Solving 'Still life' with Soft Constraints and Bucket Elimination

Keywords: constraint satisfaction, problem solving

Abstract

In this paper we study the aplicability of *bucket elimination* (BE) to the problem of finding *still-life patterns*. Very recently, it has been tackled using *integer programming* and *constraint programming*, both of them being search-based methods. We show that BE, which is based on *dynamic programming*, provides an exponentially lower worst-case complexity than search methods. Then, we empirically show that BE is quite competitive with search-based approaches. Finally, we show how BE can be adapted to exploit the problem symmetries, with which in several cases we outperform previous results.

1 Introduction

The game of *life* was invented in the late 60s by John Horton Conway and was later popularized by Martin Gardner [Gardner, 1970]. Given an infinite checkerboard, the only player places checkers on some of its squares. Each square is a *cell*. If there is a checker on it, the cell is *alive*, else it is *dead*. Each cell has eight *neighbors*: the eight cells that share one or two corners with it. The state of the board evolves iteratively according to three rules: (i) if a cell has exactly two living neighbors then its state remains the same in the next iteration, (ii) if a cell has exactly three living neighbors then it is alive in the next iteration and (iii) if a cell has fewer than two or more than three living neighbors, then it is dead in the next iteration.

While conceptually simple, the game has proven mathematically interesting and has attracted a lot of curiosity, as can be seen in,

`home.interserv.com/~mniemiec/lifepage.htm`

Maximum density stable patterns (also called *still lifes*) are board configurations with a maximal number of living cells which do not change along time. They can be seen as an academic simplification of a standard issue in discrete dynamic systems. [Elkies, 1998] has shown that for the infinite board the maximum density is $1/2$. In this paper we are concerned with finite patterns. In particular, we consider $n \times n$ still lifes, for which no polynomial method is known. This problem has been recently included in the *CSPlib*[1] repository of challenging constraint satisfaction problems.

In [Bosch and Trick, 2002] still life is solved using *integer programming* and *constraint programming*, both of them being search-based methods. Their best results were obtained with a hybrid approach which combines the two techniques and exploits the problem symmetries to reduce the search space. With their algorithm, they solved the $n = 15$ case in about 8 days of cpu with a modern computer. Another interesting work can be found in [Smith, 2002] where pure constraint programming techniques are used, and the problem is solved in its dual form. Although not explicitly mentioned, these two works use algorithms with worst-case time complexity $O(2^{(n^2)})$ and polynomial space.

In this paper we find still lifes using *dynamic programming*. We model the problem as a *weighted constraint satisfaction problem* (WCSP) [Bistarelli *et al.*, 1999] and solve it with *bucket elimination* (BE) [Dechter, 1999]. BE is a generic algorithm suitable for many automated reasoning and optimization problems. It is often overlooked due to its exponential space complexity. Here we show that for the still life problem it is highly competitive. In the theoretical side, we show that its time complexity is $\Theta(n^2 \times 2^{3n})$, which means an exponential improvement over search-based methods. Regarding space, the complexity is $\Theta(n \times 2^{2n})$. In the practical side we show that plain BE is much faster than basic search algorithms and comparable to sophisticated search methods. Our implementation of BE solves the $n = 14$ case in less than 30 hours. The $n = 15$ case cannot be solved with our computer due to space exhaustion. A nice feature of BE is that it can compute, with no extra cost, the number of optimal solutions. Thus, we report, for the first time, the number of still lifes up to $n = 14$.

An additional contribution of this paper is that we have adapted BE to exploit some of the problem symmetries, with which the speed is nearly doubled and the space requirement is halved (the $n = 14$ case is solved in about 15 hours, but we still could not solve the $n = 15$ case).

When n is too large to solve optimally with current methods, some authors [Bosch and Trick, 2002; Smith, 2002] find symmetric optimal solutions. We have also adapted BE to solve the problem subject to a vertical reflection symmetry

[1] `www.csplib.org`

and have solved the $n = 28$ case for the first time.

The structure of this paper is as follows: In Section 2 we give preliminary definitions. In Section 3 we show how the still life problem is modelled as a WCSP and solved with BE. In Section 4 we adapt BE to exploit problem symmetries. In Section 5 we modify BE to find symmetrical solutions. Finally, Section 6 summarizes the conclusions of our work.

2 Preliminaries

A *Constraint satisfaction problem* (CSP) [Tsang, 1993] is defined by a tuple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of *variables* taking values from their finite *domains* ($D_i \in D$ is the domain of x_i). C is a set of *constraints*, which prohibit the assignment of some combinations of values. A constraint $c \in C$ is a *relation* over a subset of variables $var(c)$, called its *scope*. For each assignment t of all variables in $var(c)$, $t \in c$ iff t is allowed by the constraint. A *solution* to the CSP is an complete assignment that satisfies every constraint. Constraints can be given explicitly as tables of permitted tuples, or implicitly as mathematical expressions or computing procedures.

Weighted constraint satisfaction problems (WCSP) [Bistarelli *et al.*, 1999] augment the CSP model by letting the user express preferences among solutions. In WCSP, constraints are replaced by cost functions (also called *soft constraints*). Forbidden assignments receive cost ∞. Permitted assignments receive finite costs that express their degree of preference. The *valuation* of an assignment t is the sum of costs of all functions whose scope is assigned by t. A *solution* to the WCSP is a complete assignment with a finite valuation. The task of interest is to *find the solution with the lowest valuation*.

A WCSP instance is graphically depicted by means of its *interaction* or *constraint graph*, which has one node per variable and one edge connecting any two nodes whose variables appear in the same scope of some cost function.

Bucket elimination (BE) [Dechter, 1999; Bertele and Brioschi, 1972] is a generic algorithm that can be used for WCSP solving. It is based upon two operators over functions. For the WCSP case they are:

- The *sum* of two functions f and g denoted $(f + g)$ is a new function with scope $var(f) \cup var(g)$ which returns for each tuple the sum of costs of f and g,

$$(f + g)(t) = f(t) + g(t)$$

- The *elimination* of variable x_i from f, denoted $f \Downarrow i$, is a new function with scope $var(f) - \{x_i\}$ which returns for each tuple t the minimum cost extension of t to x_i,

$$(f \Downarrow i)(t) = \min_{a \in D_i} \{f(t \cdot (x_i, a))\}$$

  where $t \cdot (x_i, a)$ means the extension of t to the assignment of a to x_i. Observe that when f is a unary function (*i.e.*, arity one), eliminating the only variable in its scope produces a constant.

Example 1 *Let $f(x_1, x_2) = x_1 + x_2$ and $g(x_1, x_3) = x_1 x_3$. The sum of f and g is $(f + g)(x_1, x_2, x_3) = x_1 + x_2 + x_1 x_3$. If domains are integers in the interval $[1..10]$, the elimination of x_1 from f is $(f \Downarrow 1))(x_2) = 1 + x_2$. The subsequent elimination of x_2, produces constant 2 (i.e, $((f \Downarrow 1) \Downarrow 2) = 2$).*

function BE(X, D, C)
1. **for** $i = n$ **downto** 1 **do**
2. $\quad B_i := \{f \in C | \, x_i \in var(f)\}$
3. $\quad g_i := (\sum_{f \in B_i} f) \Downarrow i$;
4. $\quad C := (C \cup \{g_i\}) - B_i$;
5. **endfor**
6. $t := \emptyset$;
7. **for** $i = 1$ **to** n **do**
8. $\quad v := \text{argmin}_{a \in D_i} \{(\sum_{f \in B_i} f)(t \cdot (x_i, a))\}$
9. $\quad t := t \cdot (x_i, v)$;
10. **endfor**
11. **return**(C, t);

endfunction

Figure 1: Bucket Elimination. (X, D, C) is the WCSP instance to be solved. The algorithm returns the optimal cost in C and one optimal assignment in t.

In the previous example, resulting functions were expressed intensionally for clarity reasons. Unfortunately, in general, the result of summing functions or eliminating variables cannot be expressed intensionally by algebraic expressions. Therefore, BE stores intermediate results extensionally in tables, which causes its high space complexity.

BE (Figure 1) uses an arbitrary variable ordering o that we assume, without loss of generality, lexicographical (i.e, $o = (x_1, x_2, \ldots, x_n)$). BE works in two phases. In the first phase (lines 1-5), the algorithm eliminates variables one by one, from last to first, according to o. The elimination of variable x_i is done as follows: the algorithm stores the so called *bucket* of x_i, noted B_i, which contains all cost function having x_i in their scope (Line 2). Next, BE computes a new function g_i by summing all functions in B_i and subsequently eliminating x_i (line 3). Then, B_i is replaced by g_i in the problem (line 4). The new problem does not contain x_i (all functions mentioning x_i were removed from C) but preserves the value of the optimal cost. The elimination of the last variable produces an empty-scope function (*i.e.*, a constant) which is the optimal cost of the problem. The second phase (lines 6-10) generates an assignment of variables producing the optimal cost as follows: starting from an empty assignment t (line 6), variables are assigned from first to last according to o. The value for x_i is the best value regarding the extension of t with respect to the sum of functions in B_i (lines 8,9).

An additional feature of BE is that it can easily compute, at no extra cost, the number of optimal solutions. More than that, *all* optimal solutions can be easily retrieved from the buckets computed during the process (see [Dechter, 1999] for details).

The complexity of BE depends on the problem structure, as captured by its constraint graph G, and the ordering o. The *induced graph* of G relative to o, denoted $G^*(o)$, is obtained by processing the nodes in reverse order of o. For each node

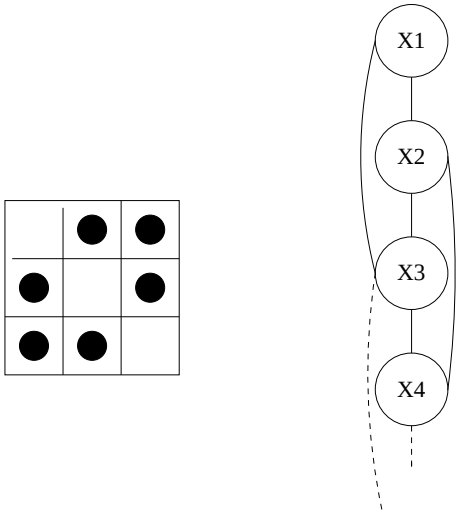

Figure 2: *Left:* A 3×3 still life pattern. *Right:* Constraint graph of still life.

all its earlier neighbors are connected, including neighbors connected by previously added edges. Given a graph and an ordering of its nodes, the *width* of a node is the number of edges connecting it to nodes lower in the ordering. The *induced width of a graph*, denoted $w^*(o)$, is the maximum width of nodes in the induced graph.

Theorem 1 *[Dechter, 1999] The complexity of* BE *along ordering o is time $O(Q \times n \times d^{w^*(o)+1})$ and space $O(n \times d^{w^*(o)})$, where d is the largest domain size and Q is the cost of evaluating cost functions (usually assumed $O(1)$).*

3 Finding still lifes with BE

3.1 Modelling still life as a WCSP

The *still life* problem consist on finding a $n \times n$ stable pattern of maximum density in the game of life, where all cells outside the pattern are assumed to be dead. Considering the rules of the game, it is clear that in stable patterns all *living cells must have exactly two or three living neighbors* and *dead cells must not have three living neighbors*. Besides, *boundary rows and columns must not have more than two adjacent living cells*, since it would produce living cells outside the $n \times n$ region. Figure 2 (left) shows a 3×3 still life.

Still life can be easily modelled as a WCSP. We use a compact formulation with n variables, one for every row. Variable x_i is associated to the i-th row. Its domain D_i is the set of sequences of n bits. The j-th bit of value a, noted a_j, indicates the state of the j-th cell of the row. If a_j takes value 1 the corresponding cell is alive, else it is dead. Let a, b and c be domain values. We define $Z(a)$ as the number of zeroes in a. $S(a,b,c,j)$ is a boolean predicate indicating whether b_j is stable if a, b and c are three consecutive board rows (a is above b, and b is above c).

The problem has n cost functions f_i (with $i = 1,..,n$). For $i = 2,..,n-1$, f_i is ternary, with scope $var(f_i) = \{x_{i-1}, x_i, x_{i+1}\}$. If the arguments represent an unstable configuration it returns ∞, else it returns the number of zeroes in the middle row. Formally,

$$f_i(a,b,c) = \begin{cases} \infty & : \exists_{1 \le j \le n} \neg S(a,b,c,j) \\ \infty & : a_1 = b_1 = c_1 = 1 \\ \infty & : a_n = b_n = c_n = 1 \\ Z(b) & : \text{otherwise} \end{cases}$$

Functions f_1 and f_n are binary. They are equivalent to the ternary cost functions, but assuming dead cells above the top row and below the bottom row, respectively. The scope of f_1 is $\{x_1, x_2\}$ and it is defined as,

$$f_1(b,c) = \begin{cases} \infty & : \exists_{1 \le j \le n} \neg S(\vec{0},b,c,j) \\ Z(b) & : \text{otherwise} \end{cases}$$

where $\vec{0}$ denotes the *all zeroes* string of bits. Similarly, the scope of f_n is $\{x_{n-1}, x_n\}$ and it is defined as,

$$f_n(a,b) = \begin{cases} \infty & : \exists_{1 \le j \le n} \neg S(a,b,\vec{0},j) \\ Z(b) & : \text{otherwise} \end{cases}$$

Note that computing $f_i(a,b,c)$, $f_1(b,c)$ and $f_n(a,b)$ is $\Theta(n)$.

function BE(n)
1. **for** $a, b \in [0..2^n - 1]$ **do**
2. $\quad g_n(a,b) := \mathtt{min}_{c \in [0..2^n-1]}\{f_{n-1}(a,b,c) + f_n(b,c)\}$;
3. **endfor**
4. **for** $i = n-1$ **downto** 3 **do**
5. $\quad$ **for** $a, b \in [0..2^n - 1]$ **do**
6. $\quad\quad g_i(a,b) := \mathtt{min}_{c \in [0..2^n-1]}\{f_{i-1}(a,b,c) + g_{i+1}(b,c)\}$;
7. $\quad$ **endfor**
8. **endfor**
9. $(x_1, x_2) := \mathtt{argmin}_{a,b \in [0..2^n-1]}\{g_3(a,b) + f_1(a,b)\}$;
10. $opt := g_3(x_1, x_2) + f_1(x_1, x_2)$;
11. **for** $i = 3$ **to** n **do**
12. $\quad x_i := \mathtt{argmin}_{c \in [0..2^n-1]}\{f_{i-1}(x_{i-2}, x_{i-1}, c) + g_{i+1}(x_{i-1}, c)\}$;
13. **endfor**
14. $x_n := \mathtt{argmin}_{c \in [0..2^n-1]}\{f_{n-1}(x_{i-2}, x_{i-1}, c) + f_n(x_{i-1}, c)\}$;
15. **return**$(opt, (x_1, x_2, \ldots, x_n))$;

endfunction

Figure 3: Bucket Elimination for the still life problem. The algorithm returns the optimal value in *opt* and the optimal assignment in $(x_1, x_2, \ldots, x_n)$

3.2 BE for still life

The constraint graph of our still life formulation is a sequence of size 3 cliques (Figure 2, right). The induced graph $G^*(o)$ with $o = (x_1, x_2, \ldots, x_n)$ does not have new edges (i.e, $G^*(o) = G$). Consequently, the induced width is $w^*(o) = 2$. Since domains have size 2^n, by Theorem 1, the complexity of BE is time $O(n^2 \times 2^{3n})$ and space $O(n \times 2^{2n})$.

The sequential structure of the constriant graph makes the implementation of BE very simple (see Figure 3). Sequences of bits of size n are represented by integers in the interval $[0..2^n - 1]$. In the first phase, we process variables from last to first. Buckets are implicitelly computed. The bucket of x_n is $B_n = \{f_n, f_{n-1}\}$ (these are the only cost function

having x_n in their scope). B_n is used to compute a new binary cost function g_n with scope $\{x_{n-2}, x_{n-1}\}$ (lines 1-3). By construction, $g_n(a,b)$ is the cost of the best extention of $(x_{n-2} = a,\ x_{n-1} = b)$ to the eliminated variable x_n. The bucket of x_{n-1} is $B_{n-1} = \{g_n, f_{n-2}\}$. It is used to compute g_{n-1} with scope $\{x_{n-3}, x_{n-2}\}$ (lines 5-7, first iteration). $g_{n-1}(a,b)$ is the cost of the best extension of $(x_{n-3} = a,\ x_{n-2} = b)$ to the eliminated variables x_{n-1} and x_n. Subsequent iterations of the loop eliminate subsequent variables. In the last iteration variable x_3 is eliminated. When the algorithm reaches line 9, the current problem contains two cost functions: g_3, which contains the optimal extensions of each potential assignment of x_1 and x_2 to the rest of variables, and f_1. Instead of continuing the elimination of variables, we found more efficient to solve the current problem with a brute-force exhaustive search (line 9). Variables x_1 and x_2 are assigned with their optimal values (line 9) and the optimal cost is assigned to *opt* (line 10).

In the second phase (lines 11-14), we process variables from first to last. We assign to each variable the best value according to its bucket and previously assigned variables.

It is easy to verify the complexity of the algorithm. Regarding space it is $\Theta(n \times 2^{2n})$, due to the space required to store extensively functions g_i, which have 2^{2n} entries each. Regarding time, the critical part of the algorithm is the execution of lines 4-8. Line 6 has complexity $\Theta(n \times 2^n)$ (finding the minimum of 2^n alternatives, the computation of each one being $\Theta(n)$). It has to be executed $\Theta(n \times 2^{2n})$ times, which makes a global complexity of $\Theta(n^2 \times 2^{3n})$. Observe that the complexity of BE in the still life problem is an exponential improvement over search algorithms.

There is a simple average-case time optimization that we found very effective. Observe that lines 2 and 6 require the evaluation of $f_i(a,b,c)$ with a and b fixed and varying c. All values of c such that $f_i(a,b,c) = \infty$ are irrelevant because they cannot provide the minimum valuation. Let u_{ab} be the smallest value such that $f_i(a,b,u_{ab}) \neq \infty$. Clearly line 6 (similarly line 2) can be replaced by:

$$g_i(a,b) := \texttt{min}_{c \in [u_{ab}..2^n-1]}\{f_{i-1}(a,b,c) + g_{i+1}(b,c)\};$$

which in many cases reduces the interval size drastically. Since all f_i in the original problem are essentially equal (the only difference is their scope) value u_{ab} is common to all f_i (with $i = 2..n-1$). For each a, b, we compute u_{ab} during a pre-process and store it in a table that is used to speed up every variable elimination. Note that this table has 2^{2n}. Thus, it does not affect the space complexity of the algorithm.

3.3 Experimental Results

Table 4 reports the results that we obtained with a 1 Ghz Pentium III machine with 1 Gb of memory. From left to right, the first three columns report: problem size, solution cost (as the number of living cells) and number of optimal solutions (most of them have never been reported before). The fourth column reports the CPU time of our executions (BE) in seconds. For comparison purposes, the fifth, sixth and seventh columns show times obtained in [Bosch and Trick, 2002] with basic constraint programming (CP), integer programming (IP), and a sophisticated hybrid algorithm (CP/IP-sym) which exploits the problem symmetries (see Section 4). In their experiments, they used a 650 Mhz Pentium III with 196 Mb of memory. Time comparison should be done with caution, because machines and are different. Note as well that times in [Bosch and Trick, 2002] were obtained using a commercial solver, while our times have been obtained with our *ad-hoc* implementation. On the one side, our implementation was made specifically for the still life problem, which has the advantage of optimizing the use of space and specializing some parts of the code. On the other side, our implementation is a prototype, which is in disadvantage with respect to commercial solvers, developed during months or years. Having said that, it can be observed that BE clearly outperforms basic CP and IP by orders of magnitude. While CP and IP algorithms cannot solve the problem beyond $n = 8$ in less than half an hour, BE can solve the $n = 12$ case subject to the same time limit. The $n = 14$ case is the largest instance that we could solve due to space exhaustion (see Figure 5). As a matter of fact, the original code could not be executed for the $n = 14$ case. We solved it by disabling the *counting solutions* feature which deallocates some memory. We computed the number of solutions in a different execution with a slower machine with more memory space. Comparing BE with the CP/IP hybrid we observe that both algorithms give very similar times (BE is faster, but within the same order of magnitude). Given the simplicity of the BE algorithm we consider it a very satisfactory result. An additional observation is that BE scales up very regularly, each execution requiring roughly eight times more time and four times more space than the previous, which is in clear accordance with the algorithm complexity.

n	cost (n. sol.)	BE	CP	IP	CP/IP-sym
5	16(1)	0	0	1	0
6	18(48)	0	1	23	0
7	28(2)	0	10	7	0
8	36(1)	1	189	65	2
9	43(76)	4	> 1500	> 1500	51
10	54(3590)	27	*	*	147
11	64(73)	210	*	*	373
12	76(129126)	1638	*	*	30370
13	90(1682)	13788	*	*	30729
14	104(11)	10^5	*	*	5×10^5
15	119	*	*	*	7×10^5

Figure 4: Experimental results of four different algorithms on the still life problem. Times are in seconds.

4 Exploiting problem symmetries

Still life is a highly symmetric problem. For any stable pattern, it is possible to create an equivalent pattern by: (i) rotating the board by 90, 180 or 170 degrees, (ii) reflecting the board horizontally, vertically or along one diagonal or (iii) doing any combination of rotations and reflections. Search methods proposed in [Bosch and Trick, 2002] and [Smith, 2002] exploit that fact by cutting off some search paths that only contain solutions that are symmetric of previously processed ones.

In the following we show how BE can also be adapted to

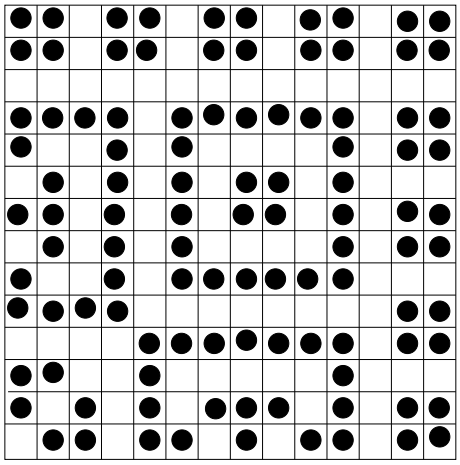

Figure 5: A 14×14 still life pattern.

take advantage of some of the symmetries.

Let's assume that n is an even number (the odd case is similar). Consider the algorithm of Figure 3 and assume that we stop the execution after the elimination of variable $x_{\frac{n}{2}+2}$. The execution of $x_{\frac{n}{2}+2}$ produces $g_{\frac{n}{2}+2}$, with scope $\{x_{\frac{n}{2}}, x_{\frac{n}{2}+1}\}$. Now, we reverse the order of elimination of the remaining variables: The elimination x_1 produces a new function g_1 with scope $\{x_2, x_3\}$, the elimination of variable x_2 produces g_2 with scope $\{x_3, x_4\}$ and so on until the elimination of variable $x_{\frac{n}{2}-1}$, which produces $g_{\frac{n}{2}-1}$ with scope $\{x_{\frac{n}{2}}, x_{\frac{n}{2}+1}\}$. At this point, the current problem has two variables $\{x_{\frac{n}{2}}, x_{\frac{n}{2}+1}\}$ and two cost functions $\{g_{\frac{n}{2}+2}, g_{\frac{n}{2}-1}\}$. Due to the 180 rotation symmetry it is the same to eliminate $x_1, \ldots, x_{\frac{n}{2}-1}$ or rotate the board by 180 degrees and eliminate $x_n, \ldots, x_{\frac{n}{2}+2}$. Therefore, for all a and b it holds that

$$g_{\frac{n}{2}-1}(a, b) = g_{\frac{n}{2}+2}(\bar{b}, \bar{a})$$

Where $\bar{a}$ (respectively, $\bar{b}$) is the reflection of value a (respectively, b). But we do not have to reflect values: due to the vertical reflection symmetry we have that,

$$g_{\frac{n}{2}+2}(\bar{b}, \bar{a}) = g_{\frac{n}{2}+2}(b, a)$$

Consequently, it is unnecessary to eliminate variables $x_1, \ldots, x_{\frac{n}{2}-1}$ because its effect can be obtained from the elimination of $x_n, \ldots, x_{\frac{n}{2}+2}$. Algorithm BE-sym (Figure 6) implements this idea (the second phase is omitted). Since BE-sym eliminates half of the variables, it is roughly twice faster than BE and requires half its memory.

Table 7 reports the results obtained with BE-sym. The first column tells the size of the problem. The second column indicates times obtained with BE-sym. To facilitate comparison, the third column reports results obtained with BE and the fourth column reports the best times obtained by [Bosch and Trick, 2002] with their hybrid CP/IP algorithm which also exploits symmetries (again, be aware of the different machines). Comparing BE *vs.* BE-sym, the experiments confirm that BE-sym is twice faster. Although BE-sym requires less memory than BE, we still could not execute the $n = 15$ case. Comparing it with the CP/IP hybrid, it can be observed that BE-sym seems to be systematically faster.

function BE-sym(n)
1. **for** $a, b \in [0..2^n - 1]$ **do**
2. $\quad g_n(a, b) := \mathtt{min}_{c \in [0..2^n-1]}\{f_{n-1}(a, b, c) + f_n(b, c)\}$;
3. **for** $i = n - 1$ **downto** $n/2 + 2$ **do**
4. $\quad$ **for** $a, b \in [0..2^n - 1]$ **do**
5. $\quad\quad g_i(a, b) := \mathtt{min}_{c \in [0..2^n-1]}\{f_{i-1}(a, b, c) + g_{i+1}(b, c)\}$;
6. **endfor**
7. $opt := \mathtt{min}_{a,b \in [0..2^n-1]}\{g_{\frac{n}{2}+2}(a, b) + g_{\frac{n}{2}+2}(b, a)\}$;
6. **return**(opt);

endfunction

Figure 6: Bucket Elimination exploiting symmetries (assume n even).

n	BE-sym	BE	CP/IP-sym
9	2	4	51
10	14	27	147
11	120	210	373
12	813	1638	30360
13	7223	13788	30729
14	6×10^4	10^5	6×10^5
15	*	*	1.4×10^6

Figure 7: Experimetal results of three algorithms on the still life problem.

5 Restricting to symmetric still life

When n is too large to solve optimally with current methods, previous authors proposed finding symmetric optimal solutions. In [Bosch and Trick, 2002] optimal horizontally symmetric solutions for $n = 18$ are found, and in [Smith, 2002] optimal 90 degrees rotational symmetric solutions for $n = 18$ are also found.

We followed the same approach and adapted BE to consider vertically symmetric patterns. With our formulation, changes are straightforward: we only need to reduce domains to symmetrical values. Lets assume that n is an even number (the odd case is similar). We represent symmetric sequences of bits of lenght n by considering the left side of the sequence (clearly, the symmetrical right part can be obtained by reversing the left part), which can be implemented as integers in the interval $[0..2^{\frac{n}{2}} - 1]$. It is easy to see that the complexity of BE is now time $\Theta(n^2 \times 2^{3n/2})$ and space $\Theta(n \times 2^n)$, which means that the size of problems that we can solve should be doubled. Observe that this problem has exactly the same symmetries as the original problem. Consequently, we can still use the BE-sym algorithm.

Figure 9 reports the results that we obtained with BE-sym. The first column contains the problem size (we only solved even values of n), the second column reports the optimal value as number of living cells, the third column reports the number of solutions and the fourth column reports CPU time obtained with the BE-sym algorithm. As predicted, we solve up to the $n = 28$ case (Figure 8). The $n = 30$ case could not be execute due to space exhaustion. These results improve significatively over the previous works of [Bosch and Trick, 2002; Smith, 2002].

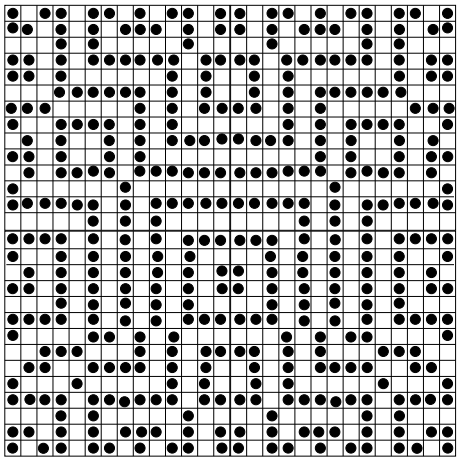

Figure 8: A 28×28 symmetric still life. The optimal value is 406 living cells.

n	opt. cost	n. sol.	BE-sym
10	52	133	0
12	76	8	0
14	104	1	0
16	136	3	0
18	170	4	10
20	208	1813	81
22	252	635	633
24	300	5363	4620
26	350	55246	37600
28	406	12718	1.7×10^5

Figure 9: Experimental results on the obtention of vertical reflection symmetric still lifes with BE.

6 Conclusion

Bucket Elimination is often believed to be an algorithm of little practical interest due to its exponential space complexity. In this paper we showed that it is extremely competitive for the still life problem. We showed that it provides a much lower worst-case time complexity than search-based methods which makes it systematically faster in practice. The space complexity drawback comes to scene where search methods fail due to their exponential time complexity. We reported some results, which we think are new: the number of optimal solutions up to $n = 14$ and the optimal cost and the number of solutions of vertically symmetric still lifes up to $n = 28$.

As far as we know, there is no previous work on how to adapt BE to exploit symmetries. We enhanced the performance of our BE implementation by considering some of the problem symmetries. We belive that it is a preliminary step towards a wider (although possibly limited) practical applicability of BE.

References

[Bertele and Brioschi, 1972] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[Bistarelli *et al.*, 1999] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. *Constraints*, 4:199–240, 1999.

[Bosch and Trick, 2002] R. Bosch and M. Trick. Constraint programming and hybrid formulations for three life designs. In *Proceedings of the International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CP-AI-OR'02*, pages 77–91, 2002.

[Dechter, 1999] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[Elkies, 1998] N.D. Elkies. *The still-life density problem and its generalisations*, pages 228–253. Institute of Math. Kyiv, 1998.

[Gardner, 1970] M. Gardner. The fantastic combinations of john conway's new solitary game. *Scientific American*, 223:120–123, 1970.

[Smith, 2002] B. Smith. A dual graph translation of a problem in life. In *Proc. of CP-2002*, pages 0–1, Ithaca, USA, 2002. LNCS. Springer Verlag.

[Tsang, 1993] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, 1993.