Search Space Reduction and Russian Doll Search

Kenil C.K. Cheng and **Roland H.C. Yap**
National University of Singapore
3 Science Drive 2, Singapore, 117543
{chengchi, ryap}@comp.nus.edu.sg

Abstract

In a constraint optimization problem (COP), many feasible valuations lead to the same objective value. This often means a huge search space and poor performance in the propagation between the objective and problem variables. In this paper, we propose a different modeling and search strategy which focuses on the cost function. We show that by constructing a dual model on the objective variables, we can get strong propagation between the objective variables and the problem variables which allows search on the objective variables. We explain why and when searching on the objective variables can lead to large gains. We present a new Russian Doll Search algorithm, ORDS, which works on objective variables with dynamic variable ordering. Finally, we demonstrate using the hard Still-Life optimization problem the benefits of changing to the objective function model and ORDS.

Introduction

In this paper, we propose to do search on the objective (auxiliary) variables for constraint optimization problems. The contributions of this paper are as follows. Firstly, we show the importance of modeling the problem which takes into account both the problem and objective variables. By using a dual model, one can get strong and efficient propagation between the problem and objective variables. This approach allows search on the objective variables to propagate to the problem variables and vice versa; hence, it can significantly improve the amount of propagation from new bounds in branch and bound search (BnB). We explain when this approach can be beneficial.

When the problem constraints of a constraint optimization problem are tight, the Russian Doll Search (RDS) algorithm (Verfaillie, Lemaître, & Schiex 1996) often computes tight bounds on the objectives of the sub-problems, which drastically prune the search tree. We give a new variant of RDS, ORDS, which searches on the objective variables. It is also more flexible than RDS as it can use dynamic variable ordering. The use of ORDS can give further gains to search in the objective space.

Finally, we experiment with two objective variable models on the difficult Still-Life optimization problem. Still-Life is very difficult for both pure CP and IP techniques to solve using branch and bound. We exemplify our approach with the super-cell (Smith 2002) and the super-row (Cheng & Yap 2006) models. The results show that, for $n = 10$, simply shifting the branch and bound search from the problem space to the objective space leads to a $13.8\times$ search space reduction. Applying ORDS in the objective space gives a $158\times$ reduction in search space. Using the super-row model, which has an even smaller objective space, gives a $2445.8\times$ search space reduction. We can go to $n = 16$ with 11002 backtracks, which is the largest instance solved by complete branch and bound on a pure CSP model. Our approach can be easily applied to solve other optimization problems in CP systems such as SICStus Prolog, ILOG Solver, etc. The promise is that significant gains can be obtained on optimization problems whose have a size difference between the problem and the objective spaces.

Preliminaries

A *constraint satisfaction problem* (CSP) is a pair $\mathcal{P} = \langle X, \mathcal{C} \rangle$, where $X = \{x_1, \ldots, x_n\}$ is a set of *variables*, and $\mathcal{C}$ is a set of *constraints*.[1] Each variable x_i can only take values from its *domain* $dom(x_i)$, which is a set of integers. A *valuation* θ is a mapping of variables to integer values, written as $\{(x_1, d_1), \ldots, (x_n, d_n)\}$. Let $vars$ be the function that returns the set of (free) variables appearing in a constraint or valuation. A k-ary constraint $C \in \mathcal{C}$ on an ordered set of k distinct variables, sometimes written as $C(x_1, \ldots, x_k)$, is a subset of the Cartesian product $dom(x_1) \times \cdots \times dom(x_k)$ that restricts the values the variables in C can take simultaneously. A valuation θ *satisfies* C, a.k.a. a *solution* of C, iff $\theta \in C$. Solving a CSP requires finding a value for each variable from its domain so that all constraints are satisfied.

Two constraints C_1 and C_2 are *equivalent*, denoted by $C_1 \equiv C_2$, iff for any valuation θ, we have $\theta \in C_1 \iff \theta \in C_2$. A constraint C is *generalized arc consistent* (GAC) iff for every $x_i \in vars(C)$ and every $d \in dom(x_i)$, $C \wedge (x_i = d) \not\equiv false$.

We can transform a CSP $\mathcal{P} = \langle X, \mathcal{C} \rangle$ into another CSP $\mathcal{P}' = \langle X', \mathcal{C}' \rangle$ using *dual encoding* (Dechter & Pearl 1989): For every $C_i \in \mathcal{C}$ we make a *dual variable* $x'_i \in X'$ whose domain is C_i. For any $C_i, C_j \in \mathcal{C}$ such that

[1] In this paper we use the terms "a set of constraints" and "a conjunction of constraints" interchangeably.

$vars(C_i) \cap vars(C_j) \neq \emptyset$, we create a binary constraint $C'_{i,j}(x'_i, x'_j) \in \mathcal{C}'$ such that $\{(x'_i, \theta_i), (x'_j, \theta_j)\} \in C'_{i,j}$ iff $\theta_i \wedge \theta_j \not\equiv false$. The CSP $\mathcal{P}'$ is also called a *dual model* of $\mathcal{P}$. By construction, we can map every solution in $\mathcal{P}$ to exactly one solution in $\mathcal{P}'$ and vice versa.

A *constraint optimization problem* (COP) is a pair $\mathcal{Q} = \langle \mathcal{P}, z \rangle$ where $\mathcal{P} = \langle X \cup Y, \mathcal{C} \rangle$ is a CSP which has at least one constraint $C_j(y_j, x_1, \ldots, x_k) \in \mathcal{C}$ of the form

$$y_j = f_j(x_1, \ldots, x_k)$$

where $y_j \in Y$, and f_j is a function that maps each valuation of $\{x_1, \ldots, x_k\} \subseteq X$ to some integer $d \in dom(y_j)$. The *objective* z is defined as

$$z = \sum_{y \in Y} y. \quad (1)$$

Our goal is to find the *optimal value*, i.e. the maximal value, of z for $\mathcal{Q}$. The solution with optimal z is the *optimal solution*. While the optimal value is unique, there may be many optimal solutions. In the rest of this paper, we call every $x_i \in X$ a *problem variable* and every $y_j \in Y$ an *objective variable*. Finally, we refer to the Cartesian product of the domains of the problem variables X as the *problem space (spanned on X)*, and that of the objective variables Y as the *objective space (spanned on Y)*. To avoid confusion, we will use the notation $\mathcal{P}$-var and $\mathcal{O}$-var to refer to a problem and an objective variable respectively, and $\mathcal{P}$-space and $\mathcal{O}$-space to the problem and the objective space respectively.

Objective Space verses Problem Space

Consider a constraint optimization problem $\mathcal{Q} = \langle \mathcal{P}, z \rangle$ where $\mathcal{P} = \langle X \cup Y, \mathcal{C} \rangle$ is a CSP that comprises n constraints $x_i \neq x_{i+1} \iff y_i = 1$. $X = \{x_1, \ldots, x_{n+1}\}$ and $Y = \{y_1, \ldots, y_n\}$. Every $\mathcal{P}$-var x_i has a domain $\{1, \ldots, d\}$ and the $\mathcal{O}$-vars have $0/1$ domains. The goal is to maximize $z = y_1 + \cdots + y_n$.

In this example, finding the optimal value for z by depth-first branch and bound search (BnB) in the $\mathcal{O}$-space requires $O(n)$ backtracks. One potential sequence of search paths is

$$\begin{array}{c} \{(y_1,0),(y_2,0),\ldots,(y_{n-1},0),(y_n,0)\} \\ \{(y_1,0),(y_2,0),\ldots,(y_{n-1},0),(y_n,1)\} \\ \{(y_1,0),(y_2,0),\ldots,(y_{n-1},1),(y_n,1)\} \\ \cdots \\ \{(y_1,0),(y_2,1),\ldots,(y_{n-1},1),(y_n,1)\} \\ \{(y_1,1),(y_2,1),\ldots,(y_{n-1},1),(y_n,1)\} \end{array}$$

Since all constraints $x_i = x_{i+1}$ or $x_i \neq x_{i+1}$ form a chain-like structure, maintaining GAC during search is enough to check the satisfiability of constraints (Freuder 1982).

The number of backtracks in the $\mathcal{P}$-space, however, is much larger and grows quadratically. Figure 1 gives the search time (in seconds) for $d = 10$ and $d = 50$ against various n in SICStus Prolog 3.12.7. The usual BnB in the $\mathcal{P}$-space is much worse here because many solutions are mapped to the same value — given any value in $dom(x_i)$, there are $d-1$ values in $dom(x_{i+1})$ to make $y_i = 1$. Hence, the search is likely to visit similar regions that lead to the same objective value. Conversely, the lower bounds generated during the branch and bound search are ineffective for pruning as different combinations of 1's among y_j's can satisfy the bound condition, leaving too many choices for x_i's.

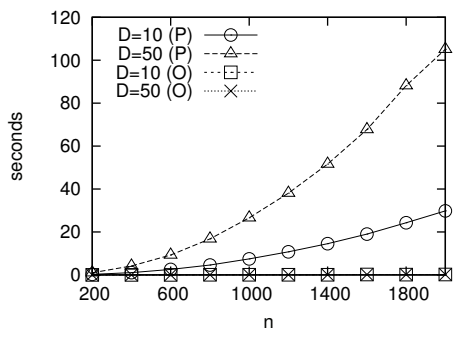

Figure 1: The search time in seconds on the y-axis with domain sizes $d = 10, 50$ against the number of variables n. "O" means the $\mathcal{O}$-vars Y and "P" means the $\mathcal{P}$-vars X.

The cost of a search depends on both the size of the search space and the time complexity of the consistency (or satisfiability testing) algorithm. Search in the $\mathcal{O}$-space will be beneficial if we can test the satisfiability with much less search in the $\mathcal{P}$-space. Since enforcing GAC on the dual model is strictly stronger than on the original model (Stergiou & Walsh 1999), searching in the $\mathcal{O}$-space may be preferred if we work on the dual encoded CSP. Consider a constraint optimization problem $\mathcal{Q} = \langle \mathcal{P}, z \rangle$ where $\mathcal{P} = \langle X \cup Y, \mathcal{C} \rangle$, the size of X and Y are respectively n and m, the maximum arity of the constraints in $\mathcal{C}$ is k, the maximum domain size of variables in X and Y are d and e, respectively, and $\mathcal{C}$ contains h constraints. Thus, the size of $\mathcal{P}$-space is $O(d^n)$ and the size of the $\mathcal{O}$-space is $O(e^m)$. Maintaining GAC on $\mathcal{C}$ takes $O(hkd^k)$ time (Bessière & Règin 1997), while on the constraints of the dual model of $\mathcal{P}$ takes $O(h^3 d^k)$ time (Samaras & Stergiou 2005). Assuming the propagation between the $\mathcal{O}$-vars, the $\mathcal{P}$-vars and z takes $O(g)$ time, and we can check the satisfiability of the problem constraints in $O(f)$ time, it takes $O(d^n \cdot hkd^k \cdot g)$ time to search in the $\mathcal{P}$-space and $O(e^m \cdot h^3 d^k \cdot g \cdot f)$ time to search in the $\mathcal{O}$-space. Hence, a search in the $\mathcal{O}$-space is faster when $e^m h^2 f$ is significantly smaller than $d^n k$. Notice that for tight or tractable constraints, f can be a simple polynomial of n, d, etc. For instance, we found f to be almost constant for the Still-Life problem, which means search in the $\mathcal{P}$-space is almost unnecessary.

Russian Doll Search on the Objective Space

We now describe ORDS, an extension of the RDS algorithm (Verfaillie, Lemaître, & Schiex 1996), which explores the $\mathcal{O}$-space directly. Figure 2 shows the pseudo-code. Given a constraint optimization problem $\mathcal{Q} = \langle \mathcal{P}, z \rangle$ where $\mathcal{P} = \langle X \cup Y, \mathcal{C} \rangle$ is a CSP, $Y = \{y_1, \ldots, y_m\}$ is the set of $\mathcal{O}$-vars and z is the objective defined in (1), ORDS finds the optimal value of z by executing the branch and bound search (BnB)

m times. At the j-th iteration, the objective

$$z_j = \sum_{i=1}^{j} y_i$$

is maximized by searching in the $\mathcal{O}$-space spanned on $Y' = \{y_1, \ldots, y_j\}$ (line 1). Assume the optimal value is ub. Since the value of z_j can only be smaller when more constraints are considered in later iterations (as well as in the original problem), ub serves as a "safe" upper bound on z_j, i.e. $z_j \leq ub$. Unlike the original RDS which employs additional book-keeping data structures to maintain these upper bounds, ORDS has a simpler implementation by harnessing the underlying constraint solver — we decompose the objective in (1) as m ternary equality constraints:

$$z_j = \begin{cases} y_j + z_{j-1} & \text{if } 1 \leq j \leq m \\ 0 & \text{otherwise} \end{cases}$$

so that $z = z_m$. The upper bound is enforced by adding a unary constraint $z_j \leq ub$ to the set of constraints $\mathcal{C}'$ (line 2).[2] The maximum value of the original objective z is returned after the m-th iteration for z_m (line 3).

The branch and bound algorithm BnB is straightforward. Initially, a naive lower bound $lb = -\infty$ is assigned to the objective z. This lower bound is increased iteratively until the constraints $\mathcal{C}' \cup \{z > lb\}$ are unsatisfiable (line 4). The optimal value is then lb. Depth-first search (DFS) is applied to determine whether the constraints are satisfiable: Recursively, the variables in Y are instantiated. It returns *true* if a solution exists and *false* otherwise.

Notice that all bounds on z_j's are "automatically" maintained by constraint propagation. This makes ORDS very easy to implement and couple with the constraint solver tightly in a CP system. In addition, the requirement of a static variable ordering by the original RDS algorithm is removed; the "multi-way" propagation among the constraints ensures the bounds are correctly computed and checked under arbitrary order of instantiations during search. This flexibility allows the use of dynamic variable ordering heuristics.

```
ORDS(Y, C)
begin
    C' ← C ∪ {z_0 = 0}
    for j ← 1 to m do
        C' ← C' ∪ {z_j = y_j + z_{j-1}}
        Y' ← {y_1, ..., y_j}
1       ub ← BnB(Y', C', z_j)
2       C' ← C' ∪ {z_j ≤ ub}
3   return max(dom(z_m))
end

BnB(Y, C, z)
begin
    lb ← −∞
    C' ← C ∪ {z > lb}
4   while DFS(Y, C') returns true do
        lb ← lb + 1
        C' ← C' ∪ {z > lb}
    return lb
end

DFS(Y, C)
begin
    if C is unsatisfiable then return false
    if Y = ∅ then return true
    select a variable y ∈ Y
    foreach d ∈ dom(y) do
        if DFS(Y − {y}, C ∪ {y = d}) returns true then
            return true
    return false
end
```

Figure 2: Pseudo-code of ORDS and BnB.

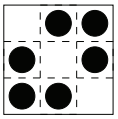

Figure 3: A 3×3 still-life pattern.

A Case Study: Modeling the Still-Life Problem

The Still-Life problem (Bosch & Trick 2004) is a very hard constraint optimization problem because of its gigantic $\mathcal{P}$-space — $O(2^{n^2})$ — where n is the instance size. We will investigate two dual models with much smaller $\mathcal{O}$-space: $O(1.241^{n^2})$ and $O((2n)^{n/3})$.

The game of Life (Gardner 1970) is played on an infinite (checker) board where each square is called a *cell*. Each cell has eight neighbors. A cell is *alive* if there is a checker on it. Otherwise it is *dead*. The state of a board at time t determines its state at time $t + 1$ based on the following three rules:

- If a cell has exactly 2 living neighbors at time t, its state remains unchanged at time $t + 1$. This is the "life" constraint.
- If a cell has exactly 3 living neighbors at time t, then it is alive at time $t + 1$. This is the "birth" constraint.
- If a cell has less than 2 or more than 3 living neighbors at time t, it is dead at time $t + 1$. These are the "death by isolation" and "death by over-crowding" constraints respectively.

It is natural from the definition to think of a cell and its eight neighbors as forming a "unit". (Smith 2002) calls such 3×3 square of cells a *super-cell*.

A *Still-Life* pattern of a board is one that remains unchanged over time. The *density* of a region is the number of living cells within that region. The *Still-Life* problem in an $n \times n$ square of a board (all the rest of the board is dead) is to determine a Still-Life pattern with the maximum density. Figure 3 shows a 3×3 Still-Life pattern. In the remainder of the paper, we can assume without loss of generality that n is a multiple of 3 since it is always possible to enlarge the

[2] The same technique is also applied in (Benoist & Lemaitre 2003) to implement RDS for a special type of weighted CSPs.

board by padding dead cells to size $n' \times n'$ where n' is a multiple of 3.

Still-Life in an $n \times n$ region can be modeled in a straightforward fashion as a CSP. Each cell at the i-th row and the j-th column (i.e. at (i, j)) is associated with a $0/1$ variable $x_{i,j}$ which is 1 if the cell is alive and is 0 otherwise. Let

$$N_{i,j} = \{x_{i\pm d, j\pm e} : d, e \in \{0, 1\} \wedge d + e > 0\}$$

be the neighbors of $x_{i,j}$. The birth and death conditions can be formulated as

$$\begin{aligned} SC_{i,j} \quad \equiv \quad & (x_{i,j} = 1 \rightarrow \textstyle\sum_{u \in N_{i,j}} u \in \{2, 3\}) \wedge \\ & (x_{i,j} = 0 \rightarrow \textstyle\sum_{u \in N_{i,j}} u \neq 3). \end{aligned}$$

This is called a *super-cell constraint*. The arity of a $SC_{i,j}$ constraint is 9. Extra constraints are added on every three adjacent cells along the border to forbid all of them from being alive. Solving the Still-Life problem is to maximize the number of living cells: $z = \sum_{1 \leq i,j \leq n} x_{i,j}$.

While is this simple model is correct, it is inefficient in a sense that very little constraint propagation can be triggered during a branch and bound search. The currently best (pure) CSP models employ dual encoding (Smith 2002) or high arity ad hoc constraints (Cheng & Yap 2006) to boost the efficiency of propagation. We describe these models and show we can obtain better results using `ORDS`.

The super-cell model

This model is basically identical to the one in (Smith 2002). There are nine $0/1$ variables in a super-cell at (i, j) are replaced by a single *super-cell variable* $u_{i,j}$ whose domain ranges from 0 to 511. We define a new constraint $M_{i,j}$[3]

$$\begin{aligned} u_{i,j} = \quad & x_{i-1,j-1} + 2x_{i-1,j} + 4x_{i-1,j+1} + 8x_{i,j-1} + \\ & 16x_{i,j} + 32x_{i,j+1} + 64x_{i+1,j-1} + 128x_{i+1,j} + \\ & 256x_{i+1,j+1} \end{aligned}$$

that maps between the variables using binary encoding. We call the $\frac{n^2}{9}$ disjoint super-cells that partition the board *partitioning super-cells*.

To get rid of the $0/1$ variables, we associate one $\mathcal{O}$-var $y_{i,j}$ for every partitioning super-cell so that the density of the board can be calculated. We get the constraint

$$D_{i,j} \equiv (y_{i,j} = \sum_{-1 \leq d,e \leq 1} x_{i+d,j+e}).$$

We can directly connect the variables $u_{i,j}$ and $y_{i,j}$ using

$$OBJ_{i,j} \equiv \exists_{x \in S}(D_{i,j} \wedge M_{ij})$$

where $S = vars(SC_{i,j})$, the cells in the super-cell at (i, j).

We need finally two more binary constraints to link up the adjacent super-cells. For the horizontally adjacent super-cells at (i, j) and $(i, j + 1)$, we have

$$H_{i,j} \equiv \exists_{x \in S}(M_{i,j} \wedge SC_{i,j} \wedge SC_{i,j+1} \wedge M_{i,j+1})$$

where $S = vars(SC_{i,j}) \cup vars(SC_{i,j+1})$. Similarly,

$$V_{i,j} \equiv \exists_{x \in S}(M_{i,j} \wedge SC_{i,j} \wedge SC_{i+1,j} \wedge M_{i+1,j})$$

is defined for the vertically adjacent super-cells at (i, j) and $(i + 1, j)$, where $S = vars(SC_{i,j}) \cup vars(SC_{i+1,j})$.

To summarize, we have in this model n^2 super-cell variables. Any two adjacent super-cells are linked with either $H_{i,j}$ or $V_{i,j}$. The border constraints are integrated into the relevant super-cells by deleting the infeasible values from their domain. The density z of the board is connected to the objective variables via an equality constraint

$$z = \sum_{1 \leq d,e \leq n/3} y_{3d-1,3e-1}.$$

We note that the constraints $D_{i,j}$ and $M_{i,j}$, as well as the Boolean variables $x_{i,j}$, are used for construction only. They are never in the final model.

The Still-Life problem has several natural symmetries, say, the board can be flipped horizontally, vertically or diagonally (Petrie, Smith, & Yorke-Smith 2004). Although breaking these symmetries are crucial to reduce the search space, adding symmetries breaking constraints are more problematic when all the original $0/1$ variables are removed; all such constraints have to be posted on the $\mathcal{O}$-vars. This may possibly change the "shape" of the symmetry breaking constraints. For example, the rotational symmetry, previously defined on the patterns of the whole board, becomes the permutations of the local densities among super-cells. Although some (weak) symmetries such as "the upper half the board has less or equal number of living cells as the lower half" can be easily handled, some stronger symmetries cannot be expressed this way, because of the padded dead cells at the boundaries of the board. For instance, when $n = 8$, one row (column) of dead cells are padded on the top (right) of the board. Obviously, the vertical symmetry cannot be broken by adding a constraint on the $\mathcal{O}$-vars. In our implementation, only one more symmetry is removed — the diagonal one — using the global `lex_chain` constraint.

Before we move on to the next model, we describe an efficiency issue specific to the super-cell variables: the naive domain $\{0, \ldots, 511\}$ is correct and yet very inefficient. This is because many values are indeed infeasible because of the (adjacent) super-cells and are removed immediately when the constraints are posted. In fact, only 259 patterns are valid for an inner super-cell (Smith 2002). Therefore many "holes" exist between 0 and 511. For a constraint solver, e.g. SICStus Prolog, which represents a variable domain as a list (or array) of ranges, a lot of holes mean a time-consuming traversal of the domain during propagation. We tackle this problem by a bootstrapping approach: all valid patterns of a super-cell are generated in the presence of its neighbors; these patterns are then mapped to an integer between 1 and 259; all constraints (e.g. $H_{i,j}$) are finally constructed using this mapping. This method significantly reduces the amount of holes and reduces the search time approximately by half.

Finally, we report the sizes of the $\mathcal{P}$-space and $\mathcal{O}$-space[4] of this model. As every super-cell ($\mathcal{P}$-var) contains at most

[3] We include subscripts for all constraints so that we can refer to them without their scopes. In the actual implementation, we just need to define the constraints once and instantiate them with different variables.

[4] To distinguish various $\mathcal{P}$-spaces and $\mathcal{O}$-spaces among models, we may use the model name as a prefix, e.g. *super-cell $\mathcal{P}$-space*.

259 patterns and there are $n^2/9$ partitioning super-cells, the size of the $\mathcal{P}$-space is $O(259^{n^2/9}) = O(1.854^{n^2})$. The size of the $\mathcal{O}$-space is $O(7^{n^2/9}) = O(1.241^{n^2})$ as the density of a super-cell ranges from 0 to 6.

The super-row model

The super-cell $\mathcal{O}$-space is much smaller than the super-cell $\mathcal{P}$-space; however, their size are still of the same order of magnitude. Trading space for time, we consider another model which brings the exponent down from n^2 to n.

Our model is very similar to the one given in (Cheng & Yap 2006), except we focus on the $\mathcal{O}$-space. The building block of this model is a row of super-cells called a *super-row*. The *super-row constraint* is defined as

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

Since it contains $3 \times n$ variables, a dual model using binary encoding is impractical — it means constructing variables with domains from 0 to $8^n - 1$. Instead, we view SR_i as a set variable[5] whose domain comprises all feasible patterns of a super-row (w.r.t. the border constraints). In other words, every $\theta \in SR_i$ is a set of living cells, and $x_{i,j} \in \theta$ iff the cell at (i, j) is alive. In this way, we can represent it compactly using binary decision diagram (BDD) (Bryant 1986; Cheng & Yap 2006) and implement set domain propagation efficiently by means of BDD manipulations (Lagoon & Stuckey 2004).

We divide the board into $\frac{n}{3}$ disjoint *partitioning super-rows*, and create an $\mathcal{O}$-var y_i for each of them. The variables are connected with the constraint $CARD_i(y_i, SR_i)$:

$$k \in dom(y_i) \iff \{\theta \in SR_i : |\theta| = k\} \neq \emptyset$$

where the operator $|\cdot|$ returns the cardinality of a set. As a result, the density z of the board is given by the constraint $z = y_2 + y_5 + y_8 + \cdots + y_{n-1}$.

If an $\mathcal{O}$-var y_i is assigned, the domain of the $\mathcal{O}$-var y_{i+3} (of the next partitioning super-row) is re-computed using

$$\exists_{x \in vars(SR_i)}(CARD_i \wedge SR_{i+1} \wedge SR_{i+2} \wedge CARD_{i+3}).$$

The size of the super-row $\mathcal{O}$-space is $O((2n)^{n/3})$ due to the fact that every super-cell has at most 6 living cells, each super-row has $n/3$ partitioning super-cells and there are $n/3$ partitioning super-rows.

We did not implement symmetry breaking for this model as we are not aware of any algorithm that works on BDDs.

Empirical Results

In this section, we present experimental results which demonstrate the effectiveness of `BnB` in the $\mathcal{O}$-space versus $\mathcal{P}$-space. We also compare the addition of `ORDS` versus `BnB`. We implemented the super-cell model in SICStus Prolog 3.12.7 (Carlsson & others 2006).[6] The super-row model was implemented in C++;[7] the BDD package BuDDy 2.4[8] was used to manipulate BDDs. We give the total number of backtracks (bt) and the overall search time ($time$) for each Still-Life instance n. The column den shows the maximum densities. When the super-row model is tested, we list also the time for creating the BDDs ($init$). Time is measured in seconds; the entry *timeout* means the execution exceeded one hour.

`BnB` uses a static variable ordering in which the board is instantiated top-down, left-to-right or right-to-left (on alternating super-rows). `ORDS` iterates over the sub-problems in reversed order, namely right-to-left or left-to-right, bottom-up. Although `ORDS` can use dynamic ordering, we applied static ordering because it gives better results on the Still-Life problem (Cheng & Yap 2006). The values in a domain is enumerated in descending order.

Table 1 gives the results on the super-cell model when the search is done on the partitioning super-cell variables (i.e. the super-cell $\mathcal{P}$-space). `ORDS` can still be used because of the functional nature of $OBJ_{i,j}$ — once the pattern is fixed, so is the density. Hence, the only change to the source code is to make the super-cell variables the search variables.

`ORDS` significantly outperforms `BnB`. When $n = 10$, `ORDS` has 4.4 times less backtracks and is 6.3 times faster. However, neither `ORDS` nor `BnB` can solve the instance $n = 11$ within 60 minutes.

The propagation speed is slow. When $n = 10$, on average `ORDS` and `BnB` respectively backtrack just 151 and 112 times/second. We conjecture the runtime performance will improve if specialized algorithm for dual encoded constraints (e.g. (Samaras & Stergiou 2005)) is used. Currently, we had to implement the binary constraints $H_{i,j}$ and $V_{i,j}$ using the global `case` constraint, which was designed for general n-ary constraints, and also the `element` constraint for the functional constraint $OBJ_{i,j}$.

		ORDS		BnB	
n	den	$time$	bt	$time$	bt
6	18	0.11	16	0.09	48
7	28	0.72	57	0.66	74
8	36	2.73	370	3.94	402
9	43	11.22	1950	59.89	9141
10	54	465.92	70530	3418.38	381543

Table 1: Results on super-cell model: $\mathcal{P}$-vars.

Using the same super-cell model, Table 2 shows that `BnB` in the $\mathcal{O}$-space (rather than the $\mathcal{P}$-space) cuts 93% backtracks and 86% runtime for the instance $n = 10$; `ORDS` can now solve instances up to $n = 12$ in 8 minutes. In (Cheng & Yap 2006), the same instance was solved in almost 8 hours (in the super-cell $\mathcal{P}$-space) with 33 millions backtracks. Since this super-cell model is the same as the one in (Smith 2002), the speedup is due to the drastic size difference between the $\mathcal{O}$-space and the $\mathcal{P}$-space.

To guarantee correct results, `ORDS` and `BnB` assign also the super-cell variables after the density variables. These additional instantiations cause no more backtracks except for $n = 12$, `ORDS` has 1 extra backtrack. In other words, en-

[5] We will freely switch between the set and the logical representations, whichever provides a more simple/natural explanation.

[6] On a PC with a 2.6GHz P4 and 1 GB RAM.

[7] On a MacBook with a 2 GHz Intel Core 2 Duo and 1 GB RAM.

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

		ORDS		BnB	
n	den	$time$	bt	$time$	bt
6	18	0.11	12	0.03	6
7	28	0.62	14	0.52	39
8	36	1.02	32	1.67	95
9	43	2.51	117	4.69	268
10	54	43.28	2410	472.30	27554
11	64	302.52	12181	timeout	
12	76	471.17	21971		

Table 2: Results on super-cell model: $\mathcal{O}$-vars.

forcing GAC on the dual model of the Still-Life problem is strong enough to avoid much search in the $\mathcal{P}$-space.

Another observation is that, when a super-cell (with at most 6 living cells) is added between two consecutive iterations t and $t+1$ of ORDS, very often the difference of the optimal densities of the corresponding sub-problems is 5, i.e. $ub_{t+1} = ub_t + 5$. This indicates even slightly improved bounds on the sub-problems may induce significant pruning in subsequent searches.

Finally, we present the results on the super-row model in Table 3. Due to the small super-row $\mathcal{O}$-space, even BnB alone can solve the instance $n = 13$ in around 5 minutes. ORDS finds the optimal value for $n = 16$ in 245 minutes with merely 11002 backtracks. To the best our knowledge, this is the largest instance solved with complete branch and bound search on a pure CSP model. Indeed, our answer is the only witness to the optimal value reported by (Larrosa, Morancho, & Niso 2005).

			ORDS		BnB	
n	den	$init$	$time$	bt	$time$	bt
6	18	0.01	0.00	4	0.00	4
7	28	0.04	0.00	7	0.01	10
8	36	0.08	0.01	29	0.00	30
9	43	0.12	0.03	65	0.04	76
10	54	0.16	0.08	156	0.08	188
11	64	0.22	0.56	385	0.95	442
12	76	0.28	2.33	336	15.03	564
13	90	0.36	18.06	1708	257.41	2990
14	104	0.44	177.24	2789	*timeout*	
15	119	0.53	1635.87	3030		
16	136	0.78	14708.90	11002		

Table 3: Results on super-row model: $\mathcal{O}$-vars.

Conclusion

Constraint optimization problems are commonly solved by a (simple) branch and bound search on the problem variables. The auxiliary/objective variables are present to connect the problem variables to the global objective. We showed this formulation could lead to repeated or fruitless search efforts. We explained that dual encoding allows us to maximize the objective by directly traversing the objective space, potentially without any search on the problem variables. We illustrated how this can reduce the search space significantly. Then, we described a specialized RDS algorithm, ORDS, which further reduces the size of the objective space. We gave a more simple and flexible (dynamic variable ordering is allowed in ORDS, but not in the original RDS) implementation based on constraints. We demonstrated the power of our new approach with the Still-Life problem. We are the first to solve the instance $n = 16$ using complete branch and bound search on a pure CSP model. We plan to test our approach on more benchmarks and to develop a (dynamic) variable ordering heuristic that explores the $\mathcal{O}$-space and the $\mathcal{P}$-space more effectively. We expect our ideas will suit well with the hybrid algorithm that combines search and variable elimination.

References

Benoist, T., and Lemaitre, M. 2003. An elegant and efficient way of implementing Russian Doll Search for variable weighted CSP. In *Intl. Workshop on Soft Constraints*.

Bessière, C., and Règin, J.-C. 1997. Arc consistency for general constraint networks: Preliminary results. In *IJCAI*, 398–404.

Bosch, R., and Trick, M. 2004. Constraint programming and hybrid formulations for three Life designs. *Annals of Operations Research* 130:41–56.

Bryant, R. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comp.* 35(8):667–691.

Carlsson, M., et al. 2006. *SICStus Prolog User's Manual*.

Cheng, K., and Yap, R. 2006. Applying ad-hoc global constraints with the case constraint to Still-life. *Constraints* 11(2–3):91–114.

Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 38:353–366.

Freuder, E. 1982. A sufficient condition for backtrack-free search. *JACM* 29(1):24–32.

Gardner, M. 1970. The fantastic combinations of John Conway's new solitaire game. *Sci. Amer.* 223:120–123.

Lagoon, V., and Stuckey, P. 2004. Set domain propagation using ROBDDs. In *CP*, 347–361.

Larrosa, J.; Morancho, E.; and Niso, D. 2005. On the practical applicability of bucket elimination: Still-life as a case study. *JAIR* 23:21–440.

Petrie, K. E.; Smith, B. M.; and Yorke-Smith, N. 2004. Dynamic symmetry breaking in constraint programming and linear programming hybrids. In *European Starting AI Researcher Symp.*

Samaras, N., and Stergiou, K. 2005. Binary encodings of non-binary constraint satisfaction problems: Algorithms and experimental results. *JAIR* 24:641–684.

Smith, B. 2002. A dual graph translation of a problem in 'Life'. In *CP*, 402–414.

Stergiou, K., and Walsh, T. 1999. Encodings of non-binary constraint satisfaction problems. In *AAAI*, 163–168.

Verfaillie, G.; Lemaître, M.; and Schiex, T. 1996. Russian Doll Search. In *AAAI*, 181–187.