CS31920 Advanced Algorithms

Assignment: Applying Linear Programming

Academic Year 2020–2021

Thomas Jansen

1 Introduction

This is the assignment for CS31920 'Advanced Algorithms' and comprises 40% of the total marks for the module. It gives information about how and when to hand in, specifies the tasks you have to perform, how you will format your report, and highlights the criteria from which the assignment will be marked. Please follow these instructions carefully. If anything is unclear or you have any questions, please, send an email to `t.jansen@aber.ac.uk` and ask! Questions can also be asked on Blackboard or the module's Discord server.

2 Hand-In

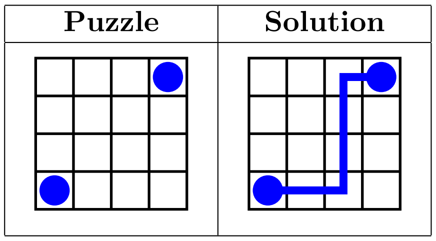
You are asked to submit your work via Blackboard. The deadline for submission is Friday, the 27th of November, 2020, at 1pm. By submitting your work you are acknowledging that it is your own work and that you are aware of both the University's and the Department's views on plagiarism. Section 6 contains more information.

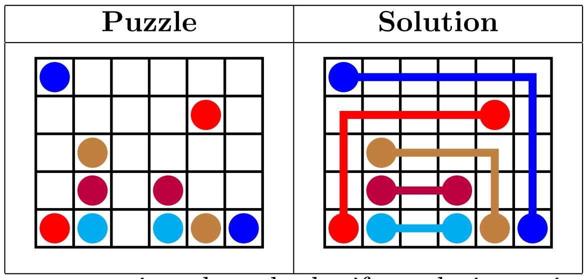
3 Task

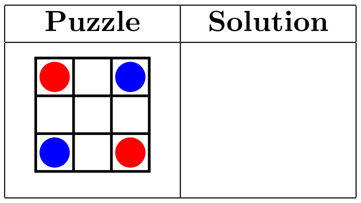
You have to come up with a way of solving the problem described below using linear programming, implementing a small C program that actually solves instances of the problem using GLPK, and write a report on how you did this.

3.1 Problem description

We are concerned with a small puzzle game for a single player, played on a rectangular grid. On the grid are pairs of squares marked with equal colours (each colour appearing exactly twice). You are asked to find a connection for all pairs simultaneously so that the path between each pair makes only use of horizontal and vertical steps (not diagonal steps) and each field of the grid is used by at most one connection or mark. Below are three examples of inputs with possible solutions. Note that possible solutions need not be unique: the first example has many different solutions. Not all inputs are necessarily solvable: the third example does not have a solution.







Write a program using linear programming that checks if a solution exists and that computes and outputs a solution if one exists.

Hint: It can be useful to think of this problem as a maximum flow problem. You have learned how to solve maximum flow problems by means of linear programming. You could try dealing with flows of different colours by having one flow problem for each colour. The only places where the different colours interact are the squares: no square can by used by more than one colour. You could share the squares among all flow problems while keeping the other aspects separate. When describing your solution it is important to give a clear, understandable, and precise description. Using figures or tables might help with that.

3.2 Implementation part

For the sake of this assignment we will encode inputs with simple text files and use the same format for the output. We encode colours with single digit numbers 1, 2, ..., 9 and empty squares with 0. Using this format the input for the second example and its solution would be encoded as follows.

Puzzle	**Solution**
`100000`	`111111`
`000020`	`222221`
`030000`	`233331`
`040400`	`244431`
`250531`	`255531`

On Blackboard as part of the assignment you will find a small C program that reads input files (using a function `int readInput(char *filename)`) and stores them as an array `int *input`. The input has `int numRows` rows and `int numCols` columns, the numbering for the rows and columns is $0, 1, \ldots,$ `numRows` $- 1$ and $0, 1, \ldots,$ `numCols` $- 1$. The number at position (i, j) can be assessed as `input[i*numCols+j]`. The function also creates an array `int *solution` of equal size that is used to store a solution. The variables `numRows`, `numCols`, `input`, and `solution` are global variables.

The program (via its main function `int main(int argc, char **argv)`) takes filenames as parameters on the command line. For each filename it tries to read the puzzle from the file and build up the data structures as described above. If this is successful it then calls a function `int computeSolution(void)` that is supposed to compute a valid solution. If a valid solution exists the solution is stored in `int *solution` and `int computeSolution(void)` returns 1. Otherwise (if no valid solution exists) `int computeSolution(void)` returns 0.

In the version of the file you find on Blackboard the implementation of `int computeSolution(void)` is incomplete and the function always returns 0.

You are asked to change `int computeSolution(void)` so that it uses linear programming (using GLPK) to solve this problem. You should use a single call of `glp_simplex` to solve the problem. You are not to use any other problem solving methods from the GLPK library, in particular not any kind of (mixed) integer programming.

Do **not** change `int readInput(char *filename)` or `int main(int argc, char **argv)`. You are free to add arbitrary other functions that can be called from `int computeSolution(void)`. You can print status information if you find this useful.

There will be some small input files provided so that you can test your implementation. When assessing your work different tests will be used. It is important that your implementation is not specific to the provided test files but works for all possible inputs.

3.3 Report

You are required to write a report about what you did that contains the following sections. Further details about the requirements are in Section 4.

1. **Introduction** Give a brief summary of your understanding of the task that allows someone who does not know the assignment brief to understand what you are doing. Your introduction should also contain a brief overview of the remainder of the report.
2. **Modelling** Describe how you design a linear program to solve the problem. Make sure to describe in detail how and why the linear program you design actually provides a correct answer to the problem.
3. **Implementation** Describe briefly your implementation. If you think there is anything remarkable or interesting about your implementation make sure to point it out and explain it. Also point out where you think that you did good work from a software engineering point of view. There is no need for a lengthy discussion of your complete implementation.
4. **Discussion and Conclusion** Briefly reflect on your approach and discuss if you are happy with it and the results it produces. Reflect on whether using linear programming to solve this problem is a good idea. If you think an alternative approach would be better explain what and why. Also, give a rough indication on how you would mark your own work (on a scale of 0–100). Use the marking scheme in Section 5 as a guide.
5. **References** Provide a list of all material that you used for your report and implementation. You can use any referencing style you like but you need to apply it correctly and consistently. The list of references needs to list at the very least the GLPK reference manual. If you used any sources to get ideas for your model (Section 2) you need to cite them in the section and list them here. If you used any help with the implementation you need to mention this when discussing your implementation (Section 3) and list the sources here. This includes the use of online sources (like discussion forums or public code repositories). In case of online sources make sure to provide the full URL and date of last access.

4 Requirements

You are asked to submit two files: a C file containing the complete source code (i. e., the file I provided which when you submit it contains your code as part of it), and a single file in PDF containing your report. The C file needs to be compilable and linkable with a standard ANSI C compiler (e. g., gcc). The only non-standard library you can use is GLPK.

There are no strict lower or upper limits on the word count of your report. The length will have no direct influence on your marks. Marks are determined only by its quality. Note that too short reports that omit significant aspects as well as too long reports that are repetitive or include information that is not relevant will not receive full marks.

Both files must be submitted via Blackboard before the deadline. Note that there are two different TurnItIn submission points, one for the report and one for the source code. **The submission deadline for the essay is Friday, the 27th of November 2020 (1pm).**

5 Marking Scheme

This assignment is worth 40% of the overall mark for CS31920. The report carries 70% of the assignment mark (28% of the module mark), the implementation carries the remaining 30% (12% of the module mark).

This report will be assessed under departmental assessment criteria in appendix AE (Practical Work) of the Student Handbook (`http://impacs-inter.dcs.aber.ac.uk/images/editor-content/Documentation/Handbooks/Appendices/AppendixAC.pdf`). The marks breakdown for the report is as follows:

- 10 marks for following the format guidelines (i. e., report is single PDF document submitted via Blackboard before the deadline, containing the five required sections including references)
- 20 marks for Section 2 (Modelling)
- 10 marks each for each of the four other sections

The mark breakdown sums to 70.

The implementation will be assessed concentrating on functionality and readability. The marks breakdown is as follows.

- 10 marks for a file that correctly handles the known example files and computes the correct result for each of them
- 10 marks for a file that correctly handles the additional test files and computes the correct result for each of them
- 10 marks for readability of the code (including useful comments)

The mark breakdown sums to 30.

6 Plagiarism

Please follow the guidelines from the Student Handbook (`https://impacs-inter.dcs.aber.ac.uk/images/editorcontent/Documentation/Handbooks/Student_Handbook_20202021_CompSci.pdf`) to help you avoid straying from legitimate and desirable cooperation into the area of plagiarism.

- Append a bibliography to your work listing all the sources you have used, including electronic ones. Do not forget to include [1] as source!
- Surround all direct quotations with inverted commas, and cite the precise source (including page numbers, or the URL and the date you accessed it if the source is on the Web) either in a footnote or in parentheses directly after the quotation.
- Use direct quotations sparingly and make sure that the bulk of the work is in your own words.
- Even if you do not use direct quotations, important ideas still need to be credited.
- Remember that it is your own input that gives a piece of work merit. Whatever sources you have used, the structure and presentation of the argument should be your own. If you are using electronic sources, do not cut and paste sections into your work. If you are using books or papers, put them aside when you actually sit down to write. In this way you will not be tempted to copy in material that you do not understand, or be at risk of unintentionally copying in more material than a brief quotation, or of accidentally leaving quotations unmarked.

References

[1] A. Makhorin (2017): GNU Linear Programming Kit Reference Manual for GLPK Version 4.65. `https://www.gnu.org/software/glpk/` last visited 05/10/2020