CS4402 P1: Late Binding Solitaire

160004864

9 March 2020

1 Introduction

In this practical we were required to model and solve the game of Late Binding Solitaire, a variant of Solitaire which is a family of single-player card games. The model is formulated in Essence Prime[1] and Savilerow[2] is then used to translate it into the language of a constraint solver such as Minion[3]. Following that, we were required to evaluate the performance of our model on 25 instances (19 solvable and 6 unsolvable), using metrics such as the `SolverNodes`, `SolverSolveTime` and `Savile Row TotalTime`. Furthermore, we were require to use an instance generator to produce more problem instances and explore the lengths of instance (where length n is the number of cards) to which our model would scale.

2 Variables and Domains

In this section I will be describing the way in which I chose the variables and domains to represent the problem.

2.1 Constants

- STEPS: this constant is a domain of integer in the range $1..n-1$ where n is the number of cards in the instance. This represents the number of steps required to reach the goal state from the initial state. This is $n-1$ as each step or move, a card is moved and placed on top of another one, thus reducing the number of piles by one and the goal state is one in which only one pile remains.
- CARDS: this constant is a domain of integer in the range of 0..51 representing the cards in a deck of cards.
- STEPSNO: this constant holds the number of steps required to reach the final solution from the initial solution. This will always be equal to $n-1$.

2.2 Variables

- states: this is a two-dimensional matrix of decision variables of size $n * n$, with each decision variable being in the domain of the range $-1..51$. This is the domain as describe above for the CARDS constant with the addition of the -1 which is reserved to represent 'empty' spaces, which occur when a pile is moved on top of another one. Each row represents the piles of cards in each state, with the first row being the initial state, and each subsequent row is the state of the cards after each move. Essentially, the ith row of this matrix represents the state of the cards resulting after the $i - 1th$ move.

- actionFrom: this is a one-dimensional matrix of decision variables of size $n-1$ where each decision variable in this matrix lies in the range $1..n$. The ith decision variable in this matrix represents the index of the card being moved at the ith step. Since $n - 1$ steps are required to find a solution, this matrix will have a length equal to that value.

- actionTo: this is a one-dimensional matrix of decision variables of size $n - 1$ where each decision variable in this matrix lies in the range $1..n$. It serves a similar purpose to the `actionFrom` matrix, with the difference that it stores the destination index of the pile being moved i.e. to which index the pile being moved is placed.

- actionValue: this is a one-dimensional matrix of decision variables of size $n - 1$ where each decision variable in this matrix lies in the range $0..51$. Essentially it keeps the value of the top-most card of the pile being moved at each step. While it isn't used to solve the problem, as one can access this value by looking at the states matrix to find this value, it is useful while analysing solutions to verify that the right pile is being moved at each step.

3 Constraints

In this section I will be describing the set of constraints I've added to my model. These are described in the same order as they appear in the `lateBindingSolitaire.eprime` file.

- Starting state: I've included a constraint which maps the cards as given in the `.param` file to the first row of the states matrix, representing the initial state, before any moves have taken place.

- Move to itself: I've included a constraint which restricts the index of the pile being moved at each step from being equal to destination's index. In essence, it states that a pile can't be moved on itself.

- Setting -1 after each move: I've included a constraint which explicitly states that the ith row of the state matrix should have $i - 1$ number of

-1s at the end. This represents the number of 'empty spaces' after each move.

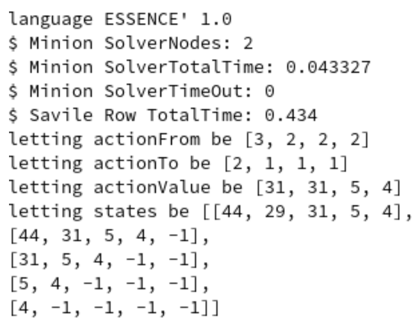
- Unchanged cards after each move: This constraint states that cards that are at an index smaller than the index of the card being moved and are not at the destination index, remain unchanged.
- Shifting cards after each move: This constraint in combination with the 2 constraints above make sure that the new state occurring after each step is correctly modelled. This particular constraint states that all the cards at an index larger than the index of the card being moved are shifted one to the left, leaving the -1s to their right.
- Can't move a -1: This constraint explicitly states that the value of the card being moved has to be greater or equal to 0. This is done to prevent the mistake of the model thinking that -1 is indeed a valid value for a card.
- All steps must be a jump of 1 or 3 to the left: This constraint states that the *ith* value of the actionFrom matrix has to be larger than the *ith* value of the actionTo matrix by 1 or 3, indicating that a card may only be moved one to the left or three to the left (skip 2).
- Legal moves are that of the same suit or same value: This constraint states that a legal move is either one in which the destination card value is the same as the source card (the results of the MOD 13 operation on the two cards are equal) or the two cards belong in the same suit (the results of the DIV 13 operation on the two cards are equal).

4 Example Solutions

In this section we will be reviewing the solution the model produces for three specific instances in order to demonstrate that it is correct. Specifically we will be looking closely to the solutions given in three instances.

4.1 LBS5_0

The `LBS5_0.param.solution` file is as follows:



```
language ESSENCE' 1.0
$ Minion SolverNodes: 2
$ Minion SolverTotalTime: 0.043327
$ Minion SolverTimeOut: 0
$ Savile Row TotalTime: 0.434
letting actionFrom be [3, 2, 2, 2]
letting actionTo be [2, 1, 1, 1]
letting actionValue be [31, 31, 5, 4]
letting states be [[44, 29, 31, 5, 4],
[44, 31, 5, 4, -1],
[31, 5, 4, -1, -1],
[5, 4, -1, -1, -1],
[4, -1, -1, -1, -1]]
```

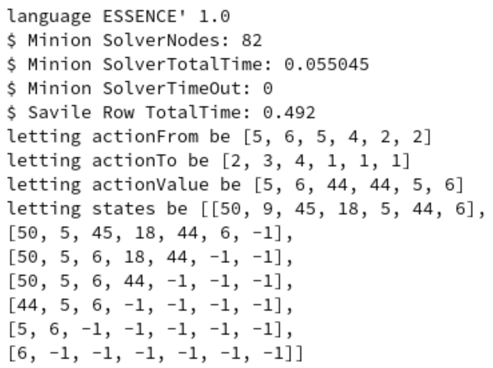
The representation of this solution in terms of cards with values and suits is:

1. $6\spadesuit, 4\clubsuit, 6\clubsuit, 6\heartsuit, 5\heartsuit$ - move $6\clubsuit$ to $4\clubsuit$
2. $6\spadesuit, 6\clubsuit, 6\heartsuit, 5\heartsuit$ - move $6\clubsuit$ to $6\spadesuit$
3. $6\clubsuit, 6\heartsuit, 5\heartsuit$ - move $6\heartsuit$ to $6\clubsuit$
4. $6\heartsuit, 5\heartsuit$ - move $5\heartsuit$ to $6\heartsuit$
5. $5\heartsuit$

As we can see, the solution is correct as all the moves are legal, and there is only one pile remaining after the final step.

4.2 LBS7_47

The `LBS7_47.param.solution` file is as follows:



```
language ESSENCE' 1.0
$ Minion SolverNodes: 82
$ Minion SolverTotalTime: 0.055045
$ Minion SolverTimeOut: 0
$ Savile Row TotalTime: 0.492
letting actionFrom be [5, 6, 5, 4, 2, 2]
letting actionTo be [2, 3, 4, 1, 1, 1]
letting actionValue be [5, 6, 44, 44, 5, 6]
letting states be [[50, 9, 45, 18, 5, 44, 6],
[50, 5, 45, 18, 44, 6, -1],
[50, 5, 6, 18, 44, -1, -1],
[50, 5, 6, 44, -1, -1, -1],
[44, 5, 6, -1, -1, -1, -1],
[5, 6, -1, -1, -1, -1, -1],
[6, -1, -1, -1, -1, -1, -1]]
```

The representation of this solution in terms of cards with values and suits it:

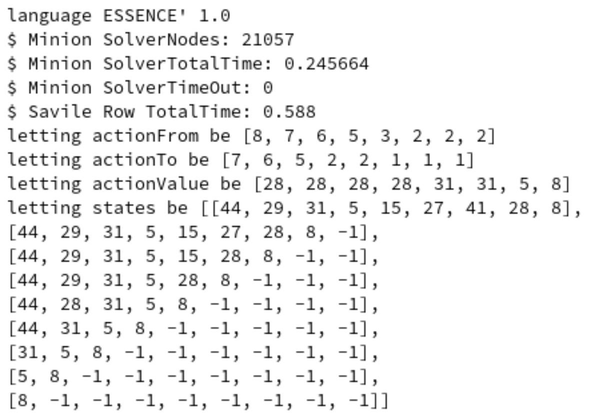
1. $Q\spadesuit, 10\heartsuit, 7\spadesuit, 6\diamondsuit, 6\heartsuit, 6\spadesuit, 7\heartsuit$ - move $6\heartsuit$ to $10\heartsuit$
2. $Q\spadesuit, 6\heartsuit, 7\spadesuit, 6\diamondsuit, 6\spadesuit, 7\heartsuit$ - move $7\heartsuit$ to $7\spadesuit$
3. $Q\spadesuit, 6\heartsuit, 7\heartsuit, 6\diamondsuit, 6\spadesuit$ - move $6\spadesuit$ to $6\diamondsuit$

4. $Q\spadesuit, 6\heartsuit, 7\heartsuit, 6\spadesuit$ - move $6\spadesuit$ to $Q\spadesuit$
5. $6\spadesuit, 6\heartsuit, 7\heartsuit$ - move $6\heartsuit$ to $6\spadesuit$
6. $6\heartsuit, 7\heartsuit$ - move $7\heartsuit$ to $6\heartsuit$
7. $7\heartsuit$

Once again, the solution is correct as all the moves are legal, and there is only one pile remaining after the final step.

4.3 LBS9_0

The `LBS9_0.param.solution` file is as follows:



```
language ESSENCE' 1.0
$ Minion SolverNodes: 21057
$ Minion SolverTotalTime: 0.245664
$ Minion SolverTimeOut: 0
$ Savile Row TotalTime: 0.588
letting actionFrom be [8, 7, 6, 5, 3, 2, 2, 2]
letting actionTo be [7, 6, 5, 2, 2, 1, 1, 1]
letting actionValue be [28, 28, 28, 28, 31, 31, 5, 8]
letting states be [[44, 29, 31, 5, 15, 27, 41, 28, 8],
[44, 29, 31, 5, 15, 27, 28, 8, -1],
[44, 29, 31, 5, 15, 28, 8, -1, -1],
[44, 29, 31, 5, 28, 8, -1, -1, -1],
[44, 28, 31, 5, 8, -1, -1, -1, -1],
[44, 31, 5, 8, -1, -1, -1, -1, -1],
[31, 5, 8, -1, -1, -1, -1, -1, -1],
[5, 8, -1, -1, -1, -1, -1, -1, -1],
[8, -1, -1, -1, -1, -1, -1, -1, -1]]
```

The representation of this solution in terms of cards with values and suits it:

1. $6\spadesuit, 4\clubsuit, 6\clubsuit, 6\heartsuit, 3\diamondsuit, 2\clubsuit, 3\spadesuit, 3\clubsuit, 9\heartsuit$ - move $3\clubsuit$ to $3\spadesuit$
2. $6\spadesuit, 4\clubsuit, 6\clubsuit, 6\heartsuit, 3\diamondsuit, 2\clubsuit, 3\clubsuit, 9\heartsuit$ - move $3\clubsuit$ to $2\clubsuit$
3. $6\spadesuit, 4\clubsuit, 6\clubsuit, 6\heartsuit, 3\diamondsuit, 3\clubsuit, 9\heartsuit$ - move $3\clubsuit$ to $3\diamondsuit$
4. $6\spadesuit, 4\clubsuit, 6\clubsuit, 6\heartsuit, 3\clubsuit, 9\heartsuit$ - move $3\clubsuit$ to $4\clubsuit$
5. $6\spadesuit, 3\clubsuit, 6\clubsuit, 6\heartsuit, 9\heartsuit$ - move $6\clubsuit$ to $3\clubsuit$
6. $6\spadesuit, 6\clubsuit, 6\heartsuit, 9\heartsuit$ - move $6\clubsuit$ to $6\spadesuit$
7. $6\clubsuit, 6\heartsuit, 9\heartsuit$ - move $6\heartsuit$ to $6\clubsuit$
8. $6\heartsuit, 9\heartsuit$ - move $9\heartsuit$ to $6\heartsuit$
9. $9\heartsuit$

As seen above, the model is producing the correct solution for this problem instance as well.

5 Empirical evaluation

In this section I will be describing my experimental setup, record and analyse the output of our model for a set of instances.

5.1 Scripts

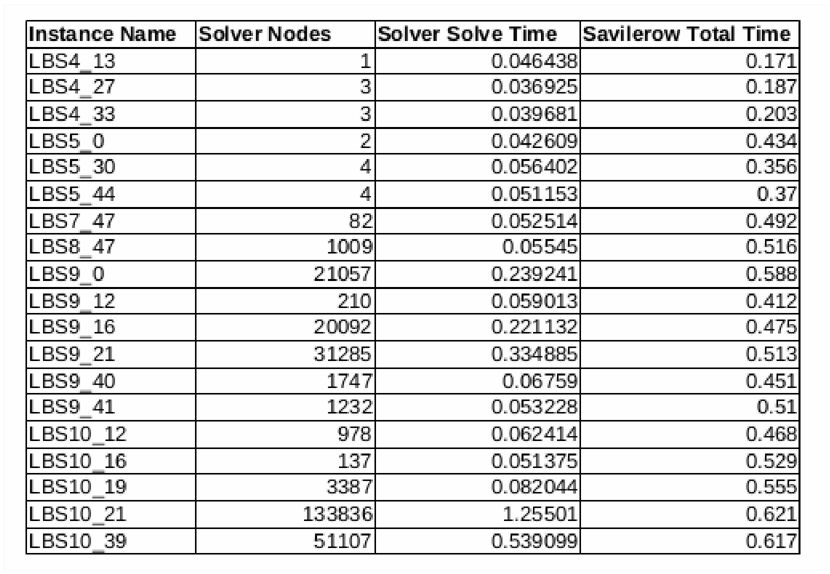
In order to automate the process of running Savilerow[2] for each instance, I've created a number of scripts.

- `run_solvable_linux.sh`: Runs all the provided, solvable instances of the problem.
- `run_solvable.sh`: Same as above but for macOS.
- `run_unsolvable_linux.sh`: Runs all the provided, unsolvable instances of the problem.
- `run_unsolvable.sh`: Same as above but for macOS.

Note: in order to run these scripts you may need to give execute privileges to them using `chmod +x <script_name>`

5.2 Solvable instances

The following is the empirical data collected on the solvable instances:

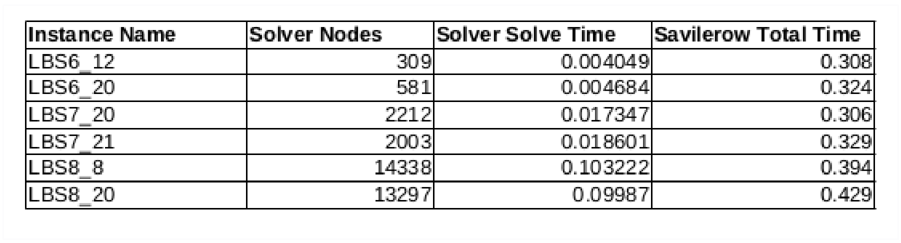


Instance Name	Solver Nodes	Solver Solve Time	Savilerow Total Time
LBS4_13	1	0.046438	0.171
LBS4_27	3	0.036925	0.187
LBS4_33	3	0.039681	0.203
LBS5_0	2	0.042609	0.434
LBS5_30	4	0.056402	0.356
LBS5_44	4	0.051153	0.37
LBS7_47	82	0.052514	0.492
LBS8_47	1009	0.05545	0.516
LBS9_0	21057	0.239241	0.588
LBS9_12	210	0.059013	0.412
LBS9_16	20092	0.221132	0.475
LBS9_21	31285	0.334885	0.513
LBS9_40	1747	0.06759	0.451
LBS9_41	1232	0.053228	0.51
LBS10_12	978	0.062414	0.468
LBS10_16	137	0.051375	0.529
LBS10_19	3387	0.082044	0.555
LBS10_21	133836	1.25501	0.621
LBS10_39	51107	0.539099	0.617

Reading the provided table, one can easily see that there's a positive correlation between the instance length and the values recorded for the `Solver Nodes`, `Solver Solve Time` and `Savilerow Total Time`. However, we can see that there are dramatic differences between instances of the same length. For example, compare `LBS9_0` with `LBS9_12`. The former uses about 100 times as many solver nodes and the solver requires almost 5 times as much time to solve it. This indicates that indeed, instances of the same length can be of dramatically different difficulties. An interesting point is that the last column, `Savilerow Total Time` scales with the instance length and it doesn't seem to be affected greatly by the relative difficulty of the instance at each length n. This makes sense as Savilerow translates the Essence Prime model into a language understandable by whatever solver is used as a backend and is not concerned with the actual solving process.

5.3 Unsolvable Instances

The following is the empirical data collected on the unsolvable instances:



Instance Name	Solver Nodes	Solver Solve Time	Savilerow Total Time
LBS6_12	309	0.004049	0.308
LBS6_20	581	0.004684	0.324
LBS7_20	2212	0.017347	0.306
LBS7_21	2003	0.018601	0.329
LBS8_8	14338	0.103222	0.394
LBS8_20	13297	0.09987	0.429

Upon examination, one can immediately notice that the third column, `Savilerow Total Time` scales with the instance length n similarly to the way it does for the solvable instances. This isn't surprising as whether the instance is solvable or not, Savilerow will carry out exactly the same process. Furthermore, `Solver Nodes` also scales very quickly as n increases and so does the `Solver Solve Time` (though not as aggressively). However, there doesn't seem to be much variation between the relative difficulty of instances of the same length. This is expected as since they are all unsolvable, they all belong in the same difficulty category.

5.4 Generated Instances

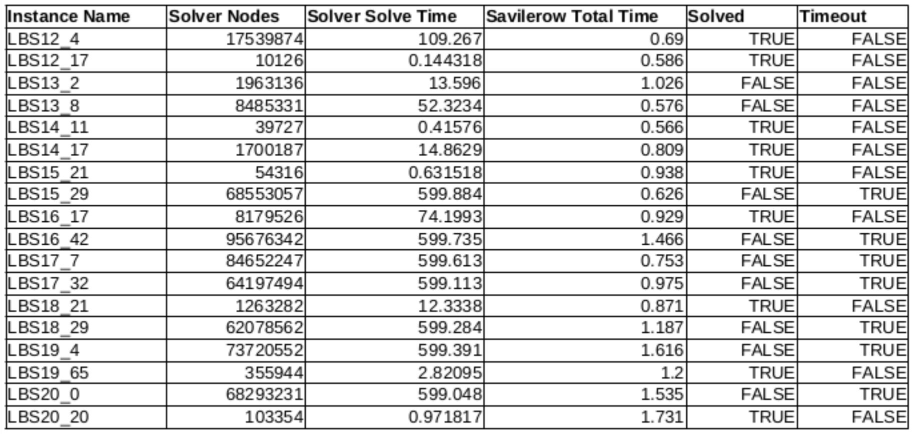
In order to collect more empirical data, I used the supplied instance generator to create new instances. I then used Savilerow to solve them and following that I collected the data produced for analysis. I have provided a number of scripts for both of these sub-tasks:

- `generate_instances.sh`: This script uses the generator to produce instances of the problem. It produces two instances for each length n between (and including) 12 and 20 using an arbitrary seed. Note that we don't know whether the generator will produce a solvable or unsolvable instance.

- `run_generated_linux.sh`: This script runs Savilerow for each of the generated instances.

- `run_generated.sh`: Same as above but for macOS.

Note: in order to run these scripts you may need to give execute privileges to them using `chmod +x <script_name>`
Since the purpose of this part is to see how the model scales with larger instances of the problem, it is necessary to introduce a reasonable timeout value. In this experiment I've used the value of 10 minutes (600 seconds) as the time limit for the solver, using the `-solver-options "-cpulimit 600"` parameter. The following table describes the results of this experiment:



Instance Name	Solver Nodes	Solver Solve Time	Savilerow Total Time	Solved	Timeout
LBS12_4	17539874	109.267	0.69	TRUE	FALSE
LBS12_17	10126	0.144318	0.586	TRUE	FALSE
LBS13_2	1963136	13.596	1.026	FALSE	FALSE
LBS13_8	8485331	52.3234	0.576	FALSE	FALSE
LBS14_11	39727	0.41576	0.566	TRUE	FALSE
LBS14_17	1700187	14.8629	0.809	TRUE	FALSE
LBS15_21	54316	0.631518	0.938	TRUE	FALSE
LBS15_29	68553057	599.884	0.626	FALSE	TRUE
LBS16_17	8179526	74.1993	0.929	TRUE	FALSE
LBS16_42	95676342	599.735	1.466	FALSE	TRUE
LBS17_7	84652247	599.613	0.753	FALSE	TRUE
LBS17_32	64197494	599.113	0.975	FALSE	TRUE
LBS18_21	1263282	12.3338	0.871	TRUE	FALSE
LBS18_29	62078562	599.284	1.187	FALSE	TRUE
LBS19_4	73720552	599.391	1.616	FALSE	TRUE
LBS19_65	355944	2.82095	1.2	TRUE	FALSE
LBS20_0	68293231	599.048	1.535	FALSE	TRUE
LBS20_20	103354	0.971817	1.731	TRUE	FALSE

By looking at the table above, one can make several observations:

1. The solver performs well for problem instances of length n of up to 14, as the first instance it times out for is instance `LBS15_29`.

2. For solvable instances, the `Solver Solve Time` values are quite low. Notice that instance `LBS20_20` took the constraint solver less than a second to solve.

3. The values for `Solver Nodes` seem to depend more on whether the instance is solvable rather than the length n of the instance.

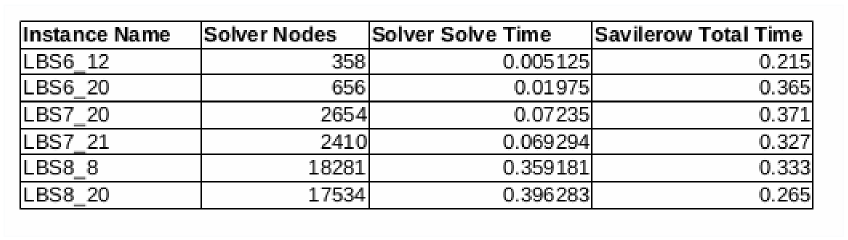
4. The value for `Savilerow Total Time` seems to scale with the length n of the problem instance.

The approach used to collect the data above is limited in that for instances for which the solver timed out, we can't know whether the solver would either eventually find a solution or find that no solution is possible, if given enough time.

6 Extensions

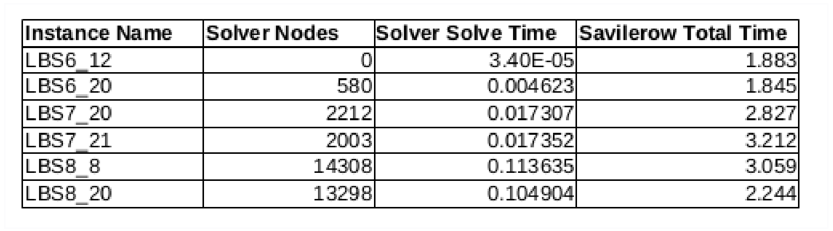
6.1 Optimisations

Upon reviewing the manual provided by Saliverow[2], I found that Savilerow performs optimisation by default. Namely is performs the `-O2` optimisation which does the `-O1` optimisation, in which 'variables that are equal are unified, and a form of common subexpression elimination is applied'. Furthermore it performs 'filtering of variable domains and aggregation'. In order to provide a benchmark showing how the `-O2` optimisation affects performance, I use the `-O0` optimisation which turns of all optimisations. I used the unsolvable instances for this as the lack of variance in difficulty between instances of the same length, will give a better and more consistent view of the optimisation achieved. The following described the collected data:



Instance Name	Solver Nodes	Solver Solve Time	Savilerow Total Time
LBS6_12	358	0.005125	0.215
LBS6_20	656	0.01975	0.365
LBS7_20	2654	0.07235	0.371
LBS7_21	2410	0.069294	0.327
LBS8_8	18281	0.359181	0.333
LBS8_20	17534	0.396283	0.265

Upon examination and comparing it with the performance of the model when the default `-O2` optimisation was applied, we can see some interesting differences. Firstly, values for the solver nodes and the solver solve time are higher in all instances. This shows that the optimisation applied has a significant effect on the performance. Consider the values for the `Solver Solve Time` for instance `LBS8_8`. When the optimisation is applied, the problem is solved about 3.5 times faster. Furthermore, when no optimisation is applied, the value for `Savilerow Total Time` is lower. This is not surprising as with the optimisations switched off, Savilerow does not spend time any carrying out optimisations when translating the model into the language understandable by the constraint solver.
The manual describes one more optimisation option `-O3` which in addition to `-O2` it 'enables tabulation and associative-commutative common subexpression elimination'. In order to see how this affects the performance, I collected the data produced when the model attempts to solve the unsolvable instances provided. The following is the result:



Instance Name	Solver Nodes	Solver Solve Time	Savilerow Total Time
LBS6_12	0	3.40E-05	1.883
LBS6_20	580	0.004623	1.845
LBS7_20	2212	0.017307	2.827
LBS7_21	2003	0.017352	3.212
LBS8_8	14308	0.113635	3.059
LBS8_20	13298	0.104904	2.244

While the model performs better in this optimisation when compared to when all the optimisations were turned off, there doesn't seem to be much difference with the results for the `-O2` optimisation. Interestingly enough, they have very similar values for the `Solver Nodes` and `Solver Solve Time`, certainly due to the fact that they both carry out the `-O2` optimisation. However, the values for the `Savilerow Total Time` are much higher under the `-O3` optimisation, by about a factor of 10. This is an indication that this particular problem does not benefit from the optimisations offered by `-O3`, an optimisation however which has a significant negative impact on the time Savilerow needs to execute.

7 Evaluation & Conclusion

7.1 Evaluation

After completing the practical, I reviewed my work in order to assess what I could have been done better. If I had more time, I'd improve the model I defined in `lateBindingSolitaire.eprime`. For example, I could incorporate many constraints into fewer, larger ones. Though this may sacrifice some program modularity, it could've made the program more efficient, scaling better with instances of larger length. Furthermore, I noticed that one of my constraints, specifically the one that states that a card cannot be moved to the position it's already in, is redundant as I later define that the indices of the source and destination cards, have to have a difference of either 1 or 3. Removing redundant constraints could have also further improved my program's efficiency. Furthermore, if I had more time I'd collect empirical data more rigorously as due to the limited time I had available, I could not afford setting the solver's time limit to a value higher than 10 minutes.

7.2 Conclusion

Overall, I'm very pleased with the result of my work as I've managed to describe a model in Essence Prime[1] that can be used to solve instances of the Late Binding Solitaire game, as required by the specification. What I found most difficult was using the constraint programming (CP) paradigm for the first time, a paradigm which is very different from the imperative programming paradigm that I'm used to following. However, this was also the most interesting part of the endeavour as I can see how powerful and useful this paradigm can be for solving a vast set of real-life problems. In conclusion, this practical was very successful in achieving its learning outcomes as it provided me with experience in modelling CP problems and it also gave me some insight into the different tools that may be used to solve such problems.

References

[1] EssencePrime. (2020). Retrieved 8 March 2020, from http://www.csplib.org/Languages/EssencePrime/

[2] Savile Row constraint modelling assistant. (2020). Retrieved 8 March 2020, from https://savilerow.cs.st-andrews.ac.uk/

[3] (2020). Retrieved 8 March 2020, from https://constraintmodelling.org/minion/