# A Complete Solution to the Maximum Density Still Life Problem☆

Geoffrey Chu, Peter J. Stuckey

*NICTA Victoria Laboratory,*
*Department of Computing and Information Systems,*
*University of Melbourne, Australia*

**Abstract**

The Maximum Density Still Life Problem (CSPLib prob032) is to find the maximum number of live cells that can fit in an $n \times n$ region of an infinite board, so that the board is stable under the rules of Conway's Game of Life. It is considered a very difficult problem and has a raw search space of $O(2^{n^2})$. Previous state of the art methods could only solve up to $n = 20$. We give a powerful reformulation of the problem into one of minimizing "wastage" instead of maximizing the number of live cells. This reformulation allows us to compute very strong upper bounds on the number of live cells, which dramatically reduces the search space. It also gives us significant insights into the nature of the problem. By combining these insights with several powerful techniques: remodeling, lazy clause generation, bounded dynamic programming, relaxations, and custom search, we are able to solve the Maximum Density Still Life Problem for all $n$. This is possible because the Maximum Density Still Life Problem is in fact well behaved mathematically for sufficiently large $n$ (around $n > 200$) and if such very large instances can be solved, then there exists ways to construct provably optimal solutions for all $n$ from a finite set of base solutions. Thus we show that the Maximum Density Still Life Problem has a closed form solution and does not require exponential time to solve.

## 1. Introduction

The *Game of Life* was invented by John Horton Conway and is played on an infinite board made up of square cells. The game takes place through discrete time steps. Each cell $c$ in the board is either alive or dead during each time period. The live/dead state of cell $c$ at time $t+1$, denoted as $state(c, t+1)$, can be obtained from the number $l$ of live neighbors of $c$ at time $t$ and from

☆This paper includes and significantly extends the earlier work [1]

$state(c,t)$ as follows:

$$state(c,t+1) = \begin{cases} l < 2 & \text{dead} & \text{[Death by isolation]} \\ l = 2 & state(c,t) & \text{[Stable condition]} \\ l = 3 & \text{alive} & \text{[Birth condition]} \\ l > 3 & \text{dead} & \text{[Death by overcrowding]} \end{cases}$$

The board is said to be a *still life* at time $t$ if it is stable under these rules, i.e., it is identical at $t+1$. For example, an empty board is a still life. Given a finite $n \times n$ region where all cells outside must be dead, the *Maximum Density Still Life Problem* is to compute the maximum density of live cells that can appear in the $n \times n$ region in a still life, or equivalently, the maximum number of live cells that can appear in the $n \times n$ region.

The raw search space of the Still Life Problem has size $O(2^{n^2})$ and it is extremely difficult even for small values of $n$. Previous search methods using integer programming (IP) [2] and constraint programming (CP) [3] could only solve up to $n = 9$, while a CP/IP hybrid method with symmetry breaking [3] could solve up to $n = 15$. An attempt using bucket elimination [4] reduced the time complexity to $O(n^2 2^{3n})$ but increased the space complexity to $O(n 2^{2n})$. This method could solve up to $n = 14$ before it ran out of memory. A subsequent improvement that combined bucket elimination with search [5] used less memory and was able to solve up to $n = 20$.

In this paper, we combine some mathematical insights into the Still Life Problem with several powerful search techniques to completely solve the problem for all $n$. This is possible because the Still Life Problem becomes well behaved mathematically for sufficiently large $n$ (around $n > 200$). The overall solution plan has four parts: 1) use complete search with a model that propagates strongly to solve the problem for all "small" $n$ ($n \leq 50$), 2) use bounded dynamic programming on a relaxation of the problem to prove a closed form upper bound on live cells for all medium and large $n$ ($n > 50$) 3) use a custom search to look for special form solutions to prove lower bounds on live cells for medium $n$ (around $50 < n \leq 200$), 4) look for special form periodic solutions that can be tiled to construct arbitrarily large solutions to prove lower bounds on live cells for large $n$ (around $n > 200$). The lower and upper bounds proved in parts 2, 3 and 4 coincide, thus they are the optimums for those $n$. Each of these parts require some mathematical insights into the problem as well as the appropriate application of search techniques. We give a brief overview of them here.

*Part 1.* In Section 2 we give a new insightful proof that the maximum density of live cells in the infinite case ($n = \infty$) is $\frac{1}{2}$. The proof is based on counting "wastage". Wastage is calculated by looking at each $3 \times 3$ pattern and seeing how much space we have "wasted" by not fitting in enough live cells into the local area. This proof allows us to reformulate the Maximum Density Still Life Problem into one of minimizing wastage rather than maximizing the number of live cells. The new model gives very tight lower bounds on wastage that dramatically increases the pruning strength of the model. In Section 3 we show how this model, coupled with a simple lookahead, allows a Lazy Clause Generation [6] solver to solve the problem up to around $n = 50$ using complete search.

*Part 2.* In Section 4, we conjecture that for sufficiently large $n$, all wastage which is forced to occur by the still life constraints are forced by only the constraints near the edge of the $n \times n$ region. That is, only the boundary conditions cause suboptimality compared to the optimal density of $\frac{1}{2}$ in the infinite case. If this conjecture holds, then it is possible to get a very good or optimal lower bound on the wastage (and thus upper bound on live cells) simply by relaxing the Still Life Problem onto its boundary and solving it, i.e., ignore all constraints other than then those within the first $k$ rows of the edge of the $n \times n$ region for some small $k$. This relaxed problem has the interesting property that the pathwidth of its constraint graph is $O(k)$ instead of the $O(n)$ of the original. There exists various techniques for solving such low pathwidth problems which can reduce the complexity from $O(2^{nk})$ to only $O(n2^{2k})$, e.g., caching [7], nogood learning [6, 8], dynamic programming [9], variable elimination [4]. In Section 5 we show how to use bounded dynamic programming [10] to solve the boundary relaxation. For fixed and small $k$, these relaxed problems can be solved in $O(n)$ time. Furthermore, due to the translational symmetry in the problem, we can solve the boundary relaxation for all $n$ using induction by examining a finite number of base cases. Thus we can derive a closed form expression which gives a very tight upper bound on the number of live cells.

*Part 3.* In Section 6, we conjecture that for sufficiently large $n$, there always exists optimal solutions of the following form: wastage only exists at the four $4 \times 4$ corners of the board, or in the one row beyond the edge of the board. Based on this conjecture, we search for these special form solutions using a variant of limited discrepancy search with dynamic relaxations as a lookahead. Such a search can find optimal solutions for up to $n = 200$ or so. We know that the solution is optimal if the number of live cells in the solution coincides with the upper bound on live cells proved in part 2.

*Part 4.* The Still Life Problem becomes mathematically well behaved for sufficiently large $n$. This raises the possibility that optimal solutions can be constructed in a systematic way. In Section 7 we find optimal solutions for $n \sim 200$ which are periodic, and which satisfy certain other constraints. If such solutions are found, then they can be tiled indefinitely to produce arbitrarily large, provably optimal solutions.

We conclude the paper in Section 8.

## 2. Wastage Reformulation

The maximum density of live cells in a still life on an infinite board is known to be $\frac{1}{2}$ [11]. However, this proof is quite complex and only applies to the infinite case. In this section we provide a much simpler proof that can easily be extended to the finite case and gives much better insight into the possible sub-patterns that can occur in an optimal solution.

**Theorem 1.** *The maximum density of live cells in a still life on an infinite board is* $\frac{1}{2}$.

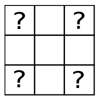
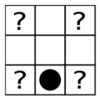
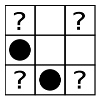
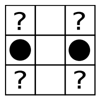
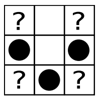
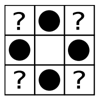
| | | | | | | |
|---|---|---|---|---|---|---|
| Pattern: |  |  |  |  |  |  |
| Beneficiaries | {} | { S } | {S, W} | {E, W} | {E, W} | {} |
| Wastage | 2 | 1 | 0 | 0 | 0 | 2 |

Table 1: Possible patterns around dead cells, showing where they donate their tokens and any wastage.

Proof. Consider any configuration of the board which is a still life. We show that the density of live cells in this configuration is $\leq \frac{1}{2}$. We initially assign 2 tokens to each cell in the board. We will show below that there exists a way to redistribute these tokens such that: 1) each live cell ends up with $\geq 4$ tokens, and 2) each token either remains at the original cell to which it was assigned, or is redistributed to one of the 4 orthogonal neighbours. If such a redistribution exists, then in any $n \times n$ region of the infinite board, if $L$ is the number of live cells, then: $2(n^2 + 4n) \geq 4L$. This is because at most $2(n^2 + 4n)$ tokens could have ended up in the $n \times n$ region after redistribution, and each live cell must have $\geq 4$ of them. Rearranging, we get $L/n^2 \leq \frac{1}{2} + 2/n$, where the LHS is the density of live cells. As $n$ approaches infinity, the RHS approaches $\frac{1}{2}$ and the result follows.

We now describe a redistribution of tokens satisfying the above two properties, i.e., 1) each live cell ends up with $\geq 4$ tokens, and 2) each token either remains at the original cell or is redistributed to one of the 4 neighbours. The redistribution occurs in two phases. In the first phase, the tokens of a dead cell are redistributed only to its orthogonal neighboring live cells, i.e. those that share an edge with the dead cell. Table 1 shows all possible patterns of orthogonal neighbors (up to symmetries) around a dead cell. Live cells are marked with a black dot, dead cells are unmarked, and cells whose state is irrelevant for our purposes are marked with a "?". Each pattern indicates the beneficiaries, i.e., the North, East, South or West neighbors that receive 1 token from the center dead cell, as well as a number indicating the "wastage", which we will discuss later.

As can be seen from Table 1, a dead cell gives 1 token to each of its live orthogonal neighbors if it has $\leq 2$ live orthogonal neighbors, 1 token to the two opposing live orthogonal neighbors if it has 3, and gives no tokens if it has 0 or 4 orthogonal neighbors. Given this set of redistribution rules, it is sufficient to examine the 3 bordering cells on each side of a live cell to determine how many tokens are obtained from the orthogonal neighbor on that side. For example, the token obtained by the central live cell from its South neighbor is illustrated in Table 2.

The tokens obtained by a live cell can therefore be computed by simply adding up the tokens obtained from its four orthogonal neighbors. Since each

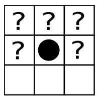
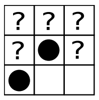
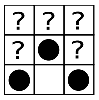
| | | | |
|---|---|---|---|
| Pattern: |  |  |  |
| token received | 1 | 1 | 0 |

Table 2: Contributions to the tokens of a live cell from its South neighbor.

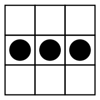
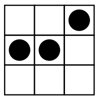
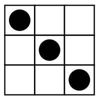
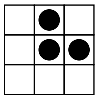
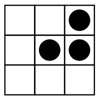
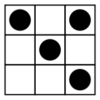
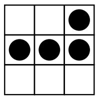
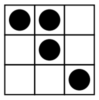
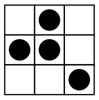
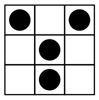
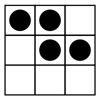
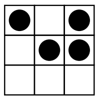
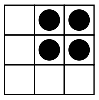
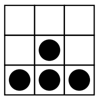
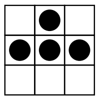
| | | | | | |
|---|---|---|---|---|---|
| Pattern: |  |  |  |  |  |
| Benefactors: | {N,S} | {N,E,S} | {N,E,S,W} | {S,W} | {N,S,W} |
| Wastage: | 0 | 1 | 2 | 0 | 1 |
| Pattern: |  |  |  |  |  |
| Benefactors: | {S,W} | {N,S} | {E,S,W} | {E,S} | {E,W} |
| Wastage: | 0 | 0 | 1 | 0 | 0 |
| Pattern: |  |  |  |  |  |
| Benefactors: | {S,W} | {S,W} | {S,W} | {N,E,W} | {S} |
| Wastage: | 0 | 0 | 0 | 0 | 0 |

Table 3: Possible patterns around a live cell showing token benefactors and any wastage.

live cell starts off with 2 tokens, it must receive at least 2 extra tokens to end up with $\geq 4$ tokens. Let us look at all possible patterns around a live cell and see where the cell will receive tokens from. Table 3 shows all possible neighborhoods of a live cell (up to symmetries). For each pattern, it shows the benefactors, i.e., the North, East, South or West neighbors that give 1 token to the live cell, as well as a number indicating the "wastage", which we will discuss later. As can be seen from the table, after the first redistribution phase, almost every live cell already has the required 4 tokens. The only exceptions are the live cells at the center of the last pattern in Table 3, which only received 1 extra token and so currently has 3 tokens.

In the second phase of redistribution, we fix up these remaining cases. Note that due to the still life constraints, the last two patterns in Table 3 always occur together in unique pairs such that the last pattern is one cell down from the second last. To see this, suppose somewhere, the second last pattern occurs. The bottom middle live cell in the second last pattern already has three live neighbours. The still life constraints says that it cannot have more than 3 live neighbours, thus the next row from the bottom of the second last pattern has to be three dead cells, which then forms the last pattern one cell down as claimed. Similarly, suppose somewhere, the last pattern occurs. The top middle live cell in the last pattern already has three live neighbours. The still life constraints says that it cannot have more than 3 live neighbours, thus the next row from the top of the last pattern has to be three dead cells, which then forms the second last pattern one cell up as claimed.

Also, note that the second last pattern has one extra unneeded token since it received 3 from its neighbours in the first redistribution phase. Thus, in the second phase of redistribution, for each live cell at the center of the second last pattern, we take 1 of the 2 original tokens it had and redistribute it to the live cell at the bottom middle of the pattern, which must be the center live cell of a last pattern. After this, all live cells have $\geq 4$ tokens, and each token either remained where it was originally, or was redistributed to one of the orthogonal neighbours. Thus this redistribution has the two properties we claimed and our proof is complete. $\square$

The above proof is not only much simpler than that of [11], it also provides us with good insight into how to compute useful bounds for the case where the board is finite. In particular, it tells us which $3 \times 3$ patterns are optimal and which are suboptimal with respect to maximizing the number of live cells. Let us define *wastage* as follows. For a dead cell, let the wastage of the cell be the number of tokens which was not given to any of the adjacent live cells during redistribution (shown in Table 1). For a live cell, let the wastage of the cell be the number of excess tokens above 4 it has after redistribution (shown in Table 3). Every live cell must end up with $\geq 4$ tokens after redistribution. Thus every 4 tokens which are *not* used to satisfy the $\geq 4$ token requirement for some live cell reduces the number of live cell in the region by 1. In other words, the $3 \times 3$ patterns which have 0 wastage are precisely the ones which are packing in live cells optimally, whereas the ones with $> 0$ wastage precisely the ones which are suboptimal and should be avoided.

Since the number of live cells and the wastage is inversely related, we can reformulate the objective function in the Maximum Density Still Life Problem as follows. For each cell $c$, let $P(c)$ be the $3 \times 3$ pattern around that cell. Note that if $c$ is on the edge of the $n \times n$ region, the dead cells beyond the edge are also included in this pattern. Let $q(P)$ be the wastage for each $3 \times 3$ pattern as listed in Tables 1 and 3. Define $w(c)$ for each cell $c$ as follows. If $c$ is within the $n \times n$ region, then $w(c) = q(P(c))$. If $c$ is in the row immediately beyond the $n \times n$ region and shares an edge with it (there are $4n$ such cells), then $w(c) = 1$ if the cell in the $n \times n$ region with which it shares an edge is dead, and $w(c) = 0$ otherwise. For all other $c$, let $w(c) = 0$. Let $W = \sum w(c)$ over all cells.

**Theorem 2.** *Wastage and live cells are related by*

$$live\_cells = \frac{n^2}{2} + n - \frac{W}{4} \tag{1}$$

Proof. We adapt the proof for the infinite board to a finite $n \times n$ region. Let us assign 2 tokens to each cell within the $n \times n$ region, 1 token to each of the $4n$ cell in the row immediately beyond the edge of the $n \times n$ region, and 0 tokens to all cells beyond. In the first redistribution phase, for each dead cell within the $n \times n$ region, we redistribute its token among its live orthogonal neighbors exactly as we did before. For each dead cell in the row immediately beyond the $n \times n$ region, we redistribute its 1 token to the cell in the $n \times n$ region with which it shares an edge if it is a live cell and do nothing otherwise.

Once again, the still life constraints force the last two $3 \times 3$ patterns listed in Table 3 to occur in unique pairs. Note that it is impossible for them to exist on the boundary so that one is in the $n \times n$ region and the other not, as that violates the still life constraints. In the second phase of the redistribution, for each live cell at the center of the second last pattern, we take 1 of the 2 original tokens it had and redistribute it to the live cell at the bottom middle of the pattern, which must be the center live cell of a last pattern.

After this, all live cells have $\geq 4$ tokens. The undistributed tokens of a dead cell in the $n \times n$ region is exactly given by the wastage count shown in Table 1. The excess tokens (above 4) of a live cell in the $n \times n$ region is exactly given by the wastage count shown in Table 3. The undistributed tokens of a dead cell in the row just beyond the $n \times n$ region is 1 if the cell in the $n \times n$ region with which it shares an edge is dead, and 0 otherwise. As can be seen, these wastage numbers correspond exactly with $w(c)$ as defined above. Now, 4 times the number of live cells will be equal to the total number of tokens minus the total wastage. Hence we end up with Equation (1). $\square$

We can trivially derive some upper bounds on the number of live cells using this equation. Clearly $W \geq 0$ and, thus, we have: live cells $\leq \lfloor \frac{n^2}{2} + n \rfloor$. Also, by the still life constraints, there cannot be three consecutive live cells along the edge of the $n \times n$ region. Hence, there is always at least 1 wastage per 3 cells along the edge and we can improve the bound to: live cells $\leq \lfloor \frac{n^2}{2} + n - \lfloor \frac{1}{3} n \rfloor \rfloor$. While this bound is very close to the optimal value for small $n$, it differs from the true optimum by $O(n)$ and will diverge from the optimum for large $n$.

## 3. Solving Small $n$ with Complete Search

The power of a branch and bound algorithm is hugely dependent on how strong a bound we can prove on the objective at each node in the search tree. The stronger the bound we can prove, the earlier we can prune off failed subtrees. The naive Still Life model based on counting the number of live cells is very weak, because the upper bound on the number of live cells that propagation can prove is usually very weak and search generally does not fail until the board is at least half filled.

Remodeling the Still Life Problem in terms of minimizing wastage instead of maximizing live cells allows us to propagate much stronger bounds on the objective, as it is easy to tell how much space has already been wasted in the parts of the board that are labeled. Let `sl_waste` be a width 10 table which specifies the wastage value of each $3 \times 3$ pattern satisfying the still life constraints. For example, the entries corresponding to the first and second patterns in Table 3 would be $(0, 1, 0, 0, 1, 0, 0, 1, 0, 0)$ and $(0, 1, 0, 0, 1, 0, 1, 0, 0, 1)$. A single table constraint using `sl_waste` will be sufficient to enforce the still life constraints and set the value of the wastage variable. We propose the following simple MiniZinc [12] model:

```
int: n; % instance parameter

array [0..n+1,0..n+1] of var 0..1: x; % cell live/dead status
array [0..n+1,0..n+1] of var 0..2: w; % cell wastage
var 0..2*n*n+4*n: total_wastage;
var 0..n*n: live_cells;

% still life and wastage constraints in n by n region
constraint forall (i,j in 1..n) (
table(sl_waste, [x[i-1,j-1], x[i,j-1], x[i+1,j-1], x[i-1,j],
x[i,j], x[i+1,j], x[i-1,j+1], x[i,j+1], x[i+1,j+1], w[i,j]]));

% boundary conditions
constraint forall (i in 0..n+1) (
x[i,0] = 0 /\ x[0,i] = 0 /\ x[i,n+1] = 0 /\ x[n+1,i] = 0);

constraint forall (i in 1..n-2) (
sum (j in i..i+2) (x[1,j]) <= 2 /\
sum (j in i..i+2) (x[n,j]) <= 2 /\
sum (j in i..i+2) (x[j,1]) <= 2 /\
sum (j in i..i+2) (x[j,n]) <= 2
);

% wastage constraints for boundary
constraint forall (i in 1..n) (w[i,0] = 1 - x[i,1]);
constraint forall (i in 1..n) (w[0,i] = 1 - x[1,i]);
constraint forall (i in 1..n) (w[i,n+1] = 1 - x[i,n]);
constraint forall (i in 1..n) (w[n+1,i] = 1 - x[n,i]);

% objective function
constraint total_wastage = sum (i,j in 0..n+1) (w[i,j]);
constraint live_cells = (2*n*n+4*n - total_wastage)/4;

solve maximize live_cells;
```

This basic model is capable of counting the wastage in the labeled parts of the board and using it to enforce an upper bound on the number of live cells. However, it is also important to get a good lower bound on the wastage that must occur in the parts of the board which are not yet labeled. The simplest bound we can get is that there must be at least 1 wastage per 3 cells along any unlabeled edge. We can implement this by adding a few lines and modifying the constraint on total wastage:

```
% wastage per 3 edge cells along edge i
array [1..4,1..n/3] of var 1..3: ew;

constraint forall (i in 1..n/3) (
ew[1,i] = sum (j in 3*i-2..3*i) (w[0,j]) /\
ew[2,i] = sum (j in 3*i-2..3*i) (w[j,0]) /\
ew[3,i] = sum (j in 3*i-2..3*i) (w[n+1,j]) /\
ew[4,i] = sum (j in 3*i-2..3*i) (w[j,n+1])
);

constraint total_wastage = sum (i,j in 1..n) (w[i,j]) +
sum (i in 1..4, j in 1.. n/3) (ew[i,j]) +
sum (i in n/3*3+1..n) (w[0,i] + w[i,0] + w[n+1,i] + w[i,n+1]);
```

In this modified model, the wastage of groups of three consecutive edge cells are summed together into variables `ew[i,j]` before being added to the objective. Since each `ew[i,j]` variable has a lower bound of 1, this facilitates the "at least 1 wastage per 3 edge cells" lookahead rule. However, this model is still a bit too weak for the bigger instances, and we use a more advanced lookahead based on relaxations. We defer detailed discussion of this to Section 4.

The search strategy is also important. We use a labeling strategy where we label from the boundary of the board inwards. We first label the first 3 rows of each edge and each $8 \times 8$ corner. Thereafter, we label one row in at a time in concentric squares. The reason for this labeling strategy will become much clearer in view of the insights discussed in Section 4. Basically, most wastage that is forced to occur by the constraints occurs along the boundary of the $n \times n$ region, and thus labeling those cells first increases the bound on the objective the quickest, allowing us to detect suboptimal assignments earlier.

Finally, we must note that a certain feature of the solver we used is critical for solving this problem effectively. One major problem with any search strategy for the Still Life problem is that it is possible to make a "mistake" in labeling that makes the subtree unsatisfiable, but propagation may not be able to notice this until many decision levels later. A normal constraint programming solver will take an exponential amount of nodes to backtrack to this mistake and fix it. The solver we use however, is the Lazy Clause Generation [6] solver CHUFFED, which supports conflict analysis and backjumping. Thus CHUFFED can analyze conflicts and immediately backjump to the mistake and fix it without having to waste an exponential amount of time searching in the failed subtree. If the conflict analysis is turned off so that CHUFFED behaves as a normal CP solver,

| $n$ | opt. | time |
|---|---|---|
| 21 | 232 | 7.0 |
| 22 | 253 | 82.4 |
| 23 | 276 | 5.1 |
| 24 | 302 | 35.4 |
| 25 | 326 | 0.4 |
| 26 | 353 | 116.3 |
| 27 | 379 | 98.9 |
| 28 | 407 | 48.8 |
| 29 | 437 | 76.6 |
| 30 | 467 | 240.8 |
| 31 | 497 | 121.3 |
| 32 | 531 | 86.9 |
| 33 | 563 | 54.2 |
| 34 | 598 | 2.0 |
| 35 | 633 | 155.2 |
| 36 | 668 | 165.1 |
| 37 | 706 | 161.2 |
| 38 | 744 | 193.0 |
| 39 | 782 | 312.2 |
| 40 | 824 | 17.6 |
| 41 | 864 | 89.2 |
| 42 | 907 | 25696 |
| 43 | 949 | 402.3 |
| 44 | 993 | 103.3 |
| 45 | 1039 | 218.4 |
| 46 | 1085 | 403.2 |
| 47 | 1132 | 200.2 |
| 48 | 1181 | 65.3 |
| 49 | 1229 | 456.9 |
| 50 | 1280 | 585.1 |

Table 4: Optimum number of live cells in the Maximum Density Still Life Problem found by complete search and the time in seconds to find and prove them.

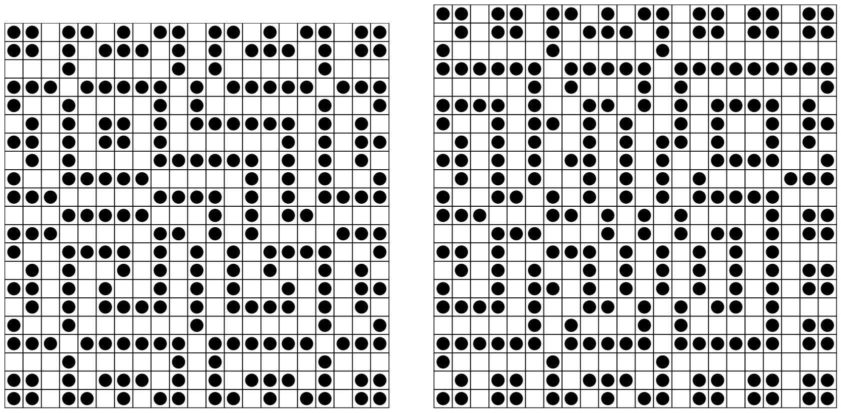


Figure 1: Optimal solutions for $n = 21$ and $n = 22$.

then the optimal solution cannot be found for any $21 < n \leq 30$ within a 1 hour timeout (we did not try larger $n$ where this presumably continues to hold).

In Table 4, we show the results for $21 \leq n \leq 50$ ($n \leq 20$ have previously been solved). Clearly, we are already able to solve instances which are much larger than the previous state of the art methods. Optimal solutions for $n = 21$ and $n = 22$ are shown in Figure 1. We can solve instances somewhat larger than $n = 50$ using complete search, but the run time grows very quickly. Instead, we use better methods to tackle larger $n$ in the next few sections.

## 4. Upper Bounds for Large $n$

To solve the problem for larger $n$, we need more mathematical insight into the problem. We make the following conjecture:

**Conjecture 1.** *For sufficiently large $n$, all "forced" wastage occuring in an optimal soluton is caused by the still life constraints within $k$ rows from the edge of the $n \times n$ region, where $k$ is some small, fixed constant.* □

By "forced" wastage, we mean wastage that is unavoidable due to the constraints of the problem, as opposed to "unforced" wastage that may occur simply because we picked a suboptimal labelling of the board. In the later sections of this paper, we will prove experimentally that Conjecture 1 does indeed hold, but this requires solving the Still Life problem for all $n$. For now, we treat it as a conjecture. Conjecture 1 is inspired by the following two facts. Firstly, in the optimal solutions for $n \leq 20$ that were found by previous methods, it was often the case that wastage only appeared in the corners or within the first 3 rows from the edge. Secondly, we already know that there exist wastage free labellings in the infinite case, thus there is nothing inherent in the still life constraints which force wastage. Instead, it would appear that it is the boundary conditions in the finite case which is forcing the extra wastage to occur.

If the conjecture holds, then it should be possible to relax the Still Life problem onto just the boundary variables (variables within $k$ rows of the edge) and still derive the same wastage lower bound. This has the advantage that: 1) such a relaxed problem has far fewer variables ($O(nk)$ instead of $O(n^2)$) and thus a much smaller search space, 2) such a relaxed problem has low pathwidth ($O(k)$ instead of $O(n)$) and there exist various techniques that can take advantage of this to reduce the search space even further to $O(n2^{2k})$, e.g., caching, nogood learning, dynamic programming, variable elimination.

It is well known that any lower bound we prove for the objective function in the relaxed problem is also a valid lower bound for the original problem. However, this bound may or may not be the optimal bound for the original problem. If we relaxed the problem too much, then the bound we derive from the relaxed problem will be weaker than the optimal bound for the original problem. Now, if Conjecture 1 holds, then there should exist some small $k$ such that the bound from relaxing the problem onto a width $k$ boundary should still be optimal for the original problem. We need to find this $k$.

The complexity of solving the relaxation is $O(n2^{2k})$, so choosing $k$ too large will make the problem intractable. We wish to find the smallest $k$ such that the relaxation is sufficient to prove the optimal bounds for the original problem. We performed a series of experiments to try to guess what this minimal $k$ is. We define the edge still life problem $edge(n, k)$ in MiniZinc as follows:

```
int: n, k; % instance parameters

array [0..n+1,0..k+1] of var 0..1: x; % cell live/dead status
array [1..n,0..k] of var 0..2: w; % cell wastage

% still life and wastage constraints in n by k region
constraint forall (i in 1..n, j in 1..k) (
table(wastage, [x[i-1,j-1], x[i,j-1], x[i+1,j-1], x[i-1,j],
x[i,j], x[i+1,j], x[i-1,j+1], x[i,j+1], x[i+1,j+1], w[i,j]]));
```

| $k$ | $r(k)$ (exact) | $r(k)$ (decimal) | castles | blocks |
|---|---|---|---|---|
| 1-2 | 1/3 | 0.333 | 0 | 1 |
| 3-11 | 4/11 | 0.364 | 1 | 1 |
| 12 | 7/19 | 0.368 | 2 | 1 |
| $\geq$ 13 | 10/27 | 0.370 | 3 | 1 |

Table 5: Ratio of wastage to edge cells in an optimal edge for different edge width, and number of patterns that appear in the first three rows.

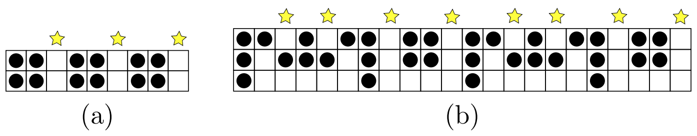


Figure 2: Optimal edge patterns for a width 2 and width 3 boundary relaxation. Wastage is highlighted with a star.

```
% boundary conditions
constraint forall (i in 0..n+1) (x[i,0] = 0);

constraint forall (i in 1..n-2) ( sum (j in i..i+2)(x[1,j]) <= 2);

% wastage constraints for boundary
constraint forall (i in 1..n) (w[i,0] = 1 - x[i,1]);

% objective function
solve minimize sum (i in 1..n, j in 0..k) (w[i,j]);
```

Let $m(n,k)$ be the minimum wastage for problem $edge(n,k)$. For each $k$, we calculate the edge wastage ratio $r(k)$ as: $r(k) = \lim_{n\to\infty} m(n,k)/n$. These limits exist and can be calculated using a combination of dynamic programming and mathematical induction in a finite amount of time. This is because the optimal edge patterns all become periodic for sufficiently large $n$ for any $k$. We discuss this in more detail below. See Table 5 for the values of $r(k)$ and Figure 2 for a sample of optimal edge patterns.

First, we note that $r(1) = r(2) = 1/3 = 0.333$ gives us the trivial edge wastage bound discussed in Section 2. The optimal edge pattern for $k = 2$ is unique and periodic and is shown in Figure 2(a). However, this optimal width 2 edge cannot be extended to a width 3 edge without introducing additional wastage. For a width 3 edge, $r(3) = 4/11 = 0.364$ and the optimal edge pattern is once again unique and periodic and is shown in Figure 2(b). We will refer to the pattern shown in the first 8 columns of Figure 2(b) the “castle” pattern and the next 3 columns of Figure 2(b) the “block” pattern. The castle pattern has length 8 with 3 wastage and the block pattern has length 3 with 1 wastage. As $k$ increases, $r(k)$ continues to increase slightly. At $k = 13$, we have $r(13) = 10/27 = 0.370$. The optimal edge is no longer unique or periodic, however, all of them are identical within the first 3 rows up to translational symmetry, and are composed of three castle patterns followed by a block pattern, which gives $3*3+1 = 10$ wastage per $3*8+3 = 27$ cells. All of the optimal edge patterns only have wastage in the row beyond the edge, which is an interesting fact to note. Beyond, $k = 13$, the value of $r(k)$ plateaus, and we stopped our experimentation at $k = 20$. We show later in Section 7 that there exists arbitrarily thick edge patterns which achieve exactly an edge wastage ratio of $r(13) = 10/27$. So in fact, $r(k) = 10/27$ for all $k \geq 13$.

What the edge wastage ratio results show is that the still life constraints up to a depth of 13 are capable of forcing wastage to occur. We should not relax these constraints away, as that will degrade the bound that we can prove using the relaxed problem. Thus we need to look at an edge relaxation of at least width 13. However, the fact that $r(k)$ plateaus after $k = 13$ also suggests that width 13 is sufficient, i.e. the still life constraints beyond a depth of 13 can have no effect on forcing edge wastage, and we can relax them away without changing the bound. This turns out to be correct, as our results later show.

## 5. Solving the Boundary Relaxation

We wish to solve a relaxed form of the Still Life problem where we only consider a width 13 boundary. The technique we choose to use is bounded dynamic programming [10]. We discuss the choice of this technique and show how it can be applied. First of all, it is necessary to choose one of the methods that can exploit the low pathwidth of the relaxed problem. Of these, dynamic programming is by far the easiest to implement. However, we use a variant of dynamic programming called bounded dynamic programming [10]. For some kinds of dynamic programs, instead of calculating the exact value for every subproblem, it is actually sufficient to prove a bound on the value of many of the subproblems.

**Example 1.** Consider the recursion:

$$a(n) = min\{a(n-1)+1, b(n-1)+2\}$$
$$b(n) = min\{a(n-1)+3, b(n-1)+1\}$$

Suppose we want to calculate the value of $a(n)$. Suppose we have already calculated that $a(n-1) = 6$. Now, we need to find some information on $b(n-1)$. However, we note that one of the terms in the *min*: $a(n-1)+1 = 7$, is already known, therefore if it is known that $b(n-1) \geq 5$, then its exact value is irrelevant as far as the value of $a(n)$ is concerned. We only want to know: is $b(n-1) \geq 5$, or if it is $< 5$, then what is its exact value? Thus we only wish to answer a bounded query on the value of $b(n-1)$, where we do not care what its exact value is if it is above or below a certain range.

On certain types of problems, bounded dynamic programming can be exponentially faster than normal dynamic programming, as there are a lot of subproblems whose exact values are irrelevant and all we need to do is to prove a certain bound on their value. Proving a weak bound is often exponentially faster than finding the exact value. In the case of the Still Life problem, there are many, many ways to label the cells, and most of them lead to large amounts of wastage. Therefore, we often just need to prove that a subproblem is sufficiently bad (has at least a certain amount of wastage), rather than calculate exactly how bad it is. Bounded dynamic programming is a rigorous way to do this.

We illustrate how dynamic programming can be applied with a simplified example. Consider a more constrained version of $edge(n, k)$ where we have additional boundary conditions which fix the first 2 and last 2 columns. We denote this problem by $edge(n, k, s_1, s_2, e_1, e_2)$ where the model has the additional lines:

```
% instance parameters
int: s1, s2, e1, e2;

% additional boundary conditions
constraint s1 = sum (i in 0..k-1) (x[1,i] * 2^i);
constraint s2 = sum (i in 0..k-1) (x[2,i] * 2^i);
constraint e1 = sum (i in 0..k-1) (x[n,i] * 2^i);
constraint e2 = sum (i in 0..k-1) (x[n-1,i] * 2^i);
```

As can be seen, we are denoting the value of a column of $k$ cells by a single number in $\{0, \ldots, 2^k - 1\}$, where the value of the cells are being interpreted as the binary digits of this number. We define the Boolean function $s(a, b, c)$ where $a, b, c \in \{0, \ldots, 2^k - 1\}$ as *true* if three consecutive columns labeled as $a$, $b$, and $c$ in that order do not violate the still life constraints, and *false* otherwise. We define the integer function $w(a, b, c)$ where $a, b, c \in \{0, \ldots, 2^k - 1\}$ as the wastage in the central column $b$ if three consecutive columns are labeled as $a$, $b$, and $c$ in that order.

Let $m(n, k, s_1, s_2, e_1, e_2)$ be the minimum wastage for problem $edge(n, k, s_1, s_2, e_1, e_2)$. Then the following recursive formulas hold: $\forall n \geq 3, s_1, s_2, e_1, e_2 \in \{0, \ldots, 2^k - 1\}$,

$$m(n, k, s_1, s_2, e_1, e_2) = min \; \{ \; m(n-1, k, s_1, s_2, e_2, e_3) + w(e_1, e_2, e_3) \mid e_3 \in \{0, \ldots, 2^k - 1\}, s(e_1, e_2, e_3) \; \}.$$

Let's consider why. Firstly, consider any solution of $edge(n, k, s_1, s_2, e_1, e_2)$. We must have fixed the $(n-3)$th column to something. Let it be denoted by $e_3$. To satisfy the still life constraints, we must have $s(e_1, e_2, e_3)$ be true. For each such solution, $w(e_1, e_2, e_3)$ gives us the wastage in the $n-2$th column. If we project this solution onto the first $n-1$ columns, then we clearly have a solution for $edge(n-1, k, s_1, s_2, e_2, e_3)$. Hence the minimum wastage among all such solutions is $m(n-1, k, s_1, s_2, e_2, e_3) + w(e_1, e_2, e_3)$. Taking the minimum of these over all possible $e_3$ gives $m(n, k, s_1, s_2, e_1, e_2)$, and hence the recursive formula.

Such a set of recursive formulas can be solved using dynamic programming in $O(n)$. We can use precalculated values of $m(n, k, s_1, s_2, e_1, e_2)$ as a wastage lookahead in the complete search method described in Section 3. Given a partially filled edge where the last two columns are given by $s_1$ and $s_2$, we can look up $min_{i,j} m(n, k, s_1, s_2, i, j)$ to get a lower bound on the wastage in the remaining cells of the edge. This will give a bound of approximately 10/27 wastage per remaining edge cell instead of the trivial 1/3 wastage per remaining edge cell we had before. Although the difference is small, it is quite crucial for solving the larger $n$ with complete search.

Now, it turns out that for any $k$, for sufficiently large $n$, the values of $m(n, k, s_1, s_2, e_1, e_2)$ become periodic. This is formalized by the following theorem.

**Theorem 3.** *If for some constants $M$, $p$, $q$, we have: $\forall s_1, s_2, e_1, e_2$, $m(M + p, k, s_1, s_2, e_1, e_2) = m(M, k, s_1, s_2, e_1, e_2) + q$, then $\forall n \geq M, \forall s_1, s_2, e_1, e_2, m(n + p, k, s_1, s_2, e_1, e_2) = m(n, k, s_1, s_2, e_1, e_2) + q$. That is, once $n \geq M$, the values of $m$ simply increase by $q$ every $p$ cells.*

Proof. The proof is by induction. The base case of $n = M$ is true by assumption. Suppose it is true for $n = t$. Consider $n = t + 1$. We have:

$$\begin{aligned}
& m(t+1+p, k, s_1, s_2, e_1, e_2) \\
= \; & min\{m(t+p, k, s_1, s_2, e_2, e_3) + w(e_1, e_2, e_3) | e_3 \in \{0, \ldots, 2^k - 1\}, s(e_1, e_2, e_3)\} \\
= \; & q + min\{m(t, k, s_1, s_2, e_2, e_3) + w(e_1, e_2, e_3) | e_3 \in \{0, \ldots, 2^k - 1\}, s(e_1, e_2, e_3)\} \\
= \; & q + m(t+1, k, s_1, s_2, e_1, e_2) \qquad \Box
\end{aligned}$$

Theorem 3 tells us that if we ever reach an $n$ where the $m$ values become periodic, then it remains periodic for all larger $n$. This always happens due to the translational symmetry in the Still Life Problem. For $k = 13$, the values typically become periodic by the time we reach $n = 200$. Once this happens, we can derive closed form equations for $m(n, k, s_1, s_2, e_1, e_2)$ for all larger $n$. Thus, we can calculate the values of $m(n, k, s_1, s_2, e_1, e_2)$ for all $n$ using a constant number of mathematical operations.

We now describe the full relaxed problem which we use to derive our bounds. We begin with the basic model described in Section 3. We relax the problem in two steps. In the first step, we eliminate the depth 14+ cells to leave a width 13 boundary. That is, for each $x[i, j]$ where $13 < i, j \leq n - 13$, we eliminate $x[i, j]$ from the problem by existentially quantifying it in every constraint in which it appears. Existential quantification of a variable in a constraint weakens the constraint, so this is a valid relaxation. In the second step, we cut a "slit" in this width 13 boundary in order to reduce the pathwidth further. The "slit" goes from the 1st to 13th columns between the 13th and 14th row. For each constraint $c$ which includes variables from both sides of the slit, i.e., both variables with $i \leq 13$ and variables with $i \geq 14$, $1 \leq j \leq 13$, we create two copies of $c$. In one copy, we existentially quantify all variables below the slit. In the other, we existentially quantify all variables above the slit. Once again, this is a valid relaxation. The resulting relaxed problem is shown graphically in Figure 3.

The reason for the first part of the relaxation is to get rid of the variables and constraints which we conjecture are irrelevant for forcing the lower bound. The reason for the second part of the relaxation is to lower the pathwidth so that the problem becomes tractable. Recall that the complexity of solving a problem with pathwidth $w$ is $O(2^w)$. Without the "slit", the pathwidth of the

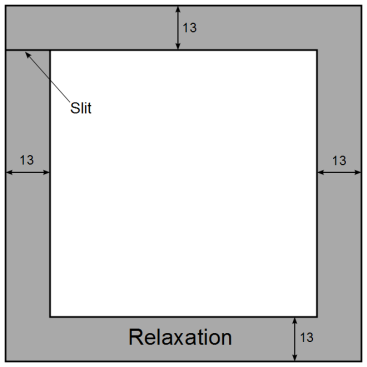


Figure 3: Boundary relaxation for calculating wastage lower bounds.

relaxed problem is around $5 * 13 = 65$, which is too high to be solved. With the slit, this drops to around $3 * 13 = 39$, which is solvable.

Once these relaxation are performed, we can simplify the objective function of the resulting relaxed problem. Consider $w[i, j]$ where $14 < i, j \leq n - 14$. These variables occur in exactly one table constraint (involving the 3x3 pattern centered at that coordinate), and as a positive term in the objective function. For such $w[i, j]$ variables, since all 9 of the $x[i, j]$ variables in the corresponding table constraint has been existentially quantified, $w[i, j]$ is free to take any value allowed by the table constraint. In particular, in any optimal solution, it would be set to its lower bound of 0. Thus we can simplify the objective function of the relaxed problem by removing all such $w[i, j]$ terms. Similarly, if two out of three columns of the $x[i, j]$ in a 3x3 pattern are existentially quantified, then $w[i, j]$ can always be set to 0. So we can also remove $x[i, j]$ where $i = 14$ or $n - 13$ and $14 < j \leq n - 14$, or where $j = 14$ or $n - 13$ and $14 < i \leq n - 14$. The resulting problem is of the same form as the simplified problem we showed above (except it has corners and multiple edges). It is easy to see that the same kind of recursive formulation into a dynamic program is possible. Thus the relaxed problem can be completely solved for all $n$ using bounded dynamic programming very efficiently.

Table 6 shows the wastage lower bounds and the corresponding live cell upper bounds we derive for $21 \leq n \leq 56$. We note several things. Firstly, all the bounds calculated by bounded dynamic programming on the relaxed problem are consistent with the optimal live cell values calculated by complete search in Section 3. Secondly, the bounds on the relaxed problem are often the optimal bounds for the original problem (the only exceptions being $n = 24, 26, 28, 38$). This is consistent with Conjecture 1, which stated that for sufficiently large $n$, the relaxed problem should have the same bound as the original. It would appear that the "sufficiently large $n$" in the conjecture is $n > 38$. Of course, to fully prove Conjecture 1, we must prove it for all $n > 38$. We do this in Section 7.

For $n \geq 61$, the wastage lower bounds becomes periodic and is given by the equations in Figure 4. The corresponding live cell upper bound is given by the

| $n$ | waste lb. | live ub. |
|---|---|---|
| 20 | 40 | 210 |
| 21 | 36 | 232 |
| 22 | 42 | 253 |
| 23 | 43 | 276 |
| 24 | 40 | 302* |
| 25 | 46 | 326 |
| 26 | 44 | 353* |
| 27 | 48 | 379 |
| 28 | 51 | 407* |
| 29 | 48 | 437 |
| 30 | 53 | 466 |
| 31 | 55 | 497 |
| 32 | 53 | 530 |
| 33 | 56 | 563 |
| 34 | 56 | 598 |
| 35 | 59 | 632 |
| 36 | 62 | 668 |
| 37 | 60 | 706 |
| 38 | 64 | 744* |
| 39 | 67 | 782 |
| 40 | 64 | 824 |
| 41 | 68 | 864 |
| 42 | 68 | 907 |
| 43 | 71 | 949 |
| 44 | 73 | 993 |
| 45 | 72 | 1039 |
| 46 | 76 | 1085 |
| 47 | 78 | 1132 |
| 48 | 76 | 1181 |
| 49 | 80 | 1229 |
| 50 | 80 | 1280 |
| 51 | 80 | 1331 |
| 52 | 85 | 1382 |
| 53 | 84 | 1436 |
| 54 | 87 | 1490 |
| 55 | 89 | 1545 |
| 56 | 88 | 1602 |
| 57 | 91 | 1658 |
| 58 | 92 | 1717 |
| 59 | 92 | 1776 |
| 60 | 97 | 1835 |
| 61 | 96 | 1897 |

Table 6: Lower bounds on wastage and corresponding upper bounds on live cells as calculated by bounded dynamic programming on a problem relaxation. Instances where the bound on the relaxed problem differed from the original problem are indicated with an asterisk.

$$
\mathit{wastage} \geq \begin{cases}
\lfloor 40/27 * n + 5 \rfloor, & n \equiv \; 5, 13, 21, 24, 32, 35, 40, 43, 48, 51 \quad \text{mod } 54 \\
\lfloor 40/27 * n + 6 \rfloor, & n \equiv \; 0, 2, 7, 8, 10, 14, 15, 16, 18, 22, 23, 26, \\
& \quad 29, 30, 34, 37, 38, 42, 45, 46, 50, 53 \quad \text{mod } 54 \\
\lfloor 40/27 * n + 7 \rfloor, & n \equiv \; 1, 3, 4, 9, 11, 12, 17, 19, 20, 25, 27, 28, \\
& \quad 31, 36, 39, 41, 44, 47, 49, 52 \quad \text{mod } 54 \\
\lfloor 40/27 * n + 8 \rfloor, & n \equiv \; 6, 33 \quad \text{mod } 54
\end{cases}
$$

Figure 4: Closed form equations for the wastage lower bound for $n \geq 61$.

equations in Figure 5. We shall see in the next two sections that these live cell upper bounds are in fact achievable, so the equations in Figure 5 actually give us the optimal number of live cells for $n \geq 61$.

## 6. Lower Bounds for Large $n$

Lower bounds on live cells can be proved by finding actual solutions to the problem. If the number of live cells in the solution found coincides with the upper bound proved in Section 5, then we know that it is an optimal solution and we have solved the problem for that $n$. However, finding optimal solutions is very hard, because the default search space of the Still Life Problem is extremely large. The model given in Section 3 was good enough for us to solve up to

$$
\mathit{livecells} \leq \begin{cases}
\lfloor n^2/2 + 17/27 * n - 2 \rfloor & n \equiv \; 0, 1, 3, 8, 9, 11, 16, 17, 19, 25, 27, \\
& \quad 31, 33, 39, 41, 47, 49 \quad \text{mod } 54 \\
\lfloor n^2/2 + 17/27 * n - 1 \rfloor & n \equiv \; 2, 4, 5, 6, 7, 10, 12, 13, 14, 15, 18, \\
& \quad 20, 21, 22, 23, 24, 26, 28, 29, 30, \\
& \quad 32, 34, 35, 36, 37, 38, 40, 42, 43, \\
& \quad 44, 45, 46, 48, 50, 51, 52, 53 \quad \text{mod } 54
\end{cases}
$$

Figure 5: Closed form equations for the live cell upper bound for $n \geq 61$.

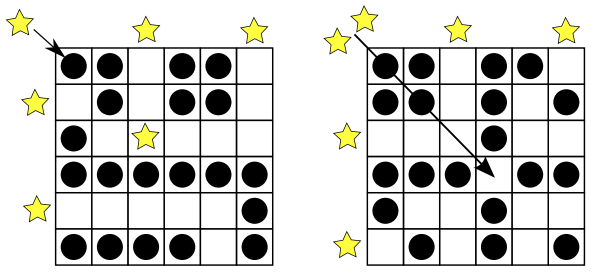


Figure 6: The two best ways to fill a corner (up to symmetry). Wastage is highlighted with a star. Note that there are two units of wastage in the cell in the 4th column and 4th row in the second pattern.

around $n = 50$ using complete search. However, to go beyond that size, we need something that reduces our search space much further. We make the following conjecture:

**Conjecture 2.** *For sufficiently large $n$, there always exists optimal solutions of the following form: wastage only exists at the four $4 \times 4$ corners of the board, or in the one row beyond the edge of the board.* □

Conjecture 2 is supported by the experiments described in Section 4. We reasoned that for sufficiently large $n$, the edges should follow one of the optimal edge patterns, and we know that none of them have any wastage other than in the one row beyond the edge of the board. Similarly, there are only two ways to label a corner with minimum wastage (see Figure 6) and both of them only have wastage within the $4 \times 4$ corners. And finally, the center of the board should be wastage free, because by Conjecture 1, the constraints in the center of the board do not force any wastage, and any wastage there would simply make the solution suboptimal.

Assuming that Conjecture 2 holds, we can look only for solutions of this special form. Note that searching only for solutions of this special form gives an incomplete search on the Still Life Problem. However, incomplete search is perfectly sufficient for proving lower bounds on the number of live cells, since the solution itself is the proof. To solve for all $n$, we must be able to find an optimal solution for every single $n$. We use two further techniques to reduce the search space: 1) dynamic relaxations as lookahead, and 2) a customized limited discrepancy search.

In Section 4, we described how we can perform a relaxation onto the boundary of the board in order to derive a lower bound on the wastage. We can do the same thing during search. Our search strategy is to label the board 8 rows at a time from top to bottom. Conjecture 1 tells us that for sufficiently large $n$, wastage is only forced by the boundary constraints. In a subproblem however, the boundary also includes the values of the cells we have labeled. If we relax the subproblem onto the boundary of the unlabeled region, we should be able to derive a very strong lower bound on the wastage in the unlabeled region of the board. Thus, at each search node, we relax the remaining problem onto:

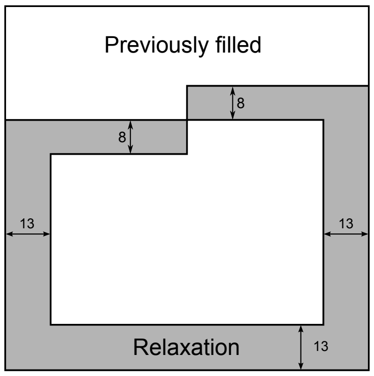


Figure 7: Dynamic relaxation lookahead for still life search.

the unlabeled parts of the width 13 boundary, and the 8 unlabeled rows below the last row we labeled (see Figure 7).

We note several things. Firstly, the set of variables involved in the relaxation is different at each search depth. Secondly, the boundary conditions of the relaxation are different even for two nodes at the same search depth if their cells in the last two rows are labeled differently. Thus each of these relaxed subproblems are different and have to be solved separately to derive the wastage lower bound. Now, as before, we can solve each of these in $O(n)$ using dynamic programming. However, we can do better by noting that the relaxed problem at any node is very similar to the relaxed problem at its parent node, differing only by one column of 8 variables. By appropriately caching the solutions to the relaxed subproblem at the parent node, we can solve the relaxed problem at each node in $O(1)$. The cost is still non-trivial, however, the wastage lower bounds derived by these relaxations are very strong and provide a large amount of pruning, so it is well worth it.

One might wonder why we use a different search order than that described in Section 3, where we label the board from outside in rather than from top to bottom. This is because the relaxation lookahead already tells us how much wastage there has to be in the unlabeled parts of the boundary, thus it is unnecessary to actually label it to force the wastage lower bound up. It is better not to label the boundary yet, as there are exponentially many ways to do it, but most of them are probably bad. It is better to fix the variables which are close to those variables already fixed so that we can detect inconsistencies earlier, hence the top to bottom strategy.

We also use the relaxation lookahead as a branching heuristic. When faced with the choice of labeling the next column of 8 variables as any one of the 256 possible values, our relaxation lookahead is able to tell us how much the wastage lower bound will increase by, given each of those choices. We order the choices according to how low the wastage lower bound given by the lookahead is. This is far superior to a naive, greedy branching heuristic which orders the choices based on the wastage in only the labeled part of the board, as it is often the case that greedily minimizing wastage locally will cause much more wastage further on. Our lookahead is capable of seeing the wastage caused further on and will not pick such locally optimal but globally suboptimal choices.

We use a modified version of limited discrepancy search [13]. Firstly, rather than defining a discrepancy as any choice which is not the first choice given by the branching heuristic, we define it as the amount that the wastage lower bound given by the relaxation lookahead would increase by if we made this decision. This means that there can be ties, i.e. multiple choices which are all equally good according to the lookahead. In such cases, we randomly tie-break between them. Secondly, we add randomized restarts to the search. At randomized points in time, the search will backtrack by a random number of rows. This is important, because despite our lookahead, there still exist vast subtrees where no optimal solutions can be found, and a complete search will take an exponential number of nodes to backtrack out of them. The combination of random restarts, plus random tie-breaking between equally good choices, is very effective. Using these techniques, we are able to find optimal solutions to instances as large as $n = 200$ in several hours on average. An optimal solution for $n = 100$ is shown in Figure 8.

## 7. Constructing Optimal Solutions for Arbitrarily Large $n$

Through our experimentation and analysis, it became clear that the Still Life Problem is actually fairly well behaved for sufficiently large $n$. We have the following properties: 1) the wastage lower bound is periodic in $n$, 2) there exists optimal periodic edge patterns which achieves this wastage lower bound, 3) it is "easy" to fill in the center of the board without any wastage. Combining these facts together, it seems possible that with a bit of work, we can construct closed form optimal solutions for arbitrarily large $n$.

In Section 5, we already worked out wastage lower bounds for all $n$. If we can construct solutions that achieve these wastage lower bounds for all $n$, then we are done. This can be done by solving instances of the Still Life Problem under additional periodic constraints, so that parts of the solution can be tiled indefinitely to produce arbitrarily large optimal solutions (see Figure 9). The initial solution is broken up by two horizontal and two vertical cuts into: 4 corner pieces, 4 edge pieces, and 1 center piece. The edge pieces and center pieces must satisfy periodic constraints so that they can be tiled and still satisfy the still life constraints. Furthermore, there are strong restrictions on the amount of wastage that can occur in these tiled pieces. We know that for sufficiently large $n$, the wastage lower bounds from Section 4 has a wastage to edge cell ratio of exactly 10/27. Thus to hit this lower bound, our 4 periodic edge pieces must have precisely this wastage ratio. Also, the periodic center piece must be completely wastage free.

Now, for the periodic edge pieces to have precisely a wastage to edge cell ratio of 10/27, their period must be a multiple of 27. Unfortunately, if the center piece is $27 \times 27$, then it is impossible for it to be wastage free, since it has an odd number of cells. Thus the minimum period we can use is $2 * 27 = 54$. The aim then is to solve an instance with $n = n'$ under additional period 54 constraints and wastage constraints on the edge and center pieces. If an optimal solution can be found, then it can be tiled to create optimal solutions for $n = n' + k * 54$

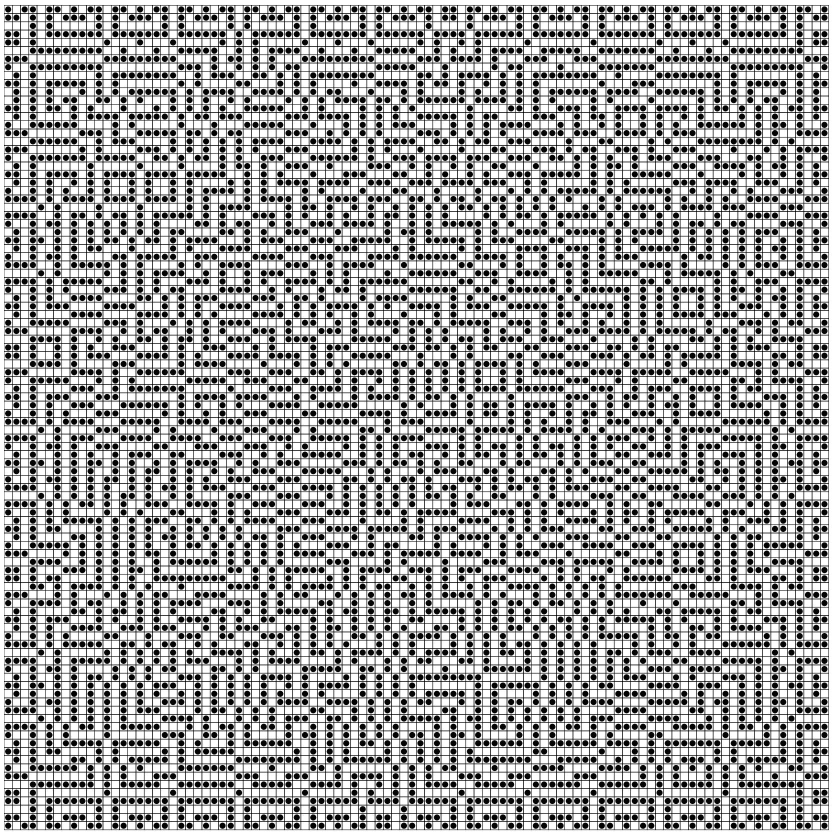


Figure 8: Optimal solution for $n = 100$.

for any $k \in \mathbb{N}$. Clearly, to solve for all $n$, this has to be done for each value of $n'$ mod 54, so there are 54 cases. We found that $n'$ had to be around 150 or higher before the Still Life problem became sufficiently well behaved that the periodic version was solvable. So we had to solve 54 instances of size 150+, one for each modulus class mod 54, under the additional periodic constraints and wastage constraints.

Rather than directly solving such problems from scratch, we decided to utilize the solutions that we had already found, and try to extend them into periodic solutions by splicing in a periodic section. We first take an optimal solution for $n = n' - 54$. We cut it into two pieces vertically at some point. We then move the two pieces apart by 54 cells and try to fill up the gap with a periodic 54 section which satisfies the additional wastage constraints. This periodic 54 section will end up being two of the periodic edge pieces. Secondly, we take this new solution and cut it horizontally at some point. We then move the two pieces apart by 54 cells and do the same again. This new section will end up being the other two periodic edge pieces plus the center piece. Now we end up with

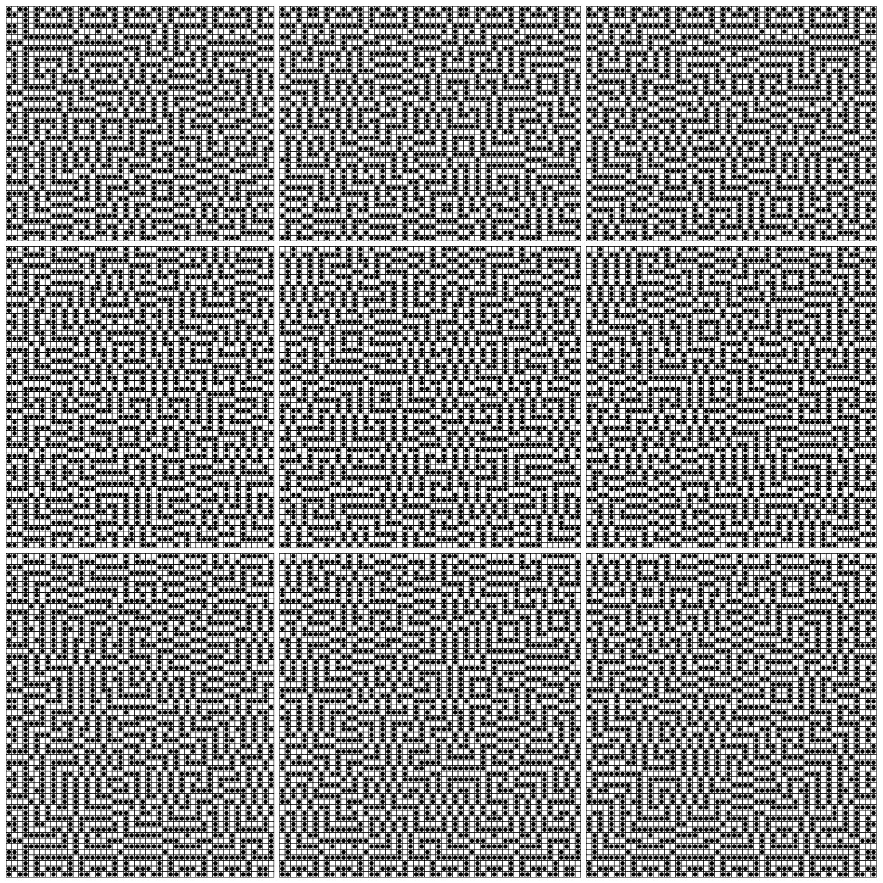


Figure 9: A periodic optimal solution where the center sections can be tiled indefinitely to produce optimal solutions for $n = 100 + k * 54$.

an optimal solution for $n = n'$ which satisfies the periodic constraints and the wastage constraints. Here is a MiniZinc model for the first step. The second step is analogous.

```
int: n; % instance parameter

% four columns covering the splice point from sol of n = n' - 54
array [1..4,1..n] of 0..1: s;

array [1..54,0..n+1] of var 0..1: x; % cell live/dead status

% still life and wastage constraints in 54 by n region
constraint forall (i in 2..53, j in 1..n) (
table(sl_waste, [x[i-1,j-1], x[i,j-1], x[i+1,j-1], x[i-1,j],
x[i,j], x[i+1,j], x[i-1,j+1], x[i,j+1], x[i+1,j+1], 0]));

% boundary conditions
constraint forall (i in 0..n+1) (
x[i,0] = 0 /\ x[i,n+1] = 0);

constraint forall (i in 1..52) (
sum (j in i..i+2) (x[j,1]) <= 2 /\
sum (j in i..i+2) (x[j,n]) <= 2
);

constraint forall (i in 1..n) (
x[1,i] = s[3,i] /\ x[2,i] = s[4,i] /\
x[n-1,i] = s[1,i] /\ x[n,i] = s[2,i]);

% wastage constraints for boundary

constraint sum (i in 1..54) (x[i,1]) = 34;
constraint sum (i in 1..54) (x[i,n]) = 34;

solve maximize satisfy;
```

This splicing does not always succeed as the constraints are very, very strong. Whether it is satisfiable or not depends on the initial solution and the point at which we make the cut. In particular, the cuts must be made at a point where both edges have already transitioned into the optimal edge pattern of 3 castles per 1 block periodic pattern. If we could not solve a particular instance after a reasonable time, we tried a different cut point or tried it using a different solution of $n = n' - 54$. The success rate was around 80%, so most of them succeeded on the first try. After approximately 3000 hours of computation, we were able to find periodic solutions for all 54 cases, and thus the Still Life Problem was solved for all large $n$. The Still Life shown in Figure 9 is one such periodic solution where the center sections can be tiled indefinitely to produce optimal solutions for $n = 100 + k * 54$.

These results prove Conjecture 1, Conjecture 2, and closes off the Maximum Density Still Life Problem for all $n$. We restate our results for clarity:

**Theorem 4.** *For $n > 39$, all "forced" wastage is caused by the still life constraints within 13 rows from the edge of the $n \times n$ region.*

**Theorem 5.** *For $n > 50$, there always exists optimal solutions of the following form: wastage only exists at the four $4 \times 4$ corners of the board, or in the one row beyond the edge of the board.*

**Theorem 6.** *For $n \leq 60$, the maximum number of live cells that can appear in an $n \times n$ still life is given in Table 7. For $n \geq 61$, the maximum number of live cells that can appear in an $n \times n$ still life is given by:*

| $n$ | live cells |
|---|---|
| 1 | 0 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 16 |
| 6 | 18 |
| 7 | 28 |
| 8 | 36 |
| 9 | 43 |
| 10 | 54 |
| 11 | 64 |
| 12 | 76 |
| 13 | 90 |
| 14 | 104 |
| 15 | 119 |
| 16 | 136 |
| 17 | 152 |
| 18 | 171 |
| 19 | 190 |
| 20 | 210 |
| 21 | 232 |
| 22 | 253 |
| 23 | 276 |
| 24 | 302 |
| 25 | 326 |
| 26 | 353 |
| 27 | 379 |
| 28 | 407 |
| 29 | 437 |
| 30 | 467 |
| 31 | 497 |
| 32 | 531 |
| 33 | 563 |
| 34 | 598 |
| 35 | 633 |
| 36 | 668 |
| 37 | 706 |
| 38 | 744 |
| 39 | 782 |
| 40 | 824 |
| 41 | 864 |
| 42 | 907 |
| 43 | 949 |
| 44 | 993 |
| 45 | 1039 |
| 46 | 1085 |
| 47 | 1132 |
| 48 | 1181 |
| 49 | 1229 |
| 50 | 1280 |
| 51 | 1331 |
| 52 | 1382 |
| 53 | 1436 |
| 54 | 1490 |
| 55 | 1545 |
| 56 | 1602 |
| 57 | 1658 |
| 58 | 1717 |
| 59 | 1776 |
| 60 | 1835 |

Table 7: Maximum number of live cells in an $n \times n$ still life for $n \leq 60$.

$$\begin{array}{lll l}
\lfloor n^2/2 + 17/27 * n - 2 \rfloor & n \equiv & 0,1,3,8,9,11,16,17,19,25,27, & \\
 & & 31,33,39,41,47,49 & \bmod 54 \\
\lfloor n^2/2 + 17/27 * n - 1 \rfloor & n \equiv & 2,4,5,6,7,10,12,13,14,15,18, & \\
 & & 20,21,22,23,24,26,28,29,30, & \\
 & & 32,34,35,36,37,38,40,42,43, & \\
 & & 44,45,46,48,50,51,52,53 & \bmod 54
\end{array}$$

## 8. Conclusion

We have solved the Maximum Density Still Life Problem for all $n$ by combining mathematical insights into the problem with appropriate applications of remodeling, lazy clause generation, bounded dynamic programming, relaxations, and custom search. The complete solution consists of four parts: 1) complete search which can solve $n \leq 50$, 2) bounded dynamic programming with relaxation to prove optimal live cell upper bounds for $n > 50$, 3) incomplete search for special form solutions which can prove optimal live cell lower bounds for $50 < n \leq 200$, 4) incomplete search to find optimal periodic solutions which can be tiled to construct arbitrarily large solutions that prove the optimal live cell lower bounds for $n > 200$. The optimal values for all $n$ are given in Theorem 6. Optimal solutions for small and medium $n$ and periodic optimal solutions for large $n$ can be found at `www.csse.unimelb.edu.au/~pjs/still-life/`. The total time taken to completely solve the Maximum Density Still Life Problem for all $n$ was approximately 3000 hours.

*Acknowledgements*

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

## References

[1] G. Chu, P. Stuckey, M. Garcia de la Banda, Using relaxations in maximum density still life, in: I. Gent (Ed.), Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, Vol. 5732 of LNCS, Springer-Verlag, 2009, pp. 258–273.

[2] R. Bosch, Integer programming and Conway's game of life, SIAM Review 41 (3) (1999) 596–604.

[3] R. Bosch, M. Trick, Constraint programming and hybrid formulations for three life designs, Annals OR 130 (1-4) (2004) 41–56.

[4] J. Larrosa, R. Dechter, Boosting search with variable elimination in constraint optimization and constraint satisfaction problems, Constraints 8 (3) (2003) 303–326.

[5] J. Larrosa, E. Morancho, D. Niso, On the practical use of variable elimination in constraint optimization problems: 'Still-life' as a case study, Journal of Artificial Intelligence Research 23 (2005) 421–440.

[6] O. Ohrimenko, P. Stuckey, M. Codish, Propagation via lazy clause generation, Constraints 14 (3) (2009) 357–391.

[7] B. M. Smith, Caching search states in permutation problems, in: P. van Beek (Ed.), Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, Vol. 3709 of Lecture Notes in Computer Science, Springer, 2005, pp. 637–651.

[8] T. Feydy, P. J. Stuckey, Lazy Clause Generation Reengineered, in: I. P. Gent (Ed.), Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, Vol. 5732 of Lecture Notes in Computer Science, Springer, 2009, pp. 352–366.

[9] R. Bellman, Dynamic Programming, Princeton University Press, 1957.

[10] J. Puchinger, P. Stuckey, Automating branch-and-bound for dynamic programs, in: R. Glück, O. de Moor (Eds.), Proceedings of the ACM SIGPLAN 2008 Workshop on Partial Evaluation and Program Manipulation (PEPM '08), ACM, 2008, pp. 81–89.

[11] N. Elkies, The still-life density problem and its generalizations, Voronoi's Impact on Modern Science: Book I (1998) 228–253 arXiv:math/9905194v1.

[12] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, G. Tack, Minizinc: Towards a standard CP modelling language, in: C. Bessiere (Ed.), Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming, Vol. 4741 of LNCS, Springer-Verlag, 2007, pp. 529–543.

[13] W. D. Harvey, M. L. Ginsberg, Limited discrepancy search, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1995, pp. 607–615.