A Multi-Level Memetic/Exact Hybrid Algorithm for the Still Life Problem

José E. Gallardo, Carlos Cotta, and Antonio J. Fernández

Dept. Lenguajes y Ciencias de la Computación, ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 - Málaga, Spain.
{pepeg,ccottap,afdez}@lcc.uma.es

Abstract. Bucket elimination (BE) is an exact technique based on variable elimination. It has been recently used with encouraging results as a mechanism for recombining solutions in a memetic algorithm (MA) for the still life problem, a hard constraint optimization problem based on Conway's game of life. This paper studies expanded multi-level models in which this exact/metaheuristic hybrid is further hybridized with branch-and-bound techniques. A novel variable clustering based recombination operator is also explored, with the aim of reducing the inherent time complexity of BE. Multi-parent recombination issues are analyzed as well. The obtained results are of higher quality than any previous metaheuristic approach, with large instances being solved to optimality.

1 Introduction

Conway's game of life [1] consists of an infinite checkerboard in which the only player places checkers on some of its squares. Each square has eight neighbors: the eight cells that share one or two corners with it. A cell is alive if there is a checker on it, and dead otherwise. The state of the board evolves iteratively according to three rules: (i) if a cell has exactly two living neighbors then its state remains the same in the next iteration, (ii) if a cell has exactly three living neighbors then it is alive in the next iteration, and (iii) if a cell has fewer than two or more than three living neighbors, then it is dead in the next iteration. The *maximum density still life problem* (MDSLP) is a challenging constraint optimization problem based on Conway's game. The problem is to find stable configurations, called *still lifes*, consisting of finite board configurations (of size $n \times n$) with a maximum number of living cells not changing along time. This problem has many practical applications in the control of discrete systems [2, 3] and is very hard to solve; though it has not been proven to be NP-hard, no polynomial-time algorithm for it is known.

The MDSLP has been tackled using different approaches. Bosch and Trick [4] used a hybrid approach mixing integer programming and constraint programming to solve the cases for $n = 14$ and $n = 15$ in about 6 and 8 days of CPU time respectively. Smith [5] considered a pure constraint programming approach to tackle the problem and proposed a formulation of the problem as a constraint satisfaction problem with 0-1 variables and non-binary constraints.

A dual formulation of the problem was also considered, and it was proven that this dual representation outperformed the initial one (although it could only solve instances up to $n = 10$). The best results for this problem were reported in [6], showing the usefulness of *bucket elimination* (BE). Their basic approach could solve the problem for $n = 14$ in about 10^5 seconds. Further improvements pushed the solvability boundary forward to $n = 20$ in about the same time. At any rate, it is clear that these exact approaches are inherently limited for increasing problem sizes, and their capabilities as anytime algorithms are unclear. Later, Cheng and Yap [7] tackled the problem via the use of ad-hoc constraints, but their results are far from the ones obtained previously by Larrosa *et al.*

To the best of our knowledge, the only evolutionary approach to the problem has been proposed by Gallardo *et al.* [8]. Their work showed that a MA endowed with BE could provide optimal or near-optimal solutions at an acceptable computational cost. A study of partial Lamarckism was also conducted, revealing that applying always the BE operator provides the best results. In this paper, we consider extended hybrid models in which the hybridization with exact techniques takes place at two levels: inside the MA, as an embedded operator, and outside it, in a cooperative model. We also study variants based on an alternative recombination operator, and on multi-parent recombination [9]. Experimental results reveal that the performance of the algorithm is improved significantly, showing that MAs stand as a practical alternative to exact techniques employed so far to obtain still-life patterns.

2 Bucket Elimination and the Still Life Problem

Bucket elimination [10] is a generic algorithm particularly adequate for solving *weighted constraint satisfaction problems* (WCSPs) [11]. A WCSP is defined by a set $X = \{x_1, \cdots, x_n\}$ of variables taking values from a set D of finite domains ($D_i \in D$ is the domain of x_i) and a set F of cost functions (also called soft constraints). Each $f \in F$ is defined over a subset of variables $var(f) \subseteq X$, called its *scope*. For each assignment t of all variables in the scope of a soft constraint f, $t \in f$ (i.e., t is *permitted*) if, and only if, t is allowed by the soft constraint. A complete assignment that satisfies every soft constraint represents a solution to the WCSP. The valuation of an assignment t is defined as the sum of costs of all functions whose scope is assigned by t. Permitted assignments receive finite costs expressing their degree of preference and forbidden assignments receive cost ∞. The optimization goal consists of finding the solution with the lowest valuation.

BE is based upon two operators over functions: (1) the sum of two functions f and g denoted $(f+g)$ is a new function with scope $var(f) \cup var(g)$ which returns for each tuple the sum of costs of f and g defined as $(f + g)(t) = f(t) + g(t)$; (2) the elimination of variable x_i from f, denoted $f \Downarrow i$, is a new function with scope $var(f) - \{x_i\}$ which returns for each tuple t the minimum cost extension of t to x_i, defined as $(f \cdot i)(t) = min_{a \in D_i}\{f(t \cdot (x_i, a))\}$ where $t \cdot (x_i, a)$ means the extension of t to the assignment of a to x_i. Observe that when f is a unary function, eliminating the only variable in its scope produces a constant.

BE works in two phases. In the first phase, the algorithm eliminates variables one at a time in reverse order according to an arbitrary variable ordering o. In the second phase, the optimal assignment is computed processing variables in increasing order. The elimination of variable x_i is done as follows: initially, all cost functions in F having x_i in their scope are stored in B_i (the so called *bucket of* x_i). Next, BE creates a new function g_i defined as the sum of all functions in B_i in which variable x_i has been eliminated. Then, this function is added to F, which is also updated by removing the functions in B_i. The consequence is that the new F does not contain x_i (all functions mentioning x_i were removed) but preserves the value of the optimal cost. The elimination of the last variable produces an empty scope function (i.e., a constant) which is the optimal cost of the problem. The second phase generates an optimal assignment of variables. It uses the set of buckets that were computed in the first phase: starting from an empty assignment t, variables are assigned from first to last according to o. The optimal value for x_i is the best value regarding the extension of t with respect to the sum of functions in B_i.

In order to apply the general BE template to the MDSLP, let us first introduce some notation. A board configuration for a $n \times n$ instance will be represented by a n-dimensional vector $(r_1, r_2, \ldots, r_n)$. Each vector component encodes (as a binary string) a row, so that the j-th bit of row r_i (noted r_{ij}) indicates the state of the j-th cell of the i-th row (a value of 1 represents an alive cell and a value of 0 a dead cell). Let $Zeroes(r)$ be the number of zeroes in binary string r and let $Adjacents(r)$ be the maximum number of adjacent living cells in row r. If r_i is a row and r_{i-1} and r_{i+1} are the rows above and below r, then $Stable(r_{i-1}, r, r_{i+1})$ is a predicate satisfied if, and only if, all cells in r are stable.

The formulation has n cost functions f_i ($i \in \{1..n\}$). For $i \in \{2..n-1\}$, f_i is ternary with scope $var(f_i) = \{r_{i-1}, r_i, r_{i+1}\}$ and is defined as:

$$f_i(a,b,c) = \begin{cases} \infty & : \neg Stable(a,b,c) \\ \infty & : a_1 = b_1 = c_1 = 1 \\ \infty & : a_n = b_n = c_n = 1 \\ Zeroes(b) & : \text{otherwise} \end{cases} \quad (1)$$

As to f_1 and f_n, they are binary with scopes $var(f_1) = \{r_1, r_2\}$ and $var(f_n) = \{r_{n-1}, r_n\}$, and are defined similarly to $f_i(\cdot)$, assuming a boundary of dead cells. Notice in these definitions that stability is not only required within the pattern, but also in the surrounding dead cells.

Due to the sequential structure of the corresponding constraint graph [6], the model can be readily approached with BE. Figure 1 shows the corresponding algorithm. Function BE takes two parameters: n, the size of the instance to be solved, and D, the domain for each variable (row) in the solution. If domain D is set to $\{0..2^n - 1\}$ (i.e., a set containing all possible rows) the function implements an exact method that returns the optimal solution for the problem instance (as the number of dead cells) and a vector corresponding to the rows of that solution.

```
      function BE(n, D)
 1:       for a, b ∈ D do
 2:           g_n(a, b) := min_{c∈D}{f_{n−1}(a, b, c) + f_n(b, c)}
 3:       end for
 4:       for i := n − 1 downto 3 do
 5:           for a, b ∈ D do
 6:               g_i(a, b) := min_{c∈D}{f_{i−1}(a, b, c) + g_{i+1}(b, c)}
 7:           end for
 8:       end for
 9:       (r_1, r_2) := argmin_{a,b∈D}{g_3(a, b) + f_1(a, b)}
10:       opt := g_3(r_1, r_2) + f_1(r_1, r_2)
11:       for i := 3 to n − 1 do
12:           r_i := argmin_{c∈D}{f_{i−1}(r_{i−2}, r_{i−1}, c) + g_{i+1}(r_{i−1}, c)}
13:       end for
14:       r_n := argmin_{c∈D}{f_{n−1}(r_{n−2}, r_{n−1}, c) + f_n(r_{n−1}, c)}
15:       return (opt, (r_1, r_2, . . . , r_n))
      end function
```

Fig. 1. Bucket Elimination for the MDSLP.

3 Memetic and Hybrid Algorithms for the MDSLP

As mentioned before, the algorithmic model we consider is based on the hybridization of MAs with exact techniques at two levels: within the MA (as an embedded operator), and outside it (in a cooperative model). An overall description of the basic hybridization scheme at the first level is provided in next subsection. Subsequently, we will explore some variants based on variable clustering and multi-parent recombination, before proceeding to the second level of hybridization.

3.1 A Memetic Algorithm with BE for the MDSLP

The MA described in [8] evolves configurations represented as binary $n \times n$ matrices; infeasible solutions are dealt via a stratified penalty-based fitness function:

$$f(r) = n^2 - \sum_{i=1}^{n}\sum_{j=1}^{n} r_{ij} + K \sum_{i=0}^{n+1}\sum_{j=0}^{n+1} \left[r'_{ij}\phi_1(\eta_{ij}) + (1 - r'_{ij})\phi_0(\eta_{ij}) \right] \qquad (2)$$

where r' is an $(n+2) \times (n+2)$ binary matrix obtained by embedding r in a frame of dead cells, K and K' are constants, η_{ij} is the number of alive neighbors of cell (i, j), and ϕ_0, $\phi_1 : \mathbb{N} \longrightarrow \mathbb{N}$ are two functions that take the number of alive neighbors of a cell, and return how many of them should be flipped to have a stable configuration (depending on whether the central cell is alive or not). Constants K and K' are set with the primary goal of decreasing the number of cells in an unstable state; if this were not possible, the secondary goal was to decrease the level of instability of these cells.

It turns out that this fitness function is easily decomposable, a fact that is exploited within the MA by means of a local improvement strategy based on tabu search (TS). This TS strategy explores the neighborhood $\mathcal{N}(r) =$

$\{s \mid \text{Hamming}(r, s) = 1\}$, i.e., the set of solutions obtained by flipping exactly one cell in the configuration.

The binary representation allows the use of standard recombination operators for binary strings, but these blind operators performed poorly. Hence, problem-aware operators were considered. To be precise, BE was used to implement a recombination operator that explored the dynastic potential [12] (possible children) of the solutions being recombined, providing the best solution that could be constructed without introducing implicit mutation. That is, let $x = (x_1, x_2, \cdots, x_n)$ and $y = (y_1, y_2, \cdots, y_n)$ be two board configurations for a $n \times n$ instance of the MDSLP. Then, $\text{BE}(n, \{x_1, x_2, \cdots, x_n, y_1, y_2, \cdots, y_n\})$ calculates the best feasible configuration that can be obtained by combining rows in any of the parents (x and y). Notice that the described operator can be generalized to recombine any number of board configurations like $\text{BE}(n, \cup_{x \in S}\{x_i \mid i \in \{1..n\}\})$ where S is a set comprising the solutions to be recombined. This is one of the algorithmic variants that will be explored next.

3.2 Variable Clustering and Multi-Parent Recombination

The complexity of BE depends on the problem structure (as captured by its constraint graph G) and the ordering o of variable elimination. According to [13], the complexity of BE along ordering o is time $\Theta(Q \times n \times d^{w^*(o)+1})$ and space $\Theta(n \times d^{w^*(o)})$, where d is the largest domain size, Q is the cost of evaluating cost functions (usually assumed $\Theta(1)$), and $w^*(o)$ is the maximum width of nodes in the induced graph of G relative to o (check [13] for details).

A well-known technique for reducing this computational cost in the context of constraint processing is variable clustering [14]. This approach merges several variables into a metavariable preserving the problem semantics. Inspired by this technique, variables corresponding to consecutive rows in a MDSLP solution can be clustered. We will denote by C_iBE the recombination operator that performs bucket elimination on a new domain obtained by clustering every group of i consecutive rows in a metavariable. This recombination operator thus provides the best feasible configuration that can be obtained by combining groups of i rows taken from the parents. Figure 2 shows the resulting algorithm for C_2BE when n is even. The procedure starts by defining the new domain for variables obtained by grouping every two consecutive rows in the original domain. Then, bucket elimination is performed for the new domain. The number of iterations of the loop in line 5 is reduced to one half with respect to the original algorithm, and the range for loops instantiating variables is also halved, thus reducing the time complexity of the algorithm at the expense of losing information.

One of the possibilities for alleviating the loss of alternatives for combining the information is the consideration of multi-parent recombination [9]. Following the scheme depicted in Section 3.1, an arbitrary number of solutions can contribute their constituent rows for constructing a new solution. In the worst case, this results in a linear increase in the size of domains, and thus does not affect the asymptotical complexity of BE, as long as the number of parents is bounded by a constant. One of the goals of the experimentation has been to

```
      function C_2BE (n, D)
 1:       D' := { {D_{2i-1}, D_{2i}} | i ← {1..|D|/2} }
 2:       for a, b ∈ D' do
 3:           g_n(a,b) := min_{c∈D'}{f_{n-4}(a_1,a_2,b_1) + f_{n-3}(a_2,b_1,b_2) + f_{n-2}(b_1,b_2,c_1)+
                                    f_{n-1}(b_2,c_1,c_2) + f_n(c_1,c_2)}
 4:       end for
 5:       for i := 1 to (n-6)/2 do
 6:           for a, b ∈ D' do
 7:               g_{n-i}(a,b) := min_{c∈D'}{ f_{n-2(i+2)+1}(a_1,a_2,b_1) + f_{n-2(i+2)}(a_2,b_1,b_2)+
                                             g_{n-i+1}(b,c)}
 8:           end for
 9:       end for
10:       κ := n - (n-6)/2
11:       (α, β) := argmin_{a,b∈D'}{g_κ(a,b) + f_1(a_1,a_2)}
12:       r_1 := α_1; r_2 := α_2; r_3 := β_1; r_4 := β_2
13:       opt := g_κ(α,β) + f_1(r_1,r_2)
14:       for i := 5 to n - 3 step 2 do
15:           κ := κ + 1
16:           α := argmin_{a∈D'}{f_{i-1}(r_{i-3},r_{i-2},r_{i-1}) + g_κ(r_{i-1},a)}
17:           r_i := α_1; r_{i+1} := α_2
18:       end for
19:       α := argmin_{a∈D'}{f_{n-1}(r_{n-2},a_1,a_2) + f_n(a_1,a_2)}
20:       r_{n-1} := α_1; r_n := α_2
21:       return (opt, (r_1, r_2, ..., r_n))
      end function
```

Fig. 2. BE with clusters formed by two rows for even sizes for the MDSLP.

check whether there exists some optimal tradeoff between these two strategies (variable clustering and multi-parent recombination), and indeed whether any of them can contribute to the global improvement of the hybrid algorithm.

3.3 A Beam Search Hybrid Algorithm

Gallardo *et al.* [15] have shown that hybridizing a MA with a branch-and-bound-based Beam Search (BS) algorithm can provide excellent results for some combinatorial optimization problems. We show here that this is also the case for the MDSLP. We consider a hybrid algorithm that executes the BS and the MA in an interleaved way. The goal is combining synergistically these two different approaches, exploiting the capability of BS for identifying provably good regions of the search space, and the strength of the MA for exploring these.

The resulting algorithm is depicted in Figure 3. Here, $\overline{r}$ denotes the reflection value of r, and $v{+\!\!+}r$ is the vector obtained by concatenating r to the end of v. Function Hybrid(n, k, l_0) constructs a branch and bound tree whose leaves are all possible $n \times n$ board configurations whose rows are symmetric (this symmetry constraint is required to keep the branching factor at a manageable level for the range of instance sizes considered). Internal nodes at level i represent partially specified (up to the ith row) board configurations. The tree is traversed using a BS algorithm that explores the tree in a breadth-first way maintaining only the best k nodes at each level of the tree. In order to rank nodes, a quality measure is defined on them, whose value is either ∞ if the partial configuration is unstable, or its number of dead cells otherwise. Parameter l_0 indicates how

```
      function Hybrid (n, k, l_0)
 1:     sol := ∞
 2:     q := { () }
 3:     for i := 1 to n do
 4:       q' := {}
 5:       for c ∈ q do
 6:         for r := 0 to 2^⌈n/2⌉ − 1 do
 7:           q' := q' ∪ {c++(r or r̄)}
 8:         end for
 9:       end for
10:       q := select best k nodes from q'
11:       if (i ≥ l_0) then
12:         initialize MA population with best nodes from q'
13:         run MA
14:         sol := min (sol, MA solution)
15:       end if
16:     end for
17:     return sol
      end function
```

Fig. 3. Hybrid algorithm for the MDSLP.

many levels the BS descends before starting running the MA, and can be used to control the balance between the MA and the BS. For each execution of the MA, its population is initialized using the best *popsize* nodes in the current level of exploration. Since these are partial solutions, they must be first converted into full solutions, e.g., by completing remaining rows randomly. After running the MA, its solution is used to update the incumbent solution. This process is repeated until the search tree is exhausted.

4 Experimental Results

A set of experiments for problem sizes from $n = 12$ up to $n = 20$ has been realized (recall that optimal solutions are known up to $n = 20$). The experiments were done in all cases using a steady-state MA ($popsize = 100$, $p_m = 1/n^2$, $p_X = 0.9$, binary tournament selection). Aiming to maintaining diversity, duplicated individuals were not allowed in the population. For the different versions of the hybrid algorithm described in Section 3.3, the setting of parameters was $k = 2000$ and $l_0 = 0.3n$, i.e, the best 2000 nodes were kept on each level of the BS algorithm, and 30% of the levels of the BS tree were initially descended before starting running the MA. All algorithms were run until an optimal solution was found or a time limit was exceeded. This time limit was set to 3 minutes —on a P4 (2.4GHz and 512MB RAM) under SuSE Linux— for problem instances of size 12 and were gradually incremented by 60 seconds for each size increment. For each algorithm and each instance size, 20 independent executions were run.

First of all, experiments have been done to explore the effects of multi-parent recombination in a MA endowed with BE for performing recombination as described in Section 3.1 (MA-BE). Figure 4 (left) shows the results obtained by MA-BE for different number of parents being recombined (arities 2, 4 and 8). For $arity = 2$, the algorithm was able to find the optimum solution for all in-

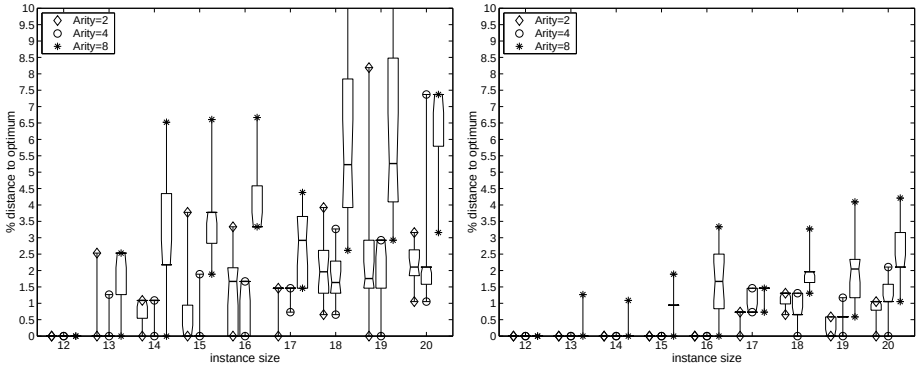

Fig. 4. Relative distances to optimum for different arities for MA-BE (left) and HYB-MA-BE (right) for sizes ranging from 12 up to 20. Each box summarizes 20 runs.

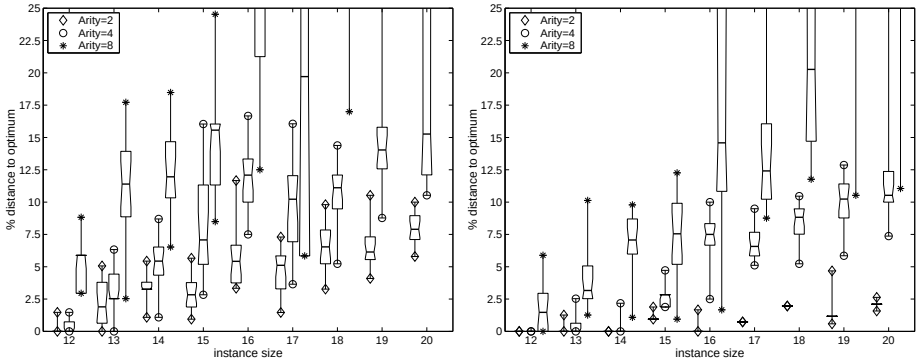

Fig. 5. Relative distances to optimum for different arities for MA-C_2BE (left) and HYB-MA-C_2BE (right) for sizes from 12 up to 20. Each box summarizes 20 runs.

stances except for $n = 18$ and $n = 20$ (the relative distance to the optimum is less than 1.04% in these cases). Note that results for $n = 19$ and $n = 20$ were obtained in just 10 and 11 minutes per run respectively. As a comparison, recall that the approach in [6] respectively requires over 15 hours and over 2 days for these same instances, and that other approaches are unaffordable for $n > 15$. Executions with $arity = 4$ cannot find optimum solutions for the remaining instances, but note that the distribution always improves. Clearly, the performance of the algorithm degrades when combining more than 4 parents due to the higher computational cost.

Subsequent experiments were conducted to evaluate the C_iBE recombination operator for $i \in \{2, 3\}$. Results are shown in Figure 5 (left) and 6 (left), and reveal that the performance of the algorithm is worse, as it only finds the optimal solution for the smallest instance sizes. The computational costs saved by clustering variables does not compensate the loss of information induced, even

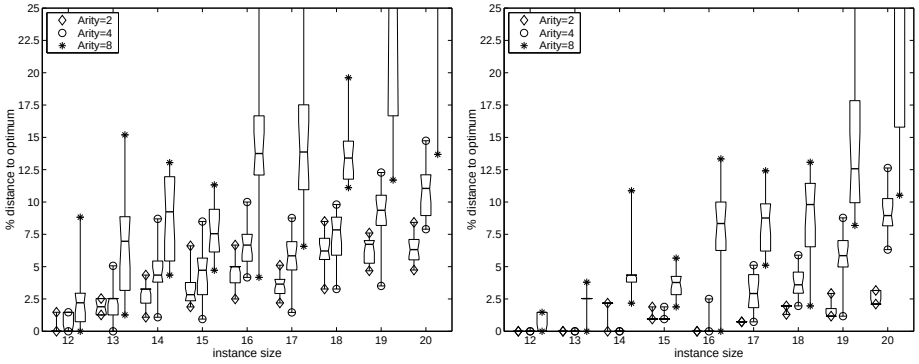

Fig. 6. Relative distances to optimum for different arities for MA-C_3BE (left) and HYB-MA-C_3BE (right) for sizes from 12 up to 20. Each box summarizes 20 runs.

in the presence of multi-parent recombination, and the combination of these two strategies is counter-productive.

We finally approach the two-level hybrid algorithm. Figures 4 (right), 5 (right), and 6 (right) show the results obtained using MA-BE, MA-C_2BE, and MA-C_3BE in the MA part. The performance is significantly improved over the original MA. Note that HYB-MA-BE using an arity of 2 parents is able to find the optimum for all cases except for $n = 18$ (this instance is solved with $arity = 4$). All distributions for different instance sizes are improved in an significant manner. For $n < 17$ and $arity \in \{2, 4\}$, the algorithm consistently finds the optimum in all runs. For other instances and $arity = 2$, the solution provided by the algorithm is always within a 1.05 % of the optimum, except for $n = 18$, for which the relative distance to the optimum for the worst solution is 1.3%. The results of HYB-MA-C_2BE and HYB-MA-C_3BE are worse than those of HYB-MA-BE, but note however that the hybridization with the BS algorithm is beneficial also in this case, as it improves the distributions with respect to MA-C_2BE and MA-C_3BE.

5 Conclusions and Future Work

The high space complexity of BE as an exact technique [10], makes this approach impractical for large instances. In this work, we have presented several proposals for the hybridization of Bucket Elimination (BE) with MAs and BS, and showed that it represents a worthwhile model. The experimental results have been very positive, solving to optimality large instances of a hard constrained problem. We have also studied the influence that variable clustering and multi-parent recombination have on the performance of the algorithm. The results indicate that variable clustering is detrimental in this problem, but multi-parent recombination can help to improve the results obtained by previous approaches.

One interesting extension to this work is to improve the bounds used in the BS algorithm. To do so, we are currently considering the technique of mini–buckets [16]. Work is in progress in this area.

Acknowledgements. This work was partially supported by Spanish MCyT under contracts TIN2004-7943-C04-01 and TIN2005-08818-C04-01.

References

1. Gardner, M.: The fantastic combinations of John Conway's new solitaire game. Scientific American **223** (1970) 120–123
2. Gardner, M.: On cellular automata, self-reproduction, the garden of Eden and the game of "life". Scientific American **224** (1971) 112–117
3. Gardner, M.: Wheels, Life, and Other Mathematical Amusements. W.H. Freeman, New York (1983)
4. Bosch, R., Trick, M.: Constraint programming and hybrid formulations for three life designs. In: CP-AI-OR. (2002) 77–91
5. Smith, B.M.: A dual graph translation of a problem in 'life'. In Hentenryck, P.V., ed.: Principles and Practice of Constraint Programming - CP'2002. Volume 2470 of Lecture Notes in Computer Science., Ithaca, NY, USA, Springer (2002) 402–414
6. Larrosa, J., Morancho, E., Niso, D.: 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
7. Cheng, K., Yap, R.: Ad-hoc global constraints for life. In van Beek, P., ed.: Principles and Practice of Constraint Programming – CP'2005. Volume 3709 of Lecture Notes in Computer Science., Sitges, Spain, Springer (2005) 182–195
8. Gallardo, J.E., Cotta, C., Fernández, A.J.: A memetic algorithm with bucket elimination for the still life problem. In Gottlieb, J., Raidl, G.R., eds.: EvoCOP. Volume 3906 of Lecture Notes in Computer Science., Springer (2006) 73–85
9. Eiben, A., Raue, P.E., Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving From Nature III. Springer-Verlag (1994) 78–87, LNCS 866
10. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence **113** (1999) 41–85
11. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. Journal of the ACM **44** (1997) 201–236
12. Radcliffe, N.: The algebra of genetic algorithms. Annals of Mathematics and Artificial Intelligence **10** (1994) 339–384
13. Larrosa, J., Morancho, E.: Solving 'still life' with soft constraints and bucket elimination. In Rossi, F., ed.: Principles and Practice of Constraint Programming - CP 2003. Volume 2833 of Lecture Notes in Computer Science., Kinsale, Ireland, Springer (2003) 466–479
14. Dechter, R., Pearl, J.: Tree clustering for constraint networks. Artificial Intelligence (1989) 353–366
15. Gallardo, J., Cotta, C., Fernández, A.: On the hybridization of memetic algorithms with branch-and-bound techniques. IEEE Transactions on Systems, Man and Cybernetics, part B (2006) (to appear).
16. Dechter, R.: Mini-buckets: A general scheme for generating approximations in automated reasoning. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan, Morgan Kaufmann (1997) 1297–1303