Nurse Rostering

A Nurse Rostering Solution with an integrated GUI

MSc in Informatics and Computer Engineering
Programming Paradigms
EIC0065-2S

António Manuel Vieira Ramadas - 201303568 (antonio.ramadas@fe.up.pt)
Guilherme Vieira Pinto - 201305803 (guilherme.vieira@fe.up.pt)
Pedro Vieira de Castro - 201305337 (up201305337@fe.up.pt)

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

July 6, 2017

1 Introduction

The need for fast, automated and efficient services has been increasing over the last decade and it does not show any signs of slowing down. Computational processing power has also seen its share of growth however it has not keep up with the tremendous rise in data quantity or with customers' needs. The need for a fast solver with valid and good solutions is an object of study for several years.

Nurse rostering (or scheduling) is a particular subset of employee scheduling that has been the focus of many researches for many years. The domain includes staffing, budgeting and short-term scheduling problems, but different staff needs is what makes nurse rostering such a challenging endeavor. Especially in an area where the scheduling of the workers (nurses in this case) has a big impact in their well-being and satisfaction which can in turn affect the working environment.

The nurse rostering problem (NRP) consists in assigning nurses to specific shifts in a way that maximizes staff requirements and nurse satisfaction and at the same time minimizes costs and nurse burndown. The satisfaction may be attributed to shift off requests granted or abide by the maximum number of consecutive working days specified in the contract. Although we count this as satisfaction, it can also be said that the lack of satisfaction is the cost of a specific scheduling.

The solution has to be the the most efficient possible to the managers of a hospital but the final user of the system will be the nurses who will have to input their specific needs into the system and will also want a graphical feedback of what their schedule will look like. This is our motivation behind the creation of a Graphical User Interface to our Nurse Rostering solver implemented in the Planning and Scheduling Methodologies course.

2 System Description

2.1 Conceptual Description

2.1.1 Functionalities

The system was implemented to solve the Nurse Rostering Problem which is a np-hard problem. The solver outputs a file according to the Second International Nurse Rostering Competition (INRC-II). It uses GRASP, Greedy Randomized Adaptive Search Procedure, to search for the best solution. The GRASP algorithm bases on the fact that sometimes multiple Hill Climbing searches, with different initial solutions outperforms more complex searches, like Tabu Search or Simulated Annealing. The solver needs to minimize the penalizations defined by the soft constraints described by the competition. Also it needs to not break hard constraints, which if it does makes the solution unfeasible.

The system was developed according to the competition regulations that stated that every schedule file had to be validated by a provided tool and then it should be sent to simulator where new history files are created. The whole process can be seen in Fig. 1.

The graphical interface allows the user to analyze the solution as its being searched. As said before, GRASP iterates through various initial solutions, searching locally for the best solution. After each local search ends, the GUI will update its information with the best schedule found. The goal is to provide a clean and accessible interface capable to better analyze the complex solutions produced. Examples of the interface may be consulted in the figures 3 and 4.

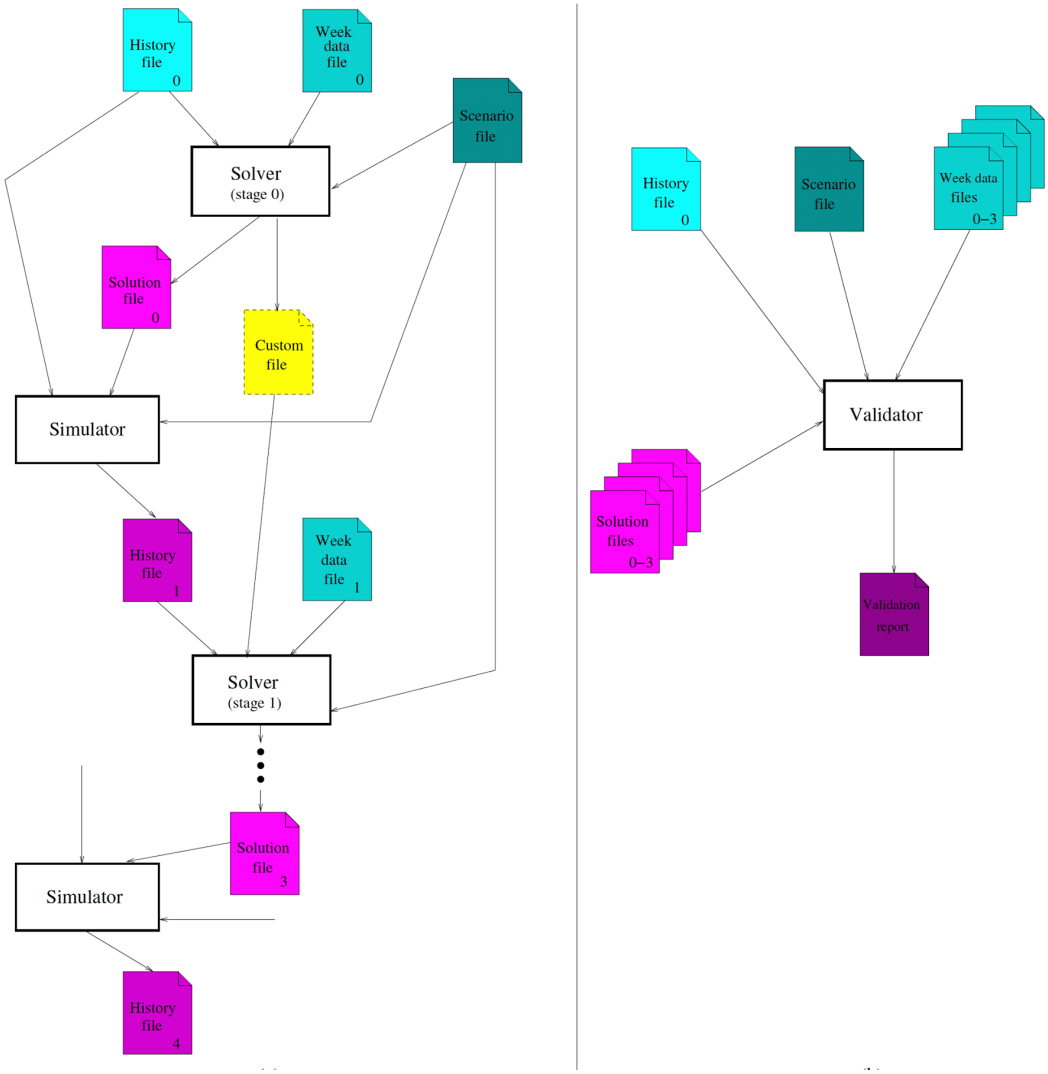

Figure 1: Diagram of how the files are dealt by the competition tools

2.1.2 Architecture

The architecture of the system is as follows:

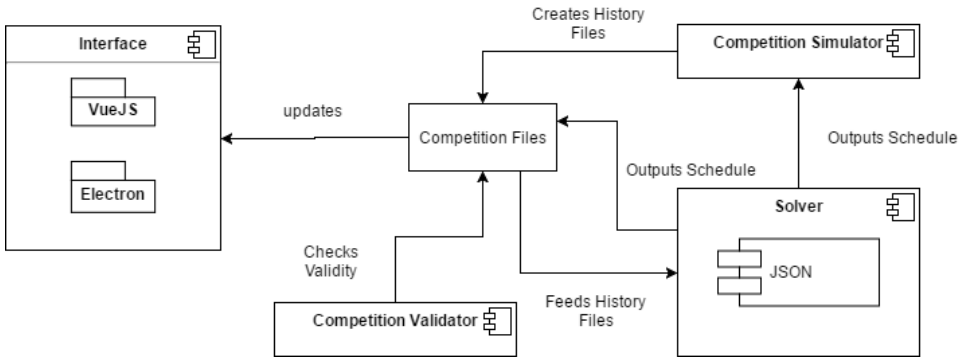

Figure 2: Architecture of the system

You can see that it all revolves around the competition files. The solver receives the history files which are created by the simulator. It then outputs the schedule of one week to the competition files folder and feeds that schedule to the simulator so it can create a new history file. This loop will happen for as many weeks as are needed.

When a new schedule is calculated, the validator will receive the new files and calculate its score according to he constraints defined in the competition. At the same time, the interface will update the graphics with the new information provided. This way, the competition specification's rules are followed and the interface is kept up to date with the new schedules. This iterative process allows the competitors to check the solutions obtained and its improvement.

2.1.3 Programming Languages

In the system, 4 programming languages are used:

- C++14
- Java
- Javascript 6
- Bash

The choice to use C++ to implement the solver was simple. They system deals with a high amount of data and a powerful algorithm that requires a lot of computation. We needed a programming language that would allow us to manage the computer resources on our own. With c++ we can more freely work with memory assignments and it allows the algorithm to run faster and with a lighter memory usage.

Also we needed a strong Object Oriented Programming language because we were dealing with highly complex objects such as a nurse or nurse's shift. These objects require a strong OOP language to support their existence as well as performing operations on them. C++ is the combination of both things we wanted. It's an improvement over C which allows the programmer to have full control over the hardware and it also supports object oriented programming, the main improvement over C. Java is chosen by the competition creators. They provide 2 jar files to be used while testing and to submit the results.

For the GUI we went for a more modern approach and we developed our interface using Electron and VueJS. Electron allows us to build desktop application with JavaScript, HTML and CSS, something that we are very familiar with. VueJS is a JavaScript Framework that allows us to create a MVVM interface, that allows a separation of the interface view from the model where all the logic is located. JavaScript is a multi-paradigm language that supports event-driven, functional, and imperative (including object-oriented) programming styles which complement the logic behind the solver. The functional paradigm in JavaScript allow us to build a more robust, concise code wise and error free interface. The imperative and objective oriented features provides the programmer with tools to deal with complex data types.

Bash is a scripting command language that specializes in the UNIX/Linux shell scripting. We use Bash to create a shell scripts that let us run all the necessary applications and still follow the specification given by the competition board. This way we can add an interface to the system without compromising the already established rules of the contest.

2.2 Implementation Description

2.2.1 Implementation Details

As previously said, the information has to be sent not just to the graphical interface but also to the competition simulator. The interface provides a clean way to execute the bash script, which properly sets up the simulator to solve the problem. The solver will then start it's data processing, fetching the data from files given by the competition. These files are parsed and it's information treated inside the solver.

Afterwards, the solver outputs the best solution found to a JSON file. After solving each and every week, a validator is then executed to validate and evaluate the overall solution.

2.2.2 Development Environment

The system was developed in an UNIX based operating system although it also works on Windows. JetBrain CLion IDE was used to develop the solver and the interface was developed using different text editors such as Sublime Text and Visual Studio Code. Our repository is hosted at GitHub.

The following modules were used in the creation of this system:

- Electron: allows the creation of a desktop application using JavaScript HTML and CSS.
- VueJS: JavaScript MVVM Framework.
- Webpack: A module bundler that solvers dependencies issues.
- Babel: to ensure JavaScript backwards compatibility (conversion from Javascript 6 to Javascript 5).
- JSON: A JSON C++ module to work with JSON files in C++ because it's not natively implemented.

2.2.3 Applications Using Our Frameworks

All the technologies we chose were carefully considered in order to develop a simple and fully functional application. We not only analysed the properties and of each tool but also looked up for similar applications using them, in order to get some ideas and perspectives of implementation.

Electron is a considerably famous framework to build cross platform desktop apps. Once we wanted to create an offline application, this tool proved to be a good choice, like it was for some well known applications like Slack, Visual Studio Code or Github Desktop.

VueJs is a progressive Javascript Framework, usefull to develop reactive components for modern web applications. VueJs is currently growing and being adopted by many developers. At the moment, there are quite a few famous applications of VueJs, highlighting Alibaba's website and even WordPress compatibility.

Finally, C++ is a very powerfull object-oriented programming language, with a set of specifications that provide great development of complex products. Many famous products are developed with C++, like Adobe's software, Apple's Mac OS X and SAP DB.

3 Conclusion

In the end, we think that the interface complements the solver extremely well. It provides a visual feedback to a complex problem and it's also good to see how the algorithm behaves. This software could very well be used in real life by a hospital or other institution that needs to create schedules efficiently and fast.

The use of different paradigms, mainly imperative and functional, allowed us to improve not only the quality of the system but also the time needed to implement all the functionalities needed. They both complement each other on this complex system with each being responsible for a crucial section of the final state of the project. Working with different paradigms demonstrated to be very efficient by having each paradigm serving a different purpose such as OOP for the solver, scripting for the automation and event-driven for the interface.

4 Improvements

The algorithm could be improved. The final result of the tests would not come in last if it were to be submitted but it has also some ground for improvement.

5 Resources

5.1 *Bibliography*

VueJS Documentation

C++14 Documentation

5.2 *Software*

JetBrain CLion 2016.3.3

VueJS

Electron

Appendices

A Appendix

Figure 3: Example of a unsolved scenario displayed in the interface

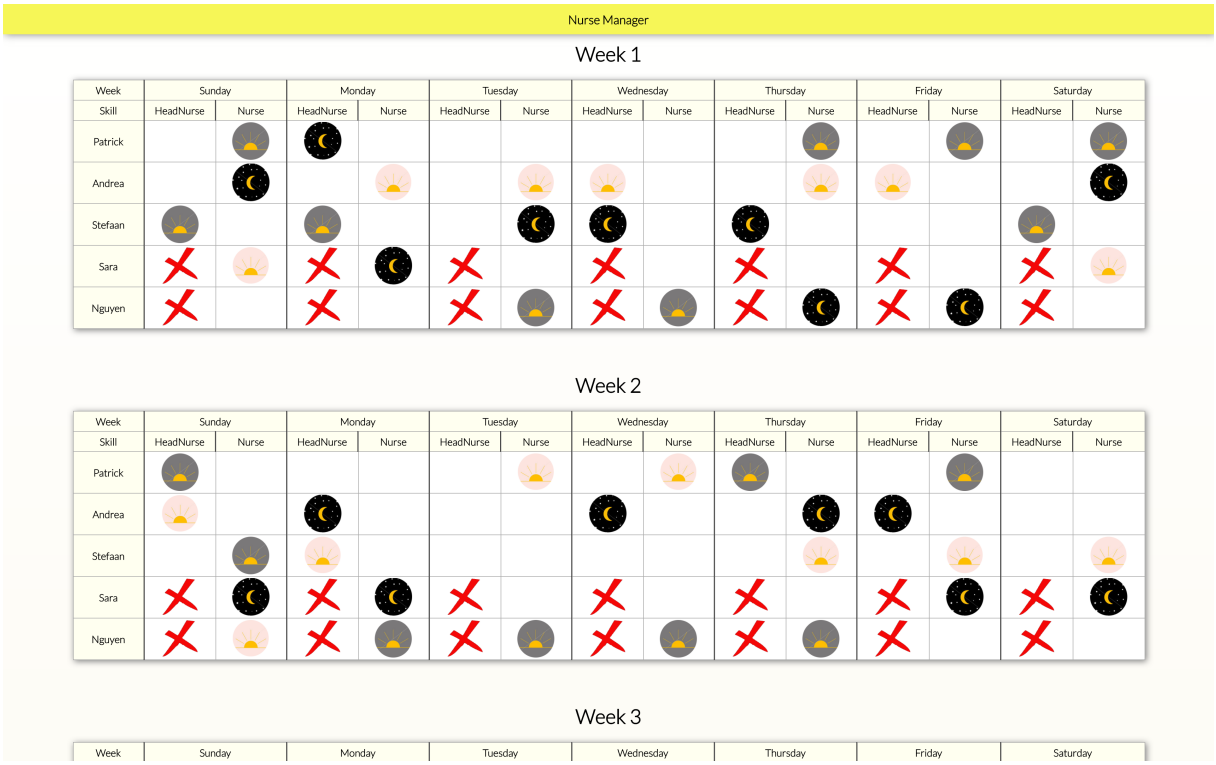

Figure 4: Example of a solved scenario displayed in the interface