Algorithms for the linear colouring arrangement problem

Bachelor's thesis

Author:
Isaac Sánchez Barrera
Opting for the Bachelor's degree
in Informatics Engineering
Computing specialisation

Director:
María José Serna Iglesias
Department of Computer Science

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya · BarcelonaTech

Defence date:
Wednesday 28th January, 2015

Abstract

This project undertakes the task of developing efficient algorithms for solving or approximating the **Minimum linear colouring arrangement** problem for graphs (MINLCA), a variation of the **Minimum linear arrangement** problem (MINLA). In our case, the linear mapping is defined on the set of colours of a proper graph colouring instead of its vertices. It is the first approach to its algorithms and computational complexity, and we present the results on a simple, open-access benchmarking platform.

Resum

El projecte emprèn la tasca de desenvolupar algorismes eficients per resoldre o aproximar el problema de l'**arranjament lineal mínim d'una coloració** per a grafs (MINLCA), una variació del problema de l'**arranjament lineal mínim** (MINLA). En aquest cas, l'assignació lineal està definida en el conjunt de colors d'una coloració correcta del graf en comptes dels seus vèrtexs. És la primera aproximació als seus algorismes i complexitat computacional, i en presentem els resultats a una plataforma de comparació simple i d'accés obert.

Resumen

El proyecto lleva a cabo la tarea de desarrollar algoritmos eficientes para resolver o aproximar el problema de la **colocación lineal mínima de una coloración** para grafos (MINLCA), una variación del problema de la **colocación lineal mínima** (MINLA). En este caso, la asignación lineal está definida en el conjunto de colores de una coloración correcta del grafo en lugar de sus vértices. Es la primera aproximación a sus algoritmos y complejidad computacional, y presentamos los resultados en una plataforma de comparación simple y de acceso abierto.

Acknowledgements

First of all, I would like to thank professor Maria Serna from the Department of Computer Science (CS) for accepting to direct my Bachelor's Thesis and proposing the project herself. This project would have not been possible without her support and guidance, including all the material, help when reviewing the documents, and support during the presentation rehearsals. From the moment I was presented with the project it seemed very interesting, and a good application for everything I had studied during all my years at University, both in Computer Science (a.k.a. Informatics Engineering, Computing specialisation) and Mathematics.

I cannot but thank professors Jordi Cortadella and Jordi Petit from CS. Right at the beginning of the project they assisted me when choosing the software for developing the algorithms and looking for the state of the art of the problem.

I would also like to thank professor Carme Àlvarez from CS for having shown interest when I was seeking a director. And professor Dani Jiménez from the Department of Computer Architecture for having proposed an interesting project when it was still a bit early for me, and apologies for having answered so late.

Executing the algorithms for the project might have been impossible without using the cluster at RDlab. Therefore, I would like to thank Gabriel and Iván from RDlab for having helped me with the doubts and problems I had when using the cluster. It is fair to say they have saved the project on two occasions.

I should not forget to thank professors Marc Noy and Simeon Ball from the Departments of Applied Mathematics II and IV, respectively, for letting me attend their course on Combinatorics and Graph Theory. It has been a very interesting course, with some ideas I have been able to apply to this project, and has helped me not lose contact with the academic world (apart from this thesis).

Though not strictly related to the project, thanks to my friends Alberto, Alex, Anne, Ander, Francesc, Guillermo and Marta and my mates from the choir. The many good times I have spent with them during the last four and a half months have been invaluable.

Last but not least, thanks to my parents and sister. Without their comprehension, I would not have been able to embrace this project properly.

Contents

List of Tables

List of Figures

List of Algorithms

Part I

Introduction

1 Problem definition and scope

1.1 Problem formulation

Nowadays many real life problems, especially in engineering, can be modelled as problems in graphs. One big group of such problems are those known as *graph layout problems*, which consist in finding a way to place the vertices of a graph in a grid or in a line so that an objective mathematical function of the edge induced lengths is minimised or maximised.

Of all these layout problems, a thoroughly studied one by the computer science community is the **minimum linear arrangement** problem. The objective is to find the best way to order the vertices of a graph, which can be seen as putting them in a straight line at equidistant points, such that the sum of the induced distances between two adjacent vertices is the minimum. This can be more formally defined using mathematical notation:

Problem 1.1 (Minimum linear arrangement). Given a graph $G = (V, E)$, with $|V| = n$, find a bijective function $\varphi\colon V \longrightarrow \{1, \dots, n\}$ such that $\displaystyle\sum_{\{u,v\}\in E} |\varphi(u) - \varphi(v)|$ is the minimum possible.

However, the problem we are going to study is a variation of MINLA. Instead of putting all of the vertices in a straight line, each one in a different position, we are now allowed to group them as long as not two vertices in a group are adjacent. Mathematically speaking, the function φ only needs to be a *proper colouring* of the graph. Because of this, this problem can be known as the **minimum linear colouring arrangement** problem, or MINLCA for short.

1.2 Goals of the project

The most general goal of the project is the study of the *minimum linear colouring arrangement* problem (MINLCA), a generalisation of the well-known graph layout problem *minimum linear arrangement* (MINLA). The results obtained during the development of this project should serve as a **basis for further studies** on this problem.

We initially want to **establish the problem complexity** in a classic way (NP-hardness) and from there continue with some theoretical and practical results on certain types of graphs. In order to do this, we will **develop some heuristic algorithms** and evaluate their performance on different classes of graphs such as bipartite graphs and random graphs generated using a binomial distribution for the edges.

The heuristic algorithms we plan to develop will mostly use greedy and local search approaches, with the help of linear programming for their development. Also, whenever possible, we will use integer linear programming (ILP) models so as to get the actual optimums for the instances used. That way, we will be able to **analyse** the empirical results from a **practical point of view**. This will allow us to classify the algorithms along with the theoretical results according to their performance.

As a final step we plan to build a simple, online **benchmarking platform** to compare the obtained results and possibly a technical report including all of them. As stated at the beginning of this section, this should allow the computer science community to have the most basic results in order to study the problem in more depth.

1.3 Scope of the project

A computational problem can be studied in many different ways, from many points of view. This can cause a project like this to stall and reach nowhere, so it is important to set some limits in the ways it is going to be studied.

To start off, the time constraints for the development of the project mean we cannot study this problem for too many families of graphs. We will mainly concentrate in binomial random graphs and random geometric graphs together with some ad-hoc graph families.

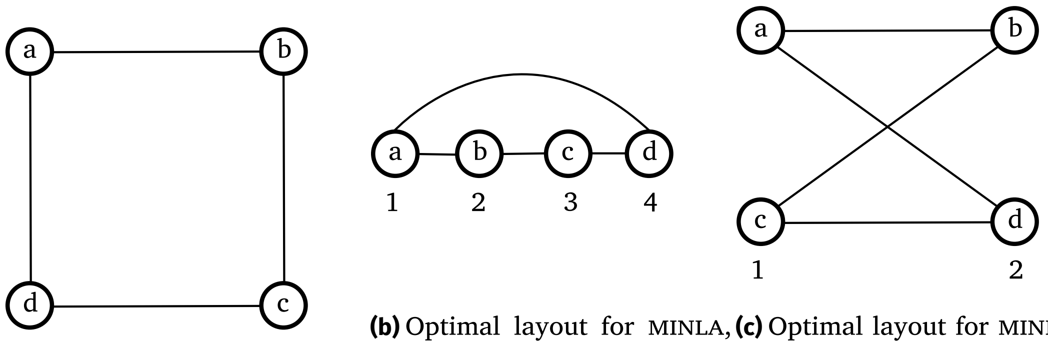
There is no need to reinvent the wheel. Because of that, we are going to use existing libraries of data structures for graph representation and software which is already well tested in order to solve the linear programming models. However, we should explain the reasons for choosing some options instead of others.

Finally, this project should be understandable on its own, including people with little to no knowledge of graph theory. As such, it should contain all needed definitions and results. What this means is not that the project should be a book on graph theory but that all required basic knowledge is included within. For deeper understanding for the reader, however, all the provided references could be of use.

2 Context and state of the art

2.1 Interest in the problem

The study of the complexity of computational problems is an important field in computer science and mathematics. As this problem can be shown to be computationally difficult, theoretical and practical results on it would be useful for the whole computer science community. It is also important to say that the *Algorithmics, Bioinformatics, Complexity and formal Methods*



(a) Original graph **(b)** Optimal layout for MINLA, cost 6 **(c)** Optimal layout for MINLCA, cost 4

Figure 2.1: Comparison between MINLA and MINLCA. The numbers below the nodes v are the $\varphi(v)$ values.

(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 particular, it is an initial generalisation to the *minimum linear arrangement* problem thoroughly studied by Jordi Petit among others, see [1], [2] for a full list of results and references to this and other graph layout problems. As such, the study of this problem will benefit to both the ALBCOM group and the computer science community in general.

Apart from being a graph layout problem, MINLCA is strongly related to other very well known graph problems: **graph colouring** and **graph homomorphism**. When regarded as a special case of the latter, it is possible to find a real world application related to computer science. The vertices of the graph can be seen as different tasks of a computer program and the edges as incompatibility rules between them (two tasks which need to be executed by different computers; but that may need to share information). Considering the computers are sequentially numbered and physically distributed in the order of the sequence, φ would the assignment of the tasks to the different computers which minimises the physical distance between incompatible tasks sharing information. Considering this, the solution might be a way to minimise the time and energy spent in information exchange, though it is an oversimplified model.

2.2 Current state of the art

Even though the minimum linear arrangement problem has been studied for many years now, our variation has not. All the existing results that we know of are unpublished and in many cases are not even results but simple conjectures.

The main difference between the two problems is on the solution φ: in the original problem it has to be a bijective mapping between the set of vertices and a set of integers, but in our case we only need it to be a *proper colouring*. Since every linear arrangement is a linear colouring arrangement in particular, all the known upper bounds for the MINLA problem are also valid (though rough) bounds for the MINLCA problem. The differences between the problems can be seen in Fig. 2.1.

Some of the algorithms for MINLA might give ideas on how to develop the algorithms for our problem, but their differences will mean that the algorithms need to be different. We plan to use local search, but using parallelisation techniques such as those in [3] falls out of the scope of this project because of the time constraints. What Petit does in that paper is to combine a technique known as *spectral sequencing*, which, as Koren and Harel [4] explain, works by doing some operations on a matrix generated with the graph information in order to get good initial solution, with a parallel version of simulated annealing.

It is interesting to notice that, while [5] by Petit is dated of 2003, the results were originally published in 2001 on his page at the Computer Science department website before Koren and Harel published their paper, and their solutions are an improvement in execution time to Petit's.

When using the ILP formulation for the problem, the current state of the art solver is the Gurobi Optimizer. When looking for a graph library for C++, one of the best options is LEMON (Library for Efficient Modelling and Optimisation in Networks) [6]. Unfortunately, there is no interface allowing to use LEMON and Gurobi simultaneously. Thus one of our objectives is to develop such interface using the C++ reference documentation [7].

LEMON does have interfaces for other solvers, but either they are not practical with our time constraints or their price is not accessible to a research project at this level. Open source projects such as the Clp solver from the COIN-OR initiative [8] take too much time to solve an ILP model, or even get stuck when there are many variables and constraints. Other commercial solvers like the CPLEX Optimizer from IBM [9] are expensive and constrained even for academic use, and Gurobi is currently one of the best available options.

Other graph libraries have been discarded in favour of LEMON, like the open source BGL (Boost Graph Library) [10] and the commercial LEDA (Library of Efficient Data types and Algorithms) [11]. The former is more complex and difficult to use than LEMON and requires the whole installation of the Boost libraries, and the latter has many restrictions when using it for free or for research purposes.

All in all, there are not any results publicly available for the problem; at least that we know of. With the use of existing results for its similar problems and theoretical results on colourings for random graphs, like the ones by Bollobás [12], this project should serve as a basis for any future studies in the subject.

3 The project within the computing specialisation

3.1 Relation with the studied subjects

Algorithmics and Advanced algorithmics These two subjects are together because of the strong link between them. On the one hand, *Algorithmics* served as a basis for the development and analysis of greedy algorithms, among others. On the other hand, *Advanced algorithmics* was partly focused in doing a more advanced study on the complexity of the problems, their approximability and the quality of the approximation algorithms.

Artificial intelligence Most of the knowledge I obtained about local search algorithms was while I was enrolled in this subject. For instance, this has helped when developing the operators which generate the neighbourhood of each node in the search space, and to decide on the compromise between having access to the whole search space and the speed of the algorithm.

Operations research This is not a subject I have studied, but I have done a strongly related one in the degree in Mathematics, *Mathematical programming*. The knowledge I got in this subject has come of use when modelling the problem as a linear programme with integrality constraints on the variables (an integer linear programme) and to understand how the solvers work.

3.2 Technical skills and level of achievement

CCO1.1 To evaluate the computational complexity of a problem, know the algorithmic strategies which can solve it and recommend, develop and implement the solution which guarantees the best performance according to the established requirements [in depth].

The main goal of the project is exactly that, to study the complexity of MINLCA and to study and implement different approaches to solving it which may have different requirements (in the use of space and time, or the quality in comparison with the optima). We have studied the problem and from there, have developed, analysed and implemented different approximation algorithms.

CCO1.3 To define, evaluate and select platforms to develop and produce hardware and software for developing computer applications and services of different complexities [a little].

In order to decide which technologies we were going to use for the development of the project, we needed to evaluate the available options and justify why we have chosen one over the rest. Because of time constraints, however, this analysis was done right at the beginning and taking the decisions with the objective of simplifying the work.

CCO2.3 To develop and evaluate interactive systems and systems that show complex information, and its application to solve person-computer interaction problems [a little].

A final goal for the project is to present the results in an online benchmarking platform (i.e., a simple website). The presentation of technical results needs a prior evaluation on what is good way to show these. Even so, the goal of the project is not to do a study on the presentation of technical results, and for that reason this part has not had a strong dedication.

CCO3.1 To implement critical code following criteria like execution time, efficiency and security [quite a lot].

Related to CCO1.1, approximation algorithms are a compromise between the optimal results and their execution time/use of memory. There is no reason to develop approximation algorithms if they are less efficient in memory and slower than their exact counterparts. To take this into account, the algorithms have been developed trying to do the computations in an incremental manner. In the implementation, we have made all our efforts to use as little memory as possible, but always without compromising the execution times.

Part II

Project management

4 Project planning

4.1 Schedule

The project was planned to take around four months (half a university year), and the final presentations for the second turn of the first half of academic year 2014-2015 should take place between 26th and 30th January, 2015. Having that in mind, the deadline for the follow-up milestone was originally thought to be in mid November, 2014. When the deadlines were published, the date

Since writing a report after doing all the work is not an option when considering the kind of project, it has been planned during the development of all the other phases. That is usually indicated as the *formalisation* and *analysis* steps in every phase.

The initial plan, explained in the following sections, is also available as a Gantt chart in Fig. 4.1.

4.2 Project planning

4.2.1 Resources

The project was planned to fit the date constraints. The resources included in the planning are:

FPI-GEP The human resources for project management, with around 2 hours per day

FPI The human resources for the project development, around 5 hours per day except during project management (max. 3 hours per day)

Computer cluster The use of the computer cluster resources. It is shown in a sequential way on the Gantt chart so as to have in mind the total computation time, but the cluster can execute more than one program simultaneously and the execution time is less than the computation time.

In order not to clutter the Gantt chart even more, it is missing a weekly two-hour meeting with the project director, planned for every Tuesday. The chart also misses the workstation used for writing the reports and developing the algorithms. Extensive explanations for the contents of the chart are to come in the following sections.

4.2.2 Project management course

Considering it is worth 3 ECTS credits, it should take around 75 hours of student work, 25 per credit. This phase had 7 steps, and all of them took more or less the same time (around 10 hours of work each step). According to the course plan, these steps were:

1. Scope definition
2. Project planning
3. Budget and sustainability
4. Initial presentation
5. Context and bibliography
6. Module for degree specialisation conditions
7. Final presentation and document

After preparing the final presentation and document, there was a presentation on Monday 21st October, 2014, at 15:00.

There is a direct precedence relationship between all the steps. In Fig. 4.1, the tasks correspond to identifiers 2 to 8. Tasks 9 and 10 are also part of this phase, and correspond to the rehearsals of the presentation for the initial milestone and the presentation itself.

4.2.3 Resources preparation

This phase, which took part simultaneously with the previous one, consists of the preparation of the development environment and the tools used to write the reports in the workstation. It also includes getting to know how the computer cluster works. This cluster is quite similar to the cluster used in another course of the Bachelor's degree, but some tests were needed to make a better use of the resources.

4.2.4 Project preparation

The initial part of the definitions and theorems had already been written during the management course, during the context and bibliography step. Two of the reference papers are a 2002 survey on graph layouts by Díaz, Petit and Serna [1] and the 2011 addenda by Petit [2]. Apart from the results on layout problems, they present many ways of generating random graphs which can be used for the creation of new instances of the problem.

Another way of getting instances for the problem is to use one of the many databases of graphs freely available (for free use) on the Internet. These databases usually include real-life graphs (tube maps, service locations, etc.) and might be useful in some cases.

This finally led to using the same instances as for MINLA and finding some other papers to develop other random graph generators.

4.2.5 Greedy algorithms development

The initial idea was to develop some greedy algorithms for the problem. The results obtained have been used as initial solutions for the execution of the ILP solver, and that is the reason for such dependency in the Gantt chart.

After discussing with the director, we thought that deterministic greedy algorithms sometimes have a bias. Having this in mind, we added, after developing the simple versions, a bit of randomness on some decisions of the algorithm. The execution for the analysis (task 24) came next, but without the randomness because preliminary tests did not show any interesting results using it.

4.2.6 Local search algorithms

When studying optimisation problems, one usual way of dealing with them is to use a local search approach; because greedy algorithms sometimes fail to give good enough results. The first thing we needed to do was developing some operators to define the neighbourhood of the search space of the problem, formalising them in the report, and finally implementing them in C++/LEMON and checking their results.

During the execution of the other algorithms (linear programming and greedy with randomisation) in the computer cluster, we can started developing the operators for local search.

This phase might took longer than the previous one in terms of development time, for the operators were not trivial to choose. Also, the execution time is usually longer, and for that reason we have planned a longer time in the cluster for these algorithms.

4.2.7 Follow-up milestone

Originally, the follow-up milestone was thought to happen in mid November. Since doing the original plan, the Barcelona School of Informatics (FIB) has published the correct dates for the January presentations and we decided with the director to **change** the date to 18th December, around one month later. This decision was taken in order to ease the execution of the project, because the original date we had decided was too conservative.

Having this in mind, all previous work to the original date has been done, local search executions have started, and the rest of the work has been rescheduled in order to use three more weeks which were available in December.

4.2.8 Result comparison and writing of conclusions

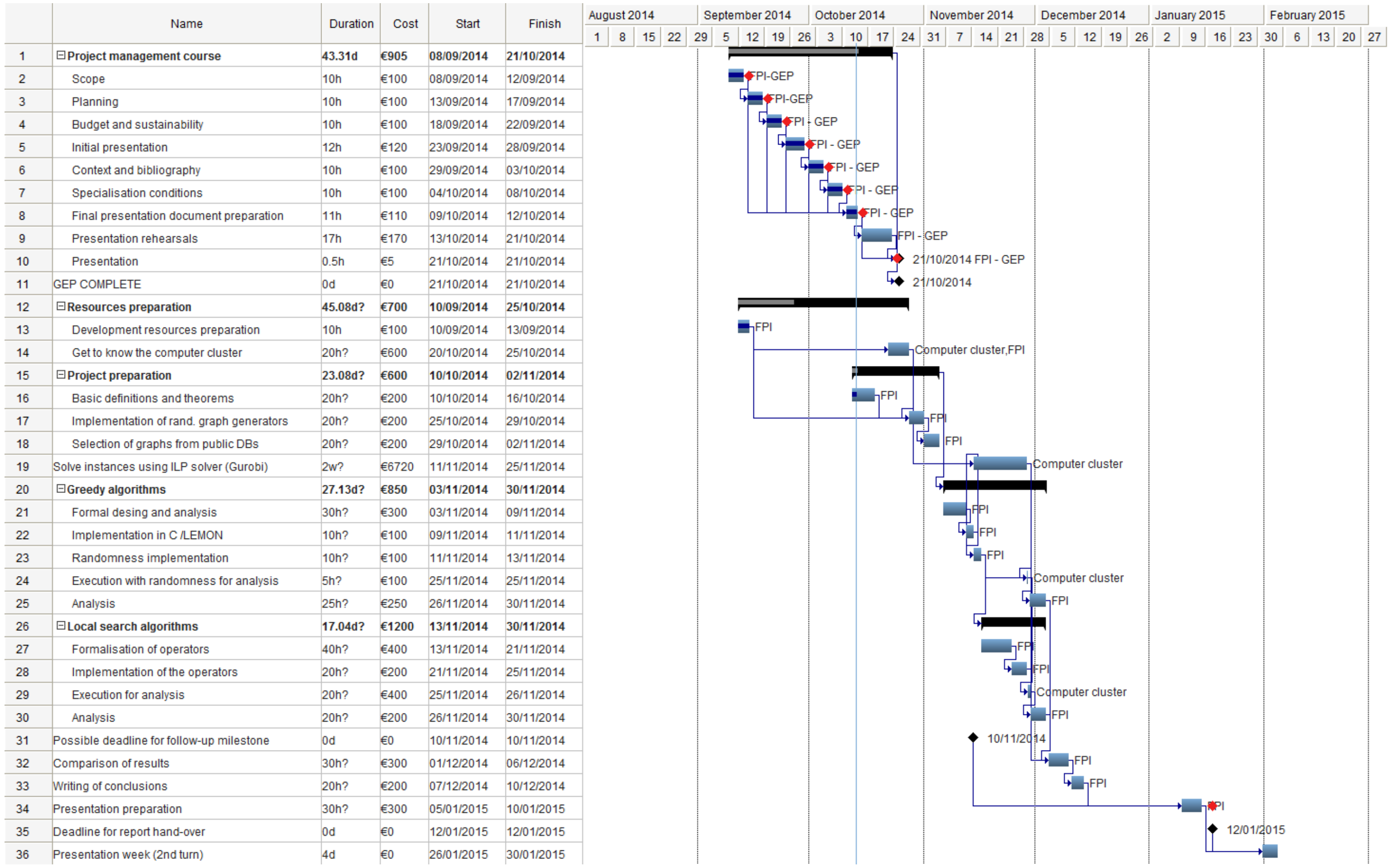
Once all executions had finished, and their results had been analysed in the previous phases, the results of every algorithm were compared and included in the benchmarking platform.

Once the results had been compared, we got to the point of writing the conclusions and finishing this report.

4.3 Alternative plans and changes in the initial plan

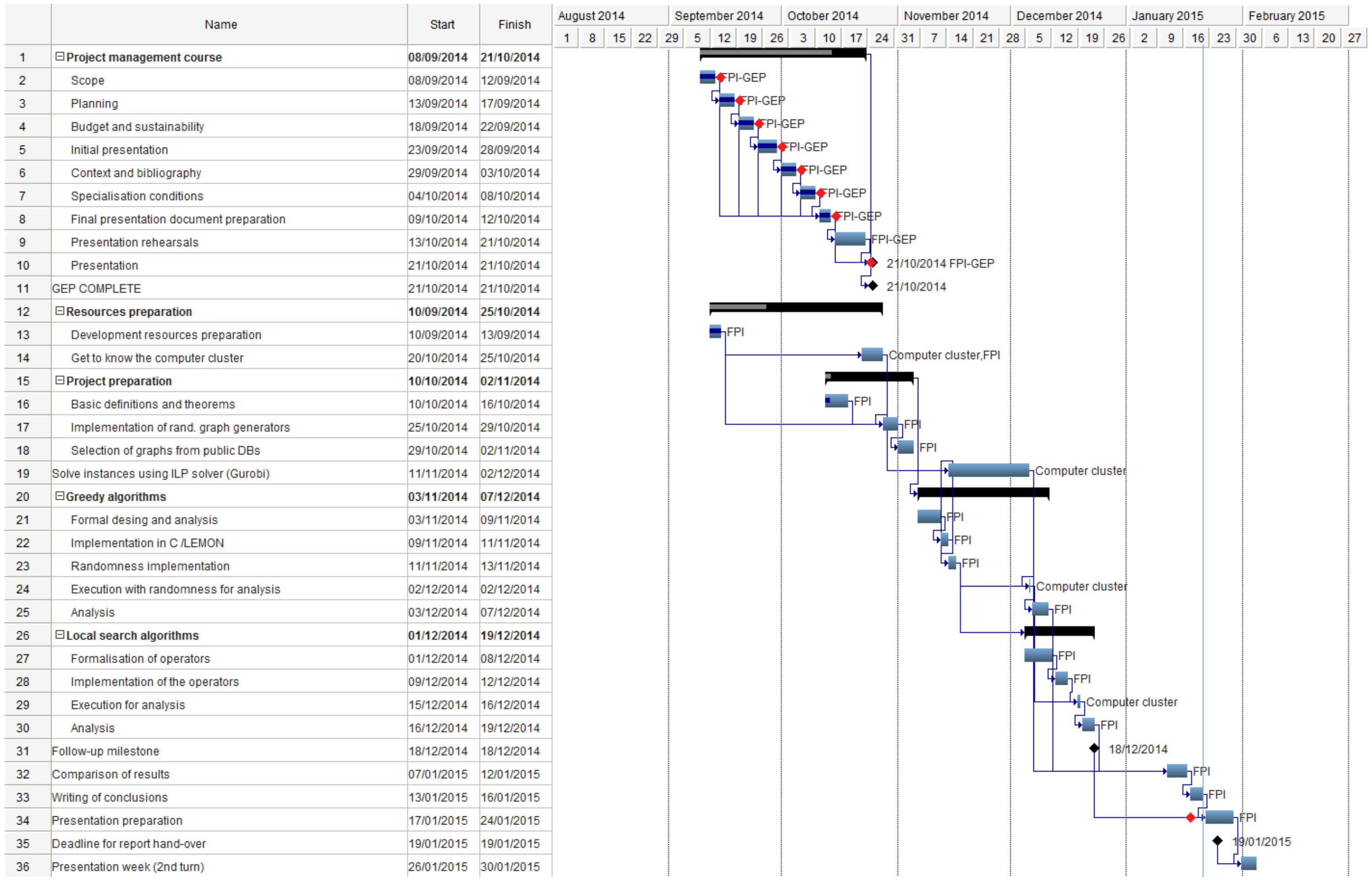
The planning was done trying to do the work in an agile and relaxed way. We initially chose to have a couple of empty weeks in December and January to overcome any possible problems. Finally, these weeks have been moved to be during the Christmas holidays, and work has been finished during January.

The initial plan is on Fig. 4.1, and the final schedule can be seen on Fig. 4.2.



	Name	Duration	Cost	Start	Finish
1	⊟ **Project management course**	**43.31d**	**€905**	**08/09/2014**	**21/10/2014**
2	Scope	10h	€100	08/09/2014	12/09/2014
3	Planning	10h	€100	13/09/2014	17/09/2014
4	Budget and sustainability	10h	€100	18/09/2014	22/09/2014
5	Initial presentation	12h	€120	23/09/2014	28/09/2014
6	Context and bibliography	10h	€100	29/09/2014	03/10/2014
7	Specialisation conditions	10h	€100	04/10/2014	08/10/2014
8	Final presentation document preparation	11h	€110	09/10/2014	12/10/2014
9	Presentation rehearsals	17h	€170	13/10/2014	21/10/2014
10	Presentation	0.5h	€5	21/10/2014	21/10/2014
11	GEP COMPLETE	0d	€0	21/10/2014	21/10/2014
12	⊟ **Resources preparation**	**45.08d?**	**€700**	**10/09/2014**	**25/10/2014**
13	Development resources preparation	10h	€100	10/09/2014	13/09/2014
14	Get to know the computer cluster	20h?	€600	20/10/2014	25/10/2014
15	⊟ **Project preparation**	**23.08d?**	**€600**	**10/10/2014**	**02/11/2014**
16	Basic definitions and theorems	20h?	€200	10/10/2014	16/10/2014
17	Implementation of rand. graph generators	20h?	€200	25/10/2014	29/10/2014
18	Selection of graphs from public DBs	20h?	€200	29/10/2014	02/11/2014
19	Solve instances using ILP solver (Gurobi)	2w?	€6720	11/11/2014	25/11/2014
20	⊟ **Greedy algorithms**	**27.13d?**	**€850**	**03/11/2014**	**30/11/2014**
21	Formal desing and analysis	30h?	€300	03/11/2014	09/11/2014
22	Implementation in C /LEMON	10h?	€100	09/11/2014	11/11/2014
23	Randomness implementation	10h?	€100	11/11/2014	13/11/2014
24	Execution with randomness for analysis	5h?	€100	25/11/2014	25/11/2014
25	Analysis	25h?	€250	26/11/2014	30/11/2014
26	⊟ **Local search algorithms**	**17.04d?**	**€1200**	**13/11/2014**	**30/11/2014**
27	Formalisation of operators	40h?	€400	13/11/2014	21/11/2014
28	Implementation of the operators	20h?	€200	21/11/2014	25/11/2014
29	Execution for analysis	20h?	€400	25/11/2014	26/11/2014
30	Analysis	20h?	€200	26/11/2014	30/11/2014
31	Possible deadline for follow-up milestone	0d	€0	10/11/2014	10/11/2014
32	Comparison of results	30h?	€300	01/12/2014	06/12/2014
33	Writing of conclusions	20h?	€200	07/12/2014	10/12/2014
34	Presentation preparation	30h?	€300	05/01/2015	10/01/2015
35	Deadline for report hand-over	0d	€0	12/01/2015	12/01/2015
36	Presentation week (2nd turn)	4d	€0	26/01/2015	30/01/2015

Figure 4.1: Initial gantt chart for the project



	Name	Start	Finish
1	**⊟Project management course**	**08/09/2014**	**21/10/2014**
2	Scope	08/09/2014	12/09/2014
3	Planning	13/09/2014	17/09/2014
4	Budget and sustainability	18/09/2014	22/09/2014
5	Initial presentation	23/09/2014	28/09/2014
6	Context and bibliography	29/09/2014	03/10/2014
7	Specialisation conditions	04/10/2014	08/10/2014
8	Final presentation document preparation	09/10/2014	12/10/2014
9	Presentation rehearsals	13/10/2014	21/10/2014
10	Presentation	21/10/2014	21/10/2014
11	GEP COMPLETE	21/10/2014	21/10/2014
12	**⊟Resources preparation**	**10/09/2014**	**25/10/2014**
13	Development resources preparation	10/09/2014	13/09/2014
14	Get to know the computer cluster	20/10/2014	25/10/2014
15	**⊟Project preparation**	**10/10/2014**	**02/11/2014**
16	Basic definitions and theorems	10/10/2014	16/10/2014
17	Implementation of rand. graph generators	25/10/2014	29/10/2014
18	Selection of graphs from public DBs	29/10/2014	02/11/2014
19	Solve instances using ILP solver (Gurobi)	11/11/2014	02/12/2014
20	**⊟Greedy algorithms**	**03/11/2014**	**07/12/2014**
21	Formal desing and analysis	03/11/2014	09/11/2014
22	Implementation in C /LEMON	09/11/2014	11/11/2014
23	Randomness implementation	11/11/2014	13/11/2014
24	Execution with randomness for analysis	02/12/2014	02/12/2014
25	Analysis	03/12/2014	07/12/2014
26	**⊟Local search algorithms**	**01/12/2014**	**19/12/2014**
27	Formalisation of operators	01/12/2014	08/12/2014
28	Implementation of the operators	09/12/2014	12/12/2014
29	Execution for analysis	15/12/2014	16/12/2014
30	Analysis	16/12/2014	19/12/2014
31	Follow-up milestone	18/12/2014	18/12/2014
32	Comparison of results	07/01/2015	12/01/2015
33	Writing of conclusions	13/01/2015	16/01/2015
34	Presentation preparation	17/01/2015	24/01/2015
35	Deadline for report hand-over	19/01/2015	19/01/2015
36	Presentation week (2nd turn)	26/01/2015	30/01/2015

Figure 4.2: Final gantt chart for the project

5 Budget, sustainability and regulations

5.1 Budget

5.1.1 Preliminary considerations

This kind of research project is usually developed within universities, and in many cases by research fellows. Quite often these projects are part of a bigger project which a research group is working on.

The Gantt chart in Fig. 4.1, used for the planning, contains the hours of work of the researcher and the computer cluster. When adding up the costs there will be some differences because in this budget analysis we have rounded some numbers to get simpler costs. Nevertheless, it is an estimation and such round errors are not considerable.

What follows is the budget as we had done it during the project management course, and as such talks about many things in the future. Final updates are considered in Section 5.1.6 and with some more depth in Section 16.1.

5.1.2 Human resources budget

As stated before, this kind of project is reasonably developed by a research fellow from a university. Reading the announcements of the FPI fellowships,[1] the gross salary for such a research fellow must be at least €16 422 per year. To make numbers a bit nicer, we will round it to €10/h. This is reasonable considering the total amount given by the government to the universities is of €20 600 per fellow.[2]

Considering the total amount of hours of the FPI when doing the project management course (90.5 hours), and the rest of the project (305 hours), it adds up to around 400 hours of work.

A part that is missing in the Gantt chart, as stated in Section 4.2.1 is a weekly two-hour meeting with the director in order to adjust the planning and make any needed corrections in the project, and that adds around 30 hours of work of the director, with an estimated salary of €50/h.

The results are available in Table 5.1.

5.1.3 Hardware budget

Whereas the execution of the algorithms will take place in a computer cluster, their development and the writing of the reports should be done on a workstation, be it a desktop computer or a

[1] *Formación del personal investigador*, granted by the Spanish government

[2] Data available in BOE 218§III, pages 69989–70010

Role	Est. salary	Est. hours	Est. cost
FPI/researcher	€10.00/h	400 h	€4000.00
Director	€50.00/h	30 h	€1500.00
Total estimated			€5500.00

Table 5.1: Human resources budget

laptop. In order to be able to work in various places, a laptop will be used. It is important to notice that laptops are usually more energy-efficient than desktop computers, and that can be another reason to choose one of the former for the development.

Excessive power is not required, but for compiling is quite interesting to have a not too low amount of RAM memory, so we can choose a laptop in the price tag of €800 (Intel Core i5 processor, 8 GB RAM).

Considering a lifespan of five years for the hardware, for a project of 4 months, the cost and depreciation estimates are in Table 5.2.

Product	Price	Units	Lifespan	Est. depreciation
Laptop	€800.00	1	5 years	€55.00
Total estimated	€800.00			€55.00

Table 5.2: Hardware budget

The cost of the computer cluster is not included because the hardware is owned by ALBCOM and other research groups, and RDlab (which is part of the CS department) offers them the maintenance. However, the cost of the hardware and maintenance is not included and is not possible to know until you have used the resources themselves. For that reason, a calculation using another provider which includes both hardware and software is available in the software budget within the software budget.

5.1.4 Software budget

Apart from the linear programming optimiser Gurobi, we will use Open Source software which is available for free. The chosen operating system is Xubuntu 14.04 [13], a Long-term support release (5 years) from the Ubuntu family of GNU/Linux operating systems, highly compatible with modern hardware and software. For compiling we will use the C++ compilers from the GNU Compiler Collection [14], and the GNU Make tools and CMake [15] to ease the building of the binaries. To edit the source codes we will use GNU Emacs [16] and Geany [17]. Finally, the documentation and final report will be built using the LuaTEX engine available in TEX live 2014 [18]. All this software is available within the Xubuntu distribution.

The code for the approximation algorithms has been built using the LEMON C++ Library for graphs, part of the COIN-OR project, as stated before. This library is available under the permissive BOOST Open Source licence, compatible with the GNU GPL, and allows its use with commercial software.

There are no costs associated to this area because the resources used are offered for free.

Software not considered during the initial planning

For the parsing and plotting of the results, we have chosen Python (CPython) 3.4 [19] and libraries NumPy 1.9 [20] and matplotlib 1.4 [21]. They are available under permissive BSD-like software licences.

Another piece of software we need to mention is Inkscape [22], an open-source vector graphics editor offered under the GNU General Public License version 2. We are using it in order to make some drawings.

Finally, the source code documentation has been generated using Doxygen [23].

Apart from their licences, they are offered free of charge, so they do not incur new fees.

Software services

Git is a great Open Source tool to maintain different versions of the code and roll-back when needed, used by small and large projects such as the Linux kernel. In order to have a private copy of the code in the cloud, the free Git service offered by BitBucket will be used.

For the Gurobi optimiser, the costs are quite more difficult to compute. It is available for free for academic research purposes, and is also available in ALBCOM's computer cluster. The estimated cost for 1 hour of computing and 2 GB of RAM in the cluster is €0.40 + VAT, but that does not include the maintenance costs of the service, as stated in the previous section. In order to do this budget we will use the by-hour licences offered by Gurobi, with a cost of $25.00 per hour (around €20.00), which also include the the hardware maintenance and software support, and 8 CPU cores and 8 GB of RAM.[3] This cost will be higher than the final real cost, for which RDlab can create the invoice to compare to once the project is finished. For this we will consider 20 hours of testing in order to know how the cluster works; two whole weeks of computation time, because we will have many instances of the problem running and ILP problems take long time to solve, and 25 hours more which include the execution of greedy and local search algorithms.

The total costs can be seen in Table 5.3.

Product	Price	Est. usage time	Est. cost
BitBucket service	0	4 months	0
Gurobi by-hour licence	€20.00/h	2 weeks and 45 hours	€7620.00
Total estimated			€7620.00

Table 5.3: Software services budget

[3]Original document at `http://www.gurobi.com/pdf/Commercial_Pricing_Gurobi_27March2013.pdf` not available anymore. Updated prices at `http://www.gurobi.com/products/licensing-and-pricing/price-list`§Cloud licenses.

5.1.5 Total budget

Adding all previous costs, the estimated budget for this project is available in Table 5.4.

Concept	Est. cost
Human resources	€6500.00
Hardware	€55.00
Software licences and services	€7620.00
Total estimated cost	€14 175.00

Table 5.4: Total budget

As stated before, the cost will probably be much lower because of the cheaper price of ALBCOM's computer cluster service.

5.1.6 Deviation control

Like we have stated in Sections 4.2.1 and 5.1.2, we planned a weekly meeting with the director. This allowed us to do an agile development, and helped us do the changes in such a way that the costs can be minimally modified.

However, there have been some changes regarding the cost of the cluster service for two reasons. On the one hand we have used the computer cluster operated by RDlab, and on the other hand the way of calculating the amount of resources used is different. A complete analysis of this topic is in Section 16.1.

5.2 Sustainability

From an economic point of view, this project tries to go through the cheap way. The budget is clearly too high for a University project done in four months, but the final cost will probably be much less than calculated because the use of the cluster will be cheaper almost for sure. It also tries to use existing technologies for the implementation of the algorithms, reducing costs and giving more time to develop the project itself. For the unrealistic numbers in the budget, but the considerations for making such calculations, we are giving our project a mark of 5 in regarding economic sustainability.

A project like this one does not have many social implications. But the use of open source software whenever possible is good from an open-knowledge point of view. And giving public access to the results when the project is finished is a way to contribute with the technology. Also, once the project is finished, the implementation of the interface between LEMON and Gurobi, will be given back to the open source community. The implementation of this feature into LEMON has been asked to the community for around 4 years.[4]. Moreover, the ALBCOM group is interested in the results of this project, so it will benefit them directly. However, there

[4]See `http://lemon.cs.elte.hu/trac/lemon/ticket/367`

are not any social implications for the rest of the population in general. For these reasons, we are giving the project a mark of 7 in this area.

Considering an environmental point of view, being an optimisation problem, it can be positive to the environment in an indirect way. If the results are positive, the algorithms developed and the results obtained could be used to minimise some costs in real-life problems related to the abstract family of graph layout problems.

A negative input to the environmental impact analysis is the use of ILP models to solve the problem. It is one of the few ways to get the global optimums for the problem, but trying to optimally solve an NP-complete problem is usually a waste of computer resources for large instances.

Computer clusters usually consume much energy, since they are very well refrigerated. Nowadays they are more efficient than they were before, and their use allows the developers to keep working on the project using the workstation.

The project itself does not create a manufacturable product, nor makes a direct use of natural resources for its results. It does use computers, which are built using minerals from areas under civil wars (and these minerals are many times the causes for these wars). But this is a problem which many projects in computer science related areas have problems, and is difficult to solve without global actions.

For all these reasons, we are giving a mark of 4 to the environmental implications.

Sustainable?	**Economic**	**Social**	**Environmental**	**Total points**
Planification	5	7	4	16

Table 5.5: Project planning sustainability table

5.3 Laws and regulations

Just like any other project, ours has to comply with some legal and ethical requirements. For the most part, 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.

As for the software and algorithms we have developed, we make use of open source libraries which allow modifications and custom redistribution as long as original copyright notices are well retained. We have not changed these libraries but worked on top of them using them as a tool for our project.

For the non open source software such as Gurobi, we have used an academic licence because all our results are solely for research purposes. The implementation of the interface between LEMON and Gurobi is planned to be given back to the community developing LEMON to be included within the project. This does not fail to comply the Gurobi licence because creating interfaces for use with other software is allowed; in order to be able to use these interfaces the users still need to install Gurobi and have a valid licence.

Other regulations for which our project has had to take care are those governing the cluster from the Computer Science Department. We can only use it for the development and execution of the algorithms. Use for other purposes is not allowed, thus using it for personal interests is not allowed. Some other fact is the shared nature of the cluster, and although it has its own ways to control abuse, common sense and fair use must be taken into account when using it.

Regarding laws and regulations for the developed software itself, we are not dealing with any personal or critical data, nor doing anything that can be on the boundaries of ethical research.

As for the licenses and intellectual property of the software developed during the project, we have to comply with the University's rules [24]. The interface between LEMON and Gurobi has been implemented at the sole discretion of the author, with no participation from the University other than evaluating its use in the project. Following from this, we have decided to offer the code under the same open source license as LEMON.

The rest of the code implementing the algorithms falls under Title 2, Article 3, Section 3 (*on the student works directed or coordinated by the faculty of UPC*) of the same document. According to the contents of the section, UPC is the owner of the exploitation rights. It is not the exclusive owner, though, and since the authors are by default owners of the exploitation rights according to the Spanish law, we have decided to release it under the same software license.

Part III

Definitions, theorems and properties

6 Basic definitions

6.1 Preliminaries

Let us begin with some basic definitions and properties. Most of them are the standard in graph theory, but they are here in order to help to understand the whole document. The notation and conventions are taken from [25] mainly.

Definition 6.1 (Graph)**.** A **simple undirected graph** G is an ordered pair of disjoint sets (V, E), where V is the set of **vertices** and $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ is the set of **edges**.

In this sense, $V(G)$ is the **vertex set** of the graph G, and $E(G)$ is its **edge set**.

Given an edge $e = \{u, v\}$, we say it **joins** vertices u and v, which are its **endvertices**, and can also be denoted by uv or vu (we are considering undirected graphs). If $e \in E(G)$, we say u and v are **adjacent** or **neighbours** in G, denoted $u \sim v$ (or $u \sim_G v$), and **incident** with edge e. And since we are considering simple graphs, there are no loops (edges joining $x \in V$ with itself) nor multiple edges (more than one edge joining the same pair of vertices).

The **order** of a graph G is $|V(G)|$ and its **size** is $|E(G)|$.

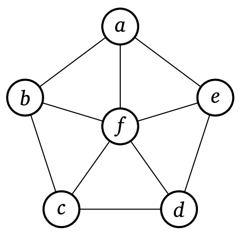
We say H is a **subgraph** of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, and it is denoted as $H \subseteq G$. When $E(H) = \{uv \in E(G) \mid u, v \in V(G)\}$, that is when H contains all edges of G joining two edges in $V' = V(H)$, the graph H is known as an **induced subgraph** (induced by V') and is denoted by $G[V']$.

Notation. Unless stated otherwise, we will use the letters $u, v, w, x, y, z, s, t, \ldots \in V = V(G)$ for the vertices in general and $e, f, g, uv = \{u, v\}, \ldots \in E = E(G)$ for the edges of a graph G.
Comment. See that graphs can be visually represented as points (the vertices) joined by lines (the edges), such as in Fig. 6.1.

Definition 6.2 (Paths and cycles)**.** A graph P is said to be a **path of length** n when $V(P) = \left\{v_0, \ldots, v_n\right\}$ and $E(P) = \left\{v_0v_1, v_1v_2, \ldots, v_{n-1}v_n\right\}$. It is denoted by P_n.

A graph C is a **cycle of length** $n \geq 3$, denoted by C_n, when $V(C) = \left\{v_1, \ldots, v_n\right\}$ and $E(C) = \left\{v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1\right\}$. When n is even it is known as an **even cycle**, and otherwise it is known as an **odd cycle**.

Definition 6.3 (Neighbourhoods and degrees)**.** Given a graph G and a vertex $v \in V(G)$, its



$V(G) = \{a, b, c, d, e, f\}$
$E(G) = \{ab, ae, af, bc, bf, cd, cf, de, df, ef\}$

Figure 6.1: Visual representation of a graph

neighbourhood is the set of vertices adjacent to v in G:

$$\Gamma_G(v) = \{u \in G \mid uv \in E(G)\}$$

The **degree of a vertex** $v \in V(G)$ is the number of edges v is incident with. Since we are considering simple graphs, it is the same as its number of neighbours:

$$\deg_G(v) = \left|\Gamma_G(v)\right|$$

The **maximal degree** and **minimal degree** of (the vertices of) G are, respectively,

$$\Delta(G) = \max_{v \in V(G)} \deg_G(V)$$

$$\delta(G) = \min_{v \in V(G)} \deg_G(V)$$

When $\delta(G) = \Delta(G) = k$ we say G is **regular of degree** k or k**-regular**.

Definition 6.4 (Complete graph)**.** The complete graph with n vertices is $K_n = (V, E)$, with $E = \{\{u, v\} \mid u, v \in V, u \neq v\}$. That is, $|V| = n$ and for any $u, v \in V,\ u \neq v$ we have $u \sim v$.

Comment. See what *complete* means: the graph contains all possible edges, which is the same as all possible pairs of vertices. For that reason, the number of edges in a complete graph is $\binom{n}{2} = \dfrac{n(n-1)}{2}$.

Definition 6.5 (Graph homomorphism)**.** Given two graphs G and H, an application $\varphi\colon G \longrightarrow H$, defined by $\varphi\colon V(G) \longrightarrow V(H)$, is a **graph homomorphism** if, for any $uv \in E(G)$, then $\varphi(u)\varphi(v) \in E(H)$. In this case we say G is H-colourable, or φ is an H-colouring of G.

We can extend this definition to map graphs to sets of integers $[k] = \{1, \dots, k\}$, seeing H as K_k, the complete graph with k vertices. This way, an application $\varphi\colon V(G) \longrightarrow [k]$ is an homomorphism, also called a **vertex colouring** of G, if, for any $uv \in E(G)$, then $\varphi(u) \neq \varphi(v)$. If such homomorphism exists, we say the graph G is k-colourable and φ is a k-colouring of G.

Definition 6.6 (Colouring family)**.** Given a graph G, the colouring family $\mathcal{F}_k$ of G is defined as the set of colourings $V(G) \longrightarrow [k]$. That is

$$\mathcal{F}_k(G) = \{\varphi\colon V(G) \longrightarrow [k] \;\big|\; uv \in E(G) \implies \varphi(u) \neq \varphi(v)\}$$

We can then define the family of all colourings,

$$\mathcal{F}(G) = \bigcup_{k\in\mathbb{N}} \mathcal{F}_k(G) = \{\varphi\colon V \longrightarrow \mathbb{N} \,\big|\, uv \in E(G) \implies \varphi(u) \neq \varphi(v)\}$$

Comment. Note that for $k' \leq k$ we have $\mathcal{F}_{k'}(G) \subseteq \mathcal{F}_k(G)$, because the homomorphisms do not need to be onto mappings.

Definition 6.7 (Chromatic number)**.** Given a graph $G = (V, E)$, its **chromatic number** $\chi(G)$ is the least k for which G is k-colourable. Equivalently,

$$\chi(G) = \min\left\{k \,\middle|\, \mathcal{F}_k(G) \neq \varnothing\right\}$$

6.2 The minimum linear colouring arrangement problem

Definition 6.8 (Linear distance of a colouring)**.** Given a graph G and a colouring φ of G, we define the **linear distance** of φ as

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

6.2.1 Parameterised formulation

Before defining our problem, it is better to start with a more classic problem such as the **graph colouring** problem, which is highly related. In fact, as the definitions show, any valid solution for our problem is also a proper colouring.

Problem 6.9 (GRAPH-COLOURING$_k$)**.** Given a graph G and an integer k, find a k-colouring $\varphi \in \mathcal{F}_k(G)$.

Problem 6.10 (GRAPH-COLOURING-DEC$_k$)**.** Given a graph G and an integer k, decide if there is a colouring $\varphi \in \mathcal{F}_k(G)$.

Definition 6.11. Given a graph G and an integer k for which $\mathcal{F}_k(G) \neq \varnothing$ we define

$$LCA_k(G) = \min_{\varphi\in\mathcal{F}_k(G)} L[\varphi](G)$$

Problem 6.12 (MINLCA$_k$)**.** Given a graph G and an integer k, find a colouring $\varphi \in \mathcal{F}_k(G)$ such that $L[\varphi](G) = LCA_k(G)$.

Problem 6.13 (MINLCA-DEC$_k$)**.** Given a graph G, and two integers k and L, decide if there is a colouring $\varphi \in \mathcal{F}_k(G)$ such that $L[\varphi](G) \leq L$.

6.2.2 Non-parameterised formulation

A more general formulation of the minimum linear colouring arrangement problem consists in leaving out the parameter k for the maximum number of allowed colours.

Definition 6.14. Given a graph G, we define

$$MinLCA(G) = \min_{\varphi \in \mathcal{F}(G)} L[\varphi](G) = \min_{k \in \mathbb{N}} LCA_k(G)$$

Problem 6.15 (MINLCA)**.** Given a graph G find a colouring $\varphi \in \mathcal{F}(G)$ such that $L[\varphi](G) = MinLCA(G)$.

Problem 6.16 (MINLCA-DEC)**.** Given a graph G, and an integer L, decide if there is a colouring $\varphi \in \mathcal{F}(G)$ such that $L[\varphi](G) \leq L$.

7 Complexity of the problem

7.1 Classic computational complexity

In this section we study the computational complexity of this problem in the classic way. After proving the NP-hardness of the problem (and completeness of the decisional formulation), we move on to see its relation to other computationally difficult problems such as **integer linear programming**.

Lemma 7.1 (Upper bound on the linear colouring arrangement)**.** *Let $G = (V, E)$ be a graph and k an integer. Then,*

$$L[\varphi](G) \leq |E|\,(k-1) \quad \forall \varphi \in \mathcal{F}_k(G)$$

Proof. We have $\max_{uv \in E} \left|\varphi(u) - \varphi(v)\right| \leq k - 1$ since $\max_{v \in V} \varphi(v) \leq k$ and $\min_{v \in V} \varphi(v) = 1$. That way,

$$L[\varphi](G) = \sum_{uv \in E} \left|\varphi(u) - \varphi(v)\right| \leq \sum_{uv \in E} (k-1) = |E|\,(k-1)$$

□

Theorem 7.2. *The decisional version of* MINLCA$_k$ *is* NP*-complete.*

Proof. First of all, we need to show that MINLCA-DEC$_k$ is in NP. Algorithm 7.1 verifies MINLCA-DEC$_k$ in polynomial time.

The running time of this algorithm is clearly polynomial. First of all, in 7.1.3 we are doing a loop linear in $n = |V|$. The other loop does at most $m = |E|$ steps. So the cost of this algorithm is $\mathcal{O}(n + m)$. If we consider the costs of the comparisons and sums are not constant, then we are simply missing a $\log n$ factor in the cost function.

We now need to prove the problem is NP-hard. In order to do so, we reduce from GRAPH-COLOURING-DEC$_k$. Suppose $\mathcal{A}(G, k, L)$ decides MINLCA-DEC$_k$ and let $L = |E|(k-1)$.

If G is k-colourable, equivalently $\mathcal{F}_k(G) \neq \varnothing$, by Lemma 7.1 we have that $\langle G, k, L \rangle$ is a yes-instance for $\mathcal{A}$.

If $\langle G, k, L\rangle$ is a yes-instance for $\mathcal{A}$, we know there is $\varphi \in \mathcal{F}_k(G)$ (whatever the value of L). We can then conclude that MINLCA-DEC$_k$ is NP-complete. □

Algorithm 7.1: Verifier for MINLCA-DEC$_k$

Precondition Let $G = (V, E)$ be a graph, $V = \{1, \dots, n\}$, $|E| = m$, represented as an adjacency list.
Let k and L be two non-negative integers.
Let φ be a possible solution for the problem (witness).

Postcondition Verifies whether φ is in $\mathcal{F}_k(G)$ and $L[\varphi](G) \leq L$.

```
1:  function MinLCA-verif($G$, $k$, $L$, $\varphi$)
2:      $LCA \leftarrow 0$
3:      for all $v \in V$ do
4:          if $\varphi(u) > k$ then                    ▷ If true, then $\varphi \notin \mathcal{F}_k(G)$
5:              Reject
6:          end if
7:      end for
8:      for all $uv \in E$ do
9:          if $\varphi(u) = \varphi(v)$ then           ▷ If true, then $\varphi$ is not a proper colouring
10:             Reject
11:         end if
12:         $LCA \leftarrow LCA + |\varphi(u) - \varphi(v)|$
13:     end for
14:     if $LCA \leq L$ then
15:         Accept
16:     else
17:         Reject
18:     end if
19: end function
```

Proposition 7.3 (MINLCA-ILP$_k$). *The following integer linear programme is a correct model for* MINLCA$_k$*:*

$$\min \sum_{uv \in E} L_{uv} \tag{7.1}$$

$$\text{s.t.} \quad x_{u,c} + x_{v,c} \leq 1 \qquad uv \in E,\ c = 1, \dots, k \tag{7.2}$$

$$\sum_{c=1}^{k} x_{v,c} = 1 \qquad v \in V \tag{7.3}$$

$$\varphi_v = \sum_{c=1}^{k} c\, x_{v,c} \qquad v \in V \tag{7.4}$$

$$\varphi_u - \varphi_v \leq L_{uv} \qquad uv \in E \tag{7.5}$$

$$\varphi_v - \varphi_u \leq L_{uv} \qquad uv \in E \tag{7.6}$$

$$x_{v,c} \in \{0, 1\} \qquad v \in V,\ c = 1, \dots, k \tag{7.7}$$

$$L_{uv} \in \{1, \dots, k-1\} \qquad uv \in E \tag{7.8}$$

See that the φ_v variables are unnecessary, but the model is easier to understand with them: they correspond to the colour of vertex v in the solution. That is, $\varphi(v) = \varphi_v$ once the programme is solved. We could change every other appearance of these variables with the definition given in 7.4.

The rest of the variables and expressions mean the following:

- *$x_{v,c}$ are binary variables (with values 0 or 1, the same as* `False` *or* `True`*). They have a value of 1 when vertex v has been assigned colour c; i.e., $\varphi(v) = c$.*
- *In order to make sure they have only one colour assigned, we have the corresponding equalities in 7.3. And to ensure there are no two adjacent vertices with the same colour, we use the inequalities in 7.2*
- *L_{uv} are the differences between the adjacent nodes u, v. Equivalently, they are the cost of the edge uv in the solution.*
- *Linear programming does not allow the use of absolute values, because they are not linear functions. For that reason, we use φ_u and φ_v to express the lower bounds for the variables L_{uv}. Their value is never higher than the difference between φ_u and φ_v because they are used in the objective function 7.1, which is a minimisation. And they are never negative because of the domain given in 7.8.*

Proof. The solution space defined by the constraints is the one we want: this is justified within the proposition itself.

The objective defined in this programme is exactly the objective for MINLCA$_k$. Notice that $L_{uv} \geq |\varphi_u - \varphi_v|$, and because the sum of L_{uv} is minimised, the value it attains is exactly $|\varphi_u - \varphi_v|$. Finally, since the solution space contains all possible k-colourings of G, the value for the objective will be the minimum possible.

See that this is a reduction from MINLCA$_k$ to optimisation ILP (integer lineal programming), which is also NP-hard. The number of variables is polynomial in $n = |V|$, since there are $|V|\,k = \mathcal{O}\left(n^2\right)$ variables $x_{v,c}$, there are $|E| = \mathcal{O}\left(n^2\right)$ variables L_{uv} and there are n variables φ_v. The number of constraints is also polynomial: there are $|E|\,k = \mathcal{O}\left(n^3\right)$ constraints of type 7.2, n of types 7.3 and 7.4 and $|E| = \mathcal{O}\left(n^2\right)$ of types 7.5 and 7.6. □

7.2 Bounds and closed results for particular graphs

Proposition 7.4 (Lower bound for MINLCA)**.** *Given a graph $G = (V, E)$, the number of edges is a lower bound for MinLCA(G). If $|E| = m$, that is*

$$m \leq \mathit{MinLCA}(G)$$

Proof. There are m edges, and each edge uv has a cost $\left|\varphi(u) - \varphi(v)\right| \geq 1$ since $\varphi(u) \neq \varphi(v)$. □

Theorem 7.5 (Rough upper bound for MINLCA)**.** *Given a graph $G = (V, E)$, the chromatic number of G and the number of edges m give us the following upper bound*

$$\mathit{MinLCA}(G) \leq (\chi(G) - 1)\ m$$

And equality holds only when $\chi(G) \leq 2$.

Proof. Let $k = \chi(G)$ and apply Lemma 7.1. And considering $m \geq 1$, there is always at least one edge with cost 1, therefore equality holds only when all edges have cost 1. □

7.2.1 Bipartite graphs

Definition 7.1 (Bipartite graph)**.** A graph $G = (V, E)$ is bipartite if V can be divided into two disjoint sets A, B, that is $V = A \cup B$ and $A \cap B = \varnothing$, such that all edges have one vertex in A and the other in B.

Theorem 7.6. *A graph G is bipartite if and only if it contains no odd cycles as subgraphs.*

Comment. See [25, p. 9] for a proof.

Lemma 7.7. *A graph G is bipartite if and only if $\chi(G) \leq 2$.*

Proof. We simply need to see the disjoint sets A, B from the definition as colours 1 and 2 for φ and vice-versa. □

Theorem 7.8 (Minimum linear colouring arrangement for bipartite graphs)**.** *A graph $G = (V, E)$, with $|E| = m$, is bipartite if and only if MinLCA$(G) = m$.*

Proof. $\Rightarrow$ We have $\chi(G) \leq 2$. Apply Proposition 7.4 and Theorem 7.5.

$\Leftarrow$ We will proceed by showing that G needs to be 2-colourable.

Suppose $\chi(G) = k \geq 3$. Let $\varphi \in \mathcal{F}$ be a colouring of G such that $L[\varphi](G) = \mathit{MinLCA}(G) =$

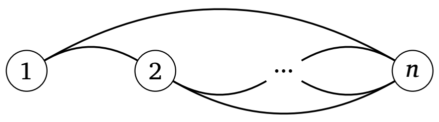


Figure 7.1: Linear colouring arrangement for a complete graph K_n

m. Let $v \in V$ such that $\varphi(v) = \max_{u \in V} \varphi(u) = r \geq k$, which exists because we need at least k colours.

Given that $L[\varphi](G) = m$, all vertices adjacent to v must have colour $r-1$. But then, v could be coloured with colour $r-2$ $\lightning$.

So $\chi(G) \leq 2$ and G is bipartite.

□

7.2.2 Complete graphs

Theorem 7.9. *If $G = (V, E)$ is the complete graph K_n, then MinLCA$(G) = \dfrac{n^3 - n}{6}$*

Proof. Since G is complete, all vertices are adjacent to each other and we need to use exactly n colours. Without loss of generality, suppose $V = [n]$ and that we choose the colouring $\varphi(v) = v$ for every $v \in V$.

Then, we have

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

If we consider the edges uv with $u < v$, since uv and vu are the same edge,

$$L[\varphi](G) = \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} (v - u)$$

changing v for $v - u$,

$$L[\varphi](G) = \sum_{u=1}^{n-1} \sum_{v=1}^{n-u} v$$

for which the inner sum can be seen as the cost of the edges incident to u which are on its right, as shown on Fig. 7.1. Now change u for $n - u$,

$$L[\varphi](G) = \sum_{u=1}^{n-1} \sum_{v=1}^{u} v$$

The inner sum is an arithmetic series from 1 to u,

$$L[\varphi](G) = \sum_{u=1}^{n-1} \frac{u\,(u+1)}{2} = \frac{1}{2}\left(\sum_{u=1}^{n-1} u^2 + \sum_{u=1}^{n-1} u\right) = \frac{1}{2}\left(\sum_{u=1}^{n-1} u^2 + \frac{n\,(n-1)}{2}\right)$$

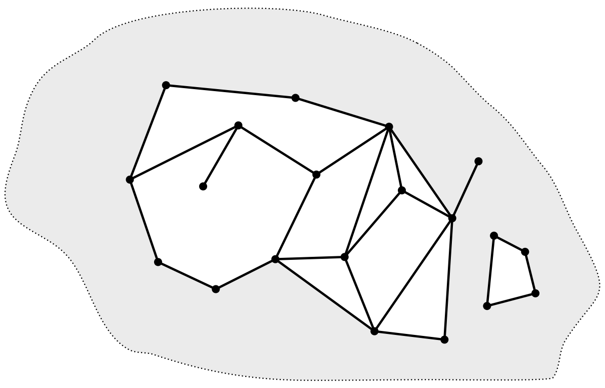


Figure 7.2: Example of a plane graph. The shaded area corresponds to the unbounded face.

and using the identity for the series of the first $n-1$ squares,

$$L[\varphi](G) = \frac{1}{2}\left(\frac{n\,(n-1)\,(2n-1)}{6} + \frac{n\,(n-1)}{2}\right) = \frac{n\,(n-1)\,(n+1)}{6} = \frac{n^3-n}{6}$$

□

Comment. The solution for a complete graph K_n is exactly the same for MINLCA and MINLA.

7.2.3 Planar graphs

Definition 7.2 (Planar graph). A graph G is planar if it can be drawn on the plane without any crossing edges. Such planar embedding is called a **plane graph**.

The regions of $\mathbb{R}^2$ obtained when removing the vertices and edges of a plane graph are its **faces**. See that there is always an **unbounded face**, which corresponds to the fundamental cycle when the graph is drawn on a sphere.

Theorem 7.10 (Four colour theorem). *If G is a planar graph, then $\chi(G) \leq 4$.*

Comment. This result is really difficult to prove. Indeed, all accepted proofs make use of computers and were not widely accepted at first. For more information on this topic, including the story of the theorem and a complete list of references, you can refer to [26].

Corollary 7.11. *If $G = (V, E)$ is planar, with $|E| = m$, then MinLCA$(G) < 3\,m$.*

Outerplanar graphs

Definition 7.3 (Outerplanar graph). A graph G is outerplanar if it is planar and there is a plane graph for which all the verices are on the boundary of the unbounded face.

Comment. An outerplanar graph can be seen as cycles with non-overlapping chords (edges going from one side to the other), possibly with a vertex in common or some paths joining them. See figure 7.3 for an example.

Theorem 7.12. *If G is an outerplanar graph, then $\chi(G) \leq 3$.*

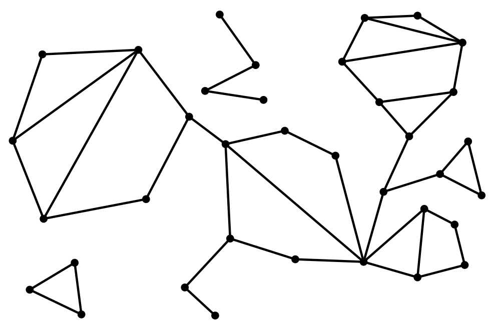


Figure 7.3: An outerplanar graph

Corollary 7.13. *If $G = (V, E)$ is outerplanar, with $|E| = m$, then MinLCA$(G) < 2\,m$.*

Part IV

Algorithms and instances

8 Exact algorithms

8.1 Integer linear programming

One way of evaluating the quality of the approximation algorithms is to compare their approximations to the exact solutions. In order to do this, we have decided to use the integer linear programme described in Proposition 7.3. The implementation contains variables to check which colours are used, with their respective constraints. They are not needed at all, in fact, the solver removes them before starting to solve the problem; but they come of use when extracting the results to do some benchmarks.

The way of working is as follows:

1. It starts by finding an optimal basic[1] solution for the relaxed[2] problem. This solution is a lower bound for the problem.
2. It then tries to find an initial integer solution which will serve as the first *incumbent* or upper bound. In order to ease this, the initial solution is given by one of the greedy algorithms described in Chapter 9.
3. Using different methods which usually involve the dual simplex algorithm, such as branch and bound and cutting planes, it tries to either increase the non-integer lower bound or decrease the incumbent.

8.1.1 Other models

For the linear arrangement problem there are other integer linear programming models such as the one described in [27], which uses *betweenness* variables. These variables use the fact that in MINLA every vertex is exactly between two others in the arrangement, except for the first and the last ones.

The results and properties are quite interesting, but we do not know if the model can be adapted to our problem because it is out of scope. Future work on our problem could include trying to adapt this formulation of MINLA to MINLCA.

[1]A vertex of the polytope described by the constraints
[2]Without the integrality constraints

9 Greedy algorithms

These algorithms are based on a breadth-first search. The contents are simply the actions to do when the algorithm visits a vertex after taking it from the pending queue.
Comment. During the analysis of the cost of the algorithms it is implicit that computation of the *linear distance* of the colouring for G has a cost of $\mathcal{O}(|V(G)| + |E(G)|)$ and, at most, a $\log|V(G)|$ factor. It is not considered because the whole algorithm has a greater cost than computing the linear distance of the colouring.

9.1 Nearest colour

The idea behind this greedy approach is quite simple. We want to get a colouring with minimal linear distance, so when the algorithm has to assign a colour to a vertex v, it checks the colour of its neighbours and increases the preference of the one colour before and the one colour after the colour of each neighbour. In other words, if $u \sim v$ and u is of colour c, the algorithm increases the preference for colours $c - 1$ and $c + 1$ and forbids colour c (otherwise, it would not be a proper colouring). It then assigns to vertex v the available colour with the highest preference.

Suppose posMax returns the position of the maximum element of an array. The algorithm for *nearest colour* is formalised in 9.1.

Proposition 9.1. *The greedy nearest colour algorithm (Algorithm 9.1) runs in polynomial time and either gives a proper colouring or returns an error.*

Proof. Let G be the graph we want to arrange, and let $V = V(G)$, $E = E(G)$, $n = |V|$, $m = |E|$.

The correctness of the algorithm is quite straightforward. When the input is $v \in V$, on line 6 it forbids the colour of all of its already coloured neighbours. And line 15 only sets $\varphi(v)$ when some colour is not forbidden, because otherwise posMax(*Colours*) is not well defined.

Regarding the cost, we can suppose $maxK \leq n$. The initialisation of *Colours* on line 2 takes $\Theta(maxK)$ steps. Notice that the loop on line 3 has a cost of $\mathcal{O}\left(\deg_G(v)\log maxK\right)$, considering *Colours* has random access in constant time and the comparisons and increments take $\mathcal{O}(\log maxK)$ time. The call to posMax(*Colours*) on line 15 can be done in linear (on *maxK*) time. So processing one vertex can be done in $\mathcal{O}\left(\deg_G(v)\log maxK + maxK\right)$ time.

If we consider the whole graph using a breadth-first search, all the steps can be done in $\mathcal{O}(m\log maxK + n\,maxK + m) = \mathcal{O}\left(n^2\log maxK\right)$ time, since $\sum_{v\in V}\deg_G(v) = 2m = \mathcal{O}\left(n^2\right)$. □

Algorithm 9.1: Greedy nearest colour algorithm

Precondition Suppose the following implicit parameters: the graph $G = (V, E)$, the maximum number of colours *maxK*, the partial k-colouring φ.
Let $v \in V$. Suppose $\varphi(v) = \texttt{Undefined}$.

Postcondition Either the subgraph induced by the already coloured vertices including v is properly coloured or the breadth-first search will stop giving an error.

```
 1: function MinLCA-greedyNearest-paintVertex($v$)
 2:     $Colours \leftarrow (0)_{i=1}^{maxK}$ ▷ Precedence for the colours. Larger is better. Initially all colours are allowed
 3:     for all $u \in \Gamma_G(v)$ do
 4:         $uC \leftarrow \varphi(u)$
 5:         if $uC \neq$ Undefined then
 6:             $Colours_{uC} \leftarrow$ Forbidden
 7:             if $uC - 1 \geq 1$ and $Colours_{uC-1} \neq$ Forbidden then
 8:                 $Colours_{uC-1} \leftarrow Colours_{uC-1} + 1$
 9:             end if
10:             if $uC + 1 \leq maxK$ and $Colours_{uC+1} \neq$ Forbidden then
11:                 $Colours_{uC+1} \leftarrow Colours_{uC+1} + 1$
12:             end if
13:         end if
14:     end for
15:     $\varphi(v) \leftarrow$ posMax($Colours$)        ▷ If all the colours are forbidden, returns an error
16: end function
```

9.2 Least cost

In this case, the colour assigned to a vertex by the algorithm is the one with the least increment of the cost at each step. A more formal description to this approach is in Algorithm 9.2.

Proposition 9.2. *The greedy least cost algorithm (Algorithm 9.1) runs in polynomial time and either gives a proper colouring or returns an error.*

Proof. Let G be the graph we want to arrange, and let $V = V(G)$, $E = E(G)$, $n = |V|$, $m = |E|$.

The correctness of the algorithm is also quite straightforward. When the input is $v \in V$, the algorithm starts by counting the number of neighbours of each colour on line 3. Initially, the best colour so far is not known, so it is set to an undefined state with potentially infinite cost on line 10. The loop starting on line 11 only considers the cost of using colours not in the neighbours (line 12), and it works by computing the linear distance of v coloured with c with all its neighbours. It is then clear that this algorithm gives a proper colouring.

As for the cost, supposing $maxK \leq n$, the initialisation of *ColourCount* takes $\Theta(maxK)$ steps. The counting of the vertices of each colour on line 3 takes $\Theta\left(\deg_G(v)\right)$ time. And since the sum on line 13 can be done in $\Theta(maxK \log maxK)$ time, the for loop on line 11 takes $\mathcal{O}\left(maxK^2 \log maxK\right)$ time. So processing one vertex can be done in $\mathcal{O}\left(\deg_G(v) + maxK^2 \log maxK\right)$

Algorithm 9.2: Greedy least cost algorithm

Precondition Suppose the following implicit parameters: the graph $G = (V, E)$, the maximum number of colours *maxK*, the partial k-colouring φ.
Let $v \in V$. Suppose $\varphi(v) = \texttt{Undefined}$.

Postcondition Either the subgraph induced by the already coloured vertices including v is properly coloured or the breadth-first search will stop giving an error.

```
1:  function MinLCA-greedyLeastCost-processVertex($v$)
2:      $ColourCount \leftarrow (0)_{i=1}^{maxK}$          ▷ Number of neighbours of each colour.
3:      for all $u \in \Gamma_G(v)$ do
4:          $uC \leftarrow \varphi(u)$
5:          if $uC \neq \texttt{Undefined}$ then
6:              $ColourCount_{uC} \leftarrow ColourCount_{uC} + 1$
7:          end if
8:      end for
9:      $colourCost \leftarrow +\infty$
10:     $bestColour \leftarrow \texttt{Undefined}$
11:     for $c \leftarrow 1$ to $maxK$ do
12:         if $ColourCount_c = 0$ then
13:             $currentCost \leftarrow \sum_{nC=1}^{maxK} |nC - c|\, ColourCount_{nC}$
14:             if $currentCost < colourCost$ then
15:                 $colourCost \leftarrow currentCost$
16:                 $bestColour \leftarrow c$
17:             end if
18:         end if
19:     end for
20:     $\varphi(v) \leftarrow bestColour$          ▷ If it is still undefined, returns error
21: end function
```

time.

And considering the whole graph using a breadth-first search, it can be done using $\mathcal{O}\left(m + n\, maxK^2 \log maxK\right) = \mathcal{O}\left(n^3 \log maxK\right)$ time. □

9.3 Discarded algorithms

There are some variations we have tried to use but have given such bad results with the initial tests that have been discarded completely.

The first variation consisted in randomising a bit the greedy approaches. With probability p, vertex v was not of the colour assigned by the greedy approach but randomly from the available ones. When p was small enough, this gave some small improvement in some cases, but it depended too much on the random seed and the graph.

The other variation we had consisted in solving the relaxed linear model (without the integrality constraints) and using the values of the colours for every vertex as preferences. Then, when the breadth-first search was visiting one vertex, the colour assigned to it was the available one with the highest preference. When the number of colours was bounded by the maximum degree of the graph, the algorithm ended up using too many colours and giving a solution with a very large cost. And when the number of colours was bounded by that of the used colours in a standard greedy approach, the preferences given by solving the linear programme made the new solution non-existent in some cases.

10 Local search algorithms

Local search algorithms work by making small modifications to an initial solution with the aim to get a better solution at each step. One way of doing local search is with the **simulated annealing** algorithm. According to Russell, Norvig and Davis [28], “annealing is the process used to temper or harden metals [...] by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low-energy crystalline state.” More generally speaking, the algorithm starts with a high-temperature on high-energy solution and jumps to lower energy solutions (better solutions) while the temperature decreases. If the solution is worse (with higher energy), it accepts it with a probability $P(T)$ which decreases exponentially the difference of energy $\Delta E(T)$ and the inverse of the temperature $\mathcal{F}(T)$ (with T the number of steps already done):

$$P(T) = \exp\left(\frac{\Delta E}{\mathcal{F}(T)}\right)$$

In order for the algorithm to work, we need to decide the **operators** which generate the new solutions or **neighbour** or **successor nodes** in the search space, and the function to determine the energy of a node. In our case, the energy is simply the linear distance of the colouring, with a negative sign because the algorithm is implemented for maximising the objective function.

What follows is the description of the approaches for the operators for generating the solutions.

10.1 Based on greedy recolourings

With this approach, we give an order to the vertices and colour them in that order using the same two greedy algorithms as in Chapter 9. So we start with an ordering of the vertices, such as that given by a breadth-first search on the graph, and generate the successors by swapping two vertices in the original ordering and recolouring the graph using the greedy algorithm on this new ordering.

In order to reduce the execution times, the recolourings are done from the first vertex changed in the order, and not from the beginning every time.

10.2 Based on changing the colours

With this approach, instead of reordering the vertices and recolouring them, we change the colour of some vertices and see what happens. The different ways of changing the colours that we will consider are the following:

By adding a new colour If we are not using the maximum number of allowed colours, we can add a new colour and randomly choose a vertex to have that colour instead of the one it had.

By changing the colour of a vertex Instead of adding a new colour, we can try to change the colour of a vertex to another colour already being used, as long as no neighbours are of that colour.

By swapping two colours We can select two different colours and change the vertices of one colour to the other and vice-versa.

When there is a gap in the used colours, which can happen when adding a new colour or changing the colour of a vertex, the solution is *compressed*. In other words, all vertices with a colour greater than the one not used are assigned their preceding colour.

It is important to notice that the colouring obtained when applying operators other than swapping two colours might not be a proper colouring, and hence not a valid solution. In this case, the algorithm discards the solution and tries to generate a new solution.

11 Chosen instances

In this chapter we show the different instances we are going to use in our benchmarking.

11.1 Used in MinLA benchmarking

Although the *minimum linear arrangement problem* was known before, in [5] the author Jordi Petit chose some particular graphs in order to do some benchmarks. Since then, many publications about the linear arrangement problem have used the same graphs when testing the algorithms.

The different instances can be classified as follows:

Graphs with known solution for MinLA There is a complete binary tree with 10 levels (`bintree10`), a 10-hypercube (`hc10`) and a 33×33 grid graph (`mesh33x33`).

Random graphs Graphs `randomA1` and `randomA2` are binomial random graphs, described in Section 11.2, with $n = 1000$ in both cases and $p = 0.01$ and $p = 0.05$ respectively. Graph `randomG4` is a random geometric graph in the unit square, see Section 11.3, with $n = 1000$

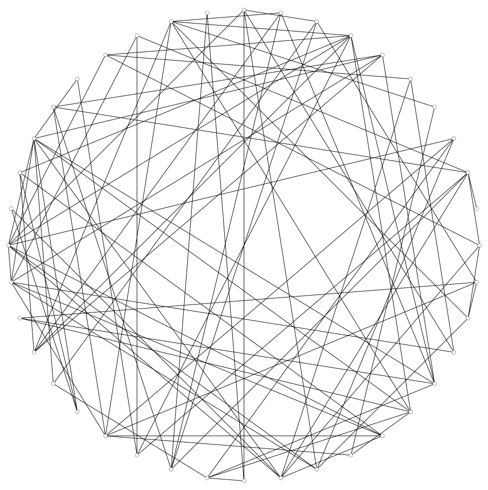


Figure 11.1: A binomial random graph with $n = 40$ and $p = 0.133$

and $r = 0.075$. Instance `randomA4` is also a binomial random graph, but with a number of edges similar to `randomG4` to study their differences.

Real life graphs These graphs come from different engineering applications. The VLSI family of graphs comes from different circuit layouts, and are `c1y` to `c5y`. Some other graphs come from finite elements discretisations for problems such as fluid dynamics (`airfoil1` and `3elt`), earthquake wave propagation (`whitaker3`) and structural mechanics (`crack`). Finally, some graphs come from graph drawing competitions and are mostly planar (`gd95c` and `gd96a` to `gd96d`).

11.2 Binomial random graphs

If G is a binomial random graph from family $\mathcal{G}(n, p)$, then $|V(G)| = n$ and every edge exists with probability p.

One reason to study these graphs is that all graphs are potentially in them. In fact, $\mathcal{G}(n, p = 0.5)$, contains all graphs (with isomorphism) with equiprobability.

Even though there are binomial random graphs on the family of MINLA instances, we are generating some new instances with p such that the expected number of edges is not within a constant factor of n^2. This can be one way generating not-too-dense graphs.

11.2.1 Choosing the edge probability

If we want the number of edges of G to be a function of the number of vertices n, that is $|E(G)| = f(n) \leq \binom{n}{2}$, we can use probability theory to choose an adequate probability p.

Consider a random variable M for the number of edges. This random variable follows a binomial distribution, $M \sim \text{Binom}\left(\binom{n}{2}, p\right)$, so if we want to have $f(n)$ edges on average, we can use the formula for the expectation of M

$$f(n) = \mathrm{E}[M] = p\binom{n}{2} = p\frac{n(n-1)}{2} \approx p\frac{n^2}{2}$$

and hence we can set $p = \dfrac{2f(n)}{n^2}$.

11.2.2 Bounds on the chromatic number

This family of graphs has been thoroughly studied. For instance, there is a result in [12, theorem 12] regarding the chromatic number which is valid for almost every graph $G \in \mathcal{G}(n,p)$:

Theorem 11.1. *Let* $0 < p < 1$ *and* $\varepsilon > 0$ *be fixed and set* $d = 1/(1-p)$. *Then, almost every* $G \in \mathcal{G}(n,p)$ *satisfies*

$$\frac{n}{\log_d n}\left(1 - 2\,\frac{\log_d \log n}{\log n}\right) \le \chi(G) \le \frac{n}{\log_d n}\left[1 + (2+\varepsilon)\,\frac{\log\log n}{\log n}\right]$$

where $\log$ *is the natural logarithm and* $\log_d$ *is in base d.*

In fact, in many cases, the chromatic number satisfies $\chi(G) \le n/\log_d n$, according to Bollobás [12, theorem 10]. These results can give us expected upper bounds when used in combination with Lemma 7.1.

11.3 Random geometric graphs

A graph G from this family $\mathcal{G}(n;r)$ of graphs has n vertices uniformly distributed in some metric space and two $u \sim v$ if $\mathrm{d}(u,v) \le r$ for some valid distance d. For instance, Díaz, Petit and Serna [1, p. 336] used the unit square with the standard Euclidean distance. However, for implementation reasons we have chosen the unit disc instead.

The reason to choose these graphs is that they are not too dense, for a small radius r which may depend on n, and usually have some interesting properties.

11.3.1 Choosing the radius

Suppose we have chosen a small radius $r > 0$. Since the distribution is uniform in the unit disc $D_1(0)$, which has an area of π, the average proportion of vertices adjacent to a vertex v (those in the disc $D_r(v)$) approaches the ratio between both disks, $\dfrac{\pi r^2}{\pi} = r^2$.

Considering this, the degree of v is $\deg(v) = n r^2$ on average. And since, by double counting, we have $2\,|E(G)| = \sum_{v \in V(G)} \deg(v) \approx 2\,n^2 r^2$, if we want $|E(G)| = f(n)$ we can take $r^2 = \dfrac{2f(n)}{n^2}$.

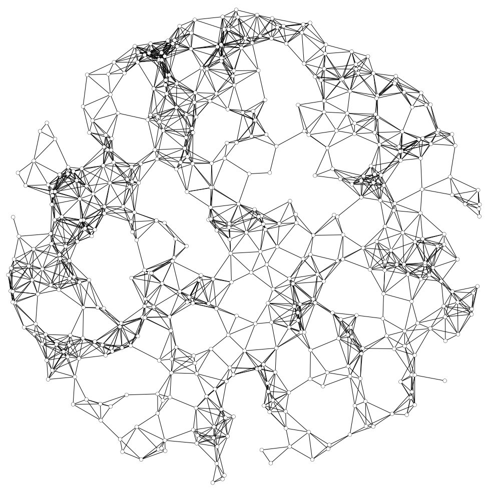


Figure 11.2: A random geometric graph with $n = 500$ and $r^2 = 0.018$

11.4 Graphs with cliques

These graphs have been chosen as a way of having more or less symmetric graphs for which we can know their optimal cost analytically in an easy way. This is in order to test if our algorithms behave well or not for *easy* instances.

11.4.1 Cycles of cliques

If we choose two integers s and t, a graph of this family has t cliques isomorphic to K_s. Then, we build a cycle connecting all the cliques, choosing two different vertices from each clique.

Corollary 11.2. *If $G = (V, E)$ is a cycle of t cliques of order s, then*

$$MinLCA(G) = t\left(1 + \frac{s^3 - s}{6}\right)$$

Proof. It is enough to notice each clique has cost $\frac{s^3 - s}{6}$ by applying Theorem 7.9, and the cycle connecting the cliques has length $2t$, with t edges from a different clique each.

If we colour the cycle using colours 1 and 2 (the subgraph is bipartite; it is an even cycle) the cliques can be optimally coloured because they have at least one edge with cost 1, which can be the one which is part of the cycle. □

11.4.2 Interconnected cliques

These graphs are thought to be locally dense graphs. If we choose two integers s and t, a graph of this family has t cliques each one isomorphic to K_s. Then, chosen a probability p, two cliques are adjacent with probability p, using the same idea as the $\mathcal{G}(t, p)$ family (with the end-vertices in each clique chosen at random).

The expected cost for this kind of graph is around $t\frac{s^3 - s}{6} + MinLCA(H)$, where $H \in \mathcal{G}(t, p)$.

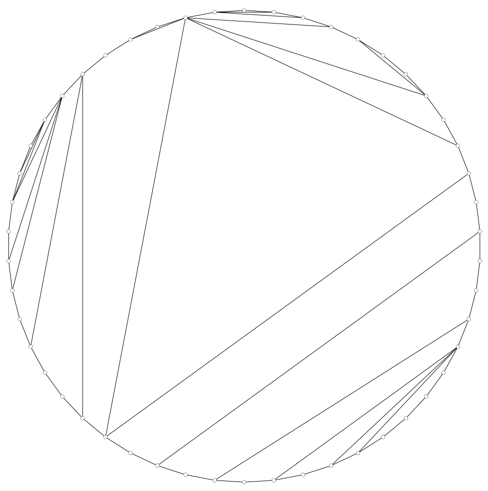


Figure 11.3: A random outerplanar graph with $n = 50$

11.5 Outerplanar graphs

For outerplanar graphs we are only considering the *blocks* (cycles with chords) because having minimally arranged the blocks, getting the colouring for the complete graph is easy. This way of working means using more than the minimum number of colours, but we are not minimising the colours but the cost of doing a linear arrangement of a colouring.

Generating random outerplanar graphs has been studied in different publications. For instance, Bodirsky and Kang [29] show a method of generating outerplanar graphs uniformly at random by means of a Las Vegas algorithm. The problem is the algorithm for generating such graphs is scattered through the text and is not easy to jump from its description to the implementation, therefore failing out of scope for the project.

In order to generate these graphs we have implemented a way which may not be uniform at all, but has given results which are different enough one from another. The description is in Algorithm 11.1, and makes use of two helper recursive functions to add the chords.

The algorithm works by breaking the outer cycle in two parts, randomly add chords crossing from one part to the other by means of Algorithm 11.2. It then adds more chords between the vertices of the same side, which we have called bridges, using Algorithm 11.3. A sample outerplanar graph generated with this method is on Fig. 11.3.

For all three algorithms, suppose `Random(`l, r`)` is a function which uniformly returns an integer from $\{l, r\}$, `RandomBoolean()` is a function following a fair Bernoulli distribution and `Edges(`$(v_1, v_2, \ldots, v_n)$`)` is a function which generates the edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}$.

Algorithm 11.1: Outerplanar graph generator

Precondition Let n be a positive integer.
Postcondition Returns an outerplanar graph G with $|V| = n$.

```
 1: function Outerplanar($n$)
 2:     $V \leftarrow [n]$
 3:     $E \leftarrow$ Edges($(1, 2, \dots, A, n, n-1, \dots, A+1, 1)$)
 4:     $A \leftarrow Random(1, n/2)$
 5:     $E \leftarrow E \cup$ CrossingChords($1, A, A+1, n$) ▷ At this moment, there are no edges between
    vertices from $\{1, \dots, A\}$ or $\{A+1, \dots, n\}$ except for those from the cycle
 6:     $BridgeLeft \leftarrow 1$
 7:     while $BridgeLeft \leq A$ do
 8:         while $BridgeLeft \leq A$ and $\deg_{(V,E)}(BridgeLeft) > 2$ do ▷ Bridges go from a lesser
    value vertex to a greater one. If the degree is greater than 2, edges might overlap.
 9:             $BridgeLeft \leftarrow BridgeLeft + 1$
10:         end while
11:         $BridgeRight \leftarrow BridgeLeft$
12:         while $BridgeRight \leq A$ and $\deg_{(V,E)}(BridgeRight) = 2$ do
13:             $BridgeRight \leftarrow BridgeRight + 1$
14:         end while
15:         if $BridgeRight \leq A$ then
16:             $E \leftarrow E \cup$ Bridges($BridgeLeft, BridgeRight$)
17:             $BridgeLeft \leftarrow BridgeRight + 1$
18:         end if
19:     end while
20:     $BridgeLeft \leftarrow A$
21:     while $BridgeLeft \leq n$ do                ▷ The same, but for the other side of the graph
22:         while $BridgeLeft \leq n$ and $\deg_{(V,E)}(BridgeLeft) > 2$ do
23:             $BridgeLeft \leftarrow BridgeLeft + 1$
24:         end while
25:         $BridgeRight \leftarrow BridgeLeft$
26:         while $BridgeRight \leq n$ and $\deg_{(V,E)}(BridgeRight) = 2$ do
27:             $BridgeRight \leftarrow BridgeRight + 1$
28:         end while
29:         if $BridgeRight \leq n$ then
30:             $E \leftarrow E \cup$ Bridges($BridgeLeft, BridgeRight$)
31:             $BridgeLeft \leftarrow BridgeRight + 1$
32:         end if
33:     end while
34:     return $G \leftarrow (V, E)$
35: end function
```

Algorithm 11.2: Crossing chords helper for generating outerplanar graphs

Precondition Let l_1, r_1, l_2, r_2 be four positive integers such that $r_1 < l_2$
Postcondition Returns some non-overlapping edges between the sets $\{l_1, \dots, r_1\}$ and $\{l_2, \dots, r_2\}$

```
1:  function CrossingChords($l_1$, $r_1$, $l_2$, $r_2$)
2:      if $r_1 < l_1$ or $r_2 < l_2$ then
3:          $E \leftarrow \varnothing$
4:      else
5:          $Top \leftarrow$ Random($l_1, r_1$)
6:          $Bottom \leftarrow$ Random($l_1, r_1$)
7:          $E \leftarrow \{\{Top, Bottom\}\}$
8:          if RandomBoolean() then    ▷ Randomly add chords to the *left* of the added chord
9:              $E \leftarrow E \cup$ CrossingChords($l_1, Top, l_2, Bottom - 1$)
10:         else
11:             $E \leftarrow E \cup$ CrossingChords($l_1, Top - 1, l_2, Bottom$)
12:         end if
13:         if RandomBoolean() then    ▷ Randomly add chords to the *right* of the added chord
14:             $E \leftarrow E \cup$ CrossingChords($Top, r_1, Bottom + 1, r_2$)
15:         else
16:             $E \leftarrow E \cup$ CrossingChords($Top + 1, r_1, Bottom, r_2$)
17:         end if
18:     end if
19:     return $E$
20: end function
```

Algorithm 11.3: Bridges helper for generating outerplanar graphs

Precondition Let l, r be positive integers.
Postcondition Returns some non-overlapping edges between vertices $\{l, \dots, r\}$ except for vertices u, v such that $v = u + 1$.

```
function Bridges($l$, $r$)
    $E \leftarrow \emptyset$
    if $l + 2 \leq r$ then    ▷ When $l + 1 = r$ we have the two vertices are adjacent in the outer cycle, so we can stop.
        if RandomBoolean() then
            $E \leftarrow \{\{l, r\}\}$
        end if
        $R \leftarrow$ Random$(1, 3)$
        if $R = 1$ then
            $E \leftarrow E \cup$ Bridges$(l + 1, r - 1)$
        else, if $R = 2$ then
            $E \leftarrow E \cup$ Bridges$(l, r - 1)$
        else
            $E \leftarrow E \cup$ Bridges$(l + 1, r)$
        end if
    end if
    return $E$
end function
```

Part V

Experimental results

12 General considerations

12.1 Implementation details

All the algorithms in this project have been implemented using the C++11 language along with the open-source graph library LEMON [6] version 1.3.1 +, part of the COIN-OR initiative.

The code has been written with a strong emphasis on portability and reusability, using C++ templates and inheritance. It should work on any system with a modern C++ compiler, subject to the availability of Gurobi in case of trying to use the linear programming models.

The results have been parsed using Python [19] version 3.4 and the plots have been generated with the use of libraries NumPy [20] version 1.9 and matplotlib [21] version 1.4.

The whole details for the implementation and executions are available on the companion benchmarking platform. This benchmarking platform is available at `http://albcom.cs.upc.edu/minlca/`.

12.2 The cluster at RDlab

The cluster used by ALBCOM at RDlab [30] works by means of the Oracle Grid Engine, a queue system which allows for assigning dedicated resources to each task. It runs on a Ubuntu 12.04.2 LTS system with an x86_64 architecture with Intel Xeon X5670 CPUs running at 2.93 GHz. The executions have been done using a single thread and a maximum of 2 GB of RAM unless stated otherwise.

The code has been compiled using C++ compiler from the GNU Compiler Collection (GCC) version 4.6 and the CMake build tool version 2.8 in release mode (uses flag `-O3` for GCC).

12.3 Random instances

We have done executions for random graphs of orders 1000, 5000, 10000, 50000, 10000, using five different seeds for each order and kind of graph.

In the case of binomial random graphs and random geometric graphs, we decided to have $f(n) = \frac{1}{2}n \log_2 n$ as the expected size (number of edges), with n the order of the graph. Therefore, we chose $p = \dfrac{\log_2 n}{n}$ and $r^2 = \dfrac{\log_2 n}{n}$, respectively.

For the graphs with cliques, we decided to have t copies of K_{10}, for $t \in \{100, 500, 1000, 5000, 1000\}$, and interconnected them using $f(t)$ edges, so the probability of two cliques being connected was $p = \dfrac{\log_2 t}{t}$.

13 Integer linear programming

13.1 Experiment design

We have done one execution for every input, with a limit of 8 threads and 8 GB of RAM instead of 2, and a time limit of 5 h for each instance (that is around 40 h of CPU time).

The executions have been done using the Gurobi solver version 5.6.0, with an initial solution given by the greedy algorithms. For Gurobi, we set a *node file start* at 500 MB. What this means is that when more than 500 MB are in use when searching through the ILP solution tree, the information of the nodes is written to disk to free memory. Having 8 threads, this means the algorithm should spend at most 4 GB (each thread has its own copy of the solution tree), but because there are some other factors spending memory not considered here, there have been instances for which the execution was aborted because of using excessive memory.

Only the instances for MINLA have been used to test the algorithm because we had started with them and the results showed to be too bad when compared to the execution times. The logs have been saved for future analysis.

13.2 Results

In this case we have only tested the integer linear programme with the instances from the Minimum linear arrangement problem. The reason is that these executions have taken much time and, in general, the results have been not of much interest. In some case there has been a 10% decrease in the cost when compared to the greedy execution when randomising the order of the vertices in the input, but in general we do not know if the results are near the optima or not.

As Table 15.1 shows, in some cases using integer programming has reached the best solutions, though we can only confirm they are the optima for `gd96c` and `small` and for bipartite graphs (those with average cost 1). When *inf* appears as the cost of the MIP solution, it means the execution was aborted due to excessive memory usage (it has happened for the three graphs with the most edges).

The results show that this is not a good approach for the problem. All executions had a time limit of 5 h of real time, which for 8 threads can mean up to 40 h of CPU time per instance, and compared to the little improvements obtained to the initial solutions this means we can discard using integer programming for this problem for the nearest future, or at least discard the model described in this document.

Not only that, the huge use of memory even after forcing to write the partial results to disc has meant aborted executions for `crack`, `randomA3` and `whitaker3`, the three largest graphs from MINLA.

If it were not by the initial solutions given by the greedy algorithms, the results would have been much worse. Initial tests showed very bad solutions even for bipartite graphs when there was no initial colouring.

14 Greedy algorithms

14.1 MinLA instances

14.1.1 Experiment design

We want to see the influence of the order when reading the graph on the results obtained with the greedy algorithms described in Chapter 9. In order to do so, we have chosen 50 randomly distributed integers from the set $\{1, \dots, 1 \times 10^9\}$ as seeds for the graph reading method. This method chooses a uniform ordering of the vertices using the *Mersenne twister* random number generator from LEMON. However, for practical reasons the edges are not added to the graph in random order. This means we are not considering some orderings which might be better when doing the breadth-first search on the graph.

14.1.2 Results

The results are in Table 14.1, and a sample boxplot for graph `randomG4` is on Fig. 14.1. The rest of the boxplots are on the benchmarking platform. See that, in general, the greedy algorithms have performed very similar one to another. Most differences can be seen when comparing the results of random graphs, and it is clear that in general the greedy least cost algorithm gives better results than the greedy nearest colour one. In fact, in all cases when doing a particular breadth-first search the cost of the solution given by the greedy least cost algorithm is at most as high as the greedy nearest colour counterpart.

14.2 Randomly generated instances

14.2.1 Experiment design

We have done one execution of both algorithms per random instance.

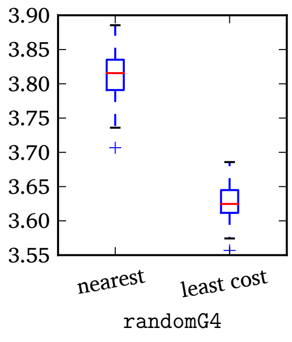


Figure 14.1: Sample boxplot of the greedy algorithms results for graph `randomG4`

14.2.2 Results

The results show many similarities with those obtained with the MINLA instances. The execution times show a good time performance when used to solve graphs with up to 10^5 vertices, not reaching 0.5 seconds in any case for the tested instances in RDlab's environment.

In the case of the graphs with cliques, the results seem to confirm the expected cost formula given in Section 11.4.2, which is a good signal.

Regarding outerplanar graphs, the results have always been using 4 colours for the selected sizes, instead of chromatic number $\chi(G) = 3$ (unless bipartite). This is a result with more importance than it may seem because outerplanar graphs are very easily colourable using the chromatic number, but it is possible it means nothing. Another interesting fact is that both algorithms have given solutions with the same cost for all outerplanar instances. They are not the best solutions, however, because the simulated annealing approach has given better solutions.

The real problem is when we take the result from Theorem 11.1 and compare it to the solutions we have got. According to the theorem, in almost every case the graph should be colourable with 3 colours, and we have needed many more. This could be for various reasons:

1. The greedy algorithms do not behave as well as initially thought and use a number of colours so high that the cost cannot be lower

2. The bound could be valid only for fixed p and the order n large enough.

			nearest colour					least cost				
				cost/$\lvert E\rvert$					cost/$\lvert E\rvert$			
Graph	$\lvert V\rvert$	$\lvert E\rvert$	avg. colours	min.	avg.	median	max.	avg. colours	min.	avg.	median	max.
3elt	4720	13722	7.12	1.56	1.58	1.58	1.59	7.30	1.55	1.56	1.56	1.58
airfoil1	4253	12289	7.22	1.56	1.58	1.58	1.60	7.12	1.54	1.56	1.56	1.58
bintree10	1023	1022	2.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00	1.00
c1y	828	1749	5.00	1.33	1.37	1.35	1.41	5.00	1.33	1.37	1.35	1.41
c2y	980	2102	4.96	1.30	1.34	1.32	1.38	4.96	1.30	1.34	1.32	1.38
c3y	1327	2844	5.00	1.32	1.35	1.34	1.38	5.00	1.32	1.35	1.34	1.38
c4y	1366	2915	5.00	1.27	1.30	1.28	1.34	5.00	1.27	1.29	1.28	1.33
c5y	1202	2557	5.00	1.30	1.32	1.31	1.36	5.00	1.30	1.32	1.31	1.36
crack	10240	30380	6.30	1.54	1.56	1.55	1.60	6.38	1.52	1.54	1.54	1.57
gd95c	62	144	5.30	1.45	1.53	1.54	1.63	5.30	1.45	1.53	1.54	1.61
gd96a	1096	1676	4.58	1.18	1.21	1.20	1.24	4.58	1.18	1.20	1.20	1.23
gd96b	111	193	2.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00	1.00
gd96c	65	125	4.08	1.19	1.25	1.26	1.30	4.08	1.19	1.25	1.26	1.30
gd96d	180	228	2.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00	1.00
hc10	1024	5120	2.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00	1.00
mesh33x33	1089	2112	2.00	1.00	1.00	1.00	1.00	2.00	1.00	1.00	1.00	1.00
randomA1	1000	4974	7.96	1.95	2.04	2.04	2.10	8.02	1.93	2.02	2.02	2.08
randomA2	1000	24738	19.74	5.44	5.59	5.58	5.77	19.44	5.34	5.46	5.46	5.58
randomA3	1000	49820	31.72	9.15	9.40	9.39	9.63	31.60	8.99	9.22	9.23	9.49
randomA4	1000	8177	10.18	2.63	2.71	2.71	2.84	10.14	2.59	2.68	2.68	2.78
randomG4	1000	8173	15.48	3.71	3.81	3.82	3.89	15.72	3.56	3.62	3.62	3.69
small	5	8	3.80	1.25	1.35	1.38	1.38	3.80	1.25	1.35	1.38	1.38
whitaker3	9800	28989	8.78	1.55	1.57	1.57	1.60	8.94	1.52	1.54	1.54	1.57

Table 14.1: Results of the greedy algorithms for the MINLA instances

15 Simulated annealing

15.1 MinLA instances

15.1.1 Experiment design

In this case, we have chosen a discretised exponential function as the temperature schedule, based on the sample Java implementation of the algorithm by Russell, Norvig and Davis [28]. If we are doing step T, and each iteration has s steps, the formula for the temperature is:

$$\mathcal{F}(T) = k \exp\left(-\lambda \left\lfloor \frac{T}{s} \right\rfloor s\right)$$

where $k > 0$ and $\lambda \in (0, 1)$ are two parameters.

The executions have been done with the following parameters, selected by plotting the probability function when $\Delta E(T) = 1$ and adjusting them manually. They are chosen to have a

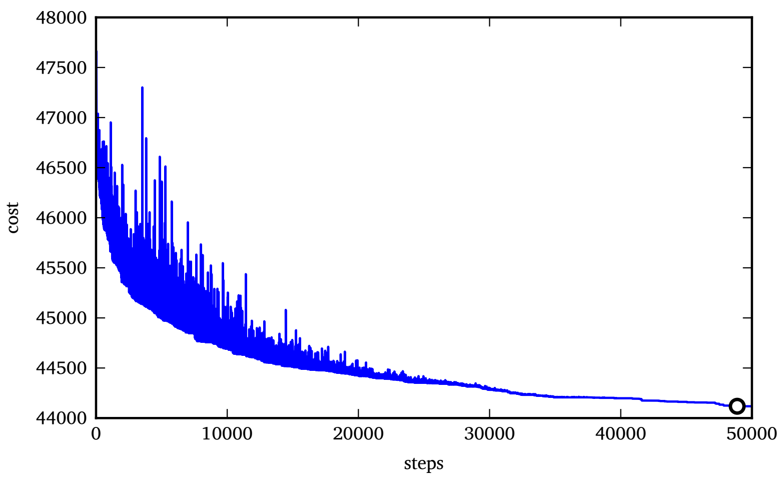


Figure 15.1: Sample cost evolution plot using simulated annealing for graph `crack` with greedy recolourings using the second group of parameters

high probability of changing to a possibly worse solution when the number of steps done is still less than a half of the maximum allowed:

First group of parameters With a maximum of 25000 steps, $s = 500$, $k = 1000$, $\lambda = 0.00035$

Second group of parameters With a maximum of 50000 steps, $s = 500$, $k = 2000$, $\lambda = 0.0002$

15.1.2 Results

Using the greedy recolourings as described in Section 10.1 with the nearest colour greedy algorithm from Section 9.1 and the second group of parameters (the ones with a *2* in Table 15.1), we have obtained some of the best results in the executions, especially when we take into account the execution time (around one minute for the largest graphs of this set of instances). The results have been the best, in general, when the initial ordering was given by doing a breadth-first search on the graph instead of using a random order for the vertices (indicated with an *r* in Table 15.1). A sample plot for the evolution of the cost using simulated annealing

The results obtained when using the method of changing the colours, described in Section 10.2, are also interesting. Sometimes they are worse than those obtained by doing the simple greedy colouring with a BFS, but that is because the initial solutions were without having randomised the order of the vertices when reading the input graphs; in fact, the initial solutions were given by doing a BFS using the nearest colour greedy approach. But the most interesting part are the execution times: they are slightly less than using the greedy recolourings, when using the second group of parameters they take around half of the time.

Graph	nearest	least	ilp	sa. col.	sa. col. (2)	sa. greedy	sa. greedy (2)	sa. greedy (r)	sa. greedy (r, 2)
3elt	1.56	1.55	1.56	1.58	1.57	1.50	1.50*	1.60	1.57
airfoil1	1.56	1.54	1.55	1.58	1.58	1.49	1.49*	1.58	1.57
bintree10	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.04	1.03
c1y	1.33	1.33	1.22	1.30	1.30	1.22	1.22*	1.25	1.24
c2y	1.30	1.30	1.17*	1.28	1.28	1.22	1.21	1.25	1.24
c3y	1.32	1.32	1.18*	1.30	1.30	1.22	1.22	1.24	1.24
c4y	1.27	1.27	1.18*	1.24	1.24	1.20	1.20	1.23	1.23
c5y	1.30	1.30	1.17*	1.26	1.26	1.21	1.21	1.25	1.25
crack	1.54	1.52	inf	1.56	1.55	1.46	1.45*	1.61	1.57
gd95c	1.45	1.45	1.38*	1.49	1.49	1.47	1.47	1.42	1.42
gd96a	1.18	1.18	1.12*	1.17	1.17	1.15	1.15	1.22	1.17
gd96b	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*
gd96c	1.19	1.19	1.16*	1.21	1.21	1.16*	1.16*	1.18	1.18
gd96d	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*
hc10	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*
mesh33x33	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.00*	1.03	1.03
randomA1	1.95	1.93	1.79	1.98	1.95	1.79	1.78*	1.86	1.84
randomA2	5.44	5.34	5.38	5.35	5.25	5.11	5.01	5.01	4.98*
randomA3	9.15	8.99	inf	8.95	8.82	8.63	8.60*	8.88	8.75
randomA4	2.63	2.59	2.62	2.60	2.57	2.39	2.37*	2.46	2.43
randomG4	3.71	3.56*	3.69	3.82	3.78	3.59	3.56	3.68	3.66
small	1.25*	1.25*	1.25*	1.25*	1.25*	1.25*	1.25*	1.25*	1.25*
whitaker3	1.55	1.52	inf	1.58	1.57	1.51	1.50*	1.64	1.61

Table 15.1: Result comparison between the algorithms. Values are cost/$|E|$. Stars indicate the best solution known for each instance.

15.2 Randomly generated instances

15.2.1 Experiment design

For this group of instances we have executed the simulated annealing search with greedy recolourings using the greedy nearest colour approach, but only with the second group of parameters described in Section 15.1.1. The instances have been the same as for the standard greedy algorithms, but limited to random geometric graphs and random outerplanar graphs with orders 1000 and 5000.

15.2.2 Results

There is not much to say about the results for these instances. The solutions are better than those given by simply doing a breadth-first search on the graph and applying the greedy colourings in that order. Perhaps the most interesting fact is that the best solutions for the chosen outerplanar graphs have been using 4 colours.

Part VI

Sustainability report

16 Sustainability analysis

As stated many times in this document, due to the nature of the project there are not many true implications from a sustainability point of view. However, we comment here the few sustainability considerations.

16.1 Timing and economic costs

The final work has used taken more time than originally planned. The reason is not because the lack of it, but because the periods without working have been inserted in the middle of the working periods. We can therefore say the planning has not been strictly followed, but the goal of having it ready for the January call has been met successfully.

Regarding the cost, as predicted in Section 5.1 during the planning phase, the true costs of the project have been much lower than estimated. According to the RDlab invoice in Fig. 16.1, we have used a total of 586 execution packs, where a pack is equivalent to 2 GB of RAM and 1 h of CPU, with a final cost of €234.40 + VAT(21%) = €283.62, quite different from the estimated €7620.00. Some more good news is there have been no added costs related to hardware, support or requirement of extra resources, with the two former being thanks to the contract between ALBCOM and the RDlab.

16.2 Social implications

The only natural implications are those derived from making the results available in an open access platform. Since we have done that, and hope to give the implementation of the Gurobi interface for LEMON back to the open-source community, results in this area are positive.

16.3 Environmental implications

The computational resources usage can be considered the most important aspect of this project. When considering the used 570.6 h of CPU in the cluster (3 weeks and 67 hours), somebody could argue that the usage of resources has been higher than the 2 weeks and 45 hours planned.

However, the plan has been done considering a different service with the 8 threads running constantly during the executions, which would be the equivalent to around 3000 execution packs in RDlab.

17 Sustainability evaluation

According to the Bachelor's Thesis regulations from FIB and the requirements in an internal document titled *El informe de sostenibilidad del TFG*, literally *The sustainability report of the Bachelor's Thesis*, all thesis must include a sustainability report based on Table 17.1.

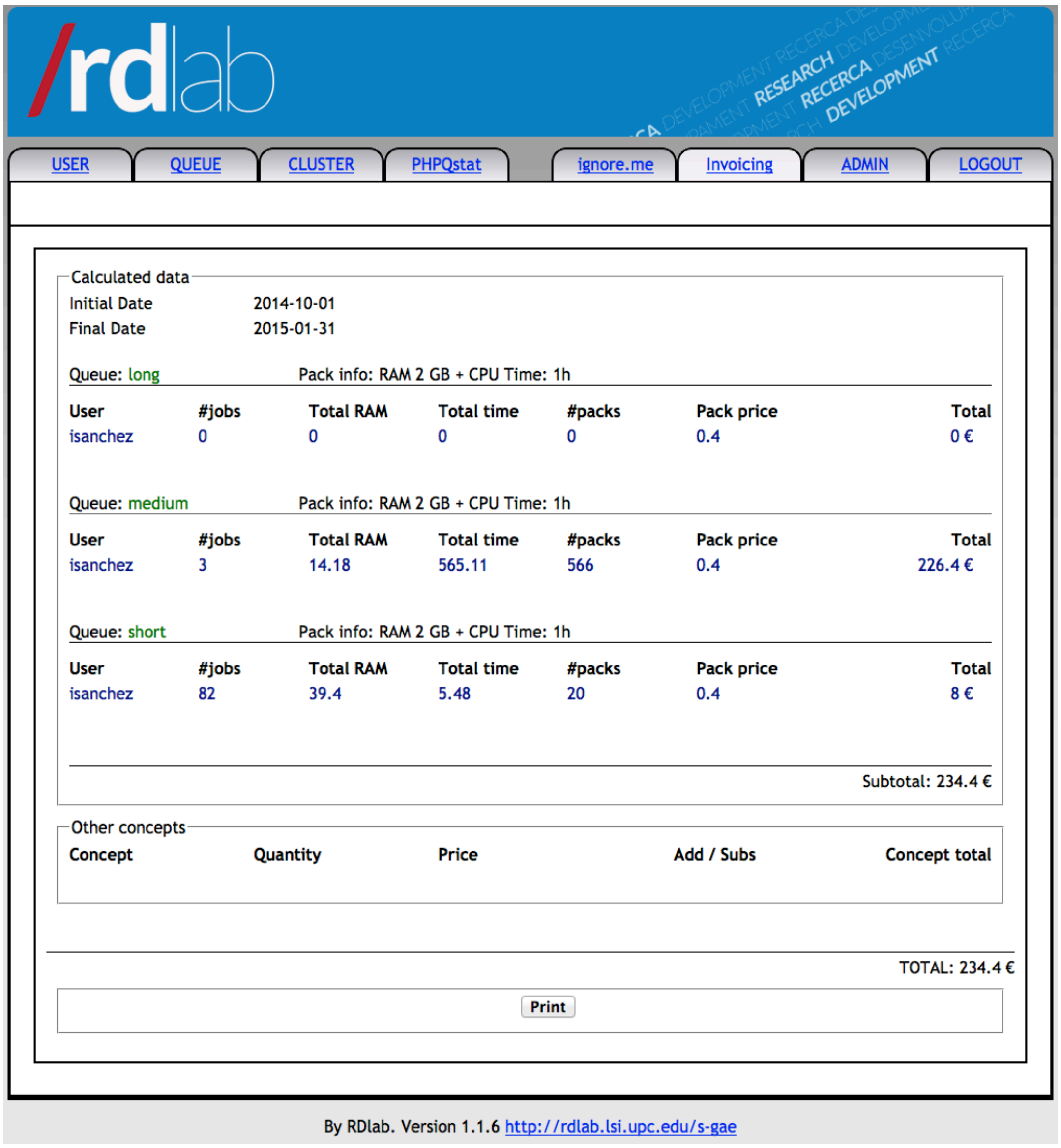
Sustainable?	Economic	Social	Environmental	Range
Planification	Economic viability	Life quality improvement	Resources analysis	(0: 10)
Results	Final cost vs. estimate	Social impact	Resources usage	(−10: 10)
Risks	Adaptation to changes	Social damages	Environmental damages	(−20: 0)
Final evaluation		(−90: 60)		

Table 17.1: Sustainability table template

We can consider there are no dangerous risks associated to any of the three aspects of the sustainability analysis, since this is purely a research project with no more possible negative implications. Taking the evaluations in the planning phase from Section 5.2 and the analysis from the previous chapter, Table 17.2 contains the final sustainability evaluation.

Sustainable?	Economic	Social	Environmental
Planification	5	7	4
Results	9	10	−1
Risks	0	0	0
Final evaluation		34	

Table 17.2: Sustainability table



Calculated data

Initial Date	2014-10-01
Final Date	2015-01-31

Queue: long — Pack info: RAM 2 GB + CPU Time: 1h

User	#jobs	Total RAM	Total time	#packs	Pack price	Total
isanchez	0	0	0	0	0.4	0 €

Queue: medium — Pack info: RAM 2 GB + CPU Time: 1h

User	#jobs	Total RAM	Total time	#packs	Pack price	Total
isanchez	3	14.18	565.11	566	0.4	226.4 €

Queue: short — Pack info: RAM 2 GB + CPU Time: 1h

User	#jobs	Total RAM	Total time	#packs	Pack price	Total
isanchez	82	39.4	5.48	20	0.4	8 €

Subtotal: 234.4 €

Other concepts

Concept	Quantity	Price	Add / Subs	Concept total

TOTAL: 234.4 €

By RDlab. Version 1.1.6 http://rdlab.lsi.upc.edu/s-gae

Figure 16.1: Invoice for cluster resources usage

Part VII

Final words

Future work

The minimum linear colouring arrangement problem offers much work to do. The available options range from trying different approaches when using local search like tabu search or changing the methods of generating the successors in the search tree, to implementing a branch and bound algorithm using backtracking, to developing new integer linear programming methods, or to using other techniques when developing new approximation algorithms.

Another path to follow would be to try and find exact algorithms for particular families of graphs. The MINLCA problem is difficult in general, but there might be cases with some practical interest which allow for polynomial-time exact algorithms (unless $\mathsf{P} = \mathsf{NP}$, then all instances would have such algorithm). For instance, bipartite graphs can be optimally solved using a linear-time algorithm, exactly the same we would use to know the two vertex classes.

From a more theoretical point of view, there are a few conjectures which can be of interest, but may not have practical implications in the short term.

Conjecture 17.1 (Number of colours in an optimal linear colouring arrangement)**.** The minimum linear colouring arrangement can be obtained using as many colours as the chromatic number. In other words, if G is a graph and $\chi(G) = k$, then

$$MinLCA(G) = LCA_k(G)$$

Comment. The best results obtained for outerplanar graphs, however, have been using 4 colours instead of 3. This could be because of the algorithms used or because they can serve as counter-examples to this conjecture.

Conjecture 17.2. If G is a graph and $\chi(G) = 3$, then

$$MinLCA(G) \leq \frac{3}{2}|E(G)|$$

Comment. This conjecture seems to be easily provable for outerplanar graphs, and might be extensible to the general case. In the case of outerplanar graphs, consider the cycles of each block obtained when *dividing* the graph through the chords. Each cycle can be 3-coloured with only one edge with cost 2, and making the right decisions this seems to show the result.

Conclusions

In this project we have done an initial study for the **minimum linear colouring arrangement** problem. We have established the initial results, developed and implemented some algorithms using different techniques. Finally, we have made everything publicly available.

There is no doubt the minimum linear colouring arrangement problem is a computationally difficult problem. There are two extreme cases, bipartite and complete graphs, which have a known optimal solution and can be found in linear time with respect to the size and order of the graph. However, most cases are difficult to solve, as can be observed in the results obtained by the simulated annealing algorithms.

Doing an extensive study of a problem is not an easy task, and can sometimes lead to unwanted situations. These situations range from getting extremely bad results, to discovering the darkness of the programming language used. Related to this second statement, the project has served to help me learn better how C++ templates work and the delicacies of multiple inheritance, which have come of great use.

I would have liked to do a more deep statistical analysis of the results. However, the lack of time derived from attending a course on graphs, having the Christmas holidays just before the presentation date and the amount of work needed to do the rest of the project have not allowed me to. All that without forgetting my way of organising my time. But all the data generated during the experiments is publicly available, allowing future work on this regard, and reminding this project has simply been the initial step to studying the MINLCA problem.

Bibliography and references

[1] J. Díaz, J. Petit and M. Serna, “A survey of graph layout problems,” *ACM Computing Surveys*, vol. 34, no. 3, pp. 313–356, Sep. 2002, ISSN: 03600300. DOI: `10.1145/568522.568523`.

[2] J. Petit, “Addenda to the Survey of Layout Problems,” *Bulletin of the European Association for Theoretical Computer Science*, no. 105, pp. 177–201, Oct. 2011, ISSN: 0252-9742. [Online]. Available: `http://www.eatcs.org/beatcs/index.php/beatcs/article/view/98`.

[3] ——, “Combining spectral sequencing and parallel simulated annealing for the MinLA problem,” *Parallel Processing Letters*, vol. 13, no. 1, pp. 77–91, Mar. 2003, ISSN: 0129-6264. DOI: `10.1142/S0129626403001161`.

[4] Y. Koren and D. Harel, “A Multi-scale Algorithm for the Linear Arrangement Problem,” in *Proceedings of the 28th International Workshop on Graph-Theoretic Concepts in Computer Science*, G. Goos, J. Hartmanis, J. van Leeuwen and L. Kučera, Eds., ser. Lecture Notes in Computer Science, vol. 2573, Berlin, Heidelberg: Springer Berlin Heidelberg, Feb. 2002, pp. 296–309, ISBN: 978-3-540-00331-1. DOI: `10.1007/3-540-36379-3`.

[5] J. Petit, “Experiments on the minimum linear arrangement problem,” *Journal of Experimental Algorithmics*, vol. 8, Jan. 2003, ISSN: 10846654. DOI: `10.1145/996546.996554`.

[6] Egerváry Research Group on Combinatorial Optimization, *Library for Efficient Modelling and Optimization in Networks (LEMON Graph Library)*, COIN-OR initiative, 2014. [Online]. Available: `http://lemon.cs.elte.hu/trac/lemon`.

[7] *Gurobi Optimizer Reference Manual*, Gurobi Optimization, Inc, 2013. [Online]. Available: `http://www.gurobi.com/documentation/5.6/reference-manual/` (visited on 14/01/2015).

[8] C. team, *Clp*, COIN-OR initiative, 2013. [Online]. Available: `https://projects.coin-or.org/Cbc`.

[9] IBM, *CPLEX Optimizer*. [Online]. Available: `http://www.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[10] J. Siek, L.-Q. Lee and A. Lumsdaine, *Boost Graph Library*, Boost C++ Libraries, 2001. [Online]. Available: `http://www.boost.org/doc/libs/1_57_0/libs/graph/doc/index.html`.

[11] Algorithmic Solutions Software GmbH, *Library of Efficient Data types and Algorithms (LEDA)*, 2012. [Online]. Available: `http://www.algorithmic-solutions.com/leda/index.htm`.

[12] B. Bollobás, “Colouring Random Graphs,” in *Random graphs*, London: Academic Press, 1985, ch. XI.3, pp. 262–267, ISBN: 0121117561.

[13] Xubuntu Community, *Xubuntu*. [Online]. Available: `http://xubuntu.org/`.

[14] GNU Project, *GNU Compiler Collection*, 2014. [Online]. Available: `https://gcc.gnu.org/`.

[15] Kitware Inc. and Insight Software Consortium, *CMake*, 2014. [Online]. Available: `http://www.cmake.org/`.

[16] GNU Project, *GNU Emacs*.

[17] G. authors, *Geany*. [Online]. Available: `http://www.geany.org/`.

[18] *TEX live*, TeX user groups. [Online]. Available: `https://www.tug.org/texlive/`.

[19] Python Software Foundation, *Python*, 2014. [Online]. Available: `https://www.python.org/`.

[20] NumPy Developers, *NumPy*, 2014. [Online]. Available: `http://www.numpy.org/`.

[21] T. matplotlib development team, *Matplotlib*, 2014. [Online]. Available: `http://matplotlib.org/`.

[22] The Inkscape Team, *Inkscape*, 2014. [Online]. Available: `https://inkscape.org/`.

[23] D. van Heesch, *Doxygen*, 2014. [Online]. Available: `http://www.doxygen.org/`.

[24] *Normativa sobre els drets de propietat industrial i intel·lectual a la UPC*, 10th Oct. 2008. [Online]. Available: `https://www.upc.edu/normatives/documents/consell-de-govern/normativa-sobre-els-drets-de-propietat-intelb7lecutal-de-la-upc` (visited on 14/01/2015).

[25] B. Bollobás, *Modern Graph Theory*, ser. Graduate Texts in Mathematics. New York, NY: Springer New York, 1998, vol. 184, ISBN: 978-0-387-98488-9. DOI: `10.1007/978-1-4612-0619-4`.

[26] Wikipedia, *Four color theorem*, Wikipedia, The Free Encyclopedia, Dec. 2014. [Online]. Available: `http://en.wikipedia.org/w/index.php?title=Four_color_theorem&oldid=636281207` (visited on 14/01/2015).

[27] A. Caprara, M. Oswald, G. Reinelt, R. Schwarz and E. Traversi, "Optimal linear arrangements using betweenness variables," *Mathematical Programming Computation*, vol. 3, no. 3, pp. 261–280, Aug. 2011, ISSN: 1867-2949. DOI: `10.1007/s12532-011-0027-7`.

[28] S. J. Russell, P. Norvig and E. Davis, *Artificial Intelligence: A Modern Approach*, Third ed. Upper Saddle River, N.J.: Prentice Hall, 2010, ISBN: 9780136042594.

[29] M. Bodirsky and M. Kang, "Generating Outerplanar Graphs Uniformly at Random," English, *Combinatorics, Probability and Computing*, vol. 15, no. 03, pp. 333–343, Apr. 2006, ISSN: 0963-5483. DOI: `10.1017/S0963548305007303`.

[30] *RDlab website*, Computer Science Department, Universitat Politècnica de Catalunya · BarcelonaTech. [Online]. Available: `http://rdlab.cs.upc.edu/index.php/en/` (visited on 14/01/2015).

[31] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*. Belmont, Massachusetts: Athena Scientific, 1997, ISBN: 1886529191.

[32] J. Díaz, M. D. Penrose, J. Petit and M. Serna, "Linear Orderings of Random Geometric Graphs," in *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science*, P. Widmayer, G. Neyer and S. Eidenbenz, Eds., ser. Lecture Notes in Computer Science, vol. 1665, Berlin, Heidelberg: Springer Berlin Heidelberg, Dec. 1999, pp. 291–302, ISBN: 978-3-540-66731-5. DOI: `10.1007/3-540-46784-X`.

[33] Wikipedia, *Graph homomorphism*, Wikipedia, The Free Encyclopedia, Jun. 2014. [Online]. Available: `http://en.wikipedia.org/w/index.php?title=Graph_homomorphism&oldid=611849953` (visited on 14/01/2015).

[34] *Gurobi Optimizer Quick Start Guide*, Gurobi Optimization, Inc, 2013. [Online]. Available: `http://www.gurobi.com/documentation/5.6/quick-start-guide/` (visited on 14/01/2015).