Solution to an assignment problem using Answer Set Programming

Yana Kozak

January 17, 2020

1 Description of our assignment problem

Given the following information:

- There are **N desks** for check-in in an airport. Those desks are identical and placed sequentially in one row: from the desk number `1` up to the desk number `N`.
- For today (from midnight to midnight) there have been scheduled **M flights** (flight 1, ... flight M). Each flight requires a certain number of desks for check-in - **num_desks**(`<flight>`) - in a specific **time interval** **[a,b[** - that is from hour `a` included up to hour `b` excluded.
- The **check-in desks** that are being **reserved** for a specific flight must be **consecutive** (they must be a sub-sequence of `1..N`).
- A desk can not be reserved for two different flights in the same moment.

, write an ASP program that for all of the scheduled flights tries to **find a way to reserve the desks** they require.

2 Solution

Facts. First, we state which atoms are true - as in Figure 1. Predicate `p(1..N)` is a shortcut for `p(1), p(2), ..., p(n)`. So we have 24 hours, 2 scheduled flights and `n` desks. `time_interval(FLIGHT, FROM, TO)` indicates when the desks are needed, while `desks_required(FLIGHT, NUMBER)` indicates the number of desks a specific flight needs for check-in.

```
hour(0..23).
flight(1..2).
desk(1..n).
time_interval(1,0,2).
time_interval(2,0,2).
desks_required(1,2).
desks_required(2,3).
```

Figure 1: Facts.

Auxiliary predicates. We needed some auxiliary predicates in order to define our solution:

- `min(FLIGHT, DESK, HOUR)` - finds the desk with the smallest ID - among all of the desks that have been reserved to a specific flight `V`, at a specific hour `O`- see Figure 2. The aggregate function `#min` has been used for this purpose.

```
min(V,N,O) :- N = #min { D: desk(D), desk_
reserved(V,D,O)}, flight(V), time_interval
(V,I,F), hour(O), O>=I, O<F.
```

Figure 2: Auxiliary predicate `min(FLIGHT, DESK, HOUR)`.

- `gauss_sum(FLIGHT, SUM, HOUR)` finds what the sum of the consecutive desks' IDs (assigned to a flight `V`, at hour `O`) should be, if the smallest of the desks' IDs is `min(FLIGHT, DESK, HOUR)` - see Figure 3. `(N*2+D-1)` is the sum of the smallest desk ID - `N` - and of the expected largest ID - that is `N+D-1`, where `D` is the number of desks reserved for that flight, at that hour.

```
gauss_sum(V,G,O) :- min(V,N,O), hour(O),
desk(A), flight(V), desks_required(V,D),
desk(D), G=(N*2+D-1)*D/2.
```

Figure 3: Auxiliary predicate `gauss_sum(FLIGHT, SUM, HOUR)`.

- `actual_sum(FLIGHT, SUM, HOUR)` - Figure 4 - sums up - using the respective `aggregate function #sum` - the IDs of all of the desk that have been reserved for a flight `V`, at hour `O`. We will compare, later, this number to the number we would get if these desks were consecutive (`gauss_sum(FLIGHT, SUM, HOUR)`) - if they are not identical, we will discard the solution.

```
actual_sum(V,N,O) :- N = #sum { D: desk(D),
 desk_reserved(V,D,O)}, hour(O), flight(V),
 time_interval(V,I,F), hour(O), O>=I, O<F.
```

Figure 4: Auxiliary predicate `actual_sum(FLIGHT, SUM, HOUR)`.

Generation of potential solutions. By writing `N{desk_reserved(FLIGHT, DESK, HOUR)[...]}N`, what we are trying to do is to reserve for each flight the desks it needs at the hours that are included in the time interval going from hour `I` included to hour `F` excluded. For this purpose we use a `cardinality constraint` - see Figure 5.

```
N{desk_reserved(V,D,O) : desk(D)}N :- flight
(V), desks_required(V,N), time_interval(V,I,
F), hour(O), O>=I, O<F.
```

Figure 5: Generation of potential solutions: `desk_reserved(FLIGHT, DESK, HOUR)`.

Trimming generated solutions. Our set of potential solutions, as it has been defined, may reserve for a flight's check-in desks that our not consecutive: we want to avoid this. So, as it have been mentioned earlier, we use our auxiliary predicates `actual_sum(FLIGHT, SUM, HOUR)` and `gauss_sum(FLIGHT, SUM, HOUR)` to assure that desks are consecutive: see Figure 6.

```
:- gauss_sum(V,N,O),actual_sum(V,S,O)
,flight(V),hour(O), N!=S.
```

Figure 6: Remove solutions in which desks reserved for a flight's check-in are not consecutive.

Now, we still can have solutions in which desks are assigned to two different flights. Figure 7 shows how we resolve this issue.

```
:- desk_reserved(V,D,O), desk_reserved
(T,D,O), hour(O), flight(V), flight(T)
, desk(D), V!=T.
```

Figure 7: Imposing that the same desk can not be assigned to two different flights.

Finally, the last restriction that must be introduced is that one which imposes that once we assigned a specific subsequence of desks for a flight's check-in, it remains the same for the entire time interval required for check-in. We simply check that the desk with smallest ID inside that subsequence of desks is the same in the different hours - in case check-in lasts more than one hour - Figure 8.

```
:- min(V,N,O), min(V,S,F), flight(V),
hour(O), hour(F), N!=S.
```

Figure 8: Imposing that the subsequnce of desks does not change in time - in case the check-in lasts more than one hour.

Visualization. We want to visualize only the data of an instance of the problem and the list of desks 'reservations'. To accomplish this, we hide the auxiliary predicates by writing the meta-statements showed in Figure 9.

```
#show.
#show desks_required/2.
#show time_interval/3.
#show desk_reserved/3.
```

Figure 9: Hiding auxiliary predicates.

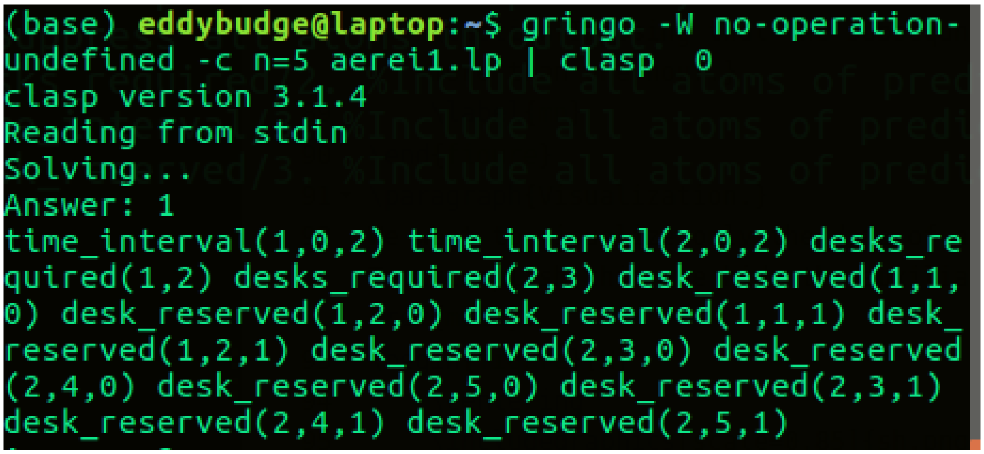
3 Testing

We will use the following `gringo` command line option `-c n=X` to define the number of desks.

First test. We have two scheduled flights: the first one requires 2 desks, while the other requires 3 of them - in the same hours (from midnight to 2 A.M.). In other words, our instance of the problem is the following:

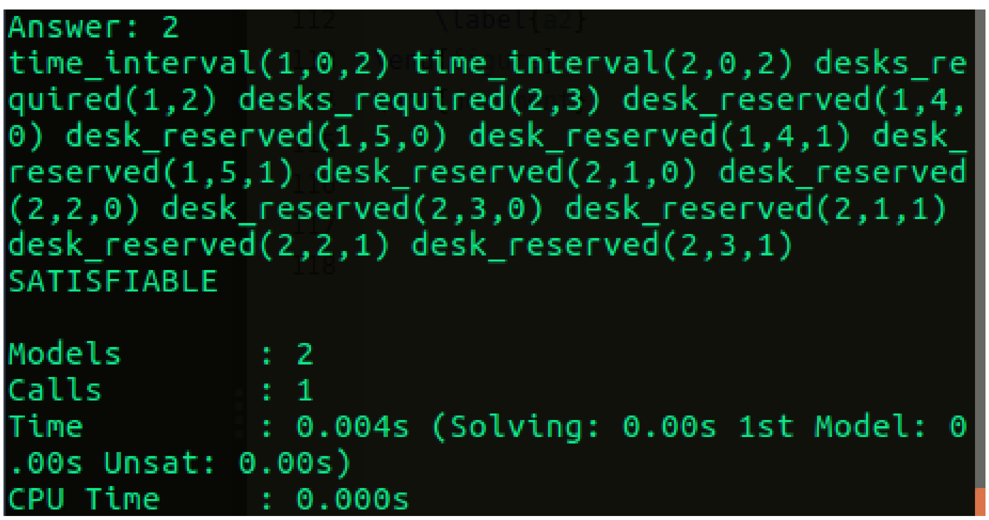
- `n=5`
- `flight(1..2)`
- `time_interval(1,0,2).`
- `time_interval(2,0,2).`
- `desks_required(1,2).`
- `desks_required(2,3).`

There have been found 2 possible solutions - see Figure 10 and 11.



```
(base) eddybudge@laptop:~$ gringo -W no-operation-
undefined -c n=5 aerei1.lp | clasp  0
clasp version 3.1.4
Reading from stdin
Solving...
Answer: 1
time_interval(1,0,2) time_interval(2,0,2) desks_re
quired(1,2) desks_required(2,3) desk_reserved(1,1,
0) desk_reserved(1,2,0) desk_reserved(1,1,1) desk_
reserved(1,2,1) desk_reserved(2,3,0) desk_reserved
(2,4,0) desk_reserved(2,5,0) desk_reserved(2,3,1)
desk_reserved(2,4,1) desk_reserved(2,5,1)
```

Figure 10: First test - first answer.



```
Answer: 2
time_interval(1,0,2) time_interval(2,0,2) desks_re
quired(1,2) desks_required(2,3) desk_reserved(1,4,
0) desk_reserved(1,5,0) desk_reserved(1,4,1) desk_
reserved(1,5,1) desk_reserved(2,1,0) desk_reserved
(2,2,0) desk_reserved(2,3,0) desk_reserved(2,1,1)
desk_reserved(2,2,1) desk_reserved(2,3,1)
SATISFIABLE

Models       : 2
Calls        : 1
Time         : 0.004s (Solving: 0.00s 1st Model: 0
.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Figure 11: First test - the second answer.

If we change now the number of desks present at the airport - from 5 it becomes 4 -, we will see that the problem cannot be solved (Figure 12).

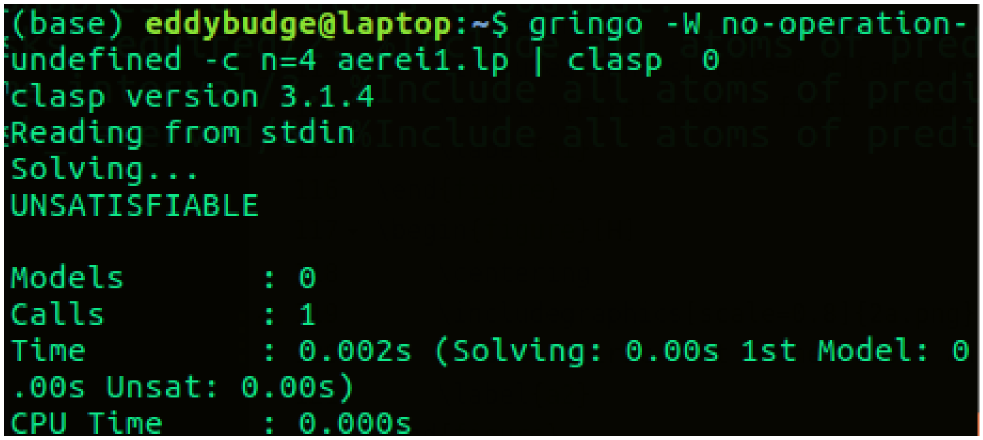
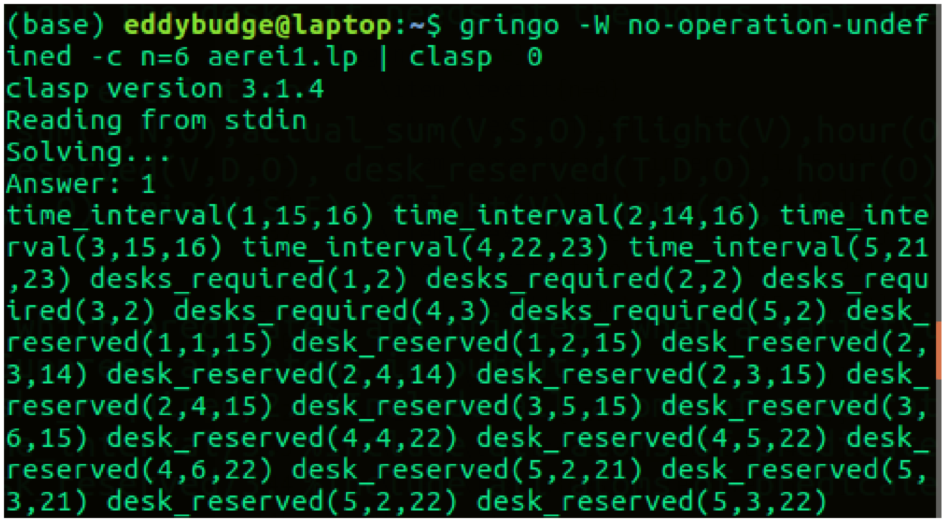


Figure 12: First test - trying with 4 desks, instead of five.

Second test. We have five scheduled flights: 3 of them should have desks reserved from 2 or 3 P.M. to 4 P.M., while the other 2 should have desks reserved from 21 or 22 P.M to 23 P.M. The instance of the problem is the following:

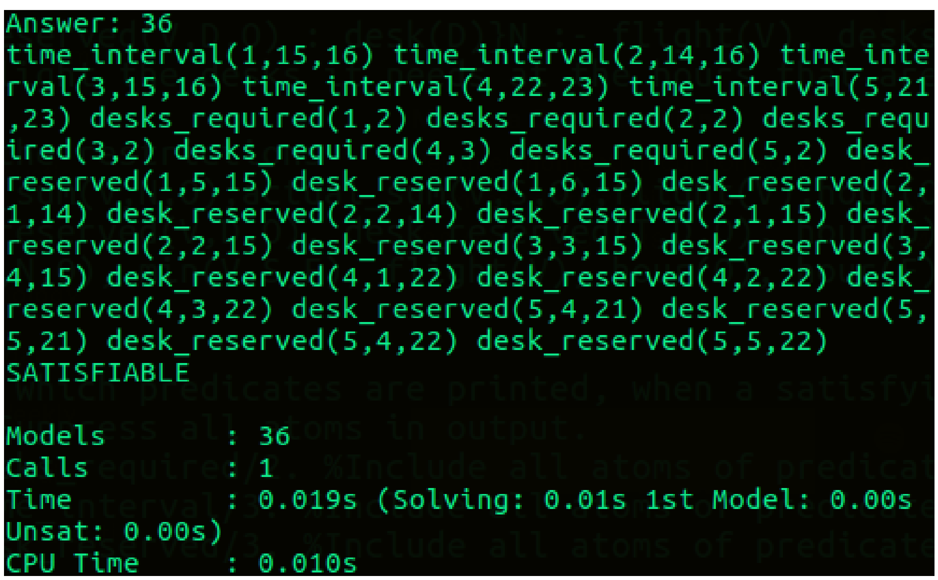
- `n=6`
- `flight(1..5)`
- `time_interval(1,15,16).`
- `time_interval(2,14,16).`
- `time_interval(3,15,16).`
- `time_interval(4,22,23).`
- `time_interval(5,21,23).`
- `desks_required(1,2).`
- `desks_required(2,2).`
- `desks_required(3,2).`
- `desks_required(4,3).`
- `desks_required(5,2).`

There have been found 36 possible solutions - see Figure 14 and 13.



```
(base) eddybudge@laptop:~$ gringo -W no-operation-undef
ined -c n=6 aerei1.lp | clasp  0
clasp version 3.1.4
Reading from stdin
Solving...
Answer: 1
time_interval(1,15,16) time_interval(2,14,16) time_inte
rval(3,15,16) time_interval(4,22,23) time_interval(5,21
,23) desks_required(1,2) desks_required(2,2) desks_requ
ired(3,2) desks_required(4,3) desks_required(5,2) desk_
reserved(1,1,15) desk_reserved(1,2,15) desk_reserved(2,
3,14) desk_reserved(2,4,14) desk_reserved(2,3,15) desk_
reserved(2,4,15) desk_reserved(3,5,15) desk_reserved(3,
6,15) desk_reserved(4,4,22) desk_reserved(4,5,22) desk_
reserved(4,6,22) desk_reserved(5,2,21) desk_reserved(5,
3,21) desk_reserved(5,2,22) desk_reserved(5,3,22)
```

Figure 13: Second test - the first answer .



```
Answer: 36
time_interval(1,15,16) time_interval(2,14,16) time_inte
rval(3,15,16) time_interval(4,22,23) time_interval(5,21
,23) desks_required(1,2) desks_required(2,2) desks_requ
ired(3,2) desks_required(4,3) desks_required(5,2) desk_
reserved(1,5,15) desk_reserved(1,6,15) desk_reserved(2,
1,14) desk_reserved(2,2,14) desk_reserved(2,1,15) desk_
reserved(2,2,15) desk_reserved(3,3,15) desk_reserved(3,
4,15) desk_reserved(4,1,22) desk_reserved(4,2,22) desk_
reserved(4,3,22) desk_reserved(5,4,21) desk_reserved(5,
5,21) desk_reserved(5,4,22) desk_reserved(5,5,22)
SATISFIABLE

Models       : 36
Calls        : 1
Time         : 0.019s (Solving: 0.01s 1st Model: 0.00s
Unsat: 0.00s)
CPU Time     : 0.010s
```

Figure 14: Second test - the last answer.