Automating a complex workplace timetabling problem

Wil Hutchens
Swansea University

CS 354 - Project Specification

Dr. Bertie Müller

3rd November 2020

Abstract. This document discusses the need and motive to create a solution to timetabling problems, and proposes a solution to a complex workplace timetabling problem. The solution uses a min-conflicts heuristics approach, starting with an inconsistent set of values for our data. Then taking steps to minimise the constraint violations until a solution is found.

Contents

1 Introduction

Timetabling is a heavily researched field in computing that has been dated back to the 1950's (*Phala, J.M 2003).* However it is also a field that needs continuous input from algorithm experts because firstly, there is no universal algorithm to be used in all cases (*Mcmullan et al 2007).* This stems from the complexity of timetabling, every institution has a different set of rules and constraints meaning they need distinct methodologies to create their timetable. Secondly, many solutions freely available are also often composed for use on a simplified data set, not actually aligning with the complexity of a real world problem *(McCollumn, 2006).*

1.1 Motivation

This project specification was selected because of a specific case that was found in a company who manages their timetable in a very inefficient way. It seemed it would make sense to design a timetabling system for them as they have resembling complexities of requirements to that of an education establishment. However their requirements differ somewhat in certain aspects, meaning a typical university or school system would not be applicable. Their current solution is using an excel document and manual calculations to figure out their entire annual timetable. This means for each area (there are more than 240, each containing a number of locations and upwards of 40 staff), someone has to work out a suitable timetable every week. This is a perfect example of why timetabling is such an important field in modern computing and algorithms.

Every school, university, college and workplace (that doesn't conform to standard hours for all staff members) is presented yearly with an often highly complicated timetabling problem. Solving these problems could add up to hundreds of thousands of hours but timetabling algorithms

have allowed people to come up with solutions to complex problems in seconds. Of course data must be inputted to systems but this is negligible in comparison to alternative methods.

1.2 Related topics

Informed backtracking (IB) is a technique used in many algorithms including those used to solve constraint satisfaction problems (CSP). Described in search techniques such as depth-first search (DFS), it is the act of searching through nodes of a data structure until either a solution is found or a case is reached that does not meet criteria, in which case the process backtracks to a previous point and takes a different path through the data structure.

Min-conflict heuristics (MCH)/ Min-conflict hill-climbing (MCHC) is described by: given a set of variables and possible values for each variable, variables are giving initial values, along with constraints to apply to the variables and knowing that variables are conflicting if their values violate a constraint, finally as stated by (Minton et al 1992), “Select a variable that is in conflict, and assign it a value that minimizes the number of conflicts.”. In the setting of this project, this means to continuously look through the variables that describe the already created timetable, and change any conflicts or constraint violations to move in the direction of a solution.

Genetic algorithms (GA) are a widely used form of machine learning. Defined by, given a target in the form of a set of constraints that need to be fulfilled and an initial permutation of data, an algorithm repeatedly tries many different permutations and the ‘fittest’ permutation has more chance of passing its attributes to the next generation. Fitness of each permutation is measured by a fitness function, that in the case of a timetabling solution would be defined as how few constraints are breached. A GA will finish when subsequent generations don’t bring any change. “A typical GA can run 10^3-10^4 possible models before converging.” (*Sergei L. Kosakovsky Pond et al 2006)* This would be a problem if, like IB, a GA were to hold previous models in memory. GA however re-writes its previous models with the next generation of models; somewhat similar to an in-place sort. This gives GA a memory advantage over IB.

NP-complete or non-deterministic polynomial complete problems are those that can be solved on a Turing machine in polynomial time. In layman’s terms this means the problem can be solved by a standard computer in a regular amount of time.

Constraint Satisfaction Problems(CSP) are described as a framework in which variables can be assigned to values, but to produce a solution to the CSP, assignments must follow the rules or constraints defined by the CSP. A more formal definition is given by *Brailsford, Sally C et al 1999: ”*a set of variables, together with a finite set of possible values that can be assigned to each variable, and a list of constraints, and values of the variables that satisfy every constraint”.

When considering a university timetabling algorithm, complexity can increase dramatically even over small data sets, especially when users have the freedom to pick any combination of subjects as is often desirable for university systems.(*Caramia M. et al 2001*). For the workplace schedule, employees can be available to work at many locations so This means time complexity will have to be greatly considered in the implementation of this project. In Big-O notation, O(n), O(log n) and O(n^2) would be acceptable. O(2^n, exponential) would not be.

2 Objectives

Our workplace problem to be solved is outlined here: there are a large number of employees who have locations they are able to work. They also have a preferred location. Employees also have a ‘rank’ between 1 and 4, 1 being the lowest and 4 being the highest. Each location requires a certain number of each of the ranks 4, 3 and 2. Each location also requires a certain number of ‘qualifications’ that any employee from rank 2 upwards can have acquired. Locations can also draw employees from other locations if they are to run out of employees without

finding a suitable schedule. Employees can also be signed up to work full time, where they must be provided with 5 days of work per week, or be signed up to part time work, where they can work any number of days from 1 to 5. Locations are open 7 days a week but this can change to just 2 days a week at certain times of the year.

The aim of this project is to create a system that can be used on many different data sets to solve varying different timetabling problems. An initial plan is to develop a system to generate a random dataset that would able to represent a real world layout of a dataset, then to to create a timetable for the given dataset. There are datasets available online that represent typical educational timetabling problems, (International timetabling competition 2003 & 2019 respectively. http://www.cs.qub.ac.uk/itc2007/Login/SecretPage.php, https://www.itc2019.org/test-instances). The generated dataset will have different objects and constraints but will be of a similar complexity to the afore mentioned.

Real world timetables have many constraints that a single algorithm could not account for for all possible users. For example, a teacher is a parent of a child at their school who can not be in their classes, a lecturer teaches on two different campuses so must be allowed time in between lectures to travel whenever they are making a change, or a company offers an afternoon off of their choosing per week to each of their staff members. This leads to a real-world applicable algorithm either having to have magical powers and understand its users every need, or for a system to implement an algorithm for the bulk of the computing but still allowing users to have an input. (*Murray K. et al 2007*) This input could be in the form of giving extra constraints personal to them, or making small changes after a solution has been given.

An abstracted list of the requirements for the final result is as such:

- Employee model and employee users able to create accounts to input information about themselves. Rank, qualifications, name, locations etc.
- A ‘manager’ model that is able to edit information about employees and run the timetable generator
- Manager able to edit details in the timetable after a completed version has been proposed
- All users able to see a well laid out view of the timetable

Detailed method of how the requirements will be fulfilled is to follow in project plan.

2.1 Methods

An initial proposal was to use an exact method. Business logic would be used to sort through the dataset, starting at employees of the highest rank and adding them randomly to suitable positions, then going to the next rank and so on. This solution has no capability to correct errors, therefore would quickly find itself in ‘local’ solutions that work for a small section of the data but do not lead to a final solution and would consequently fail. A complexity issue is also raised when a dataset of real-world applicability is used. Cooper (1976) explains well: “A number of exact methods have been proposed for scheduling a project with limited resources … however, they become computationally impracticable for problems of a realistic size, either because the model grows too large, or because the solution procedure is too lengthy”.

Another approach to solving complex timetabling problems, as explained in related topics is informed backtracking. An algorithm would start off with a small initiation of values that do not break constraints, and then a DFS would be carried out until a position is found that includes all data in the set and does not break the constraints. Further optimization can be carried out to find the best solution, i.e. the least soft constraints broken. *R.Sutar, S et al (2012)* for soft and hard constraints. IB, like an exact method, would start off with employees of the highest rank and place

them into a randomly chosen suitable position, then work through the ordering of employees. It differs when a position is reached that cannot lead to a solve, i.e. a situation where two variables' only available option is one value, even though if another path was taken, there would be other values available to be assigned to. In this case IB will return itself to a previous 'version' of its calculations and traverse down a different path. IB is a complete method so can be used to find every possible permutation of data. There are varying methods to carry out IB. Including, constraint propagation, brand-and-bound and intelligent backtracking. *(Prestwich, Steven 2002).*

Min-conflict heuristics are the most praised approach to solving CSPs. They scale much more effectively than other techniques. Our timetable will use a MCH solution. See project plan for details.

2.2 Educational timetabling comparison

Educational timetabling is a heavily researched area of computing and algorithms and therefore brings a lot of information about possible methodologies. A comparison to show that the information is applicable to our workplace schedule follows.

A typical educational timetabling problem consists of:

- Students
- Rooms available
- Lecturers/Teachers
- Modules
- Hard Constraints:
    1. Students & Lecturers can't be in two subjects at once
    2. Modules must have a room for each of their hours
    3. Rooms cannot be booked by two subjects at the same time
- Soft Constraints:
    1. Students & Lecturers should have an hour break every day
    2. Modules should have two consecutive hours once per week
    3. Don't have more than one non-consecutive lecture for any module in a single day

In comparison, our workplace scheduling problem is described as:

- Employees with a rank 1-4
- Locations with differing requirements
- Hard Constraints:
    1. Employees must not be in two locations at once
    2. Locations must have at least the required number of employees ranked 2,3 & 4 every day
    3. Locations must have at least the required number of employees with the required qualifications each day
    4. Full time employees must be offered 5 days of work per week
    5. Part time employees must be offered somewhere between 1 and 5 days per week

- Soft Constraints:
    1. Full time employees should be given at least 3 days per week at their top choice of location
    2. Part time employees should be given 2 or 3 days per week
    3. Employees should only be given work at their choice of locations

The constraints of the workplace schedule are more complex than educational scheduling in some areas and less complex than others. Figure 1 displays a comparison of these two complexities.

Educational Timetabling	Our Workplace Timetabling
3 Hard Constraints	5 Hard Constraints
3 Soft Constrains	3 Soft Constraints
8 hours per day * 5 days gives 40 positions per week to be filled	5 days per week to be filled
10s of 1000s of students	~5000 Employees

Figure 1.1

The educational timetabling does have more complexity in its 8 separate hours per day in comparison to our workplace timetabling having 1 single value per day. This however will only affect the algorithm's run time as there are 8 solutions needed for every 1 of our workplace's. The workplace timetable is also adversely affected at run-time by the extra constraints, each constraint requires a search through the data that is produced therefore run time is increased heavily.

In the final project, users must be able to create employees and locations, whilst setting requirements for locations along with setting qualifications and rank for employees. Once data has been created, an optimizable schedule should be created. As in educational timetabling, our timetabling will need to allow users to be able to make changes in order to account for circumstances that can not be included in the software, to check for and resolve inconsistencies and errors, and to reflect changes in real world circumstances, e.g. someone is ill and needs covering.

3 Project Plan

As stated, an MCH approach will be taken for the timetable. In MCH, the process of assigning and reassigning variables to values is an abstract description of the technique. In terms of our data, the variables are employees and the values are days in the timetable. Each iteration will be a new assignment of employees to days. Our approach will produce a combination of this MCH and a genetic algorithm. An MCH approach will be carried out on each version of timetable inside our genetic algorithm. A number of different timetables will be initialised.

In implementation of this project, an agile software development life cycle (SDLC) will be followed. This approach will allow a solution to be finished early and continuous improvement to be the main time consumer of the project. Advantages of this cycle include: changes in requirements to be included easily throughout the lifetime of the project. Multiple episodes of testing allow incremental improvements. The project will also be more heavily designed with the capability to make adjustments throughout.

Python would arguably be the most optimal language for use in a timetabling solution due to its data manipulation capabilities, it is however a very slow language due to being an interpreted language. Java is a compiled language meaning it has an advantage in efficiency. Java is less concise so will take more lines of code to achieve the same result but will nevertheless be the language of choice for this project.

3.1 Modelling Information

3.1.1 Modelling Data

When modelling the information, we must look at modelling the data and modelling the constraints. A Unified Modelling Language (UML) diagram has been created to described the relationships between data models.

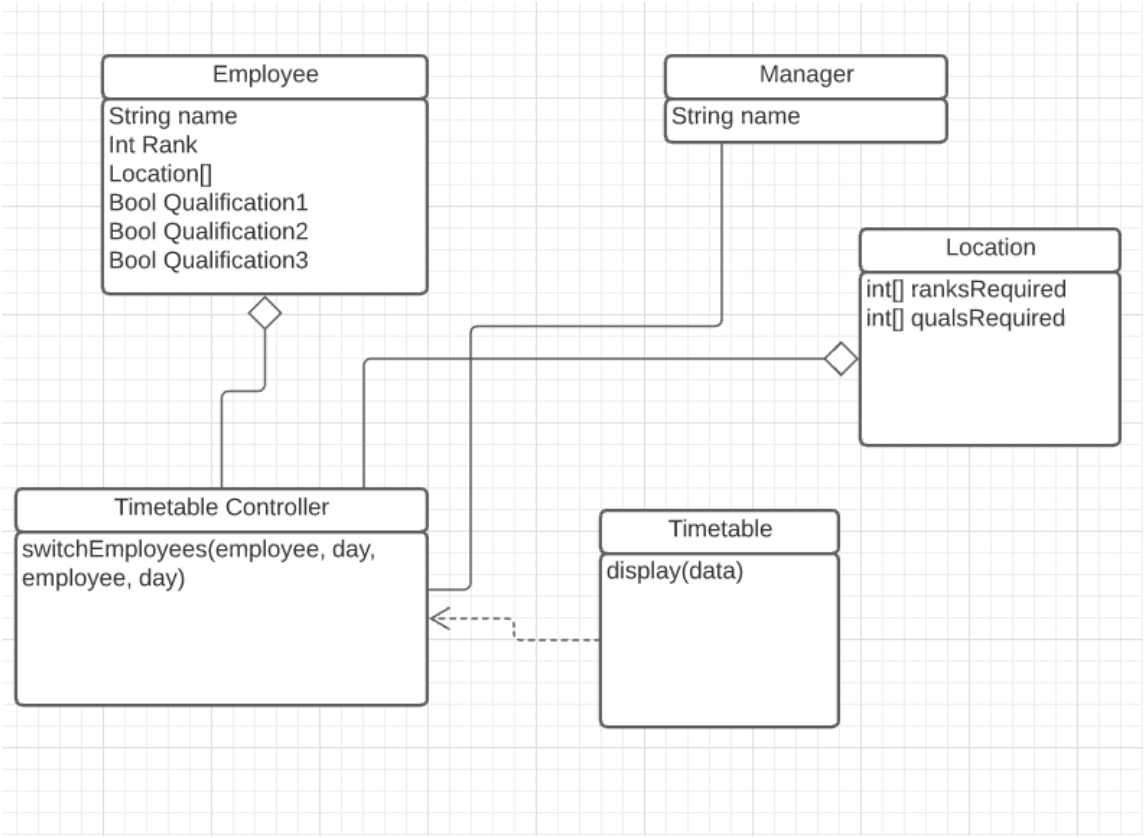

Figure 2.1 Modelling Data

When a timetable is created, there will almost certainly be changes that a manager would want to make as discussed in *Objectives.* The Timetable Controller class has function “switchEmployees()” that is dependent on a manager class. This is to allow managers to make changes to the timetable after confronted with a solution.

3.1.2 Modelling Constraints

Constraints must also be modelled in our solution. To do this, functions will have to search through the data to find and count any constraint violations that are found. A pseudocode class with functions used to search for all constraints has been written below.

```
Hard constraints:

A function for each rank 1-4 and for qualifications. Of the following format
minumumNumberOfEmployeesRankX(timetable)
    for location
        for day in timetable
            if numberOfEmployees != location.numberOfEmployeesRequiredRankX
               constraintPenalty += 1

conflictingDayForEmployees(timetable)
    for employees
        for days
            if locationsWorkingCount > 1
               constraintPenalty += 1

fullTimeEmployeesWorkingNumberOfDays(timetable)
        for employees
            if employeeDaysWorkingCount != daysRequired
               constraintPenalty += 1

Soft constraints:

employeesWorkingPreferredLocation(timetable)
    for employees where fulltime
        if daysWorkingPreferredLocation < 3
            constraintPenalty += 1 * preferredLocationChoicePenalty

partTimeEmployeesNumberOfDays(timetable)
    for employees where parttime
        if daysWorking > 3 || daysworking < 2
           constraintPenalty += 1 * partTimeNumberOfDaysPenalty

employeesWorkingChosenLocations(timetable)
    for employees
        for days
            if location isNotContainedIn(locations)
               constraintPenalty += 1 * employeesChosenLocationsPenalty
```

Figure 2.3 Constraint Modelling example

A dataset of employees, locations and their attributes must be created in early stages of development for testing purposes. To do this, a simple dataset generator class will be created that assigns incrementing sequences of strings and integers to a table of data. For locations, a list must be created using this generative technique, then members of this list will be chosen by values in the dataset. This will allow for varying sizes of datasets to be used. The workplace that inspired this project has a dataset of about 5000 users, meaning the aim will be for our solution to work on a dataset with this number of instances.

For an MCH algorithm to work, an initial permutation of data is required. In some techniques, the initial permutation is calculated using another technique. For our workplace timetable, variables will be assigned somewhat randomly. If an employee is full time, it makes sense for them to have a higher chance of being assigned to a day so a weighted choice will be included in initialisation of full time and part time employees. The pseudocode for initialising all variables is as follows:

```
for employees
    for day in timetable
        if (full_time == true)
            working(day) = math.random < 5/7 ? true : false;
        }
        else if (part_time == true) {
            working(day) = math.random < 2.5/7 ? true : false;
        }
    }
}
```

Figure 2.2. Randomly initialising data.

To know whether the next iteration of timetable has been an improvement, we must collect information on the new timetable:

- How many hard constraints are violated, a solution if not ‘feasible’ unless no hard constraints are violated.
- How many soft constraints are violated.

To figure out how good the solution is from the data:

- The sum of hard constraint violations is calculated
- The sum of soft constraints is calculated by using its weight

The weight of a soft constraint violation (CV) must be considered when implementing the project. Soft constraint weights will be a value between 0 and 1, as hard CVs will count for 1 violation and soft CVs must be weighted to be lower. An initial proposal is:

- 0.3 - Full time employees should be given at least 3 days per week at their top choice of location
- 0.4 - Part time employees should be given 2 or 3 days per week
- 0.5 - Employees should only be given work at their choice of locations

Search Method. At each iteration, the timetable will have violations reduced by making adjustments to employees’ (variables) days (values). Figure 2.3 shows pseudocode for a MCH search method (*Murray K. et al 2007).* The initial version of timetable is passed to the solve function. A best timetable is set to the initial. A process of switching conflicting values is carried out until a criterion is met. The best solution is then returned. This figure shows an iteration counter and a “canContinue” method, our solution will work similarly in that it will count the iterations and stop if a certain number has been passed, and will also stop when no more optimising is possible.

```
procedure SOLVE(initial)              // initial solution is the parameter
    iteration = 0;                    // iteration counter
    current = initial;                // current solution
    best = initial;                   // best solution
    while canContinue(current, iteration) do
        iteration = iteration + 1;
        variable = selectVariable(current);
        value = selectValue(current, variable);
        UNASSIGN(current, CONFLICTING_VARIABLES(current, variable, value));
        ASSIGN(current, variable, value);
        if better(current, best) then best = current
    end while
    return best
end procedure
```

Figure 2.3 from *Murray K. et al 2007*.

3.2 Work Schedule

This project is to be completed by spring 2021, giving an approximate time for implementation of 20 weeks. Assuming 2nd November is week 0, and the due data is week 20, splitting the work up into subsections gives a timeline for work as:

- Week 2 Created Data Model
- Week 4 Created Dataset Generating Program
- Week 6 Incorporated Constraints into Model
- Week 10 Completed simple MCH solution
- Week 11 Testing with small datasets
- Weeks 12 – 20 repeat:
    1. Expand on solution to improve competency and efficiency
    2. Test

3.3 Risk Analysis

During projection of any software, there are many possible errors that must be mitigated. In the case of this project, there will be only one person contributing work throughout, meaning version control will not be specifically necessary. That being said however, a hardware error i.e. corrupt hard drive is a real possibility so version control software will be used throughout.

Illness is a highly likely set back in any work flow, especially in modern times. To mitigate possibilities of interruptions from illness, a plan to leave at least 3 weeks spare at the end of implementation of the project leaves breathing room for any time off that may have to be taken.

4 Conclusion

We have specified the requirements for a solution to our workplace timetabling problem whilst looking at possible methodologies to solve the problem. In doing so we have compiled a viable prototype structure to begin work on producing a solution. An MCH approach has been evidenced to be the most capable algorithm to deal with timetabling problems. And has consequently been

chosen as our solution method.

References

Brailsford, Sally C (1999). *Potts, Chris N. Smith, Barbara M. European Journal of Operational Research, ISSN: 0377-2217, Vol: 119, Issue: 3, PP: 557-581*

Caramia M. (2001)*, Dell'Olmo P., Italiano G.F. New Algorithms for Examination Timetabling. In: Näher S., Wagner D. (eds) Algorithm Engineering. WAE 2000. Lecture Notes in Computer Science, vol 1982.*

Cooper, D.F. (1976)*, Heuristics for scheduling resource-constrained projects: An experimental investigation, Management Science 22, pp. 1186-1194.*

Mcmullan, Paul (2007) *& Mccollum, Barry & Burke, Edmund & Parkes, Andrew & Qu, Rong. The Second International Timetabling Competition: Examination Timetabling Track, 1.1.Queen's University Belfast, University of Nottingham. P 2.*

Mcollum, Barry (2006) *University timetabling: Bridging the gap. Masaryk. Masaryk University. P 15 – 35.*

Minton, Steven (1992) *& Andy Philips & Mark D.Johnston & Philip Laird, "Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems", P 161-205, Artificial Intelligence.*

Murray K. (2007)*, Müller T., Rudová H. Modeling and Solution of a Complex University Course Timetabling Problem. In: Burke E.K., Rudová H. (eds) Practice and Theory of Automated Timetabling VI. PATAT 2006. Lecture Notes in Computer Science, vol 3867. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-77345-0_13, 6.2.*

Phala, J.M. (2003)*. A university course timetabling problem. ORiON. Vol. 4, No. 2, pp. 92-102.*

Prestwich, Steven. *(**2002**). Local Search and Backtracking vs Non-Systematic Backtracking. AAAI 2001 Fall Symposium on Using Uncertainty within Computation.*

R.Sutar, S. and S. Bichkar, R. (2012). *University Timetabling based on Hard Constraints using Genetic Algorithm. International Journal of Computer Applications, 42(15), pp.1–7.*

Sergei L. Kosakovsky Pond (2006)*, David Posada, Michael B. Gravenor, Christopher H. Woelk, Simon D.W. Frost, GARD: a genetic algorithm for recombination detection, Bioinformatics, Volume 22, Issue 24, 15 December 2006, Pages 3096–3098, https://doi.org/10.1093/bioinformatics/btl474*