Modelling of a Mobility Meeting Scheduler

Alexandre Abreu, Pedro Seixas, Xavier Pisco
Faculty of Engineering of the University of Porto
Porto, Portugal
{up201800168,up201806227,up201806134}@up.pt

June 11, 2022

Abstract

People are willing to travel with friends during exchange periods, but, having friends in different cities, agreeing and planing a trip to a single destination can be challenging and time-consuming. Constraint programming is a powerful paradigm for solving combinatorial search problems, and this trip scheduler is one of those. This report will compare three implementations of an application that solves this problem using different tools for constraint programming.

1 Introduction

In a globalized world, people can easily have friends in other parts of the world, one situation that can make this happen in a moment where people tend to travel more is exchanges programs, which are rising substantially in the last years [Messer and Wolter, 2007]. Friends from a university will be separated for a semester, each in their own destination, but they are in a moment of their lives dedicated to knowing more about other places and cultures, so, nothing more fitting than visiting cities together. Even more so, when we consider that the most common destinations of students in exchange programs are in Europe [Van Mol and Ekamper, 2016], which is a place with lots of touristic regions with distinct cultures very close to each other [Antonakakis et al., 2015].

But this comes with one problem, the calculation of prices and scheduling of flights becomes increasingly harder when the number of people and origin cities grows. Making decisions with that many variables is hard without aid of technology. In our case, that help will consist mainly of using constraint programming to deal with lots of variables and constraints.

With this project, we aim to create an application that can help people to make decisions about their travelling plans taking into consideration their costs and times, among other variables. But, the main goal is to compare different implementations for this exact problem, using as tools: *SICStus Prolog*, *Google OR-Tools* and *IBM ILOG CPLEX Optimization Studio*, some of the most well-known constraint programming solvers [Stuckey et al., 2014].

The main idea is to have an application that can receive information of the students with their respective stay and the destinations they are pondering to go. Together with some constraints like their available hours, minimum time of the trip and even their preferences about flights, like duration and number of connections.

2 Problem

As discussed in the introduction, it takes time to find the better solution by hand, as there are different goals:

- Choose a single destination for everyone;
- Spend as little money as possible;
- Spend as much time together in the destination as possible;
- Restrict the earliest hour a person is willing to arrive in the airport to leave their city;
- Restrict the latest hour a person arrives at their city;
- Restrict the duration of flights a person is willing to take;

- Restrict the number of connections a person is willing to make;
- Define the days each person is able to be outside their city;

The application should take all these variables into account, to provide the optimal solution for a set of flights and students.

3 Formal Definition

3.1 Data Attributes

The project uses data related to flights, trips, students and groups, all of them are vectors and their definitions are regarded in this section.

3.1.1 Flights

Each flight is defined as:

$$f = \langle origin, destination, departure, arrival, duration, price \rangle$$

Where:

- **origin** and **destination** are codes of the cities or airports, following the same convention as Google Flights, which has replaced other travel websites due to its user-friendliness, processing and advanced features [Gerth, 2019];
- **departure** and **arrival** are timestamps in the same timezone as their respective locations';
- **price** is always in euros.

3.1.2 Trips

Each trip is defined as a list of flights:

$$t = \langle f_1, f_2, f_3, ..., f_n \rangle$$

Where:

- For any two consecutive flights f_i and f_{i+i}, the destination of f_i is the same as the origin of f_{i+i};
- The **origin** and **departure** of the trip are the same as of f_0;
- The **destination** and **arrival** of the trip are the same as of f_n;
- The **duration** of the trip is the time difference between its **arrival** and **departure**, in minutes, so, it includes connection times;
- The **price** of the trip is the sum of all the individual flight prices;
- The **number of connections** of the trip is the number n of flights;
- For most purposes, we treat trips as flights with a **number of connections** attribute.

3.1.3 Mobility Students

Each mobility student is defined as:

$$s = \langle city, availability, max\ connections, max\ duration, earliest\ departure, latest\ arrival \rangle$$

Where:

- **city** is the current location of the student, that is, the place where they are located during their mobility time;
- **max connections** is the maximum number of connections the student is willing to take, it includes the departure and arrival trips;

- **max duration** is the maximum duration for each trip, it is represented in minutes;
- **earliest departure** and **latest arrival** are represented in hour and minutes of the day in the same timezone used in **city**;
- **availability** is represented as a list of dates where the student is available to travel.

3.1.4 Problem Input

An input to the problem is defined as:

$$i = \langle group, minimum\ useful\ time, destinations \rangle$$

Where:

- **group** is a list of all the students;
- **minimum useful time** is the minimum value between the last arrival and first departure in the destination between the members of the group, it is represented in seconds;
- **destinations** is a list of the possible destinations.

3.1.5 Solution

Finally, a solution is defined as a list of pairs:

$$s = \langle (to_0, ti_0), (to_1, ti_1), ..., (to_n, ti_n) \rangle$$

Where:

- n is the number of people in the group;
- to_i and ti_i represent, respectively, the outgoing and incoming trips of the ith member of the group. The pair (to_i, ti_i) can be referred to as p_i;
- All incoming trip's **origin** and outgoing trips' **destination** have the same value, which represents the place all students will travel to;

3.2 Restrictions

The restriction attributes are defined for students or for the global context, this section will enumerate each one of the restrictions by their type.

Restrictions for students:

- The student will leave its city after **earliest departure**, and they will arrive before **latest arrival**. These are hard restrictions relative to the departure of the outgoing trip and arrival of the incoming trip of the student;
- The entire travel time will entirely take place within the days of the week the student is available, defined in **availability**;
- The trips that will be considered include less than **max duration** minutes and less than **max connections**.

Global restrictions:

- All people of the group must travel to the same place, that is an element of **destinations**;
- The intersection of the time each person stays in the destination is larger than **minimum useful time**;

3.3 Objective

The objectives of the application are:

- Minimize the sum of the trips' cost for every group member, referred to as **individual cost**;
- Minimize the sum of students' costs, referred to as **total cost**;
- Maximize the time all the students are together in their destination, referred to as **useful time**;
- Minimize the time when each person is in the destination, but not the entire group, referred to as **separated time**.

4 Implementation

4.1 Lists of Flight Attributes

Flights information is stored into lists, one for each attribute, so, we have a list of durations, a list of prices, and so on. By accessing them on the index i we get information for the i^{th} trip.

This was done for using the function to use a variable to access the lists, which requires a list of numbers. This is the reason all the non-numerical attributes (the timestamps and relative hours) were converted to integers.

4.2 Solution Variables

The solution variables consist of a list of pairs of trip indices, one pair for each student, with an outgoing and an incoming trip.

4.3 Dummy Trips

To avoid having conditionals in the restrictions, we added dummy variables, so that people that need to travel and people already in the destination have exactly the same constraints.

The dummy trips are created for every student interval, and are instantaneous, one happens at the start of the interval and another one at the end. In that way, a student available on an interval can take a trip to their own city at the start of the interval, pay nothing, spend no time and take no connections and end up in the destination, and the same happens for their return.

4.4 Restrictions and Dependant Variables

The restrictions use the trips on solution variables as indices for reading the concrete values from the lists of flight attributes. After reading those values, we can restrict them accordingly.

The same technique can be used to select the concrete values and aggregating them to create variables that will depend on the solution ones.

4.5 Other Implementation

Instead of having one list for each of the flight attributes, this solution used only one list for the flights and another for the destinations. Each of the used flights has a number between the negative number of students and the number of students to represent the outgoing and incoming flights, flights that are not used have a 0. As an example, the outgoing flight of the second student has a value of 2 and the incoming flight has a value of -2.

After applying all the needed constraints we executed the program with some data, and we found that it was almost 10 times slower than the other implementation with a sample dataset and wouldn't finish with the full dataset, so, we decided to not work more with this solution.

The code can still be found in the or-tools folder inside the "old" subfolder.

5 Tests and Performance

5.1 Methodology

All performance measurements only consider the time it takes for a program, already having parsed the input files, to add restrictions, calculate the dependant variables and find the optimal solution.

For *SICStus*, we executed the program 10 times for each combination of the following values for enumeration attributes:

- `leftmost`, `min`, `max`, `ff` or `ffc` to control the selection order of variables;
- `step`, `enum` or `bisect` to control the value selection for the chosen variable;
- `up` or `down` to control the exploration of the domain.

For *DOcplex*, we executed the program 10 times for each combination of the following values for enumeration attributes:

- `Auto`, `DepthFirst`, `Restart`, `MultiPoint`, `IterativeDiving` or `Neighborhood` to choose the search type.
- For variable selection, it was used the selectors `Random`, `Smallest` or `Largest` and the evaluators `DomainSize`, `DomainMax`, `DomainMin`;
- For value selection, it was used the selectors `Random`, `Smallest` or `Largest` and the `Value` evaluator;

For *OR-Tools*, we executed the program 5 times for each combination of the combination of the following values because it was slower than both of the other solvers:

- *MinDomain*, *MaxDomain*, *First*, *First*, *HighestMax* and *LowestMin* to control the selection order of the variables
- *MaxValue*, *MinValue*, *LowerHalf*, *UpperHalf* to control the exploration of the domain.

5.2 Results

5.2.1 SICStus

The two best combinations of search parameters consistently got solutions under 0.5 seconds, they are as follows:

1. The combination [`min`, `bisect`, `down`] taking 0.45 seconds on average.
2. The combination [`max`, `bisect`, `down`] taking 0.46 seconds on average.

The entire table of results can be seen here:

Next Variable	Value Selection	Domain Exploration	Average Runtime (s)
leftmost	step	up	3.20
leftmost	step	down	1.55
leftmost	enum	up	1.00
leftmost	enum	down	0.54
leftmost	bisect	up	0.48
leftmost	bisect	down	0.53
min	step	up	2.06
min	step	down	0.55
min	enum	up	0.75
min	enum	down	0.48
min	bisect	up	0.58
min	bisect	down	0.45
max	step	up	1.01
max	step	down	0.93
max	enum	up	0.76
max	enum	down	0.49
max	bisect	up	0.68
max	bisect	down	0.46
ff	step	up	1.01
ff	step	down	0.55
ff	enum	up	0.75
ff	enum	down	0.49
ff	bisect	up	0.49
ff	bisect	down	0.58
ffc	step	up	1.02
ffc	step	down	0.55
ffc	enum	up	0.75
ffc	enum	down	0.58
ffc	bisect	up	0.53
ffc	bisect	down	0.51

It is worth pointing out that the implementation in *PROLOG* is the slowest one to parse the JSON input files, taking, on average, 4.15 seconds on this part alone.

5.2.2 DOcplex

For the search type, the fastest one and the one who found a higher number of solutions was `DepthFirst` with an average runtime of 1.39 seconds. The full table of results is shown here:

Search Type	Solutions	Branches	Fails	Average Runtime (s)
Auto	68	18162	8419	6.23
DepthFirst	84	4440	1156	1.39
Restart	71	12565	5468	5.99
MultiPoint	49	2416607	1282821	Timeout
IterativeDiving	73	29284	14174	24.74
Neighborhood	0	6005929	-	Timeout

In order to test the different search phases, the search type `DepthFirst` was used, since it gave the best results overall.

Var Selector	Var Evaluator	Value Selector	Value Evaluator	Solutions	Average Runtime (s)
Default	-	Default	-	84	1.39
Random	-	Random	-	97	1.48
Random	-	Smallest	Value	91	1.64
Random	-	Largest	Value	248	1.76
Smallest	Domain Size	Random	-	166	1.38
Smallest	Domain Size	Smallest	Value	84	1.37
Smallest	Domain Size	Largest	Value	262	1.58
Largest	Domain Size	Random	-	112	12.23
Largest	Domain Size	Smallest	Value	90	11.23
Largest	Domain Size	Largest	Value	258	13.7
Smallest	Domain Max	Random	-	136	1.65
Smallest	Domain Max	Smallest	Value	89	1.35
Smallest	Domain Max	Largest	Value	242	1.65
Largest	Domain Max	Random	-	98	5.44
Largest	Domain Max	Smallest	Value	93	8.16
Largest	Domain Max	Largest	Value	271	8.38
Smallest	Domain Min	Random	-	136	1.70
Smallest	Domain Min	Smallest	Value	89	1.37
Smallest	Domain Min	Largest	Value	242	1.64
Largest	Domain Min	Random	-	91	5.26
Largest	Domain Min	Smallest	Value	77	6.85
Largest	Domain Min	Largest	Value	281	12.51

Some evaluators such as `VarImpact`, `VarLocalImpact`, `VarSuccessRate` or `ImpactOfLastBranch` for the variables, and `ValueImpact` or `ValueSuccessRate` for values, do not work with the `DepthFirst` search type. Using them was only possible with other search types, which demonstrated to be significantly slower than `DepthFirst`.

With this table, there are some results that stand out for their good performance, namely:

1. The combination [`Smallest Domain Max`, `Smallest Value`] taking 1.35 seconds on average.
2. The combination [`Smallest Domain Size`, `Smallest Value`] taking 1.37 seconds on average.
3. The combination [`Smallest Domain Min`, `Smallest Value`] taking 1.37 seconds on average.

These are small changes when compared to the `Default` phase, but with the average times being so low, they demonstrated to be quite significant.

5.2.3 OR-Tools

In this implementation, there was no big difference between the search parameters that were used, but the overall best results were achieved when the chosen variable was the first one, which started by the `destination` followed by the flights in order of student.

The entire table of results can be seen here:

Next Variable	Domain Exploration	Average Runtime (s)
MinDomain	MaxValue	12.40
MinDomain	MinValue	12.34
MinDomain	LowerHalf	12.20
MinDomain	UpperHalf	12.38
MaxDomain	MaxValue	12.36
MaxDomain	MinValue	12.54
MaxDomain	LowerHalf	12.39
MaxDomain	UpperHalf	12.09
First	MaxValue	12.27
First	MinValue	12.01
First	LowerHalf	12.04
First	UpperHalf	12.07
HighestMax	MaxValue	11.95
HighestMax	MinValue	12.78
HighestMax	LowerHalf	12.58
HighestMax	UpperHalf	12.91
LowestMin	MaxValue	12.55
LowestMin	MinValue	12.49
LowestMin	LowerHalf	12.58
LowestMin	UpperHalf	12.76

We thought about the possibility of OR-Tools finding good solutions fast but taking a long time to check every other solution, so we set the solver to stop after 5 solutions to see if that would be fast, but the solver would still take around 11 seconds to get to the first 5 solutions. In some combinations of decision strategies, it will speed up to about 9.5 seconds, but not fast enough to be useful when compared to both the other solvers.

6 Conclusions

This project demonstrated the efficiency of constraint programming, providing powerful tools to describe our problem without having to describe the low-level implementation details, resulting in very fast and easy to extend programs.

In our solution to this problem, SICStus proved to be the fastest solver, being 3 times faster than DOcplex and almost 30 times faster than OR-Tools. The only problem with our PROLOG solution was the amount of time it would need to read the JSON files that were created, 4.5 seconds compared to 0.30 seconds in Python.

In the end, both the SICStus and the DOcplex would be fast enough for a real-world product, given that the flights' information has already been saved in a JSON file.

References

[Antonakakis et al., 2015] Antonakakis, N., Dragouni, M., and Filis, G. (2015). How strong is the linkage between tourism and economic growth in europe? *Economic Modelling*, 44:142–155.

[Gerth, 2019] Gerth, K. M. (2019). *Creating sustainable competitive advantage in the German B2B lending business: the case study of Google llc.* PhD thesis.

[Messer and Wolter, 2007] Messer, D. and Wolter, S. C. (2007). Are student exchange programs worth it? *Higher education*, 54(5):647–663.

[Stuckey et al., 2014] Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., and Fischer, J. (2014). The minizinc challenge 2008–2013. *AI Magazine*, 35(2):55–60.

[Van Mol and Ekamper, 2016] Van Mol, C. and Ekamper, P. (2016). Destination cities of european exchange students. *Geografisk Tidsskrift-Danish Journal of Geography*, 116(1):85–91.