Constraint Programming and Hybrid Formulations for Life

Robert Bosch[1] and Michael Trick[2]

[1] Department of Mathematics, Oberlin College, Oberlin, OH USA, 44074
bobb@cs.oberlin.edu
[2] Graduate School of Industrial Administration, Carnegie Mellon, Pittsburgh, PA USA, 15213 trick@cmu.edu

Abstract. Conway's game of Life provides an interesting testbed for exploring issues in formulation, symmetry, and optimization with constraint programming and hybrid constraint programming/integer programming methods. We examine the issue of finding a maximum density still-life. While basic formulations are unable to solve this problem above size 8, improved formulations and symmetry handling extend this to size 13.

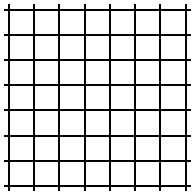
1 Introduction

John Horton Conway invented a one-player game which he called Life in the late 1960s [1]. This game, particularly once popularized by Martin Gardner in his *Scientific American* columns [7–9], developed a large literature. In the course of this work, many aspects of discrete dynamical systems were developed or illustrated by examples in Life.

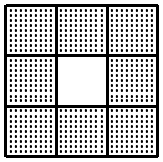
In Life, the player places checkers on some of the squares of an infinite checkerboard. The player then follows some simple rules to create a new pattern on the board, replacing the current checkers. The resulting patterns are both aesthetically interesting and mathematically interesting. Perhaps the most interesting result is that by clever placement of the checkers and appropriate interpretation of the patterns, it is possible to create a Turing-equivalent computing machine (see [1] for a summary of this work).

Bosch [2, 3] looked at a number of uses of integer programming in Life and its variants. Much of his work involved finding a *maximum density still-life*. A still-life in Life is a pattern of checkers which is not modified by the transition rules. Such a pattern represents an "end position", after which no new positions will occur. While it is easy to find such positions (by the rules, the position with no checkers is a still-life), finding very dense positions (those with lots of checkers) is a difficult computational problem.

To define the problem, we begin with the rules of Life. Life is a single-player board game, where the board is a checkerboard that extends to infinity in each direction. Each square of the board is a *cell* and each cell has eight *neighbors*: the eight cells that share one or two corners with it, as shown in figure 1.



the board



a cell and its neighbors

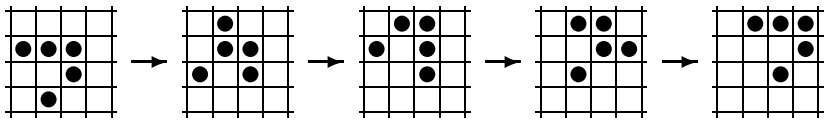
FIGURE 1

To begin the game, the player places checkers on some of the cells of the board, creating an initial pattern. A cell with a checker in it is *living* and those without are *dead*.

The pattern is then modified by applying the following rules over and over again.

- If a cell has exactly two living neighbors then its state remains the same in the new pattern (if living, it remains living; if dead, it remains dead).
- If a cell has exactly three living neighbors, then it is living in the next pattern. This is the "birth condition".
- If a cell has fewer than two or more than three living neighbors then it is dead in the next pattern. These are the "death by isolation" and "death by overcrowding" conditions respectively.

These simple rules are sufficient to generate incredibly complicated patterns and dynamics. As a simple example, consider the sequence in figure 2. This configuration of checkers "moves" one row up and one column to the right every four time iterations.

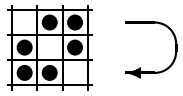


$t = 0$ $t = 1$ $t = 2$ $t = 3$ $t = 4$

FIGURE 2

A still-life is a pattern that doesn't change, so that any cell that is alive in one period is alive in the next, and every dead cell remains dead. By the rules, it is easy to see that still-lifes exist. For instance, the empty pattern is a still-life. A slightly more interesting still-life is shown in figure 3.

Given a fixed region of the board, what is the most checkers we can have in a still-life? The example in figure 3 shows that for a 3 by 3 area, it is possible to have 6 checkers. It is easy to show that it is not possible to have 7 or more, so the maximum density still-life for a 3 by 3 region is $6/9 = 2/3$ (the number



$t = 0, 1, 2 \ldots$

FIGURE 3

of checkers over the size of the region). Note that the region is still embedded in the infinite board, so that no cell outside the region can either be alive or become alive according to the rules.

There has been work done on computational and theoretical aspects of still-lifes. Elkies [6] has shown for the infinite board the maximum density is 1/2. For finite size boards, no exact formula is known for the maximum density. Callahan [4] provides software that uses local search to find still-life patterns. Cook [5] has explored the computational complexity issues of identifying when a still-life (or stable pattern in his terminology) is made up of two or more independent pieces. Niemiec [10] has put together a web page with listings of all small still-lifes.

In this paper, we look at constraint programming and hybrid methods for solving the maximum density still-life problem. This problem is interesting for constraint programming methods because:

1. Basic formulations are very straightforward in CP and quite arcane in integer programming.
2. The interplay of feasibility and optimization makes it a good example for exploring the unique role objective functions take in CP formulations.
3. The symmetry embedded in this problem is very strong, leading both to algorithmic insights and algorithmic difficulties.
4. The problem is well-known, simple to state, and difficult even for relatively small examples.

In the next section, we will provide the basic results for both constraint programming and integer programming models. We then explore two major aspects: improved formulations through handling symmetry and improved formulations by using global constraints to improve the optimization aspects.

2 Base Results

We will consider finding still-lifes in a square n by n region (rectangular or other shaped regions lead to similar results).

An initial formulation for this problem is an almost trivial exercise in constraint programming. The rules define constraints of the form: "If a cell is living then 2 or 3 of its neighbors are living; If a cell is dead then it does not have 3 living neighbors". This creates constraints of the following form(extracting from the OPL formulation; all full formulations are available at `mat.gsia.cmu.edu/LIFE`):

```
(life[i,j] = 1) =>
              2<=sum(n in neighbor) life[i+n.row][j+n.col] <=3;
(life[i,j]=0) => sum(n in neighbor) life[i+n.row][j+n.col] <>3;
```

In this formulation, `life` is a 0-1 matrix which is 1 if the cell is living and 0 otherwise. `neighbor` is a set giving the offsets to find the neighbors of a cell. Conceptually, this is an extremely easy formulation.

For integer programming, in contrast, the formulation of this problem takes a good amount of creativity. Bosch [2] examines two formulations and determines that the following is the better approach:

Let x_{ij} be 1 if cell (i, j) is alive and 0 otherwise. Let $N(i, j)$ represent the cells that are neighbors of (i, j). Then, the constraint

$$3x_{ij} + \sum_{(i',j') \in N(i,j)} x_{i'j'} \leq 6$$

enforces the "death by overcrowding condition": if a cell is alive, then at most 3 of its neighbors are alive. It is straightforward to see that if the cell is dead, then at most six of its neighbors are alive (otherwise another cell would be overcrowded).

To handle the "death by isolation condition", the following constraint suffices:

$$2x_{ij} - \sum_{(i',j') \in N(i,j)} x_{i'j'} \leq 0$$

If a cell is alive then at least two of its neighbors are also.

To prohibit violations of the birth rule, we need a somewhat more complicated constraint. Let S be a 3 element subset of $N(i, j)$. Then we need

$$-x_{ij} + \sum_{(i',j') \in S} x_{i'j'} - \sum_{(i',j') \in N(i,j)-S} x_{i'j'} \leq 2$$

In words, if all of S is alive, then either (i, j) is alive or some element of $N(i, j)-S$ is alive.

This formulation is sufficient to define an integer programming formulation for this problem (again, see the full OPL formulation at `mat.gsia.cmu.edu/LIFE`). This formulation, of course, is much less straightforward than the CP formulation.

3 Avoiding Symmetry

Before we begin trying to solve these models, it is worthwhile to explore the symmetry issues in this problem. For any still-life, it is possible to create an equivalent still-life through a number of operations:

1. Rotate the square by 90, 180 or 270 degrees
2. Reflect the pattern either horizontally, vertically, or along one long diagonal.

3. Combinations of rotations and reflections

It would be very useful to find a set of constraints that would allow us to generate only one of the equivalent patterns, ignoring the others. To date, we have not been able to find a usable constraint that can be added to either our CP or IP formulations.

Instead, we can add constraints that cut down on the redundant patterns. The simplest set of constraints recognizes that the top half and bottom half of a pattern may differ in density, so we can choose to look only at patterns where the top half is at least as dense as the bottom half. Similarly, and independently, we can restrict our search to those whose left half is at least as dense as its right half. These two simple linear constraints can be added to either the CP or the IP formulation.

At this point, we are ready to provide our first results. In Table 1, ChPo refers to "Choice Points" (or nodes of the search tree for IP formulations) and T is time in seconds (Intel 650Mz Pentium III with 196Mb memory running ILOG's OPL Studio 3.5).

Size	Value	CP w/o Sym (CP1)		CP with Sym (CP2)		IP w/o Sym (IP1)		IP with Sym (IP2)	
		T	ChPo	T	ChPo	T	ChPo	T	ChPo
4	6	.02	275	.02	259	.41	0	.37	0
5	16	.04	636	.05	636	.76	1	.50	0
6	18	1.51	24472	1.64	24083	22.68	1952	8.22	504
7	28	9.44	154151	10.07	154147	7.1	55	4.94	10
8	36	189.75	3084106	205.37	3082922	65.34	798	22.1	348

Table 1. Results on Base and Symmetry

None of these four formulations were able to solve the size 9 problem in under half an hour (for instance, the basic CP formulation took 19,637 seconds or a bit over 5 hours).

While it is clear that the symmetry breaking helps the integer programming formulation, it does not aid the CP very much (while it slightly reduces the number of choice points, the overall time goes up slightly). We will explore this in the next section.

4 Linear Constraints as a Global Constraint

Examining the details of the runs for the constraint programming model make it clear why they are inefficient. We set the search routine to begin with the first row and try to set as many cells to one as possible; it then moves down to the next row.

Consider a partial solution to the problem, with the variables instantiated for the first few rows. At this point, the constraint propagation does not go any "deeper" into the board than one row beyond what is already instantiated. For all remaining cells, their domain remains $\{0, 1\}$. The resulting bound on the objective value is very weak: with all the uninstantiated cells being possibly alive, the possible objective is very high. A very large portion of the board must be instantiated before the objective function has any force. This leads to a huge number of choice points and a long computing time. This also explains why the "anti-symmetry" constraints do not work well with this CP model: it takes too long for the search to recognize that they are binding.

To solve this, we need to provide some global information to the search on how well a partial pattern can be completed. This information could be developed a number of ways. For instance, since we know that every 3 by 3 square can have no more than 6 live cells, we could cover the board in 3 by 3 squares linked to the original formulation. We could then have the objective be set equal to a covering of the board with the 3 by 3 squares. Such a method would give some "future" information to the search. In fact, we have experimented with such an approach, and it does improve on the results so far.

There is a simpler method, however, that involves linking the integer programming approach to the constraint programming approach. Starting with the standard CP formulation, we can add the integer programming constraints as linear inequalities. We can then use the linear relaxation of these inequalities to bound the possible values of the uninstantiated variables. In doing so, the "global constraints" of the linear inequalities work together with the logical constraints to greatly improve the search.

A blind application of this approach however, is not the best solution. In particular, a natural choice would be to simply combine the two formulations. This approach does not even improve on the approaches already examined. A more efficient formulation recognizes the role the linear constraints are playing. Our goal here is to give the search a bound on the best values for the uninstantiated variables: there is a tradeoff between quality of bound and time taken to find the bound. In our integer programming formulation, for each cell there is

1. one "death by crowding" constraint
2. one "death by isolation" constraint
3. 56 "birth" constraints

We can decrease the number of linear constraints by 95% simply by not including the birth constraints. The logical constraints in the CP side will enforce this requirement.

In fact, we can explore this more closely. "Death by Isolation", while a necessary constraint, is not often binding in dense still-lifes. Again, we can allow the logical constraints to enforce this requirement and only keep the key "Death by Overcrowding" linear constraints. This further decreases the size of the linear constraints by a factor of 2.

This gives three possible formulations: HYB1 has all of the linear constraints, HYB2 removes the birth constraints, and HYB3 removes both the birth constraints and the death by isolation constraints. This gives the results in Table 2 for larger problems.

Size	Value	IP with Sym all cons. (IP2)		CP/IP with Sym all cons. (HYB1)		CP/IP with Sym no birth (HYB2)		CP/IP with Sym no birth/iso (HYB3)	
		T	ChPo	T	ChPo	T	ChPo	T	ChPo
8	36	22	348	47	1810	3	2310	2	2474
9	43	—	—	—	—	85	46345	51	48975
10	54					291	98082	147	102865
11	64					655	268520	373	318942
12	76					49166	11772869	30360	12733484
13	90					50871	10996298	30729	11845960

Table 2. Results on Hybrid Approaches

While we have developed methods based on integer programming that can also prove the optimality of the size 12 and 13 problems, the method presented here seem to be much faster (roughly a factor of 5 on the larger problems).

5 Exploiting Symmetry

As a final improvement, we also have tried to exploit the symmetry in this game by finding feasible solutions that are either horizontally or vertically symmetrical. In almost every case we have examined, there exists an optimal solution with such an axis of symmetry (an exception is the 7 by 7 instance). Modifying the formulations to generate only patterns with such symmetry requires adding constraints that force cells in the top half to take on the same values as corresponding cells in the bottom half. The result is an instance with far fewer degrees of freedom.

Table 3 gives the best solutions found by this approach (all have now been proved optimal for the symmetric problem). We conjecture that there exist no better non-symmetric solutions. We can use these values to help prove optimality by adding constraints to force the generation only of better solutions.

Size	14	15	16	17	18
Objective	104	119	136	152	170

Table 3. Symmetric, Feasible Solutions

6 Conclusions and Future Research

Finding dense still-lifes in Life is an interesting challenge for constraint programming and integer programming. The constraints to enforce the transition rules are much more natural in constraint programming, but the global view necessary to quickly prove optimality is best handled by the linear constraints of integer programming. The combination is much better than either individual approach. Key to this success are a number of insights:

1. Symmetry breaking is important, but only when the symmetry possibilities can be recognized early in the search.
2. Symmetry can be exploited to decrease problem dimension to quickly get good feasible solutions.
3. When adding linear constraints as global constraints, it is important to add only what is necessary to find good bounds.

While this method does appear to be the fastest at the moment, there is ample opportunity for improvement. In particular, the symmetry breaking being done seems ripe for improvement. Also, a more intelligent method of adding linear constraints seems a good opportunity.

References

1. E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your Mathematical Plays, Volume 2: Games in Particular*, Academic Press, London, 1982.
2. R.A. Bosch, Integer programming and Conway's game of Life, *SIAM Review* 41 no. 3 (1999) 594–604.
3. R.A. Bosch, Maximum density stable patterns in variants of Conway's game of Life, *Operations Research Letters* 27 (2000) 7–11.
4. P. Callahan, *Random Still Life Generator*, `http://www.radicaleye.com/lifepage/stilledit.html`, 2001
5. M. Cook, *Still Life Theory*, `http://paradise.caltech.edu/ cook/Workshop/CAs/2DOutTot/Life/StillLife/StillLifeTheory.html`, 2001.
6. N.D. Elkies, The still-Life density problem and its generalizations [pages 228-253 in Voronoi's Impact on Modern Science, Book 1 (P. Engel, H. Syta, eds.; Institute of Math., Kyiv, 1998)].
7. M. Gardner, *The fantastic combinations of John Conway's new solitaire game "life"*, Sci. Am., 223 (1970), pp. 120–123.
8. M. Gardner, *On cellular automata, self-reproduction, the Garden of Eden and the game "life"*, Sci. Am., 224 (1971), pp. 112–117.
9. M. Gardner, *Wheels, Life, and Other Mathematical Amusements*, W. H. Freeman, New York, 1983.
10. M.D. Niemiec, *Mark D. Niemiec's Life Page*, `http://home.interserv.com/mniemiec/lifepage.htm`, 2001.
11. W. Poundstone, *The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge*, William Morrow, New York, 1985.