model	Gecode (CP)		Backtracking model (C++)	
instance	`Linear distance`	time	`Linear distance`	time
graph_10_12_5_2_connected	10	1s	10	1s
graph_10_12_5_2_mix	11	5s	11	1s
graph_16_30_5_6_mix	2	1s	2	1s
graph_16_43_4_8_connected	13	1s	13	1s
graph_32_119_11_7_mix	172	t/o	167	t/o
graph_32_100_11_7_connected	–	t/o	87	t/o
graph_32_339_11_7_mix	628	t/o	614	t/o
graph_64_690_5_18_mix	–	t/o	–	t/o
graph_64_819_6_17_connected	988	t/o	926	t/o
graph_128_390_6_31_mix	517	t/o	483	t/o
graph_128_190_18_15_connected	–	t/o	–	t/o
graph_256_2660_177_23_mix	–	t/o	9706	t/o
graph_256_1489_10_230_connected	743	t/o	–	t/o
graph_512_18934_78_31_mix	115365	t/o	114126	t/o
graph_512_15291_68_61_connected	42617	t/o	44417	t/o
graph_1024_70404_640_2_mix	12002144	t/o	11899045	t/o
graph_1024_120404_523_5_connected	–	t/o	8181938	t/o

Table 5.3: Results for the instances with connected groups of the Gecode model and the Backtracking model side by side. In the 'time' column, if the reported time is less than the time-out (5 hours here), then the reported objective value in the '`Linear distance`' column was *proven* optimal.;

Technology	CP		C++	
Solver	Gecode		Backtracking	
model	Gecode (CP)		Backtracking model (C++)	
instance	`Linear distance`	time	`Linear distance`	time
graph_10_12_5_2_IndepSet	14	1s	14	1s
graph_16_30_5_6_IndepSet	37	12s	37	5s
graph_32_100_11_7_IndepSet	157	t/o	165	t/o
graph_64_718_7_16_IndepSet	1296	t/o	1173	t/o
graph_128_1002_55_12_IndepSet	3719	t/o	3713	t/o
graph_256_1789_29_41_IndepSet	4700	t/o	4725	t/o
graph_512_12019_103_23_IndepSet	90966	t/o	90864	t/o
graph_1024_12002_520_8_IndepSet	510476	t/o	510402	t/o

Table 5.4: Results for the instances where all groups are independent sets of the Gecode model and the Backtracking model side by side. In the 'time' column, if the reported time is less than the time-out (5 hours here), then the reported objective value in the '`Linear distance`' column was *proven* optimal.;

Chapter 6

Greedy algorithms

6.1 Greedy algorithms

According to [4], a greedy algorithm always makes the choice that looks best at the moment. In the previous sections, with enough time, the models would always find the optimum solution. In this section, we want to explore a solution in order to accomplish different objectives:

- We would like to have a model capable of handling big graphs, even if we do not find the optimum. Our aim is not to find the optimum solution, it is rather find whether there is one, especially when dealing with substantial graphs
- As we will see in the next chapter, one of the first ideas for the genetic algorithm was to work inside the solution space. To do so, we would need an initial solution, which would be provided by the greedy algorithm.
- In any case, we would like to have a solution capable of finding a solution in a very small period of time.

With this purposes in mind, we have designed the following greedy algorithm.

6.2 The Greedy model

If we take a look to the MinRLCA problem, it can be simplified to a sequence of steps with a set of choices at each step. We have to assign vertices into groups, each vertex is a step and the possible groups to be assigned are the set of choices.

6.2.1 Parameters

The input parameters have no difference with the Backtracking model.

6.2.2 Decision variables

The decision variable φ have no difference with the backtracking model.

6.2.3 Body of the model

If we boil down this model, we just have to do two design choices. Which group is assigned to which vertex, and the order in which we will assign the groups to the vertices. The rest of the model can be described as performing propagation in the same sense than int the backtracking model combined with the propagation of the Gecode model:

- The size limit propagation. We will use the same approach as the backtracking model.
- The independent set propagation. We will use the same approach as the backtracking model.
- The propagator to enforce connectivity. In this case we prefer to use an approach that consists on just erasing the impossible values from the domains of the vertices, instead of expanding the groups as in the backtracking model. We have experimentally seen better results when using this approach. Also, it allows us to choose any order of assignation.

Given a certain vertex how do we pick a group Lets imagine the situation where we have to assign a group to a vertex v. we know that maximum number of choices for a single vertex is the number of groups k. We have design an evaluation method where all the possibilities are evaluated. This method consist on creating as many new decisional variables as groups. Then, assign to each decisional variable a different group, into the vertex position. Then, perform propagation on each decisional variable, as described before, and looking to the final result (after pruning all the impossible choices of the domains of the solutions). The variables with more possibilities to have a final solution (when all vertices are assigned to a group) will be the picked one. In practice, that means selecting the variable with more groups in the domain of the non-assigned vertices. If there is a tie, we chose at random, because this way, we introduce some variation in the results an if we run the model on one particular instance several times we get different results, minimizing the probability of not finding any solution when there is one.

The order Our first idea was to assign the vertices in ascending order (as in Gecode). This approach has been the option chosen at the end, because we have seen that the other options did not improve or even decrease the solutions found. The other considered options were:

- To assign the neighbour vertices first. It lead to worse results.
- Random order. It lead to similar results.
- decreasing order. It lead to similar results.

6.2.4 The cost of the algorithm

We have seen that in order to perform one step we have to create at most k possible new decision variables. Each new decision has to be propagated and, the cost of propagate a decision can be described as follows:

- The cost of the size limit propagator. This propagator checks each group in order to know whether it is full. If that is the case, erases the group from all the other vertices. This process has a cost $O(kn)$ where k is the number of groups and n is the number of vertices.
- The cost of the independent set propagator. This propagator prune the groups from the domain of the adjacent nodes. The cost of this action is $O(km)$ where k is the number of groups and m is the number of edges.
- The cost of the propagator to enforce conectivity. This propagator performs a classic Breadth-first-search for each group, since the size of a group can be all the vertices, the cost of this propagator is $O(k(n+m))$, where k is the number of groups, n is the number of vertices and m is the number of edges.

The general cost of propagation is $O(kn + km + k(n + m)) = O(k(m + n))$. The propagation action takes place k times for each step, at most, and there are n steps. Therefore, the cost of the algorithm is:

$$O(n * k^2(n + m)) = O(k^2(n^2 + nm))$$

The final model has been uploaded with the name 'greedy-model.cpp'. We have created a makefile to simplify the compilation process.

6.3 Experiment design

The set of instances is explained in section 4.2.2.

In order to minimize the possibility of not finding any solution when there is one, we have run the model 5 times on each instance and we have chosen the best solution. In the time column of the table of results 6.1 we can see the sum of all five times.

Again, The Uppsala University has let us run all the experiments on their computers allowing us to use all their software. All experiments were run under Linux Ubuntu 18.04.1 LTS (64 bit) on an Intel(R) Xeon(R) of 2.27GHz with an 8 MB L2 cache and a 74 GB RAM (Uppsala University computers).

6.4 Results

All the results can be found in table 6.1, and a comparative table, with all the results of the Gecode, Backtracking and Greedy algorithms side by side, can be found in 6.2.

If we take a look to the value of the Linear distances of the solutions found, we will see that all the instances have much better results in the previous models. We did expect this kind of results, because, as we have said before, our aim was not to find the optimal, instead was find whether or not there was a feasible solution in a relatively small time.

The time needed to run the instances is very low, we can see that all instances run under 10 minuts, and all instances except two (the ones with 1024 vertices) run in less than 10 seconds. If we just want to do a quick check of whether exists a solution or not, this approach will do.

One of the problems of the greedy model is the false negatives, which are the instances that the algorithm does not find a solution when it exists. This only happened with the instances with 10 vertices: graph_10_12_5_2_connected and graph_10_12_5_2_mix, and the instance with 256 vertices connected. For the instance with 256 vertices, the same thing has happened with the backtracking model, and a possible explanation can be found with the backtracking results. For the small instances (10 vertices) what we have to say is that they are really hard to solve. It can seem easy because the previous models have solved them, but this is because the previous models performed exhaustive search and the search space of the models is very small. The fact that the number of groups is 5 and the size for each group is 2 makes the number of possible solutions very small compared to the non-solutions. If we take a look at the search tree of the instance graph_10_12_5_2_connected 6.1 or the search tree of the instance graph_10_12_5_2_mix 4.7, we will realize that the majority of choices lead to dead ends (The big red triangles), making for the greedy model very difficult to find the correct path.

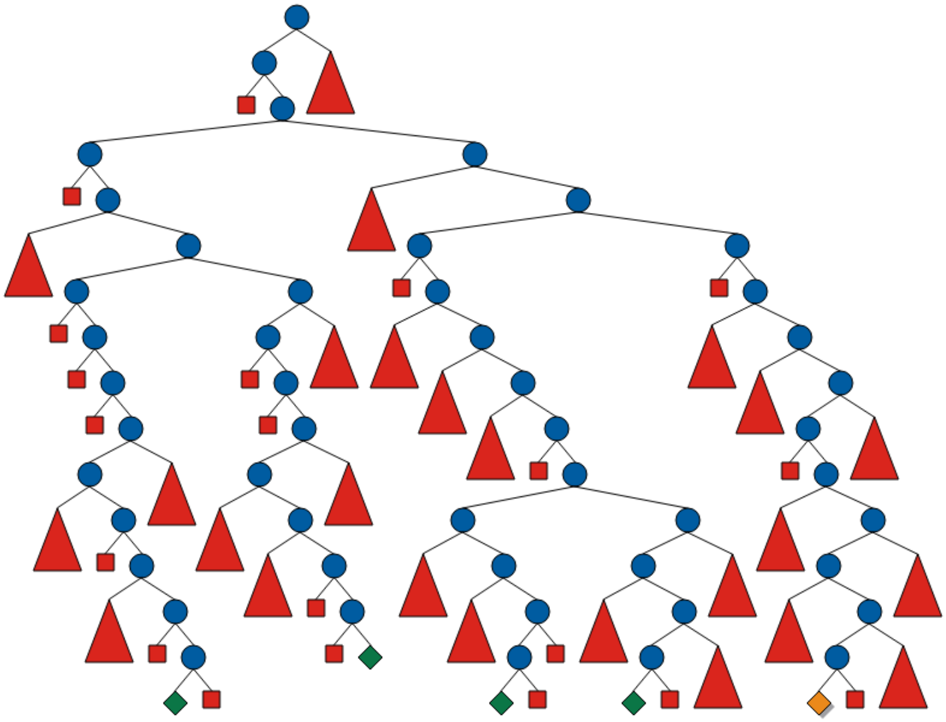


Figure 6.1: Search space of the instance graph_10_12_5_2_connected, generated with GIST

Technology	C++	
Solver	Greedy	
model	Greedy model	
instance	`Linear distance`	time
graph_10_12_5_2_connected	–	1s
graph_10_12_5_2_mix	–	1s
graph_10_12_5_2_IndepSet	21	1s
graph_16_30_5_6_mix	40	1s
graph_16_43_4_8_connected	32	1s
graph_16_30_5_6_IndepSet	55	1s
graph_32_100_11_7_IndepSet	355	1s
graph_32_119_11_7_mix	418	1s
graph_32_100_11_7_connected	314	1s
graph_32_339_11_7_mix	1171	1s
graph_64_690_5_18_mix	–	1s
graph_64_819_6_17_connected	1238	1s
graph_64_718_7_16_IndepSet	–	1s
graph_128_390_6_31_mix	728	1s
graph_128_190_18_15_connected	–	1s
graph_128_1002_55_12_IndepSet	18513	1s
graph_256_2660_177_23_mix	144277	1s
graph_256_1489_10_230_connected	–	1s
graph_256_1789_29_41_IndepSet	17970	1s
graph_512_18934_78_31_mix	478793	4s
graph_512_15291_68_61_connected	345822	8s
graph_512_12019_103_23_IndepSet	401127	1s
graph_1024_70404_640_2_mix	14908813	1m8s
graph_1024_120404_523_5_connected	20557611	8m17s
graph_1024_12002_520_8_IndepSet	2024855	3s

Table 6.1: Results for the greedy model on the chosen instances.

Technology	CP		C++		C++	
Solver	Gecode		Backtracking		Greedy	
model	Gecode (CP)		Backtracking model (C++)		Greedy model (C++)	
instance	`Linear distance`	time	`Linear distance`	time	`Linear distance`	time
graph_10_12_5_2_connected	10	1s	10	1s	–	1s
graph_10_12_5_2_mix	11	5s	11	1s	–	1s
graph_10_12_5_2_IndepSet	14	1s	14	1s	21	1s
graph_16_30_5_6_mix	2	1s	2	1s	40	1s
graph_16_43_4_8_connected	13	1s	13	1s	32	1s
graph_16_30_5_6_IndepSet	37	12s	37	5s	55	1s
graph_32_100_11_7_IndepSet	157	t/o	165	t/o	355	1s
graph_32_119_11_7_mix	172	t/o	167	t/o	418	1s
graph_32_100_11_7_connected	–	t/o	87	t/o	314	1s
graph_32_339_11_7_mix	628	t/o	614	t/o	1171	1s
graph_64_690_5_18_mix	–	t/o	–	t/o	–	1s
graph_64_819_6_17_connected	988	t/o	926	t/o	1238	1s
graph_64_718_7_16_IndepSet	1296	t/o	1173	t/o	–	1s
graph_128_390_6_31_mix	517	t/o	483	t/o	728	1s
graph_128_190_18_15_connected	–	t/o	–	t/o	–	1s
graph_128_1002_55_12_IndepSet	3719	t/o	3713	t/o	18513	1s
graph_256_2660_177_23_mix	–	t/o	9706	t/o	144277	1s
graph_256_1489_10_230_connected	743	t/o	–	t/o	–	1s
graph_256_1789_29_41_IndepSet	4700	t/o	4725	t/o	17970	1s
graph_512_18934_78_31_mix	115365	t/o	114126	t/o	478793	4s
graph_512_15291_68_61_connected	42617	t/o	44417	t/o	345822	8s
graph_512_12019_103_23_IndepSet	90966	t/o	90864	t/o	401127	1s
graph_1024_70404_640_2_mix	12002144	t/o	11899045	t/o	14908813	1m8s
graph_1024_120404_523_5_connected	–	t/o	8181938	t/o	20557611	8m17s
graph_1024_12002_520_8_IndepSet	510476	t/o	510402	t/o	2024855	3s

Table 6.2: Results for the Gecode model, the Backtracking model and the Greedy model side by side. In the 'time' column of the Gecode and the Backtracking models, if the reported time is less than the time-out (5 hours here), then the reported objective value in the '`Linear distance`' column was *proven* optimal. In the Greedy model it just indicates the time needed to execute the algorithm;

Chapter 7

Genetic algorithm

7.1 Introduction to genetic algorithms

A genetic algorithm is an heuristic method based on natural selection. This method consists on, given a population of individuals where each individual represents a solution of our problem, reproduce, mutate and select the best individuals of the population with the objective of finding the global optimum. In order to be able to use this algorithm we need a way of representing the solution of the problem that reassembles the genetic code, we will call individual to each solution. It is, also, needed a fitness function to measure the adaptability of the individual to the environment, in other words, the quality of the solution. Given a set of individuals called population and a fitness function the method requires to define three operators:

- Selection. Is the process where a subset of the population is selected to create the new generation. Usually, the selection process is a simple filter of the population where the elements with best fitness function ”survive”. We will call the subset of the best elements of the population, *elite*.
- Mutation. Is the process in which a part of the genetic code of an individual is altered, in order to create a variation of the individual that can improve its fitness function. Usually, we use a prefixed probability to decide whether perform a mutation or not.
- Crossover. Is the process for which new individuals are created, out of the old ones. The crossover happens as a recombination of the genetic code, the number of parents is usually two, because it reassemble the human evolution, but many other forms can take place. Usually, there is also a probability for the crossover operation to happen, but it is normally much higher than the mutation probability.

With all these elements in mind, let's us show how we have adapted the algorithm to deal with our problem.

7.2 Solution-space approach

Our first idea was to work in the solution-space, this seemed a good approach because all our individuals would be already solutions, and the evolution process would make the new individuals to be feasible solutions of the problem as well.

The genetic representation of the problem (a possible solution) is its restricted linear coloring arrangements φ, as we already know. Given $G = (V, E)$, the graph of figure 7.1, a possible individual can be represented as follows:

$$\varphi = [0, 1, 2, 3, 4, 0, 4, 2, 3, 1]$$

The selection operator The fitness function is the linear distance of the solution, In this case the linear distance of φ is 14. In order to implement the selection process, we sort the individuals of the population by its linear distance and select the ones with less cost.

$$L[\varphi] = \sum_{(u,v) \in E(G)} |\varphi(u) - \varphi(v)|$$

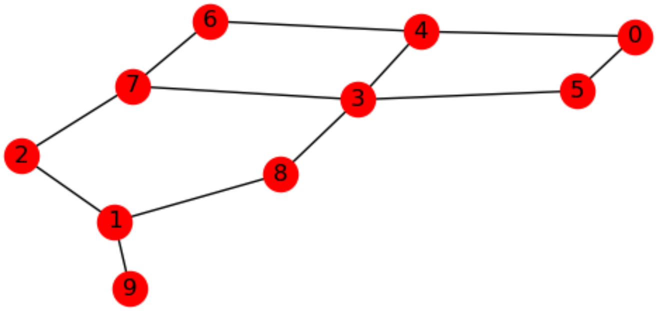


Figure 7.1: Graph of 10 vertices and 12 edges, for the instance graph_10_12_5_2_connected

The crossover operator Since the combination of 2 or more solutions is a very complicated process, we have create a crossover that creates a new individual out of another. The idea is to recombine the groups in order to create the new individual, we know that 2 groups restricted to be equal (independent sets or connected) could exchange all the vertices and the result would still be a solution. For example, given the individual already presented φ, we can exchange the vertices of the groups 2 and three producing the following individual φ_1:

$$\varphi_1 = [0, 1, 3, 2, 4, 0, 4, 3, 2, 1]$$

The linear distance of this individual is 13, a bit better than the parent.

The mutation operator Let's now consider the following individual for the instance already described:

$$\varphi_2 = [0, 3, 4, 1, 2, 0, 2, 4, 1, 3]$$

The linear distance of this individual is better than the two already presented $L[\varphi_2] = 12$. We can see that the individual φ_2 cannot be created using the described crossover operator, therefore we should be able to create a mutation operator capable of reaching this solution. The problem is that, if the mutation occurs typically in only one position of the individual (one vertex is reassigned) and the groups of the graph are complete (in this case, we have 10 vertices and 5 groups of size 2), then, the mutation operator would have to reassign several vertices in order to produce another valid solution. This process is too complicated to produce good results, and in practice, that means that part of the solution space is not explored. That is why we have discarded the option of working in the solution space.

7.3 Non-solution-space approach

In this approach, an individual would be any assignation of vertices into groups even if it does not meet the restrictions. The fitness function would be adapted to go toward the solution space, this would be accomplish by scoring the solutions by the number of restrictions that they do not meet, combined with the linear distance of the vertices that do meet the restrictions. The selection, crossover and mutation, would be the same, because now we can reach any solution (since any combination of vertices is allowed) as well as others.

In this approach all the weight falls in the fitness function. We have observed that when the fitness function is modified, it tends to produce optima that should not appear, and instead of finding good solutions just go around exploring the space of non-solutions. The problem is that finding a fitness function in these conditions is not an easy job, so we decided that we would not work in the space of non-solutions neither.

7.4 Eureka!

If we do not work in the solution space, nor in the non-solutions space, where do we work?

7.4.1 Partial-solutions space

In this approach, we use partial solutions as individuals. An example of individual can be the following:

$$\varphi = [0, 3, 4, 2, 1, 0, \{1, 4\}, \{1, 4\}, 2, 3]$$

The set values indicates which groups can be assigned in that position. As we can imagine, we will need to propagate every decision, so we will use the propagation explained in the greedy algorithm.

The crossover operator We will use the same crossover operator described in the solution-space approach. The idea is to exchange groups with the same requirements. The interesting thing is that now, we could exchange full groups with empty groups. An example would be to exchange, in φ, the vertices assigned to the group 3 for the vertices assigned to the group 4, creating the following individual:

$$\varphi = [0, 4, 3, 2, 1, 0, \{1, 3\}, \{1, 3\}, 2, 4]$$

The mutation operator This operator will now modify some of the already assigned vertices assigning them to groups, as long as the propagators allow us to do it. In other words, all individuals must be partial solutions, it is not allowed an individual which would lead to non-solutions only.

The selection operator In the backtracking model, we developed a way of calculating the linear distance of a partial solution. Let's imagine the following situation. We want to compare two partial solutions like the following ones:

$$\varphi_1 = [0, 4, 3, 2, 1, 0, \{1, 3\}, \{1, 3\}, 2, 4]$$
$$\varphi_2 = [1, 4, 3, 2, \{0, 1, 2, 3\}, \{0, 1, 2\}, \{0, 1, 2, 3\}, \{0, 1, 2, 3\}, \{0, 2, 4\}, \{0, 4\}]$$

The Linear distance of both solutions is:

$$L[\varphi_1] = 8L[\varphi_2] = 1$$

As we can see, the individual with less vertices assigned usually gets better costs. If we only take in account the linear distance as defined, we would end up with zero vertices assigned, because this is the combination with cost 0. If we try to modify the fitness function in order to assign better scored to the solutions with more vertices assigned, we could create false optima, and fall into local minimums very far from the global ones. Let's see how to solve this problem.

7.4.2 Not one, but many

The final model does not do a single genetic algorithm, but many. The idea is that using the linear distance, defined as in the backtracking model, we can compare solutions with the same number of vertices assigned. Therefore, we can create as many genetic algorithms as vertices, and do:

- Generate an initial population of p vertices as follows:
  - ∗ Let's imagine that we start the algorithm with the initial individual (zero vertices assigned). This individual, for the graph of 10 vertices and 5 groups (starting by 0) is:

    $$\varphi = [\{0,1,2,3,4\},\{0,1,2,3,4\},\{0,1,2,3,4\},\{0,1,2,3,4\},\{0,1,2,3,4\},\{0,1,2,3,4\},...]$$

  - ∗ Then we would generate an initial population assinging all the possible groups to the first vertex:

    $$\varphi_0 = [0,\{0,1,2,3,4\},...]$$
    $$\varphi_1 = [1,\{0,1,2,3,4\},...]$$
    $$\varphi_2 = [2,\{0,1,2,3,4\},...]$$
    $$\varphi_3 = [3,\{0,1,2,3,4\},...]$$
    $$\varphi_4 = [4,\{0,1,2,3,4\},...]$$

    Essentially, we are taking the individuals of the last round and creating k new individuals (the maximum number of groups) for each old individual, assigning the another vertex to all the possible groups.

- Then we perform the selection, reducing the population to a certain number of individuals, called elite.
- Then, we apply the crossover and mutation propagator, raising the number of individuals of the population again.
- Selection, again and repeat the process a fixed number of generations.
- When we have performed the genetic algorithm for x vertices assigned, we execute the next round, which consist on creating the new population from the elite of the last generation.
- We will repeat this process until $x == n$, where n is the number of vertices.

The final parameters are:

- The maximum number of individuals in the population is 200. Initially was $10 * n$, where n is the number of vertices in the graph, but it turns out to slow for the bigger instances.
- The number of individuals in the elite, is a fifth of the maximum number of individuals of the population. In the this case is 40.
- The number of generations for each round is 15 or until the sum of the linear distances of all the individuals does not improve. We now that the number of generations will be finite because the minimum linear distance of an individual is 0, then, it cannot improve forever.
- The probability of performing the crossover operator is 0.78, and the probability to perform mutation is 0.08, for each individual.

The cost We have seen before that performing propagation on an individual with n vertices, m edges and k groups is $O(k(n{+}m))$. At each round we will create at last $k{*}elite{*}num\_of\_generations$ new individuals. We will execute n rounds, therefore the final cost will be:

$$O(k^2(n+m)*15n) = O(15k^2(n^2+m))$$

The final model has been uploaded as several classes. The first one is the class Individual, the second is the class genetic-algorithm, and finally, a main. We have provided a Makefile to simplify the compilation process.

7.5 Experiment design

The set of instances is explained in section 4.2.2.

Again, The Uppsala University has let us run all the experiments on their computers allowing us to use all their software. All experiments were run under Linux Ubuntu 18.04.1 LTS (64 bit) on an Intel(R) Xeon(R) of 2.27GHz with an 8 MB L2 cache and a 74 GB RAM (Uppsala University computers).

7.6 Results

All the results can be found in table 7.1, and a comparative table with all the results of the Gecode, Backtracking and Greedy algorithms side by side can be found in 7.2.

Almost all the instances have improved all previous results. The solutions have been found in the least amount of time ever, and almost all instances have found solutions with better linear distances than the previous ones.

All the instances with equal or less than 128 vertices run in less than one minute. The only solutions that have not improved the results of the previous models are:

- The ones without solution. Even in these cases we have to take into account that the solutions for both instances in the genetic algorithm have been found in 1 second and 20 seconds respectively.
- The instances with 32 and 256 vertices with all groups restricted to be connected. We can observe that both instances restrict all groups to be connected, maybe the propagator for connectivity can be improved by using the same technique as in the backtracking model, but this would require adjustments in the order that vertices are processed. We will leave this improvement as future work.

Because of the way we have designed the genetic algorithm, a solution can only be found after performing all the previous rounds. That is why the instances with timeout do not have any results. One way to find solutions on these instances is by reducing the maximum number of individuals in the population or by reducing the number of generations for each round. The problem is that, if we do that, we are reducing the possibilities of finding a solution (we tried with several values and we realized that there was no solution to these instances). As always, it is a trade off between time and results.

We believe that the genetic algorithms have a lot to say to this problem, and, for future work, we would propose to solve the compromise between time and results

Technology	C++	
Solver	Genetic	
model	Greedy model	
instance	`Linear distance`	time
graph_10_12_5_2_connected	10	1s
graph_10_12_5_2_mix	11	1s
graph_10_12_5_2_IndepSet	14	1s
graph_16_30_5_6_mix	2	1s
graph_16_43_4_8_connected	13	1s
graph_16_30_5_6_IndepSet	37	1s
graph_32_100_11_7_IndepSet	150	1s
graph_32_119_11_7_mix	142	1s
graph_32_100_11_7_connected	100	1s
graph_32_339_11_7_mix	546	1s
graph_64_690_5_18_mix	–	1s
graph_64_819_6_17_connected	913	1s
graph_64_718_7_16_IndepSet	1176	1s
graph_128_390_6_31_mix	379	2s
graph_128_190_18_15_connected	–	2s
graph_128_1002_55_12_IndepSet	3286	25s
graph_256_2660_177_23_mix	–	6m27s
graph_256_1489_10_230_connected	756	20s
graph_256_1789_29_41_IndepSet	4133	48s
graph_512_18934_78_31_mix	108598	17m39s
graph_512_15291_68_61_connected	40387	15m41s
graph_512_12019_103_23_IndepSet	88240	12m09s
graph_1024_70404_640_2_mix	–	t/o
graph_1024_120404_523_5_connected	–	t/o
graph_1024_12002_520_8_IndepSet	437243	2h35m39s

Table 7.1: Results for the genetic model on the chosen instances.

Technology	CP		C++		C++		C++	
Solver	Gecode		Backtracking		Greedy		Genetic	
model	Gecode		Backtracking model		Greedy model		Genetic model	
instance	`L.D.`	time	`L.D.`	time	`L.D.`	time	`L.D.`	time
graph_10_12_5_2_connected	10	1s	10	1s	–	1s	10	1s
graph_10_12_5_2_mix	11	5s	11	1s	–	1s	11	1s
graph_10_12_5_2_IndepSet	14	1s	14	1s	21	1s	14	1s
graph_16_30_5_6_mix	2	1s	2	1s	40	1s	2	1s
graph_16_43_4_8_connected	13	1s	13	1s	32	1s	13	1s
graph_16_30_5_6_IndepSet	37	12s	37	5s	55	1s	37	1s
graph_32_100_11_7_IndepSet	157	t/o	165	t/o	355	1s	150	1s
graph_32_119_11_7_mix	172	t/o	167	t/o	418	1s	142	1s
graph_32_100_11_7_connected	–	t/o	87	t/o	314	1s	100	1s
graph_32_339_11_7_mix	628	t/o	614	t/o	1171	1s	546	1s
graph_64_690_5_18_mix	–	t/o	–	t/o	–	1s	–	1s
graph_64_819_6_17_connected	988	t/o	926	t/o	1238	1s	913	1s
graph_64_718_7_16_IndepSet	1296	t/o	1173	t/o	–	1s	1176	1s
graph_128_390_6_31_mix	517	t/o	483	t/o	728	1s	379	2s
graph_128_190_18_15_connected	–	t/o	–	t/o	–	1s	–	2s
graph_128_1002_55_12_IndepSet	3719	t/o	3713	t/o	18513	1s	3286	25s
graph_256_2660_177_23_mix	–	t/o	9706	t/o	144277	1s	–	6m27s
graph_256_1489_10_230_connected	743	t/o	–	t/o	–	1s	756	20s
graph_256_1789_29_41_IndepSet	4700	t/o	4725	t/o	17970	1s	4133	48s
graph_512_18934_78_31_mix	115365	t/o	114126	t/o	478793	4s	108598	17m39s
graph_512_15291_68_61_connected	42617	t/o	44417	t/o	345822	8s	40387	15m41s
graph_512_12019_103_23_IndepSet	90966	t/o	90864	t/o	401127	1s	88240	12m09s
graph_1024_70404_640_2_mix	12002144	t/o	11899045	t/o	14908813	1m8s	–	t/o
graph_1024_120404_523_5_connected	–	t/o	8181938	t/o	20557611	8m17s	–	t/o
graph_1024_12002_520_8_IndepSet	510476	t/o	510402	t/o	2024855	3s	437243	2h35m39s

Table 7.2: Results for the Gecode model, the Backtracking model, the Greedy model and the Genetic model side by side. In the 'time' column of the Gecode and the Backtracking models, if the reported time is less than the time-out (5 hours here), then the reported objective value in the '`Linear distance`' column was *proven* optimal. In the Greedy model and in the Genetic model it just indicates the time needed to execute the algorithm;

Chapter 8

Sustainability report

8.1 Sustainability analysis

Do to the nature of our project there are not many true implications from a sustainability point of view, other than the standard one in developing programs.

8.1.1 Social and environmental implications

This project has been developed within the university, in this context the social aspect can be seen as the impact that the project will have on the scientific community, how this project can have repercussions on the world and the way that the world will notice this project. This is a new problem that can model a new range of real world situations and its principal objective is to develop new algorithms. Of course, the best way that this project will have some impact on the real world is to be known for all the people who wants to study similar problems or wants to use this algorithms in some ways, so one of the most important aspects is to be available for everyone under a non restrictive software licence [24]. Even though, the social impact on the general public is low in a direct way, there can be an impact if the studied problem is ever used for some real-world application like VLSI circuit design.

8.1.2 Timing

The final work has taken more time than the originally planned. The reason is because in the constraint programming and in the genetic models we had to deal with additional difficulties, we expected this kind of delays given the magnitude of the project. Overall, we can say that the goal of having it ready for the July call has been met successfully.

Chapter 9

Final words

9.1 Conclusions

In this project we have done an initial study for the Minimum Restricted Linear Coloring Arrangement problem.

First of all, we have studied the problem from a theoretical point of view. There is no doubt the Minimum Restricted Linear Coloring Arrangement problem is a computationally difficult problem. We have studied the case for complete graphs and we have advanced with some specific instances, concluding with a conjecture on the general case.

Secondly, we have done an experimental study of the problem from different perspectives. We have created a model using only predefined constraints in MiniZinc. Although, the experiments on this model have given poor results, we have seen that it is possible to enforce connectivity using only the predefined constraints. We have made another constraint programming model using Gecode. This time, we have implemented our own propagators, leading up to much better results. Due to the nature of constraint programming, we have had to implement a branch and bound backtracking algorithm, in order to implement new ideas on how to propagate connectivity. The new model was, in general, better than the Gecode one, improving most of the results. After the backtracking model, we start looking for faster models. First, we developed a greedy algorithm which solves the problem in less than 10 minutes for all the tested cases. The solutions found are not very good, but they can be used as a first approach. Finally, we have created a genetic algorithm that has beaten all the records. I would have liked to have more time to keep improving the genetic algorithm. However, we hope to have done the first step of many in the study of the Minimum Restricted Linear Coloring Arrangement Problem.

9.2 Future work

There is a long way to go in the Minimum Restricted Linear Coloring Arrangement Problem. There is a wide range of available options: implementing new strategies to the custom propagators on the constraint programming model, continue exploring the backtracking approach to the connectivity propagator or improving the current genetic model to make it capable of handling bigger graphs.

Another path to follow would be to try to find more exact algorithms for particular families of graphs, and prove or find a counter example to the conjecture made in section 3.3.4.

Conjecture. Given a complete graph $G = K_n$ and an instance of the MinRLCA problem, the best solution φ (the one with the minimum linear distance) is the one where all labels can be ordered to be consecutive numbers. In this case the number of non-empty groups in φ is minimum.

Comment. We have created a $C++$ model, uploaded with the name 'exact-model-complete-graphs.cpp', with the idea of the conjecture, and for the few tested cases the model finds the optimum (at least compared to the results obtained with the other models). Because of time constraints, we have not included these results in the final document.

We initially planned to made publicly available the results of this project. Due to time constraints and other technical problems with the server, the task remains undone. However, the results are well documented so that it will be easy to upload in the near future

Bibliography

[1] Isaac Sánchez Barrera. Algorithms for the linear colouring arrangement problem. Bachelor's thesis, Facultat d'Informàtica de Barcelona, Universitat Politècnica de Catalunya, 2015.

[2] Janka Chlebíková. Approximating the maximally balanced connected partition problem in graphs. *Information Processing Letters*, 60(5):225 – 230, 1996.

[3] Guido Tack Christian Shulte and Mikael Z. Lagerkvist. Modeling and programming with gecode. `https://www.gecode.org/doc-latest/MPG.pdf`, 2019.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[5] Josep Díaz, Jordi Petit, and Maria J. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34:313–356, 2002.

[6] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics).* Springer, August 2005.

[7] Brankica Djurić, Jozef Kratica, Dušan Tošić, and Vladimir Filipović. Solving the maximally balanced connected partition problem in graphs by using genetic algorithm. *Computing and Informatics*, 27:341–354, 01 2008.

[8] Pierre Flener. Pierre flener web page. `http://user.it.uu.se/~pierref/`, 2019.

[9] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 47–63, New York, NY, USA, 1974. ACM.

[10] Gecode. Gecode. `https://www.gecode.org/`, 2019.

[11] GNU. Gcc. `https://gcc.gnu.org/`, 2019.

[12] L. Harper. Optimal assignments of numbers to vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, 1964.

[13] Yehuda Koren and David Harel. A multi-scale algorithm for the linear arrangement problem. In *Revised Papers from the 28th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '02, pages 296–309, London, UK, UK, 2002. Springer-Verlag.

[14] Donald L. Adolphson. Single machine job sequencing with precedence constraints. *SIAM J. Comput.*, 6:40–54, 03 1977.

[15] Dragan Matic and Milan Božić. Maximally balanced connected partition problem in graphs : Application in education, 2012.

[16] MiniZinc. The minizinc challenge. `https://www.minizinc.org/challenge.html`, 2018.

[17] MiniZinc. The minizinc handbook. `https://www.minizinc.org/doc-2.2.3/en/index.html#the-minizinc-handbook`, 2018.

[18] Graeme Mitchison and Richard Durbin. Optimal numberings of an n n array. *SIAM J. Algebraic Discrete Methods*, 7(4):571–582, October 1986.

[19] Jordi Petit. Experiments on the minimum linear arrangement problem. *J. Exp. Algorithmics*, 8, December 2003.

[20] Jordi Petit. Addenda to the survey of layout problems. *Bulletin of the EATCS*, 105:177–201, 2011.

[21] R. Ravi, Ajit Agrawal, and Philip Klein. Ordering problems approximated: Single-processor scheduling and interval graph completion. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 751–762, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[22] Christian Schulte. Graph connectivity. `https://www.gecode.org/users-archive/2009-May/002403.html`, 2009.

[23] Farhad Shahrokhi, Ondrej Sýkora, László A. Székely, and Imrich Vrto. On bipartite drawings and the linear arrangement problem. *SIAM J. Comput.*, 30:1773–1789, 2000.

[24] Richard Stallman. The right to read. *Commun. ACM*, 40(2):85–87, February 1997.

[25] UPC. Normativa sobre els drets de propietat industrial i intel·lectual a la upc. `https://www.upc.edu/normatives/ca/documents/consell-de-govern/normativa-sobre-els-drets-de-propietat-intelb7lecutal-de-la-upc/view`, 2008.

[26] UPC. Pressupost upc 2018. `https://www.upc.edu/ca/la-upc/la-institucio/fets-i-xifres/pressupost_upc_2018_final-1.pdf`, 2018.

[27] Willem-Jan van Hoeve. The alldifferent constraint: A survey. *CoRR*, cs.PL/0105015, 05 2001.

[28] Montserrat Brufau Vidal. An experimental study of the minimum linear colouring arrangement problem. Bachelor's thesis, Facultat de Matemàtiques i Estadística, Universitat Politècnica de Catalunya, 2016-2017.

[29] Neng-Fa Zhou and Håkan Kjellerstrand. The picat-sat compiler. In *PADL*, 2016.