CSP Project for Game of Life
CPP- MAI – UPC

José A. Magaña Mesa

January 2014

Introduction

This report corresponds to the implementation of the Conway's Game of Life using Constraint Satisfaction Programming techniques. The objective is to find stable configurations starting from a given setup and that the stable configuration maximizes the number of Alive Cells in the board.

The Gecode libraries have been used for this using IntVars for the model defined. As an extension, some of the different options in the library for Variable and Value selection have been evaluated to see the difference in performance. Also, a set of redundant constraints have been defined that are included in the model based on command line settings.

Implementation

Constraints

For a direct implementation of the Conway's Game of Life rules, the following constraints must be added. On them, the matrix minimodel has been used so that manipulation of cells becomes straightforward.

As it was the case for the SAT problem, the board has been widened in order to avoid having to care about the position of the cell when looking for the neighbor cells.

For each Alive Cell in the original configuration, we assign the value:

```
rel(*this, A(x,y), IRT_EQ, 1);            // the cell is alive
```

For all the cells (i,j) in the original board (not extended), the neighborhood is calculated in the variable s, that adds the number of Alive Cells. The conditions for Alive Cells and Dead Cells are added. Operator >> implements the conditional in logic (→).

```
s=A(i-1,j-1)+A(i-1,j)+A(i-1,j+1)+A(i,j-1)+A(i,j+1)+A(i+1,j-1)+A(i+1,j)+A(i+1,j+1);
rel(*this, (A(i,j)==0) >> (s!=3));
rel(*this, (A(i,j)==1) >> (s==2) || (s==3));
```

Additional conditions are added on the side cells, to prevent propagation on a potentially infinite board:

```
// horizontal
rel(*this, A(1,i)+A(1,i+1)+A(1,i+2)<3);
rel(*this, A(n,i)+A(n,i+1)+A(n,i+2)<3);
// vertical
rel(*this, A(i,1)+A(i+1,1)+A(i+2,1)<3);
rel(*this, A(i,n)+A(i+1,n)+A(i+2,n)<3);
```

The margin of the extended board is set to zero (horizontally and vertically):

```
rel(*this, A(i,0), IRT_EQ, 0);
rel(*this, A(i,n+1), IRT_EQ, 0);

rel(*this, A(0,i), IRT_EQ, 0);
rel(*this, A(n+1,i), IRT_EQ, 0);
```

Cost function

As the problem has as objective maximizing the number of Alive Cells, the cost function is directly the number of Alive Cells, that is, the sum of the values of the board. The minimum of the function is the number of Alive Cells in the initial configuration.

```
upperBound=n*n;
Cost=IntVar(*this,nAlive,upperBound);    // cost: minimum is the number of alive cells
                                         // maximum is the number of cells
rel(*this,Cost==sum(X));
```

The maximum can also be bounded based on the results of [1] and set to half the size of the board plus a delta. Nevertheless, this optimization has not been incorporated to the solution and the upper bound has been set to the number of cells in the board.

Using this optimization would highly speed up the search process as it would prune plenty of options that are not feasible solutions but that the algorithm must discard itself.

Evaluation

In order to efficiently calculate all the mentioned cases, intensive use of the parameterization of the library has been used.

solutions(0) → all the solutions must be explored

Dynamically, the Variable and Value selection are set:

setVar(----)

setVal(----)

In addition, an extra parameter has been set to decide if the redundant constraints must be incorporated to the model:

```
opt.model(0);                        // by default: no redundant
opt.model(1,"redundant");
opt.model(0,"noredundant");
```

The following command line parameters have been used:

-print-last 1 → only the last (best) solution is printed

-file-sol solsl15_2.txt → the solution is printed to a defined file

-mode solution →required to obtain the solution on the file

-model redundant /noredundant →the model may or not incorporate redundant constraints

sl15_2.txt → the input file

Some incompatibilities between the settings have limited the use of certain functionalities. It is not possible for example to export the solutions for each setting set to the same file obtaining a list. Each execution deletes the existing content on the file. The same happens if we try to obtain the statistics.

If the experiment wants to be repeated for a number of iterations so that we can extract averages to compensate for any stochasticity in the algorithm, the setting –mode time must be used but this does not produce the solution.

How to execute

To ease validation of the model, a .bat file (GoL.bat) has been provided that takes as parameters the input file: <input.txt>

The output will have the solution in a file named: sol<input.txt>

The solution includes in the last line the value of the cost function for the configuration.

The batch file can be easily converted to a script in any Linux environment where it needs to be executed.

Variable and Value Selection impact

In order to evaluate how variable and value selection influence the algorithm the following predefined methods have been selected. Some of them may not make sense for cases, as this one, where the domain is {0,1}. In both cases, the test has been limited to options that do not require non-trivial parameters. In the case of the Random selection, it has been initialized with a 0 seed for repeatability.

The execution times expressed in milliseconds for the sl15_2.txt example are shown in the table below. The results correspond to a single execution what is not a limitations as the methods are most of them, and except for ties, deterministic.

The results are colored with red for the higher (worst) execution times and green for the lower execution times.

	INT_VAR_SIZE_MAX	INT_VAR_ACTIVITY_MAX	INT_VAR_ACTIVITY_SIZE_MAX	INT_VAR_SIZE_MIN	INT_VAR_ACTIVITY_MIN	INT_VAR_ACTIVITY_SIZE_MIN	INT_VAR_RND
INT_VAL_MAX	11	34	33	11	14	15	208
INT_VALUES_MAX	11	34	33	11	15	15	244
INT_VAL_RANGE_MAX	11	33	34	11	14	14	135
INT_VAL_SPLIT_MAX	11	33	34	10	14	14	212
INT_VAL_MIN	11	98	99	11	12	12	178
INT_VALUES_MIN	11	99	100	11	11	11	192
INT_VAL_RANGE_MIN	11	113	97	11	12	11	103
INT_VAL_SPLIT_MIN	11	99	99	10	11	12	193
INT_VAL_RND	11	205	120	10	11	13	151

From the table we can observe that one of the axes dominates the other in the sense that the values follow the same order if we fix one of the values of the non-dominant axis. The dominant axis is the Variable selection. Once we fix that, the Value selection has little influence in the result (except for the Random case).

	INT_VAR_SIZE_MAX	INT_VAR_ACTIVITY_MAX	INT_VAR_ACTIVITY_SIZE_MAX	INT_VAR_SIZE_MIN	INT_VAR_ACTIVITY_MIN	INT_VAR_ACTIVITY_SIZE_MIN	INT_VAR_RND	Average	STDEV
INT_VAL_MAX	11	34	33	11	14	15	208	46.57	66.54
INT_VALUES_MAX	11	34	33	11	15	15	244	51.86	78.97
INT_VAL_RANGE_MAX	11	33	34	11	14	14	135	36.00	41.46
INT_VAL_SPLIT_MAX	11	33	34	10	14	14	212	46.86	68.07
INT_VAL_MIN	11	98	99	11	12	12	178	60.14	61.29
INT_VALUES_MIN	11	99	100	11	11	11	192	62.14	65.59
INT_VAL_RANGE_MIN	11	113	97	11	12	11	103	51.14	46.27
INT_VAL_SPLIT_MIN	11	99	99	10	11	12	193	62.14	65.80
INT_VAL_RND	11	205	120	10	11	13	151	74.43	76.49
Average	11	83.1	72.1	10.7	12.7	13	180	54.59	
STDEV	0	54.1	35.1	0.47	1.49	1.49	40.8		65.34

A basic analysis of the results brings interesting conclusions. The most favorable settings are those that use the Size of the Domain of the Variable (INT_VAR_SIZE_MIN and INT_VAR_SIZE_MAX). The best execution time is for INT_VAR_SIZE_MIN what in this case where the domain has cardinality 2, indicates that first all the cells whose value is defined either in the original configuration or by the application of the constraints are fixed and only after this the rest of cells are calculated.

The fact that the second best option is INT_VAR_SIZE_MAX can be interpreted as that the other criteria are not relevant and following any strategy based on domain size is a winning strategy, especially considering that in this case, the domain size is 2.

Good results, in some case close to the best obtained, are achieved for INT_VAR_ACTIVITY_SIZE_MIN and INT_VAR_ACTIVITY_MIN variables. No detail is provided on Gecode Documentation about how Activity is measured so it is not possible to analyze further for this case.

Regarding the Value Selection Variable, the table shows that for better performance they must be correlated with the Variable Selection. Hence, if the Variable Selection is a ???_MAX the Value Selection must also be a ???_MAX. This is especially evident in the INT_VAR_ACTIVITY_???_MAX cases where the execution time is triplicated.

The fact that the worst values are in the cases where random criteria are used is a clear indicator (despite the low statistical significance of the experiment as it has only been repeated once) that Value and Variable selection are key to achieve good performance. If the results for random where closer to the best it would have been absolutely necessary to repeat several times the results.

Another interesting result is that in most cases the number of solutions found to get to the optimal is 2.

Only in a few cases, the first solution is the optimal, curiously in all cases there is a Random element, and curiously also there is a MIN and a MAX component.

INT_VAR_ACTIVITY_MIN() - INT_VAL_RND(0), time=11 msec

INT_VAR_SIZE_MAX() - INT_VAL_RND(0), time=11 msec

INT_VAR_RND(0) - INT_VALUES_MAX(), time= 244 msec

INT_VAR_RND(0) - INT_VALUES_MIN(), time=192 msec

In other few cases, 3 solutions are required to find an optimal one. Again a Random factor appears:

INT_VAR_RND(0) - INT_VAL_RANGE_MAX(), time=135 msec

INT_VAR_RND(0) - INT_VAL_SPLIT_MAX(),time=151 msec

INT_VAR_RND(0) - INT_VAL_SPLIT_MIN(), time=193 msec

Redundant constraints

In order to try to make the search quicker and in general more efficient, some constraints have been added to the problem.

First, for those cells that are Alive in the initial configuration we can include:

```
r=expr(*this, A(x-1,y-1)+A(x-1,y)+A(x-1,y+1)+
              A(x,y-1)+A(x,y+1)+
              A(x+1,y-1)+A(x+1,y)+A(x+1,y+1));
rel(*this, (r==2) || (r==3) );
```

This is the same constraint that we added for all the cells in the board but knowing that the cell is Alive.

A second set of redundant constraints explodes the reverse conditional constraints:

```
rel(*this, (s>3) >> (A(i,j)==0));
rel(*this, (s==1) >> (A(i,j)==0));
rel(*this, (s==0) >> (A(i,j)==0));
rel(*this, (s==3) >> (A(i,j)==1));
```

For those values of the number of alive cells that only permit one value for the cell the condition is added as constraint.

Results with Redundant Constraints

Adding the redundant constraints and executing for all the configurations, the results in the following table are obtained. The first observation is that the execution is in all cases worse than without the redundant constraints. The best times are obtained for the same parameters.

	INT_VAR_SIZE_MAX	INT_VAR_ACTIVITY_MAX	INT_VAR_ACTIVITY_SIZE_MAX	INT_VAR_SIZE_MIN	INT_VAR_ACTIVITY_MIN	INT_VAR_ACTIVITY_SIZE_MIN	INT_VAR_RND
INT_VAL_MAX	18	52	52	17	23	23	323
INT_VALUES_MAX	17	53	52	17	23	23	377
INT_VAL_RANGE_MAX	17	52	52	18	22	23	206
INT_VAL_SPLIT_MAX	18	51	53	17	17	22	326
INT_VAL_MIN	18	146	148	18	17	17	275
INT_VALUES_MIN	18	147	148	18	18	17	298
INT_VAL_RANGE_MIN	18	148	147	18	18	18	156
INT_VAL_SPLIT_MIN	18	147	148	18	18	18	296
INT_VAL_RND	17	310	180	17	18	20	228

By adding the redundant constraints, the number of propagators in the model is increased from 1575(or 1574) to 3900(or 3899) depending on the settings. Surprisingly (at least for novice users of Gecode) the number of propagators does not depend only on the constraints added.

An analysis of additional performance values is required. For some of the Variable Selection policies (INT_VAR_SIZE_MIN, INT_VAR_ACTIVITY_SIZE_MIN , INT_VAR_ACTIVITY_SIZE_MAX and INT_VAR_RND) an extended analysis is performed.

The first observation is that execution time is proportional to Propagations and Nodes. The table for Propagations is shown:

	NO REDUNDANT							REDUNDANT						
	INT_VAR_SIZE_MAX	INT_VAR_ACTIVITY_MAX	INT_VAR_ACTIVITY_SIZE_MAX	INT_VAR_SIZE_MIN	INT_VAR_ACTIVITY_MIN	INT_VAR_ACTIVITY_SIZE_MIN	INT_VAR_RND	INT_VAR_SIZE_MAX	INT_VAR_ACTIVITY_MAX	INT_VAR_ACTIVITY_SIZE_MAX	INT_VAR_SIZE_MIN	INT_VAR_ACTIVITY_MIN	INT_VAR_ACTIVITY_SIZE_MIN	INT_VAR_RND
INT_VAL_MAX	65841	207012	207020	65830	90190	90209	1187080	113563	355441	356908	113563	154149	154182	1987603
INT_VALUES_MAX	65871	206987	207138	66003	90189	90214	1306573	113631	356915	355806	113555	154110	154182	2189716
INT_VAL_RANGE_MAX	65817	207013	207125	65802	90200	90211	798468	113868	356925	355796	113555	153885	154191	1335020
INT_VAL_SPLIT_MAX	66002	207013	207020	65802	90156	90176	1201893	113712	356925	355806	113563	153876	154191	2006113
INT_VAL_MIN	70155	582727	583587	70198	67839	67831	1003844	121137	961395	957993	120921	115634	115796	1667044
INT_VALUES_MIN	70155	583122	583671	70127	67871	67951	1013896	121137	961395	957993	120921	115610	115529	1691874
INT_VAL_RANGE_MIN	70155	583771	583640	70201	67884	67872	599402	121137	961397	958993	120923	115759	115694	986953
INT_VAL_SPLIT_MIN	70155	584140	583755	70098	67844	67943	1068582	120948	957867	957986	120921	115704	115823	1772813
INT_VAL_RND	65939	1202669	664670	65079	68758	78612	920508	113764	1968383	1099419	112129	118035	134461	1536844

The pattern is the same than for the model without redundant clauses, also the number of solutions is 1 for the same cases.

The results for both including redundant and non-redundant configurations are included in red15_2.txt and no red15_2.txt.

Test with 0 initial Alive Cells

For our problem, the extreme case would be start with a fully empty board. This has been tested for some values of N:

N=6	N=8	N=10
VarBranch:INT_VAR_SIZE_MAX() ValBranch: INT_VAL_MAX() GoL 6 0 36 110110 110110 000000 110110 110110 000000 16 110110 110110 000000 110110 110101 000011	VarBranch:INT_VAR_SIZE_MAX() ValBranch: INT_VAL_MAX() GoL 8 0 64 11011000 11010100 00010010 11011010 11010011 00010100 11010100 11011000 30 (other solutions ……) 11011011 11011011	VarBranch:INT_VAR_SIZE_MAX() ValBranch: INT_VAL_MAX() GoL 10 0 100 1101100110 1101010010 0001001100 1101101000 1101001000 0001010110 1111010101 1000101001 0101001010 1101100100 47 (other solutions…)

18 Initial propagators: 305 branchers: 1 Summary runtime: 0.915 (915.000 ms) solutions: 2 propagations: 923393 nodes: 12040 failures: 6018 restarts: 0 no-goods: 0 peak depth: 19	00000000 11011011 11011011 00000000 11011011 11011011 36 Initial propagators: 537 branchers: 1 Summary runtime: 2:39.163 (159163.000 ms) solutions: 4 propagations: 131291089 nodes: 1621477 failures: 810735 restarts: 0 no-goods: 0 peak depth: 34	53 1101101101 1101011011 0001000000 1101011011 1101011010 0001000001 1110111111 1001000000 0101011111 1101101001 54 After 4 hours had not finished the execution.

Whose maximum densities correspond to the optimal solutions known and described in [3].

In the case of N=10, the execution has taken a long time to finish as the whole search space has had to be explored to discard the existence of other solutions, an optimal solution is found in just a few seconds.

In all cases, very far of the optimal results recently published in [4], that proof that all maximum-density still life can be solved in constant time.

Other tests

Some additional files have been used for testing and are included in the delivery. The result has been verified to be correct for all of them. The files are in most cases the same used for the SAT project.

*sl10.txt

*sl10_1.txt

*sl10_2.txt

*sl10_3.txt

*sl15.txt

*sl15_1.txt

*sl15_2.txt

*sl15_3.txt

And their solutions are also provided, named sol<inputfile.txt>

Conclusions

Gecode provides is very comprehensive library for modelling CSP problems. In this case, we have used Integer Variables and arithmetic and logical constraints to model the problem.

The model has been evaluated successfully using the same test set that was used for the SAT implementation of the same problem.

Intuitively, it would have been expected, being a Max Optimization problem, that selecting the Maximum value(1) had been a better strategy and had served for both obtaining faster execution time and a smaller exploration of the search tree as the cost function for a solution is used as a criteria for pruning solutions. But the results show that INT_VAL_MAX/INT_VALUES_MAX does not provide better results. Variable selection is a more discriminant factor to improve performance. In fact, the binary nature of the variables makes that when one value is discarded the alternate is assigned to the variable and propagated causing very likely a lot of nodes being explored that are not feasible, and causing deep searches and the consequent backtracks.

Adding redundant constraints may not originate performance improvements and as it has been the case can even increase execution time significantly. This may be due to the constraints not being adequate to the problem as the clauses are still based on the whole neighborhood of the cell what requires all cells to have a value and hence the propagator may not be able to apply the constraint. In [2] a more complex encoding of the problem is suggested that benefits from redundant constraints that exploit the overlapping of 3x3 blocks in the board via binary encoding of the 9-ary constraint.

References

[1] Chu, Geoffrey, Peter J. Stuckey, and Maria Garcia De La Banda. "Using relaxations in maximum density still life." *Principles and Practice of Constraint Programming-CP 2009*. Springer Berlin Heidelberg, 2009. 258-273.

[2] Smith B “A Dual Graph Translation of a Problem in ‘Life’.

[3] Larrosa J., Morancho E., Niso D., “On the Practical use of Variable Elimination in Constraint Optimization Problems: ‘Still-lfe’ as a Case Study. Journal of Artificial Intelligence Research 23(2005)

[4] Geoffrey Chu, Peter J. Stuckey, A complete solution to the Maximum Density Still Life Problem, Artificial Intelligence, Volumes 184–185, June 2012, Pages 1-16,