Using relaxations in Maximum Density Still Life

Geoffrey Chu[1], Peter J. Stuckey[1], and Maria Garcia de la Banda[2]

[1] NICTA Victoria Laboratory,
Department of Computer Science and Software Engineering,
University of Melbourne, Australia
{gchu,pjs}@csse.unimelb.edu.au
[2] Faculty of Information Technology,
Monash University, Australia
mbanda@infotech.monash.edu.au

Abstract. The Maximum Density Sill-Life Problem is to fill an $n \times n$ board of cells with the maximum number of live cells so that the board is stable under the rules of Conway's Game of Life. We reformulate the problem into one of minimising "wastage" rather than maximising the number of live cells. This reformulation allows us to compute strong upper bounds on the number of live cells. By combining this reformulation with several relaxation techniques, as well as exploiting symmetries via caching, we are able to find close to optimal solutions up to size $n = 100$, and optimal solutions for instances as large as $n = 69$. The best previous method could only find optimal solutions up to $n = 20$.

1 Introduction

The *Game of Life* was invented by John Horton Conway and is played on an infinite board. Each cell c in the board is either alive or dead at time t. The live/dead state at time $t+1$ of cell c, denoted as $state(c, t+1)$, can be obtained from the number l of live neighbours of c at time t and from $state(c, t)$ 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 unchanged by these rules, i.e., it is the same at $t + 1$. For example, an empty board is a still-life. Given a finite $n \times n$ region where all other cells are dead, the *Maximum Density Still-life Problem* aims at computing the highest number of live cells that can appear in a still life for the region. The density is thus expressed as the number of live cells over the $n \times n$ region.

The raw search space of the Maximum Density Still-life Problem has size 2^{n^2}. Thus, it is extremely difficult even for "small" values of n. Previous search methods using IP [1] and CP [2] could only solve up to $n = 9$, while a CP/IP hybrid method with symmetry breaking [2] could solve up to $n = 15$. An attempt using bucket elimination [6] 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 [7], used less memory and was able to solve up to $n = 20$. In this paper we combine various techniques to allow us to solve instances up to $n = 50$ completely or almost completely, i.e. we prove upper bounds and find solutions that either achieve that upper bound or only have 1 less live cell. We also obtain solutions that are no more than 4 live cells away from our proven upper bound all the way to $n = 100$. The largest completely solved instance is $n = 69$.

The contributions of this paper are as follows:

- We give a new insightful proof that the maximum density of live cells in an infinite still life is $\frac{1}{2}$. This proof allows us to reformulate the maximum density still-life problem in terms of minimising "wastage" rather than maximising the number of live cells.
- We derive tight lower bounds on wastage (which translate into upper bounds on the number of live cells) that can be used for pruning.
- We define a static relaxation of the original problem that allows us to calculate closed form equations for an upper bound on live cells for all n. And we do this in constant time by using CP with caching. We conjecture that this upper bound is either equal to the optimum, or only 1 higher for all n.
- We identify a subset of cases for which we can improve the upper bound by 1 by performing a complete search on center perfect solutions. This completes the proof of optimality for the instances where our heuristic search was able to find a solution with 1 cell less than the initial upper bound.
- We define a heuristic incomplete search using dynamic relaxation as a lookahead method that can find optimal or near optimal solutions all the way up to $n = 100$.
- We find optimal solutions for n as large as 69, more than 3 times larger than any previous methods, and with a raw search space of 2^{4761}.

2 Wastage reformulation

The maximum density of live cells in an infinite still life is known to be $\frac{1}{2}$ [5, 4]. However, that proof is quite complex and only applies to the infinite plane. In this section we provide a much simpler proof that can easily be extended to the bounded case and gives much better insight into the possible sub-patterns that can occur in an optimal solution. The proof is as follows.

First we assign an area of 2 to each cell (assume the side length is $\sqrt{2}$). We will partition the area of each dead cell into two 1 area pieces and distribute them among its live neighbours according to the local pattern found around the dead cell. It is clear that if we can prove that all live cells on the board end up with an area ≥ 4, then it follows that the density of live cells is $\leq \frac{1}{2}$.

The area of a dead cell is assigned only to orthogonal neighbouring live cells, i.e., those that share an edge with the dead cell. We describe as "wastage" the area from a dead cell that does not need to be assigned to any live cell for the proof to work (i.e., it is not needed to reach our 4 target for live cells). Table 1 shows all possible patterns of orthogonal neighbours (up to symmetries) around

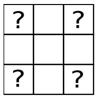
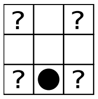
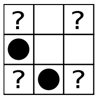
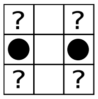
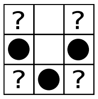
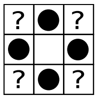
Pattern:						
Beneficiaries	{}	{ S }	{S, W}	{E, W}	{E, W}	{}
Wastage	2	1	0	0	0	2

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

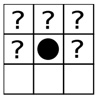
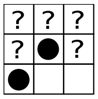
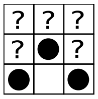
Pattern:			
Area received	1	1	0

Table 2. Contributions to the area of a live cell from its South neighbour.

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 neighbours that receive 1 area from the center dead cell, and the resulting amount of wastage. As it can be seen from the table, a dead cell gives 1 area to each of its live orthogonal neighbours if it has ≤ 2 live orthogonal neighbours, 1 area to the two opposing live orthogonal neighbours if it has 3, and no area if it has 4. Note that, in the table, wastage occurs whenever the dead cell has ≤ 1 or 4 live orthogonal neighbours. As a result, we only need to examine the 3 bordering cells on each side of a live cell, to determine how much area is obtained from the orthogonal neighbour on that side. For example, the area obtained by the central live cell from its South neighbour is illustrated in Table 2.

The area obtained by a live cell can therefore be computed by simply adding up the area obtained from its four orthogonal neighbours. Since each live cell starts off with an area of 2, it must receive at least 2 extra area to end up with an area that is ≥ 4. Let us then look at all possible patterns around a live cell and see where the cell will receive area from. Table 3 shows all possible neighbourhoods of a live cell (up to symmetries). For each pattern, it shows the benefactors, i.e., the North, East, South or West neighbours that give 1 area to the live cell, and the resulting amount of wastage, which occurs whenever a live cell receives more than 2.

Note that the last pattern does not receive sufficient extra area, just 1 from the South neighbour. However, the last two patterns always occur together in unique pairs due to the still-life constraints (each of the central live cells has 3 neighbours so the row above the last pattern, and the row below the second last pattern must only consist of dead cells). Hence, we can transfer the extra 1 area from the second last pattern to the last.

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	1	−1

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

Clearly, all live cells end up with an area ≥ 4, and this completes our proof that the maximum density on an infinite board is $\frac{1}{2}$.

The above proof is not only much simpler than that of [5, 4], it also provides us with good insight into how to compute useful bounds for the case in which the board is finite. In particular, it allows us to know exactly how much we have lost from the theoretical maximum density by looking at the amount of "wastage" produced by the patterns in the currently labeled cells.

To achieve this, we reformulate the objective function in the Bounded Maximum Density Still Life Problem as follows. For each cell c, let $P(c)$ be the 3×3 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 $w(P)$ be the wastage for each 3×3 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 $w(c) = w(P(c))$. 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 1. *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 the $n \times n$ region. Let us assign 2 area to each cell within the $n \times n$ region, and 1 area to each of the $4n$

cell in the row immediately beyond the edge of the $n \times n$ region. Now, for each dead cell within the $n \times n$ region, partition the area among its live orthogonal neighbours as before. For each dead cell in the row immediately beyond the $n \times n$ region, give its 1 area to the cell in the $n \times n$ region with which it shares an edge. Again, since the last two 3×3 patterns listed above must occur in pairs, we transfer an extra 1 area from one to the other. Note also that the second last pattern of Table 3 cannot appear on the South border (which would mean that the last pattern appeared outside the shape) since it is not stable in this position. Clearly, after the transfers, all live cells once again have ≥ 4 area, and wastage for the 3×3 patterns centered around cells within the $n \times n$ region remain the same. However, since we are in the finite case, we also have wastage for the cells which are in the row immediately beyond the edge of the $n \times n$ region. These dead cells always give 1 area to the neighbouring cell which is in the $n \times n$ region. If that cell is live, the area is received. If that cell is dead, that 1 area is wasted. The reformulation above counts all these wastage as follows. The total amount of area that was used was $2n^2$ from the cells within the $n \times n$ region and $4n$ from the $4n$ cells in the row immediately beyond the edge, for a total of $2n^2 + 4n$. Now, 4 times live cells will be equal to the total area minus all the area wasted, and hence we end up with Equation (1). □

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 a small n, it differs from the true optimum by $O(n)$ and will diverge from the optimum for a large n. We provide a better bound in the next section.

3 Closed form upper bound

Although in the infinite case there are many patterns that can achieve exactly $\frac{1}{2}$ density, it turns out that in the bounded case, the boundary constraints force significant extra wastage. As explained in the previous section, the still life constraints on the edge of the $n \times n$ region trivially force at least 1 wastage per 3 edge cells, however, it can be shown by exhaustive search that even this theoretical minimal wastage of 1/3 per edge cell is unattainable for an infinitely long edge due to the still life constraints on the inside of the $n \times n$ region.

There is also no way to label the corner without producing some extra wastage. For example, for a 6×6 corner, naively we expect to have $(6+6)/3 = 4$ wastage forced by the still life constraints on the boundary. However, due to the still life constraints within the corner, there is actually no way to label a 6×6 corner without at least 6 wastage.

Examination of the optimal solutions found by other authors show that, in all instances, all or almost all wastage is found either in the corners or within 3 rows from the edge. This leads to the following conjecture:

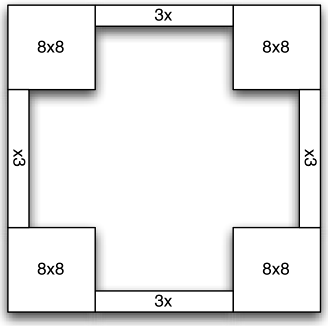


Fig. 1. The relaxed version of the problem, only filling in the 8×8 corners and the 3 rows around the edge.

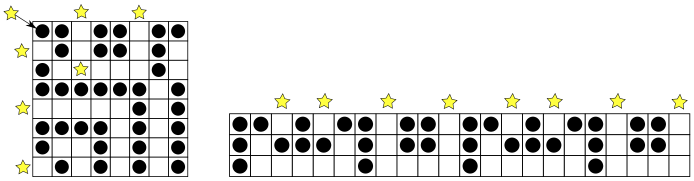
Conjecture 1. All wastage forced by the boundary constraints of an $n \times n$ region must appear either in the 8×8 corners, or within 3 rows from the edge.

If this conjecture is true, then it should be possible to find a good lower bound on the amount of forced wastage simply by examining the corners and the first few rows from the edge.

We thus perform the following relaxation of the bounded still life problem. We keep only the variables representing the four 8×8 corners of the board, as well as the variables representing cells within 3 rows of the edge (see Figure 1). All variables in the middle are removed. The still life constraints for fully surrounded cells, including cells on the edge of the $n \times n$ region, remain the same. The still life constraints for cells neighbouring on removed cells are relaxed as follows: dead cells are always considered to be consistent with the problem constraints, and live cells are considered to be consistent as long as they do not have more than 3 live neighbours (otherwise any extension to the removed cells would violate the original constraints). The objective function is modified as follows: fully surrounded cells have their wastage counted as before, live cells with neighbouring removed cells have no wastage counted, and dead cells with neighbouring removed cells have 1 wastage counted if and only if it is surrounded by ≥ 3 dead unremoved cells (since any extension to the removed cells will result in a pattern with ≥ 1 wastage). Clearly, since we have relaxed the constraints, and also potentially ignored some wastage in our count, any lower bound we get on wastage in this relaxed problem is a valid lower bound on the wastage in the original problem.

Since the constraint graph of this relaxed problem has bounded, very low path-width, it can easily be solved using CP with caching in $O(n)$ time (see [3] Theorem 13.2). In practice, solving the 8×8 corners is the hardest and takes $8s$. An example corner solution is shown in Figure 2(a). Solving the width 3 edge takes milliseconds. The results of calculating the bound from the relaxed problem for small n is shown in Table 4.

Because of the high symmetry of the edge subproblem (full translational symmetry), the edge bounds starts to take on a periodic pattern for n sufficiently



(a) (b)

Fig. 2. An optimal 8×8 North West corner pattern with 7 wastage highlighted, and (b) the periodic pattern for an optimal North edge with 4 wastage per period 11 highlighted.

large, at which point we can derive a closed form equation for their values for all n. The periodicity comes from the fact that it can be shown that the optimal periodic edge pattern (see Figure 2(b)) has period 11, and any sufficiently long optimal edge pattern will have a series of these in the middle.

Since the set of optimal solutions for the 8× 8 corners remain the same for any $n > 16$, and we can derive a closed form equation for the edge bounds for large n, we can calculate a closed form equation for the lower bound on wastage for the whole relaxed problem for any n sufficiently large. For $n \geq 50$ it is:

$$min\_wastage = \begin{cases} 8 + 16 \times \lfloor n/11 \rfloor,\ n \equiv 0,1,2 & \mod 11 \\ 12 + 16 \times \lfloor n/11 \rfloor,\ n \equiv 3,4 & \mod 11 \\ 14 + 16 \times \lfloor n/11 \rfloor,\ n \equiv 5 & \mod 11 \\ 16 + 16 \times \lfloor n/11 \rfloor,\ n \equiv 6,7 & \mod 11 \\ 18 + 16 \times \lfloor n/11 \rfloor,\ n \equiv 8 & \mod 11 \\ 20 + 16 \times \lfloor n/11 \rfloor,\ n \equiv 9,10 & \mod 11 \end{cases} \tag{2}$$

A lower bound on wastage can be converted into an upper bound on live cells using Equation (1):

$$live\_cells \leq \left\lfloor \frac{2n^2 + 4n - min\_wastage}{4} \right\rfloor \tag{3}$$

We will call the live cell upper bound calculated from our closed form wastage lower bounds the “closed form upper bound”. If Conjecture 1 is true and all forced wastage must appear within a few rows of the edge, then this closed form upper bound should be extremely close to the real optimal value.

Conjecture 2. The maximum number of live cells in an $n \times n$ region is

$$\left\lfloor \frac{2n^2 + 4n - min\_wastage}{4} \right\rfloor$$

or one less than this, where *min_wastage* is the optimal solution to the relaxed problem of Figure 1 (given by Equation (2) for $n \geq 50$). □

n	optimal	upper bound
8	36	**36**
9	43	44
10	54	55
11	64	65
12	76	77
13	90	91
14	104	**104**
15	119	120
16	136	**136**
17	152	**152**
18	171	172
19	190	**190**
20	210	**210**

Table 4. Upper bound by relaxation for small n, shown in bold if it is equal to the optimal solution.

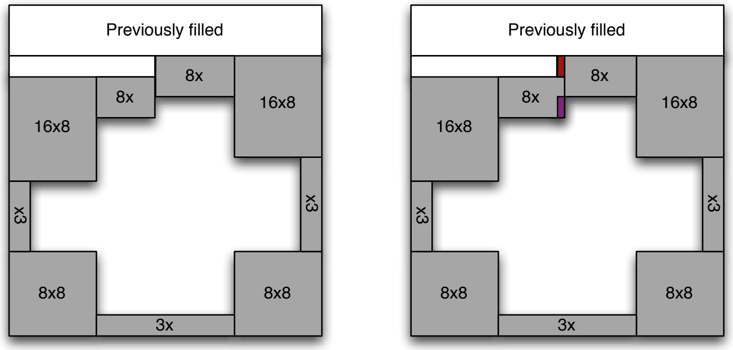
Conjecture 2 is true for previously solved n. As it can be seen from Table 4, for $n \leq 20$, our upper bound is never off the true optimum by more than 1, and is often equal to it.

As our new results in Table 5 show, our conjecture is also true for at least up to $n = 50$. The bound is also achievable exactly for n as high as 69. For larger n our solver is too weak to guarantee finding the optimal solution and thus we cannot verify the conjecture. We believe the conjecture is true because although not all 3×3 patterns are perfect (waste free), there are a large number of perfect ones and there easily appears to be enough different combinations of them to label the center of an $n \times n$ region perfectly. Indeed, since we already know that there are many ways to label an infinite board perfectly, the boundary constraints are the only constraints that can force wastage.

4 Finding optimal solutions

In the previous section, we found a relaxation that allows us to find very good upper bounds in constant time. However, in order to find an optimal solution it is still necessary to tackle the size 2^{n^2} search space. The previous best search based methods could only find optimal solutions up to $n = 15$, with a search space of 2^{225}. Here, we attempt to find solutions up to $n = 100$, which has a search space of 2^{10000}, a matter of 3000 orders of magnitude difference. Furthermore, optimal solutions are extremely rare: 1682 out of 2^{169} for $n = 13$, 11 out of 2^{196} for $n = 14$, and so on [7]. Clearly, we are going to need some extremely powerful pruning techniques.

For some values of n, the closed form upper bound calculated in the previous section is already the true upper bound. For such instances we simply need to find a solution that achieves the upper bound and, therefore, we can use an incomplete search. We take advantage of this by looking only for solutions of a particular form: those where all wastage lies in the 8×8 corners or within 3 rows of the edge, and the "center" is labeled perfectly with no wastage whatsoever.



(a) (b)

Fig. 3. Dynamic relaxation lookahead for still-life search: (a) shows darkened the area of lookahead, (b) shows the change in lookahead on labelling an additional column when super-row width is 4 and lookahead width is 8.

We call such solutions "center perfect". Our choice is motivated by the fact that, while there are many ways to label the center perfectly, there are very few ways to label the edge with the minimum amount of wastage. Since we are only allowed a very limited number of wastage on the entire board, they should not be used to deal with the center cells. Searching only for center perfect solutions, allows us to implement our solver more simply and to considerably reduce the search space.

4.1 Dynamic relaxation as lookahead

The main technique whereby we make solution finding feasible is through the use of dynamic relaxation as a lookahead technique. We label the board k rows at a time (call each set of k rows a "super-row"), and column by column within each super-row. Our implementation can set the width of the super row anywhere between 4 and 8, although our experimental evaluation has not shown any obvious difference in performance. At each node in our search tree, we perform a relaxation of the current subproblem, which we solve exactly as a lookahead. If we cannot find a (good enough) solution to the lookahead problem we fail and try another labelling.

The relaxation consists of all the unlabeled variables within k rows from the currently labeled variables, a 16×8 "thick edge" chunk of variables on each side, a width 3 edge down each side, the bottom two 8×8 corners, and the bottom width 3 edge (see Figure 3(a)). It also includes the 2 rows of already labeled variables immediately bordering on the unlabeled region. The values of these labeled variables form boundary conditions for the relaxed problem at this particular node. We will discuss the choice of this shape later on. We relax the still life constraints and objective function as before.

The relaxed subproblem is an optimisation problem. If by solving the relaxed subproblem, we find that the minimum amount of wastage among the unlabeled variables, plus the amount of wastage already found in the labeled variables, exceeds the wastage upper bound for the original problem, then clearly there are no solutions in the current branch and the node can be pruned. Since the constraint graph of the relaxed problem has low path-width, it can be solved in linear time using CP with caching. In practice though, we exploit symmetries and caching to solve it incrementally in constant time (see below). Lower bounds calculated for particular groups of variables (such as corners, edges, and super-rows) are cached. These can be reused for other subproblems. We also use these cached results as a table propagator, as they can immediately tell us which particular value choices will lead to wastage lower bounds that violate the current upper bound.

The relaxed problem can be solved incrementally in constant time as follows. When we travel to a child node, the new relaxed problem at that node is not that different from the one at the previous node (see Figure 3(a) and (b), respectively). A column of k variables from the previous relaxation has now been removed from the relaxation and labeled (new top dark column in Figure 3(b)), and a new column of k variables is added to the relaxation (new bottom dark column in Figure 3(b)). By caching previous results appropriately, it is possible to lookup all the solutions to the common part of the two relaxed problems (cached when solving the previous relaxation). Now we simply need to extend those solutions to the new column of k variables, which takes $O(2^k)$ time but is constant for a fixed k. There is sufficient time/memory to choose a k of up to 12. The larger k is, the greater the pruning achieved but the more expensive the lookahead is. In practice, $k = 8$ seems to work well. With $k = 8$, we are able to search around 1000 nodes per second and obtain a complete depth 400+ lookahead in constant time.

The choice of the shape of the lookahead is important, and is based on our conjectures as well as on extensive experimentation. The most important variables to perform lookahead on are the variables where wastage is most likely to be forced by the current boundary constraints. Conjecture 1 can be applied to our relaxed subproblems as well. It is very likely that the wastage forced by the boundary constraints can be found within a fixed number of rows from the current boundary. Thus, the variables near the boundary are the ones we should perform lookahead on. The reason for the 16×8 thick edge chunk of variables comes from our experimentation. It was found that the lookahead is by far the weakest around the “corners” of the subproblem, where we have boundary constraints on two sides. A lookahead using a thinner edge (initially width 3) often failed to see the forced wastage further in and, thus, allowed the solver to get permanently stuck around those “corners”. By using a thicker edge, we increase the ability of the lookahead to predict forced wastage and thus we get stuck less often. A 16×8 thick edge is still not sufficiently large to catch all forced wastage, but we are unable to make it any larger due to memory constraints, as the memory requirement is exponential in the width of the lookahead.

Interestingly, if Conjecture 1 is indeed true for the subproblems, and we can get around the memory problems and perform a lookahead with a sufficient width to catch all forced wastage, then the lookahead should approach the strength of an oracle, and it may be possible to find optimal solutions in roughly polynomial time! Indeed, from our results (see Table 5), the run time required to find optimal solutions does not appear to be growing anywhere near exponentially in n^2.

4.2 Search strategy

The search strategy is also very important. A traditional depth first branch and bound strategy is doomed to failure, as it will tend to greedily use up the wastage allowance to get around any problem it encounters. If it maxes out on the wastage bound early on, then it is extremely unlikely that it will be able to label the rest of the board without breaking the bound. However, we may not be able to tell this until much later on, since the center can often be labeled perfectly. Therefore, the search will reach extremely deep parts of the search tree, even though there is no chance of success and we will essentially be stuck forever. Instead, we need a smarter search strategy that has more foresight.

Intuitively, although labeling the top parts of the board optimally is difficult, it is much easier than labeling the final part of the board, since the final part of the board will have boundary constraints on every side, making it extremely difficult to get a perfect labeling with minimal wastage. Thus, we would like to save all our wastage allowance until the end. We accomplish this by splitting the search into two phases. The first phase consists of labeling all parts of the board other than the last 8 rows, and the second phase consists of the last 8 rows. In the first phase, we use a variation of limited discrepancy search (LDS), designed to preserve our wastage allowance as much as possible. We describe this in the next paragraph. If we manage to get to the last 8 rows, then we switch to a normal depth first search where the wastage allowance is used as necessary to finish off labeling the last few rows.

Our LDS-like algorithm is as follows. We define the discrepancy as the amount by which a value choice causes our wastage lower bound to increase. Note that this is totally different from defining it as the amount of wastage caused by a value choice. For example, if our lookahead tells us that a wastage of 10 is unavoidable among the unlabeled variables, and we choose a value with 1 wastage, after which our lower bound for the rest of the unlabeled variables is 9, then there is no discrepancy. This is because even though we chose a value that caused wastage, the wastage was forced. We are thus still labeling optimally (as far as our lookahead can tell). On the other hand, if the lookahead tells us that 10 wastage is unavoidable, and we choose a value with 0 wastage, after which our lower bound becomes 11, then there is a discrepancy. This is because, even though we did not cause immediate wastage, we have in fact labeled sub-optimally as this greedy choice causes more wastage later on. Discrepancy based on increases in wastage lower bounds is far better than one based on immediate wastage, as greedily avoiding wastage often leads to substantially more wastage later on! Note that by definition, there can be multiple values at each node with the same discrepancy, all of which should be searched at that discrepancy level.

Our search also differs from traditional LDS in the way that discrepancies are searched. Traditional LDS assumes that the branching heuristic is weak at the top of the tree, and therefore tries discrepancies at the top of the tree first. This involves a lot of backtracking to the top of the tree and is extremely inefficient for our case, since the branching heuristic for still life is not weak at the top of the tree and our trees are very deep. Hence, we order the search to try discrepancies at the leaves first, which makes the search much closer to depth-first search. This is substantially more efficient and is in fact crucial for solving our relaxations incrementally. We also modify LDS by adding local restarts. This is based on the fact that, for problems with as large a search space as this, complete search methods are doomed to failure, as mistakes require an exponential time to fix. Instead, we do incomplete LDS by performing local restarts at random intervals dependant on problem size, during which we backtrack by a randomised amount which averages to two super-rows. Value choices with the same number of discrepancies are randomly reordered each time a node is generated, so when a node is re-examined, the solver can take another path. Given the size of the search space, we do not wait until all nodes with the current discrepancy are searched before we try a higher discrepancy. Instead, after a reasonable amount of time (around 20 min) we give up and increase the discrepancy. Essentially, what we are doing is to try to use as little of our wastage at the top as possible and save more for the end. But if it feels improbable that we can label the top with so little wastage, then we use more of our allowance for the top.

5 Improving the upper bound

The incomplete search described in the previous section does not always find a solution equal to the closed form upper bound in the allocated time. For some of the cases in which this happens, we can perform a different kind of simplified search that also allows us to prove optimality. This simplified search is based on Equation (3) and, in particular, on the fact that wastage lower bounds are converted into live cell upper bounds by being rounded down. For instance, if our wastage lower bound gives us $live\_cells \leq \lfloor 100.75 \rfloor$, then we actually have $live\_cells \leq 100$. This means that the live cell upper bound we get is actually slightly stronger whenever the numerator is not divisible by 4. Since the upper bound is strengthened, it means that we do not always have to achieve the minimum amount of forced wastage in order to achieve the live cell upper bound. Let us define $spare = (2n^2 + 4n - min\_wastage) \mod 4$. The $spare$ value (0, 1, 2 or 3) tells you how many "unforced" (by the boundary constraints) wastage we can afford to have and still achieve the closed form upper bound. In other words, although the boundary constraints force a certain number of wastage, we can afford to have $spare$ wastage anywhere in the $n \times n$ region and still achieve the closed form upper bound.

It is interesting to consider the instances of n for which $spare = 0$. We believe that these are the instances where our closed form upper bound is most likely to be off by one. This is because when $spare = 0$, the closed form upper bound can only be achieved if there are no "unforced" wastage, and this can make the problem unsolvable. In other words, when $spare = 0$, any optimal solution that achieves the closed form upper bound must also be center perfect, since the edge and corner variables alone are sufficient to force all the wastage allowed. Thus, a complete search on center perfect solutions is sufficient to prove unsatisfiability of this value and improve the upper bound by 1. If we have already found a solution with 1 less live cell than the closed form upper bound, this will constitute a full proof of optimality. A complete search on center perfect solutions with minimum wastage is in fact quite feasible. This is because the corner, edges and center all have to be labeled absolutely perfectly with no unforced wastage, and there are very few ways to do this. For $spare > 0$ however, none of this is possible, as the spare "unforced" wastage can occur in an arbitrary position in the board, and proving that no such solution exists requires staggeringly more search.

6 Results

Our solver is written in C++ and compiled in g++ with O3 optimisation. It is run on a Xeon Pro 2.4GHz processor with 2Gb of memory. We run it for all n between 20 and 100. Smaller values of n have already been solved and are trivial for our solver. In Table 5, we list for each instance n, the lower bound (best solution found), the upper bound (marked with an asterisk if improved by 1 through complete search), the time in seconds taken to find the best solution, and the time in seconds taken to prove the upper bound. Each instance was run for at most 12 hours.

There are two precomputed tables which are used for all instances and are read from disk. These include tables for the 8×8 corner which takes $8s$ to compute and 16k memory to store, and a "thick edge" table consisting of a width 8 edge which takes 17 minutes to compute and ~400Mb of memory. Since these are calculated only once and used for all instances the time used is not reflected in the table.

As can be seen, "small" instances like $20 \leq n \leq 30$ which previously took days or were unsolvable can now be solved in a matter of seconds (after paying the above fixed costs). The problem gets substantially harder for larger n. Beyond $n \sim 40$, we can no longer reliably find the optimal solution. However, we are still able to find solutions which are no more than 3-4 cells off the true optimum all the way up to $n = 100$. The run times for the harder instances have extremely high variance due to the large search space and the scarcity of the solutions. If the solver gets lucky, it can find a solution in mere seconds. Otherwise, it can take hours. Thus, the run time numbers are more useful as an indication of what is feasible, rather than as a precise measure of how long it takes.

Our memory requirements are also quite modest compared to previous methods. The previous best method used an amount of memory exponential in n and could not be run for $n > 22$. Our solver, on the other hand, uses a polynomial amount of memory. For $n = 100$ for example, we use $\sim$ 400 Mb for the precomputed tables and $\sim$ 120 Mb for the actual search.

7 Conclusion

We reformulate the Maximum Density Still Life problem into one of minimising wastage. This allows us to calculate very tight upper bounds on the number of live cells and also gives insight into the patterns that can yield optimal solutions. Using a boundary based relaxation, we are able to prove in constant time an upper bound on the number of live cells for all n which is never off the true optimum by more than one for all $n \leq 50$. We further conjecture that this

n	lower	upper	lb. time	ub. time
20	**210**	210	0.1	0.1
21	**232**	232	0.4	0.1
22	**253**	253	3.4	0.1
23	**276**	276	0.1	0.1
24	**301**	301*	0.5	0.1
25	**326**	326	0.6	0.1
26	**352**	352*	2.1	6.2
27	**379**	379	51	0.1
28	406	407	1.4	0.1
29	**437**	437	2.5	0.1
30	**466**	466*	45	0.3
31	**497**	497	0.6	0.1
32	**530**	530*	1815	0.1
33	**563**	563	60	0.1
34	**598**	598	207	0.1
35	**632**	632*	1459	1.9
36	**668**	668	0.1	0.1
37	**706**	706	1.1	0.1
38	743	744	43	0.1
39	**782**	782*	3.4	3.3
40	**823**	823*	3.3	10.0
41	**864**	864*	553	2.1
42	906	907	1.6	0.1
43	949	950	2176	0.1
44	**993**	993*	285	3.4
45	**1039**	1039	3807	0.1
46	1084	1085*	101	3.4
47	1131	1132	244	0.1
48	1180	1181	265	0.1
49	**1229**	1229*	563	10
50	1279	1280	9.2	0.1
51	1330	1331	105	0.1
52	1381	1383	354	0.1
53	**1436**	1436	4326	0.1
54	1489	1490*	25219	3.3
55	1543	1545	296	0.1
56	1601	1602	484	0.1
57	1657	1659	816	0.1
58	1716	1717	1950	0.1
59	1774	1776	992	0.1
60	1834	1836	4.3	0.1
61	**1897**	1897	15648	0.1
62	1957	1959*	62	0.1
63	2021	2023	7594	0.1
64	2085	2087	389	0.1
65	2150	2152	137	0.1
66	2217	2218	2296	0.1
67	2284	2285	26137	0.1
68	2351	2354	3149	0.1
69	**2422**	2422	14755	0.1
70	2490	2493	641	0.1
71	2562	2564	3077	0.1
72	2634	2636	866	0.1
73	2706	2709	433	0.1
74	2781	2783	3575	0.1
75	2856	2858	5440	0.1
76	2932	2934*	5879	3.4
77	3009	3011	5298	0.1
78	3088	3090	20865	0.1
79	3166	3169	2768	0.1
80	3247	3249	8328	0.1
81	3327	3330	113	0.1
82	3410	3412	2	0.1
83	3492	3495	10849	0.1
84	3576	3579*	1083	3.6
85	3661	3664*	3666	0.6
86	3748	3751	7628	0.1
87	3835	3838	957	0.1
88	3923	3926	5047	0.1
89	4012	4015	1837	0.1
90	4102	4105*	7047	3.2
91	4193	4196	605	0.1
92	4286	4289	8843	0.1
93	4379	4382	2254	0.1
94	4473	4476	3669	0.1
95	4568	4571	10871	0.1
96	4664	4667	16801	0.1
97	4761	4764	36205	0.1
98	4859	4862*	3462	3.4
99	4958	4961	7660	0.1
100	5058	5062	15458	0.1

Table 5. Results on large max-density still life problems. Optimal answers are shown in bold. Upper bounds which are improved by complete search are shown starred. Times in seconds to one decimal place (in reality 0.1 usually represents a few milliseconds)

holds true for all n and thus we may have found the optimum value for all n up to a margin of error of 1. By using dynamic relaxation as a lookahead learning/pruning technique, we are able to produce a complete depth 400+ lookahead that can be calculated in constant time. This prunes the search so powerfully

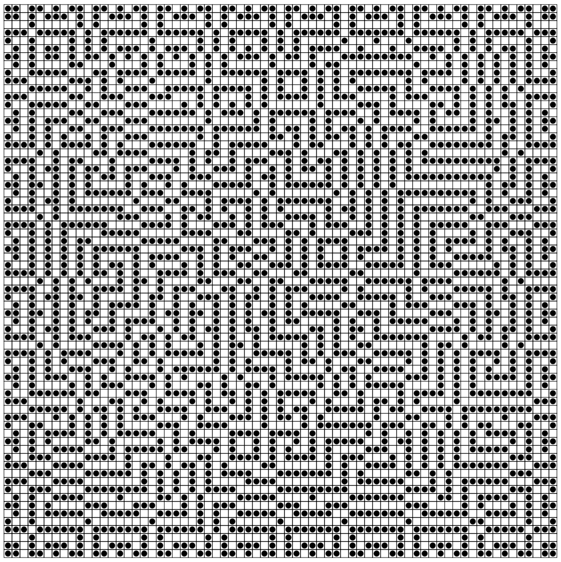


Fig. 4. An optimal solution to 69×69.

that we can find optimal (or near optimal) solutions for a problem where the search space grows as 2^{n^2}. The largest n for which the problem is completely solved is $n = 69$ (shown in Figure 4). Further, we have proved upper and lower bounds that differ by no more than 4 for all n up to 100. All of our solutions can be found at `www.csse.unimelb.edu.au/~pjs/still-life/`.

Acknowledgments. We would like to thank Michael Wybrow for helping us generate the still life pictures used in this paper. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

References

1. R. Bosch. Integer programming and conway's game of life. *SIAM Review*, 41(3):596–604, 1999.
2. 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.
3. R. Dechter. *Constraint Processing*. Morgan Kaufman, 2003.
4. N. Elkies. The still-life density problem and its generalizations. arXiv:math/9905194v1.
5. N. Elkies. The still-life density problem and its generalizations. *Voronoi's Impact on Modern Science: Book I*, pages 228–253, 1998. arXiv:math/9905194v1.
6. J. Larrosa and R. Dechter. Boosting search with variable elimination in constraint optimization and constraint satisfaction problems. *Constraints*, 8(3):303–326, 2003.
7. J. Larrosa, E. Morancho, and D. Niso. On the practical use of variable elimination in constraint optimization problems: 'still-life' as a case study. *Journal of Artificial Intelligence Research*, 2005.