A Dual Graph Translation of a Problem in 'Life'

Barbara M. Smith

University of Huddersfield
Huddersfield HD1 3DH, U.K.
b.m.smith@hud.ac.uk

Abstract. Conway's game of Life provides interesting problems in which modelling issues in constraint programming can be explored. The problem of finding a maximum density stable pattern ('still-life') is discussed. A formulation of this problem as a constraint satisfaction problem with 0-1 variables and non-binary constraints is compared with its dual graph translation into a binary CSP. The success of the dual translation is surprising, from previously-reported experience, since it has as many variables as the non-binary CSP and very large domains. An important factor is the identification of many redundant constraints: it is shown that these can safely be removed from a dual graph translation if arc consistency is maintained during search.

1 Introduction

The game of Life was invented by John Horton Conway in the 1960s and popularized by Martin Gardner in his *Scientific American* columns (e.g. [6]). Many variants of the game and problems arising from it have been studied. Here, one such problem is described and its solution using constraint programming is discussed.

Life is played on a squared board, considered to extend to infinity in all directions. Each square of the board is a cell, which at any time during the game is either alive or dead. A cell has eight neighbours, as shown in Figure 1. The configuration of live and dead cells at time t leads to a new configuration at time $t+1$ according to the rules of the game:

- if a cell has exactly three living neighbours at time t, it is alive at time $t+1$
- if a cell has exactly two living neighbours at time t, it is in the same state at time $t+1$ as it was at time t
- otherwise, the cell is dead at time $t+1$

A *still-life* is a pattern that is not changed by these rules: hence, every cell that has exactly three live neighbours is alive, and every cell that has fewer than two or more than three live neighbours is dead. An empty board is a still-life, but so are more interesting patterns. The question addressed in this paper is: on an $n \times n$ section of the board, with all the rest of the board dead, what is the densest possible still-life pattern, i.e. the pattern with the largest number of live cells? For instance, Figure 2 shows a maximum density 3×3 pattern.

P. Van Hentenryck (Ed.): CP 2002, LNCS 2470, pp. 402–414, 2002.

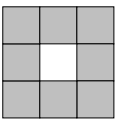

Fig. 1. A cell and its 8 neighbours

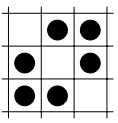

Fig. 2. A maximum density 3 × 3 still-life

Bosch and Trick [4, 3] considered integer programming and constraint programming formulations of the maximum density still-life problem. Their most successful approach [3] used a hybrid of the two. In this paper, pure constraint programming approaches are considered further.

2 A 0-1 Formulation

An obvious way to model the problem as a constraint satisfaction problem (CSP) is to use a 0-1 variable x_{ij} for the cell in row i, column j of the $n \times n$ grid, such that x_{ij} has the value 0 if the cell is dead and 1 if it is alive. Each cell is constrained by its eight neighbouring cells: if the sum of the values assigned to the neighbouring cells is 3, then the cell is alive, and if the sum is less than 2 or more than 3 then the cell is dead. Since all the cells surrounding the $n \times n$ square are dead, we cannot have a sequence of three live cells along the boundary: this is also included as a constraint.

The density of a pattern is the sum of the values of the cell variables. It is maximized by adding a constraint whenever a solution is found, that the density must be greater in any new solution; when no more solutions exist, the last one found is optimal.

The variables are assigned in lexicographic order, row by row and left to right along each row. This may not be the best ordering, but several likely alternatives have proved worse. To find dense solutions quickly, the value 1 is assigned before the value 0.

This formulation has been implemented in ILOG Solver. It is very similar to the basic constraint programming model described in [3].

The first improvement to this model is to deal with the symmetry of the problem. Given any Life pattern, there are up to seven symmetrically equivalent patterns resulting from rotating or reflecting the board. Hence, the search can explore many partial solutions which are symmetrically equivalent to dead-ends that have already been explored. This can be avoided on backtracking by adding

constraints to the new branch of the search tree to forbid the symmetric equivalents of assignments already considered. Gent & Smith [7] have implemented SBDS (Symmetry Breaking During Search) in Solver to do this. The symmetries of Life are the same as the symmetries of the n-queens problem, an example discussed in [7]. In this case, all that the SBDS user need do is write seven functions, each describing the effect of one of the symmetries on an assignment of a value to a variable. Bosch & Trick added constraints to their constraint programming formulation to remove some of the symmetry, but SBDS, which removes all of it, gives a greater reduction in search.

The results of the 0-1 formulation are given in Table 1. Search effort is measured by the number of fails (backtracks) reported by Solver. The table shows both the effort required to find the optimal solution and the total effort and running time (on a 600MHz PC). The difference between the number of fails to find the optimal solution and the total number of fails shows the effort required to prove optimality, i.e. to prove that there is no solution with greater density. Using SBDS reduces the number of fails by about a factor of 6, and the running time by about a factor of 4.

These results are better than those reported by Bosch and Trick for a constraint programming formulation. However, they achieved much better results from their hybrid approach. The constraint programming approach needs to be much improved before it is competitive with the hybrid.

Table 1. Search effort and running time to find maximum density still-lifes using a 0-1 formulation. Value = maximum density; F = number of fails (backtracks) to find the optimal solution; P = number of fails to prove optimality; sec. = running time in seconds

n	Value	No symmetry breaking			With SBDS		
		F	P	sec.	F	P	sec.
5	16	3	187	0.03	3	59	0.02
6	18	9	6030	0.88	9	1062	0.22
7	28	617	39561	6.74	607	8436	1.78
8	36	955	811542	135	450	146086	26.6
9	43	*not attempted*			259027	11065129	2140

3 Dual Graph Representation

There are several reasons for the poor performance of the 0-1 model. Firstly, the constraints between a variable and its 8 neighbouring cell variables are not helpful in guiding the search. The state of the cell can still be undetermined when 7 of its neighbours have been assigned (if it has exactly 2 or 3 live neighbours

at that point). At best, four of the neighbouring cells must be assigned before a variable's value is determined, and this is only sufficient if all four are live.

Secondly, 0-1 variables do not lend themselves to variable ordering heuristics based on domain size: as soon as a value is removed from the domain of a variable, the alternative value is assigned.

Thirdly, as pointed out in [3], it is difficult to discover whether a partial assignment can be completed to give a higher density than in the incumbent solution. The density is defined as the sum of the variables, and since any unassigned variable has the value 1 in its domain, the upper bound on the density in the unassigned part of the grid is simply the number of cells it contains.

To begin to address these difficulties, we can consider different ways of representing the problem as a CSP. The dual graph representation is a well-known translation of a CSP with non-binary constraints into a binary CSP [5]. As described in [1], for instance, the constraints of the original problem become the variables of the dual representation: the domain of a dual variable is the set of tuples that satisfy the original constraint. There is a binary constraint between two dual variables iff the corresponding constraints in the original problem share at least one variable: the constraint ensures that the dual variables assign the same value to each of the original variables that they share.

In this problem there are two kinds of non-binary constraint: the constraints between each cell and its neighbours, of arity 9, and the objective constraint, ensuring that the number of live cells in any solution is greater than in the incumbent solution. The objective constraint cannot be replaced by a dual variable, since this would require listing all the satisfying tuples and so would entail solving the problem. The dual encoding will only replace the arity 9 constraints, and so will not result in a pure binary CSP.

The variables of the dual encoding correspond to 'supercells' in the grid, i.e. 3×3 squares consisting of a cell and its eight neighbours, as in Figure 1. The supercell variable y_{ij} corresponds to the cell variables $x_{i,j}$, $x_{i,j+1}$, $x_{i,j+2}$, $x_{i+1,j}$, $x_{i+1,j+1}$, $x_{i+1,j+2}$, $x_{i+2,j}$, $x_{i+2,j+1}$ and $x_{i+2,j+2}$.

A possible value of y_{ij} can be represented as a 9-bit number, rather than a 9-tuple, by writing the following constraint between a supercell variable and the variables of its constituent cells:

$$\begin{aligned} y_{ij} = {} & x_{i,j} + 2x_{i,j+1} + 4x_{i,j+2} + 8x_{i+1,j} + 16x_{i+1,j+1} \\ & + 32x_{i+1,j+2} + 64x_{i+2,j} + 128x_{i+2,j+1} + 256x_{i+2,j+2} \end{aligned} \tag{1}$$

A supercell variable is therefore an integer variable, with values between 0 and 511.

Each possible value for a supercell variable indicates which of the 0-1 variables in (1) have the value 1, and hence which of the 9 cells are alive. In a still-life many of these values are not allowed: any 3×3 area can contain at most 6 live cells, for instance. The possible values of a supercell variable in the interior of the square just correspond to the 259 tuples satisfying the 9-ary constraint. For supercells on the boundary, the domain has to allow for the neighbouring dead cells. The domains are constructed by finding all the feasible configurations for

a 3 × 3 square which is within the $n \times n$ square but at the edge. If a value in the range 0 to 511 represents a feasible assignment for 3 × 3 square, it will be consistent with the dead cells beyond the edge, and the value is included in the domain of the variable; otherwise, this is not a possible value for a supercell in that position. The supercells in the corners of the grid have fewest possible values (74), since they must be consistent with the dead cells neighbouring them on two sides. Supercells along the edges of the grid, but not in a corner, have 148 possible values.

4 Binary Constraints

Following the standard account of the dual encoding given earlier, a constraint is required between any pair of supercell variables which share a cell variable. Figure 3 shows the supercell variable y_{33} and the supercells above and to the left of it that it shares a cell with. y_{33} also shares cells with 18 other supercells, unless n is small and it is too close to the bottom right corner of the square for all of these to exist: only one quarter of the possibilities are shown, for space reasons. In the standard dual encoding, we therefore need binary constraints between y_{33} and up to 24 other variables.

It was pointed out by Dechter & Pearl [5] that some of the constraints in the dual graph may be redundant, in the sense that eliminating them does not change the set of solutions of the problem. A constraint is redundant if there is an alternative path between the two dual variables involved such that the variables which they share are also shared by every variable along the path. The constraints along the alternative path are sufficient to ensure that, in any solution, the shared variables have the same value in every dual variable along the path, in particular in the two dual variables at the ends of the redundant constraint.

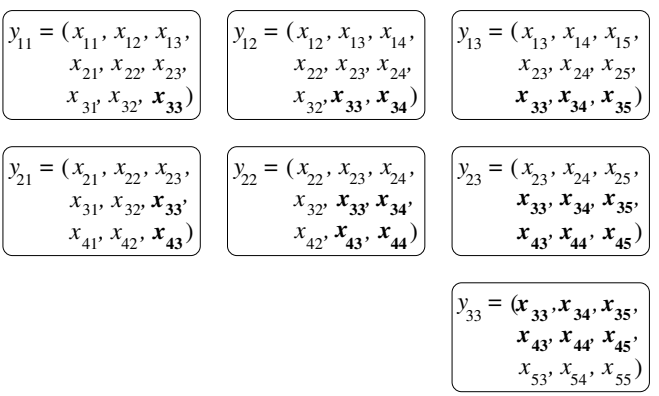

Fig. 3. Supercell variable y_{33} and some of the supercell variables which share at least one cell variable with it

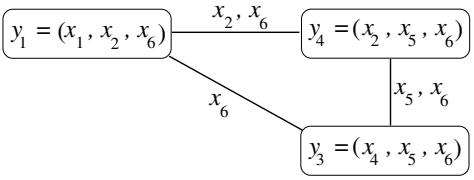

Fig. 4. Binary constraints between three dual variables

All the constraints involving y_{33} shown in Figure 3 are redundant, except for the constraint with y_{23}, which shares 6 cells with y_{33}. The other variables shown share a subset of these 6 cells with y_{33}. If there is a constraint only between pairs of variables that share 6 cells, there will always be a path between pairs of variables sharing at least one cell. For instance, y_{11} and y_{33} have just x_{33} in common: each variable along the path $y_{33} \rightarrow y_{23} \rightarrow y_{13} \rightarrow y_{12} \rightarrow y_{11}$ shares 6 cells with its predecessor and with its successor, and x_{33} is a common cell in them all. If y_{33} is assigned a value, then any consistent assignment to the variables along this path must have each variable assigning the same value to x_{33}, as required.

Although deleting redundant constraints will not change the solutions, it may increase the search effort to find a solution, which is why descriptions of the dual encoding include them. Figure 4 is based on part of a diagram in [1]. There, Bacchus and van Beek discuss using the dual encoding with the forward checking algorithm. Assigning a value to the dual variable y_1 then reduces the domains only of those variables which are constrained by y_1, so that if the redundant constraint between y_1 and y_3 is deleted, the domain of y_3 will not be reduced by an assignment to y_1 and the search may take longer.

However, if arc consistency is maintained after assignments are made, this constraint can safely be deleted: assigning a value to y_1 will still reduce the domain of y_3 appropriately via the other two constraints, because there is an alternative pathway $y_1 \rightarrow y_4 \rightarrow y_3$ with x_6 shared between all three variables. In general, if the search algorithm maintains arc consistency, redundant constraints between dual variables can, and should, be deleted. This has not previously been recognised.

Constraint programming tools such as ILOG Solver can maintain the arc consistency of the binary constraints of the dual encoding during search. The binary constraints for the still-life problem are expressed in Solver using *table constraints*: these implement the general arc consistency schema introduced by Bessière and Régin [2]. A predicate is supplied to the table constraint to evaluate, given a pair of variables and a value for each of them, whether or not the pair of values satisfy the required constraint between the variables.

In the dual representation of the still-life problem, a supercell variable y_{ij} need only be involved in four binary constraints, with $y_{i-1,j}$, $y_{i,j-1}$, $y_{i+1,j}$ and $y_{i,j+1}$. The 20 redundant constraints involving y_{ij} can be omitted. In total, approximately $2n^2$ binary constraints are needed, compared to approximately

$12n^2$ for large n if constraints between all variables sharing a cell are included (precisely, $2(n-2)(n-3)$ compared to $6(n-3)(2n-7)$, e.g. for $n = 8$, 60 constraints compared to a total of 270).

If the redundant constraints were added to the model, it could make no difference to the search, but would greatly increase the running time. Adding just the constraints between each supercell variable y_{ij} and its 'diagonal neighbours' $y_{i+1,j+1}$, $y_{i-1,j-1}$, $y_{i-1,j+1}$ and $y_{i+1,j-1}$ approximately doubles the number of constraints: for the range of problem sizes being considered, the running time increases by roughly 50%. Using all the constraints would more than double their number again, for $n = 8$. Hence, it is essential to remove the redundant constraints in order to achieve good performance.

5 Solving the Dual Model

So far, the dual model has approximately n^2 dual variables, mostly corresponding to interior supercells and so with 259 possible values, and approximately $2n^2$ binary constraints. In principle, in a dual encoding, the original variables are no longer needed. However, the objective constraint still has to be modelled, and the x_{ij} variables are kept for this purpose. The density of a pattern, as before, is the sum of the x_{ij} variables. The constraints (1) ensure that whenever a variable y_{ij} is assigned a value, its constituent x_{ij} variables are bound at the same time. Only the y_{ij} variables are used as search variables.

It is now possible to use the smallest domain variable ordering heuristic, which was not possible in the 0-1 model. This heuristic gives significantly better results than lexicographic ordering; the results given below use this heuristic. As in the 0-1 model, the value ordering used favours dense patterns, by choosing the value for a supercell variable giving the largest number of live cells.

Table 2 shows a comparison between the results of running the 0-1 encoding (repeated from Table 1) and the dual encoding. The dual encoding is doing much less search, but the running time, even without the redundant constraints, is much longer. The much higher ratio of running time to number of fails in the dual representation is because the domains of the dual variables are very large, so that maintaining arc consistency is time-consuming. Further improvements are discussed in the next section.

6 Improved Bounds

One reason for the poor performance of both models is that proving optimality takes a long time. As already noted, given a partial solution, the upper bound on the density of an extension to the rest of the grid assumes that all remaining cells can be alive. Hence, it can appear that even a sparsely populated partial grid can lead to a better solution, and it can take a long time to prove otherwise.

Since a 3×3 square can have at most 6 live cells in a still life, rather than 9, a much better bound on the cost of completing a partial solution can be found by covering the grid with such squares. Bosch & Trick [3] found that this

Table 2. Finding maximum density still-lifes with either the 0-1 encoding or the dual encoding

n	0-1 encoding		Dual encoding		
	P	sec.	F	P	sec.
5	59	0.02	15	51	0.41
6	1062	0.22	0	181	2.23
7	8436	1.78	1321	3510	16.2
8	146086	26.6	18084	53262	264
9	11065129	2140	250401	2091386	10300

gave some improvement in conjunction with their 0-1 model, although not as much as using bounds from the IP formulation. Bounds based on 3×3 squares (supercells) are relatively easy to incorporate into the dual representation, so the cost of completing a partial solution can be calculated from a set of supercell variables forming a partition of the unassigned part of the grid.

When n is a multiple of 3, the $n \times n$ grid can be partitioned into disjoint supercells and the density of the grid expressed as the sum of their densities. This defines the objective in terms of a subset of the dual variables. A side-effect is that the original variables are no longer needed; they were kept in the dual encoding only to define the objective. Hence for this special case it is possible to model the problem using just the dual variables, with all binary constraints apart from the objective constraint. However, a comparison of the running time with and without the cell variables shows that having them in the model carries a negligible overhead, when they are used only to access the states of individual cells and not as search variables.

Extending this model to the general case can be done by 'padding' the $n \times n$ grid with dead cells to the right and below it, so that the number of cells on each side is a multiple of 3, as shown in Figure 5. This again allows the density to be defined in terms of a partition of the extended grid into supercells.

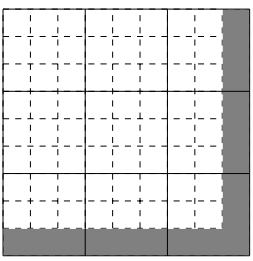

Fig. 5. An 8×8 grid, increased to 9×9 by adding dead cells

Calculating the density in this way has a dramatic effect on the search effort and the solution time: the 9×9 problem, which previously took over 2 million fails and over 10,000 sec. to solve with the dual encoding, now takes 1555 fails and 73 sec. to solve. This change makes the dual model much faster than the 0-1 model, for problems of this size.

7 Search Strategies & Results

Since the density is now defined in terms of a set of disjoint contiguous supercells, these variables can also be used as the search variables: propagation of the table constraints would assign consistent values to the remaining variables. (A simple way to implement this is to define all the supercell variables as search variables, but allow the variable ordering to select just from those that form a partition of the grid.) This reduces the number of variables to be assigned values from $(n-2)^2$ to $(n/3)^2$, e.g. from 49 to 9 in the 9×9 case. It would be simplistic to expect a similar reduction in search: the domain of each search variable, when it is selected for assignments, is likely to have been much less reduced by previous assignments than before.

With a choice of variable ordering heuristic, we have four possible search strategies: we can use all the supercell variables as the search variables or just a subset that together partition the grid, and we can assign them in lexicographic order, or choose the smallest domain.

The results of these four strategies are compared in Table 3. With lexicographic ordering, using the smaller set of search variables does reduce search,

Table 3. Comparison of four search strategies for the dual encoding

	All variables					
	Lex. ordering			Smallest domain		
n	F	P	sec.	F	P	sec.
5	11	12	1.11	0	2	0.66
6	0	13	0.79	0	13	0.78
7	75	216	7.27	5	54	3.52
8	200	530	33.4	49	206	13.9
9	592	1554	79.4	598	1555	73.4
10	150406	577258	29322	1471	225179	10900

	Partition					
	Lex. ordering			Smallest domain		
n	F	P	sec.	F	P	sec.
5	11	13	0.74	0	2	0.64
6	0	13	0.79	0	13	0.95
7	77	126	7.06	4	45	3.55
8	196	517	32.5	0	125	11.1
9	454	1171	66.7	340	893	56.8
10	102141	369332	21900	89	128905	7780

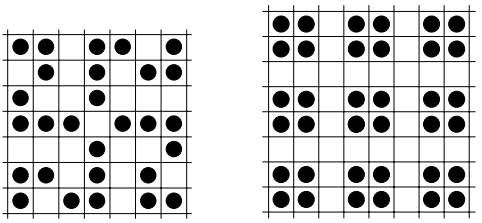

Fig. 6. The maximum density 7 × 7 and 8 × 8 still-lifes

especially for the larger problems. Using the partitioning variables in combination with the smallest domain ordering has a much greater effect, however. This ordering strategy will behave very differently depending on the set of search variables. If all the supercell variables are search variables, then after the first variable has been assigned, there will always be at least one variable having at least 6 cells in common with one already assigned, and hence a reduced domain size of at most 8: one of these variables will have the smallest domain. The improved performance of the heuristic when the search variables partition the grid may be because it then has more freedom to choose the variables in different parts of the grid.

Although the 9 × 9 problem is now easily solved, larger problems are still very difficult, as shown in Table 3 for the 10 × 10 problem. The optimal solution to this problem and to the 11 × 11 problem can be found very quickly, and it is proving optimality that is time-consuming. The time to solve the 10 × 10 problem has been improved to some extent by adding tighter bounds on the cost of completing a partial solution. Problems representing the corner of a larger grid, rather than a complete grid, of sizes 6 × 6, 6 × 9, 9 × 6 and 9 × 9, were solved optimally. For each corner size, the density of the whole grid is then constrained to be at most the maximum density of the corner plus the maximum densities of supercells covering the rest of the grid. These constraints provide tighter bounds: for instance, a 9 × 9 corner has maximum density 46, whereas the bound based on supercells suggests that a 9 × 9 square can hold 54 live cells. With these bounds, the 10 × 10 problem can be solved optimally in 55550 fails, 3300 sec. The 11 × 11 problem is still intractable for this approach, however: although the optimal solution (given by Bosch & Trick) is found immediately, with no backtracking, it has not been proved optimal in more than 10 hours running time.

An advantage of using SBDS to eliminate symmetrically-equivalent solutions is that non-isomorphic solutions can be enumerated. There is a unique optimal solution for $n = 3$, 5, 7 and 8; Figure 6 shows two of these. There are 2, 9 and 10 solutions for $n = 4$, 6 and 9 respectively, but many more for $n = 10$ and 12.

When n is too large to solve optimally with current methods, Bosch & Trick proposed finding symmetric solutions, and found optimal horizontally symmetric solutions for $n = 14$ and 15. Restricting solutions to those with 90° rotational symmetry, the dual encoding has found the optimal solution for n =18, which I

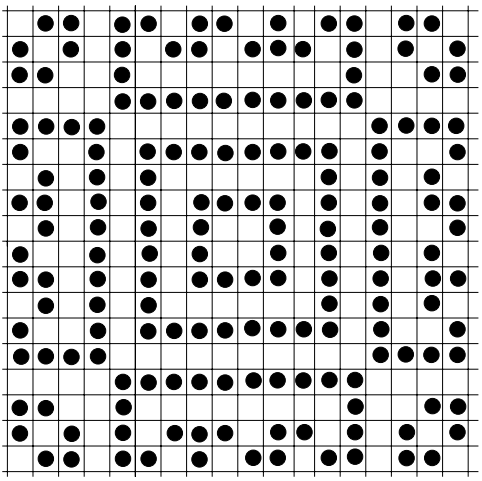

Fig. 7. A maximum density rotationally-symmetric 18 × 18 still-life

believe is new. This solution took 295 fails to find, and 76527 fails, 19900 sec. to prove optimal. It has value 168 and is shown in Figure 7.

8 Discussion & Conclusions

The final version of the dual representation outperforms the 0-1 formulation by two orders of magnitude in running time, and still more in the number of fails. The improvement over the first dual representation is partly due to the better bounding information given by expressing the density in terms of a partition into 3×3 squares, corresponding to supercell variables. This could be done by using supercell variables as an adjunct to the 0-1 formulation: Bosch & Trick reported trying this, although it appears that the improvement then was not as dramatic as in the dual encoding. The fact that the dual variables can be identified with 3×3 supercells also helps in devising good search strategies for the dual encoding, and the remaining improvement comes from basing the search itself on a subset of the dual variables representing a partition of the $n \times n$ square.

An important factor in the success of the dual representation is that redundant constraints have been removed. They have no pruning effect if arc consistency is maintained during search, and would slow down the search several times over. I believe that it has not previously been noted that redundant constraints in the dual representation should be removed if arc consistency is maintained. Redundant constraints may occur in the dual translations of other problems: a necessary condition is that three dual variables should share a common variable. A problem where this does not occur is the crossword puzzle example considered by Bacchus and van Beek [1]. The original variables correspond to the letter squares in the crossword puzzle grid, and the constraints (and so the

dual variables) to sets of consecutive letter squares forming a word. Since any letter square in a crossword grid is the intersection of at most two words, three dual variables cannot have a letter variable in common, and so there are no redundant constraints. An example where they do occur is the Golomb rulers problem, discussed by Stergiou and Walsh [8]: the original CSP has ternary constraints $x_{ji} = x_j - x_i$ for $1 \leq i < j \leq n$, defining the distance between the ith and jth marks on the ruler. The dual variables corresponding to the constraints defining x_{ji}, for $1 \leq j \leq n, j \neq i$, would then all have the variable x_i in common, giving many redundant binary constraints in the dual representation. This shows that redundant constraints can occur in the dual representation of other problems. However, as Bacchus and van Beek noted, there has been little experience reported of the effectiveness of translations into binary representations, so it is hard to say how common redundant constraints are likely to be.

It is surprising that the dual representation of the still-life problem is practicable, since it does not fit the guidelines given by Bacchus and van Beek to indicate when the dual graph representation of a problem might be successful. They suggest that the number of constraints in the original CSP should be small, and the constraints should be tight, i.e. with few of the possible tuples allowed. Hence, the dual CSP will have a small number of variables with small domains. Here the dual representation has about the same number of variables as the non-binary CSP, and the domains are very large, especially in comparison with the original 0-1 variables.

A factor not mentioned by Bacchus and van Beek is the tightness of the binary constraints in the dual translation. Once the redundant constraints have been removed, any constraint of the dual encoding links two dual variables that share 6 cell variables. These are tight constraints: once a dual variable has been assigned a value, any variable it constrains has only three unassigned cells; at most 8 values, out of possibly 259, are still left in the domain. Hence although the domains are initially large, the branching factor of the search tree is manageable. This may be a reason why the dual translation is successful in this case; the tightness of the resulting binary constraints should perhaps be a factor to take into account when considering using the dual translation of a problem.

Compared to Bosch & Trick's results [3], the dual representation solves problems up to 9×9 with much less search: their hybrid of constraint and integer programming takes 46000 choice points (slightly more than the number of fails, in optimization problems) to solve the 9×9 problem, compared with 893 for the dual encoding. On the other hand, the hybrid solves the 11×11 problem with about 10 times the effort required for the 9×9 problem, whereas it is out of reach, so far, for the dual encoding, This suggests that the dual CSP is doing better locally than the hybrid: the effects of an assignment to a supercell variable are quickly propagated to the neighbouring cells. For the smaller problems, local effects are predominant, and so the dual encoding does well. As problems get larger, local information becomes inadequate. The dual encoding does not have good enough bounds to determine the effects of assignments made in one part of the square on the overall density, and the better bounds given by the linear

constraints in the CP/IP hybrid become more significant. The two approaches seem to offer complementary advantages, and combining the two could lead to still better results.

The dual representation has so far enabled some new results to be found in the maximum density still-life problem, for instance the 18 × 18 pattern in Figure 7 and the enumeration of non-isomorphic solutions. It offers the hope of solving further problems in combination with the CP/IP hybrid. It also gives rare practical experience of the successful reformulation of a non-binary CSP using the dual graph translation.

Acknowledgments

I should like to thank Mike Trick for his helpful comments on this work. Thanks are also due to the other members of the APES group for their support: Peter van Beek in particular made encouraging comments on an early version of the paper. I am especially grateful to Ian Gent: the paper has been much improved as a result of his input.

References

[1] F. Bacchus and P. van Beek. On the Conversion Between Non-Binary and Binary Constraint Satisfaction Problems. In *Proceedings AAAI'98*, pages 311–318, 1998. 405, 407, 412

[2] C. Bessière and J.-C. Régin. Arc consistency for general constraint networks: preliminary results. In *Proceedings IJCAI'97*, volume 1, pages 398–404, 1997. 407

[3] R. Bosch and M. Trick. Constraint programming and hybrid formulations for three life designs. In N. Jussien and F. Laburthe, editors, *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 77–91, 2002. 403, 405, 408, 413

[4] R. A. Bosch. Maximum density stable patterns in variants of Conway's game of Life. *Operations Research Letters*, 27:7–11, 2000. 403

[5] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989. 405, 406

[6] M. Gardner. The fantastic combinations of John Conway's new solitaire game. *Scientific American*, 223:120–123, 1970. 402

[7] I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. In W. Horn, editor, *Proceedings ECAI'2000*, pages 599–603, 2000. 404

[8] K. Stergiou and T. Walsh. Encodings of Non-Binary Constraint Satisfaction Problems. In *Proceedings AAAI'99*, pages 163–168, 1999. 413