Constraints (2006) 11: 91–114
DOI 10.1007/s10601-006-8058-9

Applying Ad-hoc Global Constraints with the `case` Constraint to Still-Life

Kenil C. K. Cheng · Roland H. C. Yap

Abstract The Still-Life problem is challenging for CP techniques because the basic constraints of the game of Life are loose and give poor propagation for Still-Life. In this paper, we show how ad-hoc global `case` constraints can be customized to construct various models to provide much stronger propagation with CP solvers. Since we use custom ad-hoc constraints of high arity where the number of tuples to define the constraint are large, the actual constraint representation becomes important to avoid excessive space consumption. We demonstrate how to use BDDs to construct good representations for the `case` constraint which is critical for efficiency. Our results seem comparable to hybrid CP/IP models even though we are only using propagation albeit on ad-hoc global constraints. This paper shows an extensive example of how to systematically build models using different kinds of ad-hoc constraints. It also demonstrates the solving potential of ad-hoc global constraints.

Keywords Modeling · Binary decision diagram · Still-life problem · Ad-hoc constraint · Non-binary constraint

1. Introduction

The game of *Life* was invented by John Horton Conway in the late 1960s and subsequently popularized by Martin Gardner [9]. A particularly difficult problem in the game of Life is the Still-Life problem which is to find a maximum density stable pattern. It has been used by many authors to explore issues in problem formulation, symmetry and hybrid approaches using techniques from Constraint Programming (CP) as well as Integer Programming (IP).

In this paper, we have two objectives. Firstly, since Still-Life is considered to be difficult to solve by CP techniques, we want to investigate how to obtain better

K. C. K. Cheng · R. H. C. Yap (✉)
National University of Singapore, School of Computing,
3 Science Drive 2, Singapore
e-mail: ryap@comp.nus.edu.sg

K. C. K. Cheng
e-mail: chengchi@comp.nus.edu.sg

constraint propagation and search techniques for Still-Life. Secondly, and more importantly, we use Still-Life as a (difficult) representative problem for applying global constraints. Still-Life is a good reference problem for investigating global constraints. Often effective solutions for difficult problems rely on strong propagation provided by special purpose global constraints with specialized algorithms such as `all_different` or `cumulative`. However, it is not clear if such global propagators are relevant for modeling this problem. The best propagation result is with a hybrid of CP and IP. In this paper, we want to show that "pure propagation" is effective by making use of global ad-hoc constraints. We utilize the `case` global constraint. Thus, this paper is also a good case study on how to use the `case` constraint.

We begin with a basic model which is a straightforward modeling of the life, death and birth constraints of Conway's game of Life. We show how to come up with additional ad-hoc global constraints which can be used to get better propagation for the Still-Life problem. By ad-hoc global constraints, we mean arbitrary non-binary constraints. It is common in the CSP literature to work with an extensional representation for non-binary constraints. This is impractical (and intractable) when there are many feasible solutions to the constraints, making the representation too big. In our models, we make use of high arity ad-hoc constraints whose extensional representation grows exponentially in the number of variables. For example, the extensional representation of a 3×10 super-row constraint has 76 million tuples and 30 variables. We demonstrate effective methods of using BDDs to construct `case` constraints for such large ad-hoc constraints.

One might expect with the combination of high arity and very large solution spaces for the ad-hoc constraints that solving and propagation would be too expensive in time. We demonstrate using our methodology, because we focus on compact representations, that it is possible to construct effective global `case` constraints even for very big ad-hoc constraints. Although solving the ad-hoc constraints can be expensive, by effective modeling, one can get much stronger propagation than non-ad-hoc and non-binary constraints such as arithmetic. Our objective is not to get the best results for Still-Life. The best known results are with using bucket elimination [11]. Rather, we are interested in showing that it is possible to significantly improve on pure CP propagation based techniques. Furthermore, the results are competitive with CP/IP hybrids which use very sophisticated IP solvers such as CPLEX.

2. The Still-Life Problem

The game of Life is played on an infinite (checker) board where each square is called a *cell*. Each cell has eight neighbors. A cell is *alive* if there is a checker on it. Otherwise it is *dead*. The state of a board at time t determines its state at time $t+1$ based on the following three rules (see Fig. 1):

– If a cell has exactly two living neighbors at time t, its state remains unchanged at time $t+1$. This is the "life" constraint.
– If a cell has exactly three living neighbors at time t, then it is alive at time $t+1$. This is the "birth" constraint.
– If a cell has less than two or more than three living neighbors at time t, it is dead at time $t+1$. These are the "death by isolation" and "death by over-crowding" constraints, respectively.

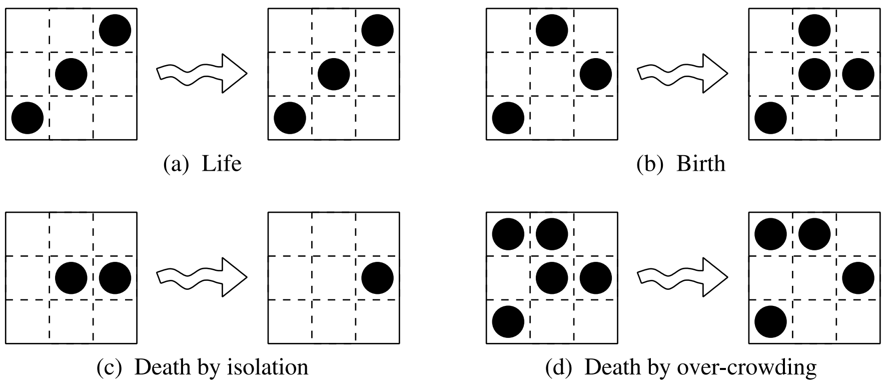


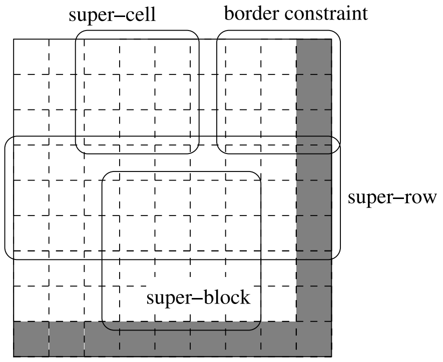
Fig. 1 The rules for the game of Life

It is natural from the problem definition to think of a cell and its eight neighbors as forming a "unit." Smith [13] calls such 3×3 square of cells a *super-cell*. We expand this view and introduce three more super-units: a *super-row* is a $3 \times n$ rectangle; a *super-column* is a $n \times 3$ rectangle; and a *super-block* is a 4×4 square of the board. Super-cells and other super-units will be the basis for constructing various models.

Figure 2 gives the graphical representation of an 8×8 board, and various super-units. Since a super-column is a transposed super-row, we will not further discuss super-columns as the methods for super-rows will hold. In the remainder of the paper, we can assume without loss of generality that n is a multiple of 3 since it is always possible to enlarge the board by padding dead cells to size $n' \times n'$ where n' is a multiple of 3.

A *Still-Life* pattern of a board is one that remains unchanged over time. The *density* of a region is the number of living cells within that region. The *Still-Life* problem in an $n \times n$ square of a board (all the rest of the board is dead) is to determine a Still-Life pattern with the maximum density. Figure 3 shows a 3×3 Still-Life pattern. A larger 9×9 Still-Life pattern is shown in Fig. 4.

Fig. 2 A 8×8 board made to 9×9 by padding dead cells (shaded). A super-cell is any 3×3 square of cells. A super-row is a $3 \times n$ (here $n = 9$) rectangle of cells. A super-block is a 4×4 square of cells



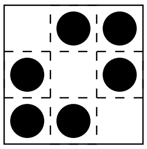


Fig. 3 A 3×3 still-life pattern

2.1. The Basic Model $\mathcal{M}_0$

Still-Life in a $n \times n$ region can be modeled in a straightforward fashion as a CSP. Each cell at the i-th row and the j-th column is associated with a 0/1 variable $x_{i,j}$ which is 1 if the cell is alive and is 0 otherwise. Throughout this paper, we will abuse notation slightly when referring to a variable to mean either the variable or the object it represents. Let $N_{i,j} = \{x_{i+d,j+e} | \ d, e \in \{-1, 0, 1\} \wedge d^2 + e^2 \neq 0\}$ be the neighbors of $x_{i,j}$. The birth and death conditions can be formulated as

$$SC_{i,j} \equiv (x_{i,j} = 1 \rightarrow 2 \leq \sum_{u \in N_{i,j}} u \leq 3) \wedge (x_{i,j} = 0 \rightarrow \sum_{u \in N_{i,j}} u \neq 3).$$

We call $SC_{i,j}$ a *super-cell (SC) constraint*. The arity of a $SC_{i,j}$ constraint is 9. Extra constraints are added on every three adjacent cells along the border to forbid all of them from being alive, otherwise a cell outside the border would become alive. Thus, as is shown in Fig. 2, a super-cell on the border, a *border constraint*, will have the shaded cells set to 0.

The objective of the Still-Life problem is to maximize the density of alive cells, i.e., *max f*, where f is

$$f = \sum_{1 \leq i,j \leq n} x_{i,j}.$$

We call this model which employs only super-cell constraints $\mathcal{M}_0$. In model $\mathcal{M}_0$, the SC constraint is implemented using arithmetic constraints and reified constraints.[1]

3. New Improved Still-Life Models

As reported by Bosch and Trick [3], the super-cell constraints are too weak for constraint propagation to be effective, which results in late detection of local inconsistency and a huge search tree. They propose a hybrid CP approach combining CP with IP. They show that the use of IP to provide global constraints can significantly reduce the search space with a large improvement in overall search time. Smith [13] uses a dual encoding on the super-cells implemented with the `table` constraint in ILOG Solver. This paper is inspired by these two approaches and we use both super-cells and a variety of other ad-hoc constraints.

One possible way to improve the efficiency of constraint propagation is to combine several constraints into a single larger constraint. The idea here is to see if

[1] Although it could also be implemented as a 9 variable `case` constraint, as in Fig. 7.

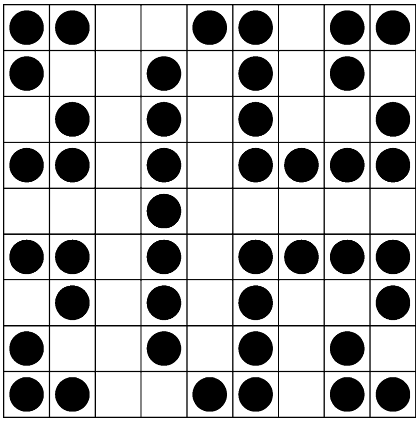


Fig. 4 A 9×9 still-life pattern

the combination can be used to get more propagation. In the rest of the paper, we show how to model Still-Life using increasingly complex ad-hoc constraints as follows:

– In model $\mathcal{M}_r$, a chain of super-cell constraints along a super-row are replaced by a single super-row constraint.
– In the model $\mathcal{M}_{r+d}$, a super-row density constraint is used in which the density of a super-row is "pushed" into the constraint itself.
– In the model $\mathcal{M}_{r+d+b}$, we join also the border constraints with the super-row density constraint.
– In the model $\mathcal{M}_{r+d+b} + SB$, instead of using super-cell constraints to link the super-rows and columns, we replace four super-cells constraints with a single super-block constraint.
– The model $\mathcal{M}_{r+d+b} + SB + DY$ adds constraints to link the density with the variables in a super-cell.
– In the model $\mathcal{M}_{r+d+b} + SB + Emb$, we insert constraints that partially link the density with selected cells in a super-row.
– The model $\mathcal{M}_{r+d+b} + SB + DY + Emb$ includes all the constraints together.

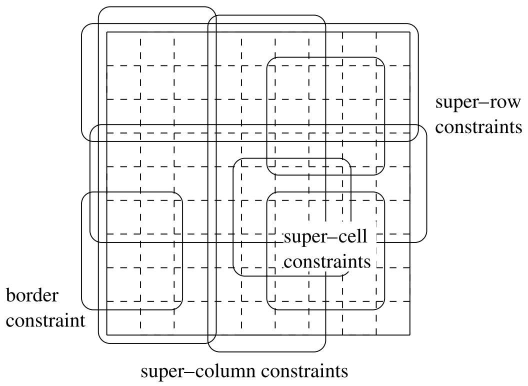
3.1. Model $\mathcal{M}_r$: Super-Rows and Super-Columns

The idea here is to investigate whether a super-cell constraint can be extended further. A natural extension is to consider a row or column of super-cells. A *super-row* (respectively *super-column*) is a horizontal $3 \times n$ rectangle (respectively $n \times 3$ rectangle for a super-column).

Model, $\mathcal{M}_r$, consists of the following constraints:

– A disjoint set of super-row SR_i constraints which partition the board into $3 \times n$ rectangles. Super-column constraints similarly partition the board in the vertical direction.
– A set of border constraints. These are as in model $\mathcal{M}_0$.
– A set of overlapping super-cell constraints that link up two adjacent super-rows. Similarly, these super-cell constraints also link up adjacent super-columns.

Fig. 5 Graphical representation of $\mathcal{M}_r$



These constraints are illustrated in Fig. 5. The super-cell constraints are needed since a super-row or super-cell does not consider cells outside its boundary. We do not replace all super-cell constraints with overlapping super-row and super-column constraints because of the high runtime overhead.

3.1.1. Modeling and Representing the Super-Row Constraint

The question then with model $\mathcal{M}_r$ is how to represent the super-row constraint which is an ad-hoc constraint. There are three different approaches to represent an ad-hoc constraint.

The most straightforward approach is to store the (non-)solutions defining the ad-hoc constraint in an extensional form [2], e.g. an array or table. This is the most common approach used in the CSP literature for dealing with general CSP problems. The `table` constraint in ILOG Solver provides this way of defining ad-hoc constraints. The main drawback is that the size of the table is determined by the number of (non-)solutions to the constraint. This can however grow exponentially with constraint arity. In practice this means that the extensional form cannot be too large, i.e., the constraints are tight (also loose constraints where the set of invalid tuples is not large) and/or the arity is small. Unfortunately in the case of super-row constraints, the arity is high and the number of solutions can be large, e.g., a 3×7 super-row has 21 variables and 145365 solutions.

The second approach is to identify or extract some arithmetic or symbolic relations from the solution set (e.g., [6, 8]). This relies on the use of global constraints which have special semantic properties which leads to a compact representation. However this approach often requires expensive pre-processing. The actual propagation provided by such specialized constraints might not be strong since one has reformulated the original constraints. Since we aim for stronger propagation and the propagation of super-cell constraints is weak, we cannot use this approach.

The last approach is to represent the solution set in some compact data structure and build a tailor-made propagation algorithm on top of it (e.g., [1]). The difference with the extensional approach is that here we try to take advantage of a special representation with special algorithms. The ad-hoc non-binary `case` constraint provided by SICStus Prolog [5] belongs to this category. We will show why it is very useful in this paper for dealing with ad-hoc constraints.

One can specify the level of consistency which the case constraint enforces, such as bounds consistency or generalized arc consistency (GAC). We use GAC (i.e., with the options on(dom(X)) and prune(dom(X))) for all ad-hoc constraints in all models. To use the case constraint, the solutions of the ad-hoc constraint should be represented as a directed acyclic graph (DAG) which is recursively defined as follows. A case DAG whose root is of the form node$(G, x, [r_1 - G_1, \ldots, r_m - G_m])$ defines the constraint

$$\langle G \rangle \equiv \bigvee_{k=1}^{m} (x \in r_k \wedge \langle G_k \rangle)$$

where each G_k is a case DAG. For instance, Fig. 6 depicts a DAG representation of C_{adhoc} with solutions

$$\{(x,1),(y,3)\},\ \{(x,2),(y,3)\},\ \{(x,2),(y,4)\},\ \{(x,3),(y,1)\},\ \{(x,3),(y,2)\},$$
$$\{(x,3),(y,4)\},\ \{(x,3),(y,5)\},\ \{(x,5),(y,3)\}$$

Obviously, the efficiency of case should be related to the compactness of the DAG used.[2] For example, suppose there are two case DAGs representing the same constraint. Assuming a sufficient size difference, we might expect that the smaller DAG should be used. The problem of finding a compact case DAG is non-trivial. This question is not addressed by SICStus Prolog which assumes that the user comes up with a DAG for case. Our approach is to make use of the fact that each $x_{i,j}$ in Still-Life is Boolean which allows us to construct a binary decision diagram (BDD) of the ad-hoc constraint. This can then be converted to a case DAG. In our experiments, we use the BDD package BuDDy 2.4 to manipulate BDDs.[3]

Binary decision diagram (BDD) [4] is a state of the art representation for propositional logic formulas which is heavily used in CAD. We can view a BDD as a special form of a case DAG: a BDD node rooted at node$(G, x, [(0..0) - G_0, (1..1) - G_1])$ defines the constraint

$$\langle G \rangle \equiv (x = 0 \wedge \langle G \rangle) \vee (x = 1 \wedge \langle G \rangle)$$

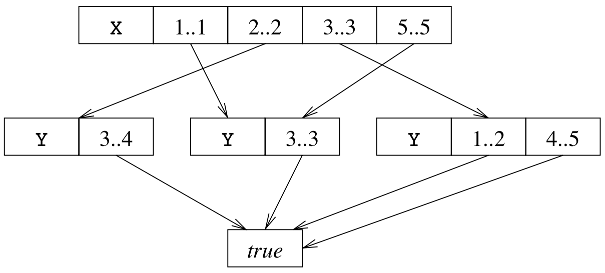
where G_0 and G_1 are BDDs.[4] In BDD terms G_0 is the *0-successor* and G_1 is the *1-successor* of x. A BDD has two terminals, namely the *0-terminal* which means *false* and the *1-terminal* which means *true*. Figure 7 shows the BDD constructed for the super-cell constraint $SC_{2,2}$. Each node x[i, j] represents a cell variable $x_{i,j}$. The solid and the dotted out-going arrows of a node point to its 1 and 0-successors, respectively. The two terminals of a BDD are drawn as two gray boxes. We compare the compactness of the BDD representation with that of the tabular representation as follows. Since $SC_{2,2}$ has 276 solutions, a table constraint needs an array with $276 \times 9 = 2484$ cells to keep all solutions. By comparison, the BDD constraint has only 36 nodes. Since a table cell and a BDD node represent the same variable $x_{i,j}$,

[2] We do not have any specific knowledge about the implementation; it is not described in the SICStus documentation.

[3] http://sourceforge.net/projects/buddy

[4] If there is no ambiguity, we sometimes use G and $\langle G \rangle$ interchangeably.

Fig. 6 A DAG representation of C_{adhoc}



we see that the BDD representation is $2484/36 = 69$ times less nodes (cells) than the equivalent tabular representation.

While a BDD may give a compact representation for many Boolean functions, it can be exponentially large in the worst case. In practice, the BDD size is often sensitive to the underlying variable ordering. As an example, suppose we want to construct a single BDD for $SC_{2,2} \wedge SC_{2,3}$ (the 3×4 super-row). Under the lexicographical ordering (i.e., $x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}, x_{2,1}, \ldots, x_{3,3}, x_{3,4}$), the BDD (Fig. 8c) has 180 nodes. Under a different spiral-like ordering. (depicted in Fig. 8a), namely,

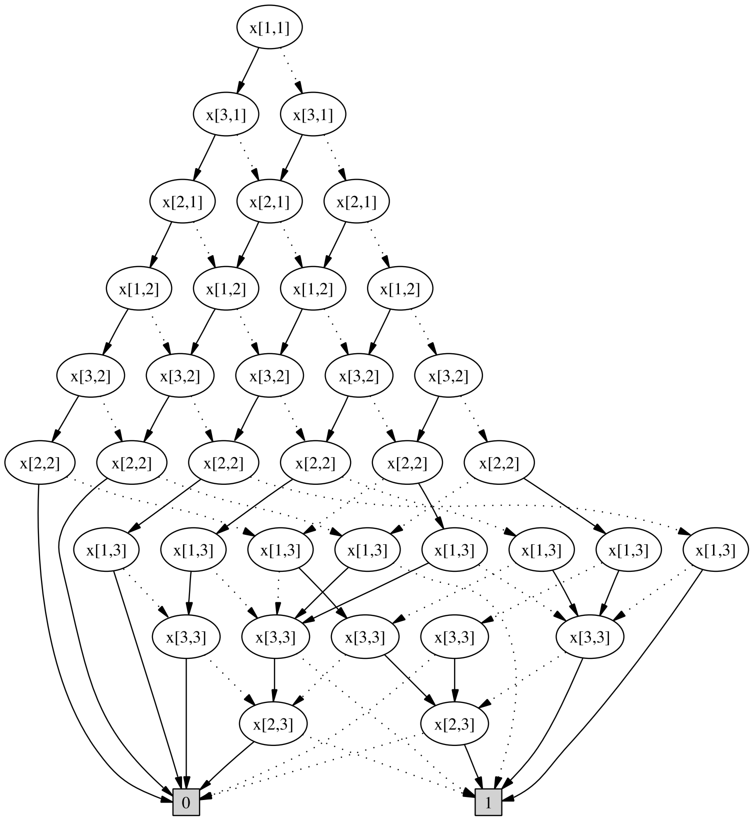


Fig. 7 The BDD representation of the super-cell constraint $SC_{2,2}$

$x_{1,1}, x_{3,1}, x_{2,1}, x_{1,2}, x_{3,2}, \ldots, x_{2,4}$, the BDD has only 95 nodes. Figure 9 shows the sizes of the BDD for the *super-row (SR) constraint*

$$SR_i \equiv \bigwedge_{j=2}^{n-1} SC_{i,j}$$

which is a conjunction of $n-2$ super-cell constraints. We observe that under a "good" (spiral-like) ordering, the size of the BDD grows linearly with n. Under a "bad" (lexicographical) ordering, however, the BDD blows up exponentially. For example, although there are approximately 76 million solutions for the $n=10$ super-row, the good BDD ordering only needs 559 nodes. The bad BDD ordering on the other hand has 1221965 nodes. At $n=40$, the good BDD only needs 2,944 nodes to encode about 1.9×10^{28} solutions. We remark that the spiral-like ordering was obtained by means of some well known guidelines on variable ordering [7], the use of BDD minimization procedures (provided by the BDD package) on small SR constraints (e.g. 3×5) and manual generalization. To the best of our knowledge, no BDD variable ordering heuristic for (general) Boolean constraints is known, which deserves future work.

In all instances in Fig. 9 under the good ordering, the BDD construction time for n from 3 to 40 takes at most 2 s. Thus, although we have constructed a special purpose ad-hoc constraint, the constraint construction time is not significant. Furthermore, this is only done once for a particular value of n. Figure 10 shows the exponential growth in the size of the extensional representation of SR_i where the vertical axis is the logarithm of the number of solutions of SR_i.

3.2. Results for Models $\mathcal{M}_0$ and $\mathcal{M}_\mathrm{r}$

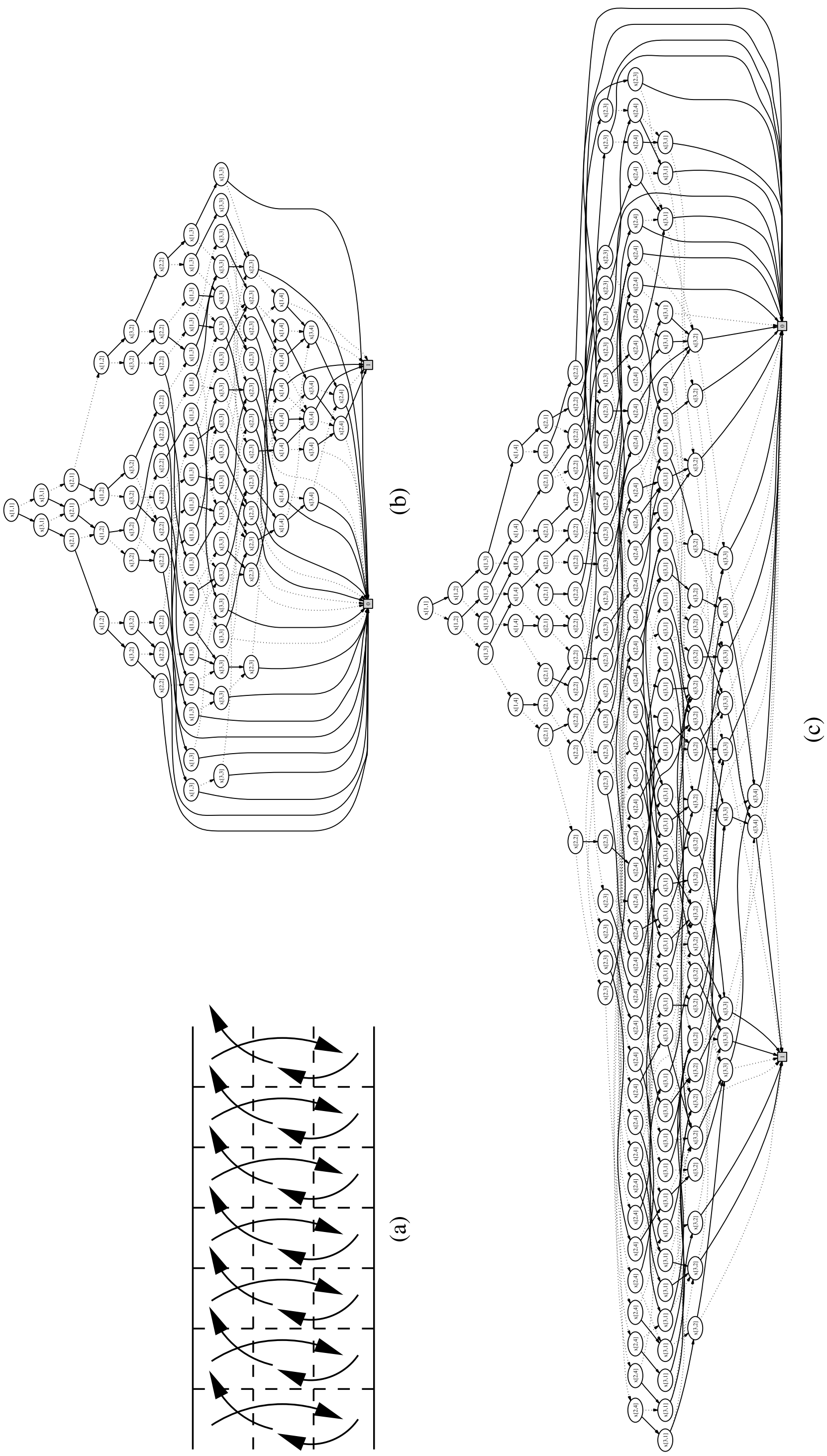
Similar to [12, 13], search is done on a set of $\frac{n^2}{9}$ disjoint super-cells which partition the board. The super-cell is treated as a single auxiliary variable $y_{i,j}$ which represents its associated sub-cells using the following constraint:

$$y_{i,j} = x_{i-1,j-1} + 2x_{i-1,j} + 4x_{i-1,j+1} + 8x_{i,j-1} + 16x_{i,j} + 32x_{i,j+1} + 64x_{i+1,j-1} + 128x_{i+1,j} + 256x_{i+1,j+1}.$$

For example, a 6×6 board is partitioned into four super-cells, namely $y_{2,2}$, $y_{2,5}$, $y_{5,2}$ and $y_{5,5}$. However, unlike in [13], the initial domains of the auxiliary variables are calculated from the constraint given and hence consist of the whole range of integers from 0 to 511.

We use a search strategy where the auxiliary variables are selected lexicographically (i.e., thus super-cells from the bottom right to top left of the board). The value selection is by using domain splitting, i.e., $y_{i,j} \leq d \vee y_{i,j} > d$ where d is the mid-point of the domain. (Domain splitting is called `bisect` in the experiments). The upper region of the domain is explored first. We found this search strategy to be better than smallest domain. A more detailed comparison with other search heuristics can be found in Section 5.

Table 1 summarizes the experimental results for three models. The columns n and *opt* give the maximum density of the Still-Life pattern in an $n \times n$ region. Our experimental platform is SICStus Prolog 3.12.1. Experiments were run on a PC


(a)
(b)
(c)

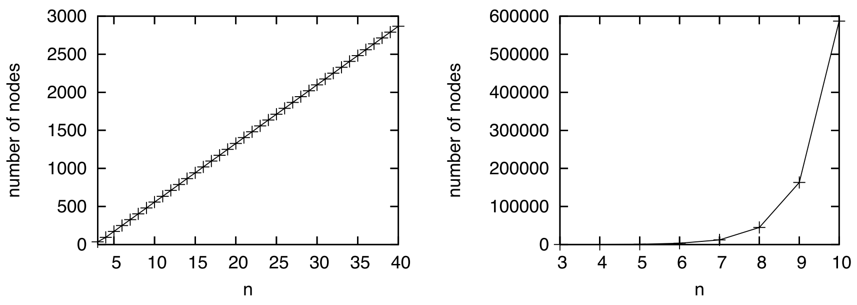


Fig. 9 Sizes of the BDD for SR_i under a "good" (*left*) and a "bad" (*right*) variable ordering

running Windows XP, with a P4 3.2 GHz CPU and 2 GB physical memory. Statistics are collected with the built-in predicate `statistics/2` (we use `runtime` and `memory`). We give the number of backtracks (*bt*), execution time (*time* in seconds) and memory used (*mem* in MB in SICStus) for Still-Life on each model. A time limit of 3,600 s is used. The entries M.O. and T.O., respectively, mean "memory out," memory exhausted error, and "time out," CPU time limit exceeded. Model $\mathcal{M}_r$(SRlex) is the $\mathcal{M}_r$ model where the `case` constraint on super-rows uses the "bad" lexicographic variable ordering. Model $\mathcal{M}_r$, on the other hand, uses the default "good" spiral variable ordering for constructing the `case` constraint.

As we have observed that the super-row constraints are not tight, model $\mathcal{M}_r$ only gives slightly more propagation than $\mathcal{M}_0$ as expected. Here, the overhead of the global constraint outweighs the search pruning and $\mathcal{M}_r$ exceeds the time limit for $n = 10$. The results on $\mathcal{M}_r$(SRlex) show the importance of having a good representation for the ad-hoc constraint: while $\mathcal{M}_r$ used only 3.6 MB for $n = 9$, $\mathcal{M}_r$(SRlex) needed 182.1 MB. $\mathcal{M}_r$ and $\mathcal{M}_r$(SRlex) are equivalent models, and hence give the same propagation. They differ only in the representation of the SR_i constraints. For $n = 10$ with $\mathcal{M}_r$(SRlex), SICStus Prolog aborted with an out of memory error.

Table 2 summarizes the experimental results from other papers: CP [3] which uses only CP super-cell constraints; CP/IP [3] as a hybrid of CP and IP; the dual encoding [13]; and super-cell variables [12]. We caution that a direct time comparison is not meaningful because different systems are used, both the software and hardware differ. Furthermore, the different papers give different number of backtracks due to differences in the model and search procedure. The column *cho* gives the number of choice-points and *bt* the number of backtracks. The super-cell variables is similar to $\mathcal{M}_0$ but it appears that our search heuristic is better for larger *n*. Our model $\mathcal{M}_0$ is similar to that in CP [3] but it also appears that our search heuristic is better.

We have not presented any results using symmetries. We found that the simple use of constraints to remove reflection symmetries did not give much pruning ([3] also found the same for their CP model). More sophisticated symmetry breaking techniques [12] could be used but we did not have a SBDS implementation in

Fig. 8 (a) The BDD variable ordering for super-row constraint. (b) The BDD that represents a 3×4 super-row under the variable ordering in (a) has 95 nodes. (c) The same BDD under lexicographical variable ordering has 180 nodes

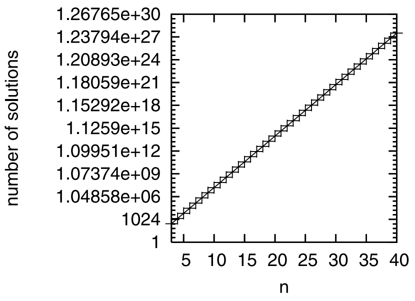


Fig. 10 Number of solutions of SR_i against different n

SICStus Prolog available to use. In any case, symmetry breaking is orthogonal to our goals in this paper.

3.3. Model $\mathcal{M}_{r+d}$: Super-Rows with Density

While the use of super-row constraints achieves slightly better pruning, it is not effective overall since more time is needed to deal with the more complex global super-row constraints. The reason why it doesn't improve propagation much is that the underlying super-cells are loose. Thus, joining them doesn't prune the search space by much.

Since the super-row constraint is loose, it suffers from the problem that unless a large part of a row is instantiated, there is not much interaction between a super-row constraint and the maximum density from the optimization function. Summing the cells in a super-row will get little propagation since most of the cells can be either 0 or 1. Hence, our next step is to modify the super-row constraint to include the density of the super-row itself:

$$SRD_i \equiv (f_i = \sum_{d \in \{-1,0,1\}} \sum_{j=1}^{n} x_{i+d,j}) \wedge SR_i$$

We call this a *super-row density (SRD) constraint.* A *SRD* constraint has $3n + 1$ variables for a row, consisting of the cells in the row and a extra variable, f_i, giving the density of this super-row. (Note we do not include the border constraints).

3.3.1. Generating the Super-Row Density Constraint

Since each super-cell has at most six living cells and it is known that the maximum density has the form $n^2/2 + O(n)$, we can assume a super-row has at least one living cell. As the maximum density of super-cells is six and a super-row can be partitioned into $n/3$ disjoint super-cells, the maximum density of a super-row is $2n$. We combine the various densities, f_i, of a super-row where $1 \leq f_i \leq 2n$ by constructing the `case` DAG for an ad-hoc SRD_i constraint as follows. We first create $2n$ BDDs, each represents the constraint

$$C_k \equiv (k = \sum_{d \in \{-1,0,1\}} \sum_{j=1}^{n} x_{i+d,j}) \wedge SR_i$$

Table 1 Experimental results on $\mathcal{M}_0$, $\mathcal{M}_r$ and $\mathcal{M}_r(SRlex)$

n	*opt*	$\mathcal{M}_0$			$\mathcal{M}_r$			$\mathcal{M}_r(SRlex)$		
		bt	*time*	*mem*	*bt*	*time*	*mem*	*bt*	*time*	*mem*
6	18	2,568	0.3	2.2	2,539	0.4	3.5	2,539	0.4	6.6
7	28	3,512	0.5	2.7	3,438	0.6	3.5	3,438	0.9	15.3
8	36	54,623	7.8	2.7	53,126	9.4	3.6	53,126	23.1	40.4
9	43	2,450,406	314.4	3.5	2,420,472	408.7	3.6	2,420,472	1,700.6	182.1
10	54	25,236,397	3,587.5	3.5		T.O.				M.O.

These are linked with an extra node(SRD_i, f_i, $[(1..1) - C_1, \ldots, (2n..2n) - C_{2n}]$). Effectively, the case DAG represents the constraint

$$SRD_i \equiv \bigvee_{k=1}^{2n} (f_i = k \wedge C_k).$$

Note that the BDDs for any two constraints may overlap, i.e., they may share the same sub-BDDs. This is an important by-product of using BDDs which makes it easier to get a compact representation.

To fully utilize the densities of the super-rows, we add an extra constraint $f = \sum f_i$ which links the densities to the overall density of the board. Similarly, we use SRD_j for super-columns and add another constraint $f = \sum e_j$ where e_j is the density of the j-th super-column. We call this model $\mathcal{M}_{r+d}$.

3.4. Model $\mathcal{M}_{r+d+b}$: Specializing for the Border

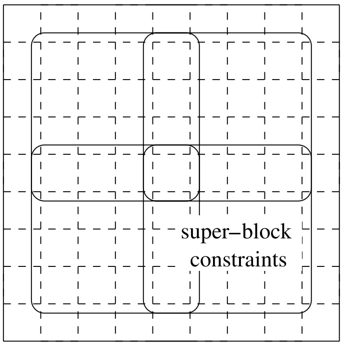
In $\mathcal{M}_r$ or $\mathcal{M}_{r+d}$, the super-row (density) constraints involve only super-cell constraints. In the next model $\mathcal{M}_{r+d+b}$, we obtain an even stronger super-row constraint by joining also the border constraints. For each $1 \leq i \leq n$, define

$$SRDB'_i \equiv SRD_i \wedge \bigwedge_{j \in \{1,n\}} (SC_{i,j} \wedge x_{i-1,j} + x_{i,j} + x_{i+1,j} \leq 3).$$

Table 2 Experimental results on CP/IP and dual encoding

n	*opt*	CP [3]		CP/IP [3]		Dual encoding [13]		Super-cell variables [12]	
		cho	*time*	*cho*	*time*	*bt*	*time*	*bt*	*time*
6	18	24,083	1.6	–	–	181	2.2	1,689	6.4
7	28	154,147	10.1	–	–	3,510	16.2	10,939	48.3
8	36	3,082,922	205.4	2,310	3	53,262	264.0	238,513	1,418.8
9	43	–	–	46,345	85	2,091,386	10,300.0	–	–
10	54	–	–	98,082	291	–	–	–	–
11	64	–	–	268,520	655	–	–	–	–
12	76	–	–	11,772,869	49,166	–	–	–	–
13	90	–	–	10,996,298	50,871	–	–	–	–

Fig. 11 Arrangement of four super-block constraints in a 9×9 board (super-row and super-column constraints not shown; border constraints are needed for $\mathcal{M}_r$ and $\mathcal{M}_{r+d}$)



Then, the super-row constraint with border for rows other than the first and last, $3 \leq i \leq n-2$, is

$$SRDB_i \equiv SRDB'_i.$$

For the first row, we get the constraint

$$SRDB_2 \equiv SRDB'_1 \wedge SRDB'_2.$$

Similarly for the last row, we get the constraint

$$SRDB_{n-1} \equiv SRDB'_{n-1} \wedge SRDB'_n.$$

They are called *super-row density border (SRDB) constraints*.

3.5. Model $\mathcal{M} + SB$: Adding Super-Blocks

Finally, we can replace the four super-cells in a super-block by a single *super-block constraint*

$$SB_{i,j} \equiv \bigwedge_{d \in \{0,1\}} \bigwedge_{e \in \{0,1\}} SC_{i+d,j+e}.$$

The BDD for a super-block constraint has 644 nodes. A model $\mathcal{M}$ which uses super-block constraints is denoted as $\mathcal{M} + SB$. Recall in Fig. 5 that super-cell constraints are needed to connect the super-row and super-column constraints among all models. Hence, we can replace these super-cell constraints with super-block constraints for any model as illustrated in Fig. 11 where one SB replaces four SC constraints.

4. Experimental Results on the Models $\mathcal{M}_{r+d}$, $\mathcal{M}_{r+d+b}$ and $\mathcal{M}_{r+d+b} + SB$

In this section, we present the experimental results on $\mathcal{M}_{r+d}$, $\mathcal{M}_{r+d+b}$ and $\mathcal{M}_{r+d+b} + SB$. We first show in Fig. 12 the sizes and the time to generate the `case` DAGs for the SR, SRD and SRDB constraints for n from 6 to 20. We include two types of SRDB constraints, namely $SRDB_2$ which is the super-row on the board edge and $SRDB_i$ which is the super-row inside. While the size of the SR constraint is

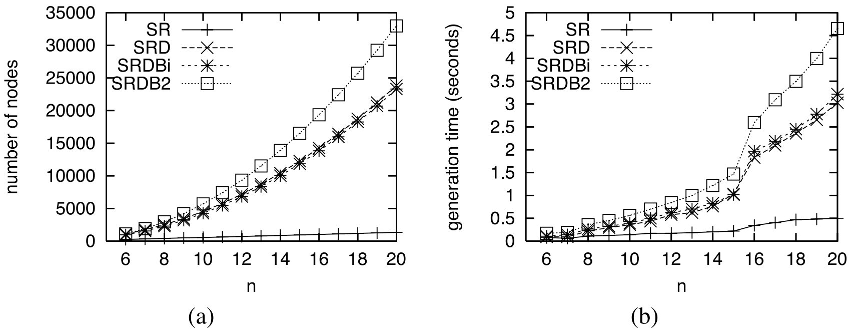


Fig. 12 (a) Number of nodes in the case constraints and (b) generation time of the BDDs for *SR*, *SRD*, $SRDB_i$ and $SRDB_2$

small, both SRD and SRDB have a slow exponential growth. This is because the join of the border and the density function perturbs the "nice" structure of the underlying SR constraint. However, we have not found this to be a problem since n is still small. Furthermore, in SICStus Prolog, we can share the same DAG between different ad-hoc constraints, Thus, for any fixed n, we only need to generate one case DAG for all SR (SRD or SRDB) constraints to share. Different occurrences of the SRD and SRDB constraints in a Still-Life problem instance use the same case DAG but on different sets of variables.

Table 3 lists the experimental results. All three models perform much better than $\mathcal{M}_0$ and $\mathcal{M}_r$. For example, while $n = 10$ is difficult for $\mathcal{M}_0$ and $\mathcal{M}_r$, it becomes much easier for the new models. This justifies the integration of the density function and the super-row constraint. However, the density function alone is not enough as we see $\mathcal{M}_{r+d}$ could not solve $n = 11$ within the time limit. Both $\mathcal{M}_{r+d+b}$ and $\mathcal{M}_{r+d+b} + SB$ could solve $n = 11$. We only attempted $n = 12$ with $\mathcal{M}_{r+d+b}$ and $\mathcal{M}_{r+d+b} + SB$ as this exceeds the time-out and $\mathcal{M}_{r+d}$ timed-out earlier at $n = 11$. Both found the maximum density within 8 h. We see that using *SB* constraints does lead to more pruning but the improvement is not so significant.

When dealing with a search space which might be too large to search systematically, one alternative is to use limited discrepancy search (LDS) [10] to find a good solution. Informally, with respect to our labeling strategy, restricting the discrepancy to k means, on the path leading to the maximum density, there are at most k choice-points in which a low branch (i.e., $y_{i,j} \leq d$) was taken. The assumption

Table 3 Experimental results on $\mathcal{M}_{r+d}$, $\mathcal{M}_{r+d+b}$ and $\mathcal{M}_{r+d+b} + SB$

n	*opt*	$\mathcal{M}_{r+d}$			$\mathcal{M}_{r+d+b}$			$\mathcal{M}_{r+d+b} + SB$		
		bt	*time*	*mem*	*bt*	*time*	*mem*	*bt*	*time*	*mem*
6	18	881	0.2	5.8	51	0.0	5.6	50	0.0	5.6
7	28	737	0.3	6.1	40	0.0	5.9	36	0.0	5.9
8	36	1,881	1.0	6.5	175	0.1	6.1	172	0.1	6.2
9	43	370,340	133.0	6.8	4,507	2.1	7.0	4,224	2.0	6.9
10	54	3,304,788	1,551.5	7.6	75,558	45.9	7.2	60,010	38.9	10.7
11	64		T.O.		1,195,619	1,034.3	7.4	1,134,526	1,013.5	10.9
12	76	–	–	–	34,235,923	28,635.8	14.1	33,164,165	28,388.0	12.3

Table 4 Experimental results on LDS = 2

n	*opt* [11]	$\mathcal{M}_0$		$\mathcal{M}_r$		$\mathcal{M}_{r+d}$		$\mathcal{M}_{r+d} + SB$		$\mathcal{M}_{r+d+b}$		$\mathcal{M}_{r+d+b} + SB$	
		best	*bt*	*best*	*bt*	*best*	*bt*	*best*	*bt*	*best*	*bt*	*best*	*bt*
6	18	**18**	54	**18**	54	**18**	49	**18**	49	**18**	25	**18**	25
7	28	27	119	27	119	27	106	27	100	**28**	39	**28**	36
8	36	**36**	96	**36**	96	**36**	71	**36**	71	**36**	40	**36**	40
9	43	40	184	40	184	40	174	40	174	42	167	42	161
10	54	47	324	47	324	47	324	47	317	**54**	237	**54**	227
11	64	**64**	252	**64**	252	**64**	202	**64**	202	**64**	169	**64**	169
12	76	48	341	48	339	48	339	48	339	75	442	75	438
13	90	77	559	77	561	77	561	77	553	85	689	85	689
14	104	100	499	100	499	100	442	100	442	100	446	100	446
15	119	–	490	–	488	–	488	–	488	109	1,035	109	1,041
16	136	–	811	–	831	–	831	–	822	129	1,304	129	1,307
17	152	144	846	144	846	144	788	144	788	144	876	144	871
18	171	–	657	–	655	–	655	–	655	153	1,636	153	1,639
19	190	–	1,099	–	1,130	–	1,130	–	1,120	177	2,518	180	2,483
20	210	196	1,307	196	1,307	196	1,233	196	1,233	196	1,495	196	1,490

behind LDS is that a good variable ordering heuristic makes fewer mistakes to find a solution. Intuitively this means that LDS has the opportunity for getting better solutions if the constraints inside a CSP model have more propagation. Note that a fairly tight lower bound on the maximum density (usually better than the LDS one) can be obtained by using the symmetrical version of the Still-Life problem. Here, we simply want to investigate the differences between the models and the effect of propagation on the maximum density.

Table 4 presents the nearest maximal density (*best*) LDS can find using a discrepancy factor of 2 for our models. The column *opt* (from [11]) gives the maximal density for different n. A dash indicates that LDS failed and was unable to find a solution within a discrepancy factor of 2. The number of backtracks is also given as a measure of the search effort for LDS. Where LDS(2) gives the optimum, it been emphasized in bold. We see that the $\mathcal{M}_0, \mathcal{M}_r, \mathcal{M}_{r+d}$ and $\mathcal{M}_{r+d} + SB$ mostly give the same results in terms of LDS for up to $n = 11$. In most cases, they manage to find the optimum with the exception of $n = 7$ and 10 where only $\mathcal{M}_{r+d+b}$ and $\mathcal{M}_{r+d+b} + SB$ are able to get the optimum density. While we do not expect that LDS(2) can give good bounds for Still-Life, we can still see the effect of the models with stronger constraints. Firstly, even when the optimum is not obtained, the better propagation of a stronger model does give improved density for the same discrepancy factor, i.e., $\mathcal{M}_{r+d+b}$ gives better bounds than $\mathcal{M}_{r+d}$ for many values of n, and $\mathcal{M}_{r+d+b} + SB$ is better than $\mathcal{M}_{r+d+b}$ for $n = 19$. Secondly, the stronger model is able to find a lower bound with LDS more often than in the weaker models.

5. Yet More Models

Our results suggest ad-hoc constraints are a practical tool to boost the efficiency of propagation. As a result, we decided to experiment with two new models with extra or modified ad-hoc constraints. Our goal is to further strengthen the link among local densities (of super-cells) and other constraints. We will explain the models, followed by the analysis of the empirical results.

5.1. Model $\mathcal{M} + DY$: Linking up Labeling Variables and (Local) Densities

Given any model $\mathcal{M}$, we produce a new model, $\mathcal{M} + DY$, which links the super-cell density with the auxiliary variable. The new ad-hoc constraint has the form

$$DY \equiv \exists_{u_1,\ldots,u_9}\left(d = \sum_{i=1}^{9} u_i \wedge y = \sum_{i=1}^{9} 2^{i-1} u_i\right)$$

where u_i is a Boolean variable and $\exists_x$ means the usual existential quantification, between every pair of the labeling variable $y_{i,j}$ and the density $d_{i,j}$ of its respective super-cell. DY defines the relation between an integer y and the number d of 1's in its binary encoding. The use of DY may lead to more propagation. For example, suppose $y \in \{3, 5\}$. The values 3 and 5 have two 1s in their binary encoding. Hence d for the super-cell can only be 2, which is encoded in the DY constraint. On the other hand, treating the two sums (of u_i's, d and y) independently, we would get $u_1 = 1$, $u_2, u_3 \in 0..1$ and for the rest $u_i = 0$. This gives weaker propagation since we only get $d \in 1..3$.

5.2. Model $\mathcal{M} + Emb$: Local Densities and Super-row Constraints

Recall the super-row density (SRD) constraints (in $\mathcal{M}_{r+d}$) are stronger in propagation than the super-row constraints (in $\mathcal{M}_r$), because of the stronger connection between the density of a super-row and the alive–dead conditions of the cells in it. In the second new model, $\mathcal{M} + Emb$, we are one step further by adding the densities of the disjoint super-cells into the SRD constraints. Without loss of generality, assume $n = 3m$, i.e., a multiple of 3. Our modified SRD constraint is

$$SRD'_i \equiv SRD_i \wedge \bigwedge_{k=1}^{m} \left(d_{i,3k-1} = \sum_{p=-1}^{1} \sum_{q=-1}^{1} x_{i+p,3k-1+q}\right).$$

The arity of SRD'_i is $3n + 1 + m = 10n/3 + 1$. The local densities $d_{i,3k-1}$'s are then linked to the density d_i of the super-row with

$$d_i = \sum_{k=1}^{m} d_{i,3k-1}.$$

However, dealing with the local densities along every path individually will cause an exponential blow-up in the `case` DAG. Our strategy is to "approximate" the values of each $d_{i,3k-1}$ in the DAG. The correctness of the model is unaffected because the use of local densities are solely for adding extra information.

We use a simplified example to describe the idea. Figure 13a shows part of the `case` DAG of a SRD constraint. For the sake of presentation, let us assume a super-cell has only three cells: x_1, x_2 and x_3. Its density is given by $d = x_1 + x_2 + x_3$. Observe that if we consider the exact value of d along each path, the DAG would degenerate into more or less a binary tree. Hence, a practical workaround is to collect the values of d only at certain locations. One may picture this as the use of

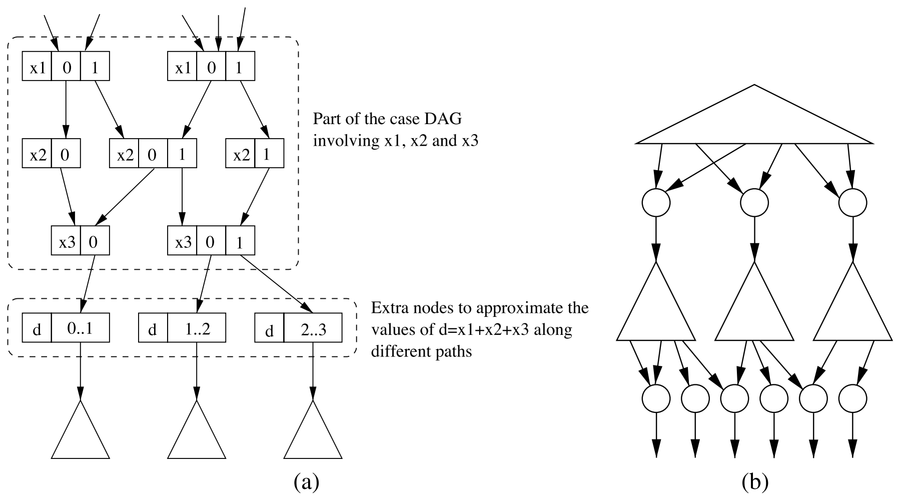

x1
0
1
x2
x3
d
0..1
1..2
2..3
Part of the case DAG involving x1, x2 and x3
Extra nodes to approximate the values of d=x1+x2+x3 along different paths
(a)
(b)

Fig. 13 Part of a `case` DAG for a (simplified) super-row density constraint in (a) micro and (b) macro levels. In (b) the extra nodes are drawn as *circles*. The *triangles* represent the DAGs for the disjoint super-cells in the super-row

some rain collectors for a reservoir. In our example DAG, all paths from the root must pass through some nodes for x_1, x_2 and x_3 and reach three different sub-DAGs. Hence, we insert three nodes for d just above the root of each sub-DAG, where each node captures all possible values of d reaching it. For instance, the left-most node represents $d \in 0..1$. To appreciate the extra pruning, consider when $d = 2$. Propagation on this new constraint immediately tells $x_2 = 1$, which is not known if the density and the super-row are not checked together.

Figure 13b shows how this construction procedure adds extra nodes. In our experiments (next section), the use of *Emb* increases the size of a SRD constraint by about 16%. The time spent on modifying the `case` DAGs varies from 0.02 to 26.78 s.

5.3. Experimental Results on the *DY* and *Emb* Models

Table 5 shows the empirical results on the new models based on $\mathcal{M}_{r+d+b} + SB$, the best model from Section 4.[5] L is the labeling option (see below). A descending value ordering is used. The column $+\ DY$ gives the number of backtracks and runtime (in seconds) for the model $\mathcal{M}_{r+d+b} + SB + DY$. Similarly the remaining columns for the $+Emb$ and $+DY + Emb$ models. For each instance (n) and labeling strategy (L), the least number of backtracks among models is highlighted in bold. The smallest number of backtracks across all models for each value of n is highlighted by underlining. Time out (T.O.) is set to 3,600 s. We do not list the memory usage so that the table fits within page margins. We found, in the worst case, the memory used by the new models roughly doubles that used by $\mathcal{M}_{r+d+b} + \text{SB}$, but never exceeds 20 MB.

[5] We remark that using the $\mathcal{M}_{r+d+b}$ model gives similar results but using the $\mathcal{M}_{r+d+b} + SB$ model gives less backtracks.

We investigate the difference among three types of value selection heuristics. Bisection is the domain splitting strategy, which we call `bisect`. The other strategies are: two-way branching, i.e., $x = d$ or $x \neq d$, called `step`; and enumeration, i.e., $x = d$ for each $d \in D(x)$. Regardless of the model in use, domain splitting (`bisect`) gives the best overall result in terms of both number of backtracks and execution time. Two-way branching (`step`), always has less backtracks, but is sometimes slower (for the new models), than enumeration (`enum`). Neither `step` nor `enum` can solve for $n = 11$ within an hour. One reason for the huge number of backtracks by `enum` and `step` is that, as the constraints are weak and the domain of the labeling variables are large, instantiating a variable near the top of the search tree may have little pruning. For instance, assigning 3, 5 or 6 to $y_{i,j}$ gives the same density $d_{i,j} = 2$ (assume no other constraint). As a result, `enum` and `step` are more likely to commit to some wrong, deep search space than `bisect`. Two-way branching is slower than enumeration for the new models because the former generally triggers more constraint checks (for both $x = d$ and $x \neq d$) than the latter (for $x = d$ only). When most checks are fruitless, the overhead of the new constraints becomes the dominating factor. Similar observations on two-way branching and enumeration are also reported by Smith and Sturdy [14].

We also compare first-fail (`ff`) with the lexicographic (default) variable ordering. We found an interesting result that for our models, `ff` almost always results in more backtracks when combined with `bisect`. When `enum` or `step` is used, `ff` is often a better choice when $n \leq 9$, with a few exceptions mainly when $n = 7$ with $+DY$. However, when $n = 10$, the number of backtracks produced by `ff` is an order of magnitude more than that by the lexicographical ordering. Overall, the lexicographical variable ordering is more robust for our models of the Still-Life problem.

We now analyze how the new models perform. Dramatically, when domain splitting is used, the new models often ($n \geq 8$) have more backtracks and take longer to finish execution. When we inspected the search trees, we found the new constraints indeed give more pruning, i.e., more "holes" in the domains of the variables. However, these holes (or bounds) seem to upset the balance of the search tree by shifting the domain splitting positions. Since for Still-Life (or generally for optimization problem) we are likely to visit a large part of the search tree, making the tree unbalanced may lead to a longer traversal. To investigate this phenomenon, we tested our four models with fixed splitting positions (i.e., the initial lower and upper bounds are always 0 and 511, respectively, and every subsequent splitting position is independent to the actual domain). This time almost all models give the same number of backtracks (see Table 6). Some exceptions are when n is 8 or 11, in which $\mathcal{M}_{r+d+b}$+$SB + DY + Emb$ has five less backtracks than others.

Comparing `enum` and `step`, we get results which would be expected, namely less backtracks with the new models. In particular, combining local densities with the super-row density constraints ($+Emb$) reduces the number of backtracks by at least an order of magnitude for all n. On the other hand, $+DY$ is less useful, sometimes even counter-productive (when $n = 7$). The overhead of the new constraints is sometimes significant for larger n. For instance, when $n = 9$, $+Emb$ with `step` and `ff` cannot finish within time limit.

Although the DY and Emb models do not give overall better results than $\mathcal{M}_{r+d+b} + SB$ with `bisect`, they give us some counter-intuitive insights. Our results suggest that, (for domain splitting) the balance of a search tree is as important as the

Table 5 Experimental results on new models

n	opt	L	$\mathcal{M}_{r+d+b}+SB$		$+DY$		$+Emb$		$+DY+Emb$	
			bt	$time$	bt	$time$	bt	$time$	bt	$time$
6	18	`bisect`	50	0.0	**49**	0.0	56	0.0	55	0.0
6	18	`ff` + `bisect`	51	0.0	**49**	0.0	62	0.0	52	0.0
6	18	`step`	1,945	0.1	1,063	0.2	96	0.0	**82**	0.0
6	18	`ff` + `step`	1,534	0.1	903	0.2	112	0.0	**92**	0.0
6	18	`enum`	4,477	0.2	2,619	0.9	261	0.1	**224**	0.1
6	18	`ff` + `enum`	3,815	0.2	2,429	0.9	260	0.1	**222**	0.1
7	28	`bisect`	36	0.0	35	0.0	37	0.0	**34**	0.0
7	28	`ff` + `bisect`	45	0.0	52	0.0	34	0.0	**30**	0.0
7	28	`step`	1,368	0.1	858	0.2	128	0.1	**117**	0.1
7	28	`ff` + `step`	992	0.0	1,451	0.2	77	0.0	**72**	0.0
7	28	`enum`	2,823	0.2	2,041	0.3	247	0.1	**220**	0.1
7	28	`ff` + `enum`	2,902	0.1	3,209	0.2	162	0.0	**150**	0.1
8	36	`bisect`	**172**	0.1	264	0.1	202	0.2	199	0.3
8	36	`ff` + `bisect`	**189**	0.1	299	0.2	219	0.2	209	0.3
8	36	`step`	1,773	0.3	1,602	0.3	364	0.2	**353**	0.3
8	36	`ff` + `step`	1,748	0.3	1,538	0.3	384	0.3	**354**	0.3
8	36	`enum`	3,253	0.4	2,818	0.4	617	0.3	**551**	0.4

8	36	`ff + enum`	3,061	0.4	2,545	0.4	616	0.3	**530**	0.4
9	43	`bisect`	**4,224**	2.0	4,653	4.3	4,946	5.2	4,722	6.6
9	43	`ff + bisect`	**4,276**	2.2	6,487	6.2	4,478	5.5	4,404	7.0
9	43	`step`	215,581	22.3	162,296	43.9	37,699	21.5	**33,002**	22.3
9	43	`ff + step`	158,405	17.7	139,620	37.7	16,476	10.9	**15,967**	12.4
9	43	`enum`	397,714	36.4	293,458	36.0	73,828	14.3	**61,792**	16.3
9	43	`ff + enum`	322,093	27.9	257,653	30.8	29,796	8.9	**28,146**	9.8
10	54	`bisect`	**60,010**	38.9	69,913	61.1	70,237	136.1	66,864	169.9
10	54	`ff + bisect`	**277,263**	182.9	348,248	401.0	737,055	1,425.8	714,733	1,749.2
10	54	`step`	2,769,162	357.6	2,241,745	610.0	542,646	424.3	**500,570**	465.5
10	54	`ff + step`	13,852,027	1,442.6	**12,075,421**	3,180.3		T.O.		T.O.
10	54	`enum`	5,293,938	6,00.2	4,221,366	480.0	949,370	289.7	**839,900**	348.5
10	54	`ff + enum`	27,657,352	2,237.8	23,184,452	2,299.7	6,029,281	2,113.5	**5,657,138**	3,043.1
11	64	`bisect`	**1,134,526**	1,013.5	1,348,722	1,373.4	1,289,849	2,902.0	1,246,668	3,514.5
11	64	`ff + bisect`	**2,881,215**	2,418.6		T.O.		T.O.		T.O.
11		`step`		T.O.		T.O.		T.O.		T.O.
11		`ff + step`		T.O.		T.O.		T.O.		T.O.
11		`enum`		T.O.		T.O.		T.O.		T.O.
11		`ff + enum`		T.O.		T.O.		T.O.		T.O.

Table 6 Experimental results on new models with fixed domain splitting positions

n	*opt*	$\mathcal{M}_{r+d+b}+SB$		$+DY$		$+Emb$		$+DY+Emb$	
		bt	*time*	*bt*	*time*	*bt*	*time*	*bt*	*time*
6	18	92	0.0	92	0.0	92	0.0	92	0.0
7	28	272	0.0	272	0.0	272	0.0	272	0.1
8	36	420	0.1	420	0.1	420	0.2	415	0.3
9	43	7,797	2.2	7,797	4.2	7,797	5.2	7,797	6.4
10	54	151,034	42.0	151,034	58.4	151,034	129.6	151,034	164.6
11	64	2,301,599	1,050.8	2,301,599	1,306.3	2,301,599	2,796.4	2,301,594	3,450.5

actual pruning power of the constraints. An interesting question is whether this holds true for other optimization problems independent of the actual domain.

6. An Overview of the Modeling Approach

Since we have covered many models some of which are quite detailed, it is useful to recap some of the general principles which we have used here. We start with the primitive model $\mathcal{M}_0$ with only weak arithmetic super-cell and border constraints. Without changing the underlying basic model, we have integrated the constraints in the model by combining some structural elements and constraints to get bigger constraints. Essentially with the basic model $\mathcal{M}_0$, we just add on more constraints to get $\mathcal{M}_{\mathrm{r}}$, $\mathcal{M}_{r+d}$ and $\mathcal{M}_{r+d+b}$ (SB, DY and others). Ad-hoc constraints turn out to be useful simply because they give a simple way of creating more complex constraints for more propagation.

From the basic $\mathcal{M}_0$ model, the first extension is to combine super-cells into super-rows. Then, we create $\mathcal{M}_{r+d}$ by pushing the local densities into super-rows. Finally in $\mathcal{M}_{r+d+b}$ super-row density constraints are integrated with border constraints. Together with super-blocks, $\mathcal{M}_{r+d+b}+SB$ is our best model for Still-Life. Moving further along the same strategy, we can further enhance the model with DY and Emb constraints. It turns out that these do not get much improvement with the best strategy and thus just contribute overhead. However we would like to emphasize that we have demonstrated a semi-systematic cookbook methodology which uses global (ad-hoc) constraints to enhance a model. The manual part of the process is to determine the nature of the combination step with the rest of the process being quite systematic. An interesting and future direction is to determine if the models developed here can be obtained through an automated procedure.

7. Discussion

We show that by using ad-hoc constraints, it is possible to get much more propagation for Still-Life. The number of backtracks is significantly better than the CP models in [3, 12, 13]. When comparing with the hybrid CP/IP model, the $\mathcal{M}_{r+d+b}+SB$ seems to be comparable (sometimes more pruning and sometimes less). While it is difficult to compare our ad-hoc global constraints with IP, we conjecture from the results that they seem to capture some of the same pruning power. In their CP/IP model, they had to remove some of the IP constraints in order to get results. This is because solving the additional IP constraints drastically increased the solver time. In our results, the cost of processing the `case` constraints increases with complexity of the constraint. The trade-off here seems to be more manageable.

The best results for Still-Life are with bucket elimination [11]. This is because of the structure of the Life constraints which are quite localized, as such, one can expect that pure CP propagation and search cannot be as good. Bucket elimination however trades time for space and has a growing exponential space requirement. In Still-Life, it turns out that bucket elimination can still solve larger problems than with a pure CP model though this paper narrows the gap. We believe that ideas from this paper are compatible with bucket elimination and this is in scope for further work. We have also not investigated the use of more powerful symmetry removal such as SBDS, this is partly because of lack of support for SBDS in SICStus Prolog. Similarly, it might be interesting to investigate a hybrid of our ad-hoc constraints and an IP model. We have not done so because SICStus Prolog is the only CP system available with the `case` constraint and it does not support any IP solver integration. We emphasize that exploiting symmetries and IP models are not a restriction of this paper but one imposed by availability of the `case` constraint.

The use of ad-hoc constraints is not limited to solving the Still-Life problem. On the contrary, ad-hoc constraints can be considered as a general tool to improve an existing CSP model at a lower level, i.e., when a model is compiled. They are therefore orthogonal to other high level optimization techniques that deal with the semantics of a model, such as the use of implied constraints.

Finally, this paper is an instructive demonstration of how to use the `case` constraint effectively. The `case` constraint is difficult to use precisely because it is unclear how to build the DAGs. We show an approach using BDDs which works well for problems with 0–1 domains. However, since the size of a BDD is sensitive to the BDD variable ordering, to make our approach practical for general problems, heuristics on good variable ordering for CSPs have to be developed.

References

1. Beldiceanu, N. (2000). Global constraints as graph properties on structured networks of elementary constraints of the same type. TR T2000-01, SICS.
2. Bessière, C., & Règin, J.-C. (1997). Arc consistency for general constraint networks: Preliminary results. In *International Joint Conference on Artificial Intelligence (IJCAI),* (pp. 398–404).
3. Bosch, R., & Trick, M. (2004). Constraint programming and hybrid formulations for three Life designs. *Annals of Operations Research, 130*, 41–56.
4. Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers, 35*(8), 667–691.
5. Carlsson, M., et al., (2005). *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, release 3.12.1 edition. ISBN 91-630-3648-7.
6. Cheng, C. K., Lee, J. H. M., & Stuckey, P. J. (2003). Box constraint collections for ad-hoc constraints. In *Principles and Practice of Constraint Programming (CP),* (pp. 214–228).
7. Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model Checking*. The MIT Press.
8. Dao, T. B. H., Lallouet, A., Legtchenko, A., & Martin, L. (2002). Indexical-based solver learning. In *Principles and Practice of Constraint Programming (CP),* (pp. 541–555). (September).
9. Gardner, M. (1970). The fantastic combinations of John Conway's new solitaire game. *Scientific American, 223*, 120–123.

10. Harvey, W. D., & Ginsberg, M. L. (1995). Limited discrepancy search. In *International Joint Conference on Artificial Intelligence (IJCAI)*.
11. Larrosa, J., Morancho, E., & Niso, D. (2005). On the practical applicability of bucket elimination: Still-life as a case study. *Journal of Artificial Intelligence Research, 23*, 21–440.
12. Petrie, K. E., Smith, B. M., & Yorke-Smith, N. (2004). Dynamic symmetry breaking in constraint programming and linear programming hybrids. In *European Starting AI Researcher Symp*.
13. Smith, B. M. (2002). A dual graph translation of a problem in 'life'. In *Principles and Practice of Constraint Programming (CP),* (pp. 402–414).
14. Smith, B. M., & Sturdy, P. (2005). Value ordering for finding all solutions. In *International Joint Conference on Artificial Intelligence (IJCAI),* (pp. 311–316).