Optimal Rectangle Packing: A Meta-CSP Approach

Michael D. Moffitt and Martha E. Pollack
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109, USA
{mmoffitt, pollackm}@eecs.umich.edu

Abstract

We present a new approach to optimal rectangle packing, an NP-complete problem that can be used to model many simple scheduling tasks. Recent attempts at incorporating artificial intelligence search techniques to the problem of rectangle packing have focused on a CSP formulation, in which partial assignments are defined to be the fixed placement of a subset of rectangles. Our technique takes a significant departure from this search space, as we instead view partial assignments as subsets of relative pairwise relationships between rectangles. This approach recalls the *meta-CSP* commonly constructed in constraint-based temporal reasoning, and is thus a candidate for several pruning techniques that have been developed in that field. We apply these to the domain of rectangle packing, and develop a suite of new techniques that exploit both the symmetry and geometry present in this particular domain. We then provide experimental results demonstrating that our approach performs competitively compared to the previous state-of-the-art on a series of benchmarks, matching or surpassing it in speed on nearly all instances. Finally, we conjecture that our technique is particularly appropriate for problems containing large rectangles, which are difficult for the fixed-placement formulation to handle efficiently.

Introduction

The problem of rectangle packing is one that has drawn attention from several diverse fields of computer science. For instance, in the context of scheduling, it can be used to represent scenarios where jobs require a fixed amount of time and resources, which compose the two dimensions of a single rectangle. The task of packing many such rectangles into an enclosing space, so as to minimize the width, height, or area of this space, allows for the minimization of makespan, resources needed, or total wasted resource. In VLSI design, rectangles represent actual physical modules that need to be placed in a spatial arrangement such that no two modules overlap.

One of the more recent approaches (Korf 2003) has cast optimal rectangle packing as a *constraint satisfaction problem* (CSP). In this formulation, a variable is created for each rectangle, whose legal values are the positions that rectangle could occupy without exceeding the boundaries of the enclosing space. In addition, there is a binary constraint between each pair of rectangles, requiring that they do not

overlap. To solve this CSP, Korf developed a backtracking algorithm, where each partial assignment is defined to be the fixed placement of a subset of rectangles. By obtaining lower bounds on the amount of wasted space at each node in the search, an efficient algorithm is constructed, which was later refined (Korf 2004) into the fastest known algorithm for optimal rectangle packing.

In this paper, we present a new approach to rectangle packing. Specifically, we cast the problem of optimally packing a set of rectangles with fixed orientations as a *meta-CSP*. In this formulation, we create a meta-variable for each pair of rectangles, whose values are the four pairwise relationships (i.e., *above*, *below*, *left of*, *right of*) that prevent that pair from overlapping. As such, commitment to the exact placement of any rectangle is not established until a consistent solution has been generated. In this regard, our approach resembles graph-based methods for optimal rectangle packing (Onodera, Taniguchi, & Tamaru 1991), which have previously been unable to cope with more than a small handful of rectangles. The key difference in our meta-CSP formulation is the introduction of modern artificial-intelligence search techniques that greatly increase the efficiency of exploring this alternative space. Some of these are drawn from literature on constraint-based temporal reasoning (Dechter, Meiri, & Pearl 1991; Stergiou & Koubarakis 1998; Tsamardinos & Pollack 2003), while others are entirely new techniques that exploit the symmetry and geometry present in this particular domain. We combine these methods to create the BLUEBLOCKER solver, and provide experimental results demonstrating that our approach performs competitively compared to the previous state-of-the-art on a series of benchmarks. Finally, we conjecture that our technique is particularly appropriate for problems containing large rectangles, which are difficult for the fixed-placement formulation to handle efficiently.

Background

Rectangle Packing

Consider the following problem: we are given a set of N jobs, where each job j requires a specific number of workers (or machines) m_j and a specific amount of uninterrupted processing time p_j. Assuming that all workers are paid the same salary and work the same hours, one may wish to find a schedule that minimizes the total cost of labor (which is proportional to the product of number of workers and the total hours worked).

This problem can be approximated as a rectangle packing problem.[1] Specifically, each job can be represented by a rectangle, with width w_j being equal to the time required p_j, and height h_j being equal to the required number of workers m_j. Minimizing the total labor cost is then equivalent to finding an enclosing rectangle of dimensions $W \times H$ whose total area is minimal. We refer to this as the *Minimal Bounding Box Problem*. The decision variant of this, where the dimensions of the enclosing space are specified, we refer to as the *Containment Problem*.

Previous work on rectangle packing has primarily focused on generating approximate or suboptimal solutions, using techniques such as genetic algorithms and simulated annealing. To facilitate these approaches, a number of data structures have been developed to represent a layout of rectangles including *sequence pairs* (Murata *et al.* 1995), *BSG structures* (Nakatake *et al.* 1996), and *O-Trees* (Guo, Cheng, & Yoshimura 1999). We omit the details of such structures, as they are largely unrelated to our formulation.

The first line of research to consider *optimal* two-dimensional compactions of rectangles (Onodera, Taniguchi, & Tamaru 1991) applied a branch-and-bound approach, using a graph algorithm to maintain consistency at each step of the backtracking search. Extensions of this method have been largely avoided, as it was shown to handle no more than six rectangles tractably. More recently, an algorithm using the O-Tree representation was able to surpass this record by efficiently handling up to nine rectangles (Chan & Markov 2004).

Rectangle Packing as a CSP

One of the more distinctive approaches to rectangle packing (Korf 2003) applied artificial intelligence search techniques with great success. In this work, rectangle packing is modeled as a binary constraint satisfaction problem. There is a variable for each rectangle, whose legal values are the positions it could occupy without exceeding the boundaries of the enclosing space. In addition, there is a binary constraint between each pair of rectangles that they cannot overlap.

To solve this CSP, a backtracking algorithm is used. At each node in the search, a rectangle is given a fixed position within the enclosing space. This is accomplished by "drawing" the rectangle (or an outline of the rectangle) onto a bitmap representing the entire area. For example, one such partial assignment is depicted in Figure 1. Here, two rectangles (in fact, squares) have been given assignments. The upper-left corner of the 6×6 square has been placed at $(0, 0)$, and the upper-left corner of the 5×5 square has been placed at $(6, 0)$. Suppose we have four more squares remaining, with dimensions 4×4, 3×3, 2×2, and 1×1; by visual inspection, it is clear that this particular partial assignment is a dead-end, as there is no place for the 4×4 square to go.

To make this search efficient, a number of powerful techniques were developed to prune large portions of the search space. These techniques rely almost entirely on the ability to compute the number of remaining empty cells in the enclosing space that will necessarily be wasted for a given
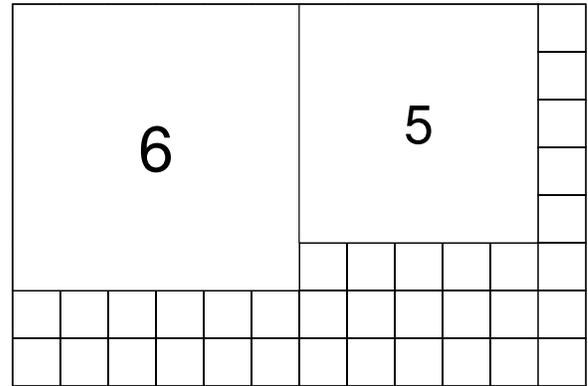

Figure 1: The original CSP search space

partial assignment. As an example, suppose the 5×5 square had not yet been placed in the above figure. Clearly, the only squares that can fit beneath the 6×6 block are those with dimensions 1×1 and 2×2. Thus, of those twelve unit cells, at least $12 - 1^2 - 2^2 = 7$ must remain wasted in any extension of this partial assignment. Since the sum of the areas of all the rectangles is 91, and the enclosing rectangle is $8 \times 12 = 96$, and $91 + 7 > 96$, this partial assignment cannot lead to a consistent solution. In Korf's system, this procedure is performed by slicing the empty space into one-dimensional strips and solving a relaxation of a related bin packing problem to determine a lower bound on the wasted space in polynomial time.

The above approach solves the Containment Problem; to address the Minimal Bounding Box Problem, the algorithm is used as a black box within an outer loop that iterates on possible dimensions of the enclosing space. This general approach was later refined (Korf 2004) to produce a solver faster than any other known engine (although the current implementation is capable of solving only square packing instances). Even the original results (Korf 2003), which are roughly a magnitude slower, are staggering compared to all previous attempts at optimal two-dimensional compaction. To date, no other solver has been able to match or even come close to the performance exhibited by Korf's methods.

Rectangle Packing as a Meta-CSP

In this section, we construct an alternative formulation for determining whether or not a set of rectangles with fixed orientations can be packed into a given enclosing space. Let parameters (w_i, h_i) denote the dimensions (width and height) of rectangle i, and let the variables (x_i, y_i) denote the position of the upper left corner of rectangle i with respect to the upper left corner of the enclosing rectangle.[2] Our goal is to find an assignment to all coordinates (x_i, y_i) such that (1) each rectangle is entirely contained within the enclosing rectangle of dimensions $W \times H$, and (2) no two rectangles overlap. The first set of constraints is achieved by:

---

[1]Richard Korf brought to our attention that the rectangle packing formulation turns out to be a bit more constraining, as it requires the group of workers assigned to each job to be "contiguous" along an artificial ordering.

[2]Extensions to 3 or more dimensions are easily accomplished by introducing additional rectangle parameters (e.g., a depth d_j) and additional coordinate variables (e.g., a z_j coordinate).

```
Solve-Meta-CSP(A, U)
If (U = ∅)
      return success
C_{i,j} ← select-variable(U), U' ← U − {C_{i,j}}
For each disjunct d_{ikj} of D(C_{i,j})
      A' ← A ∪ {C_{i,j} ← d_{ikj}}
      If (consistent(A'))
            If (Solve-Meta-CSP(A', U') = success)
                  return success
return failure
```

Figure 2: An algorithm for solving the meta-CSP

$$
\begin{array}{ll}
0 \le x_i, 0 \le y_i & \text{for } 1 \le i \le N \\
x_i + w_i \le W & \text{for } 1 \le i \le N \\
y_i + h_i \le H & \text{for } 1 \le i \le N
\end{array}
$$

We will refer to these as the *containment constraints*. The second requirement, precluding overlap between a pair of rectangles i and j, is achieved by a set of disjunctive constraints of the following form:

$$
\begin{array}{ll}
\{d_{iLj} : x_i + w_i \le x_j\} \vee & (i \text{ is to the left of } j) \\
\{d_{iRj} : x_j + w_j \le x_i\} \vee & (i \text{ is to the right of } j) \\
\{d_{iAj} : y_i + h_i \le y_j\} \vee & (i \text{ is above } j) \\
\{d_{iBj} : y_j + h_j \le y_i\} & (i \text{ is below } j) \\
\multicolumn{2}{c}{1 \le i < j \le N}
\end{array}
$$

We refer the set of such constraints as the *non-overlap constraints*. Each inequality (or *disjunct*) has been given a label, such as d_{iLj}, for reference. We will generally refer to the first pair of inequalities as the *horizontal disjuncts*, and the second pair as the *vertical disjuncts*.

This encoding lends itself to a *meta-CSP* formulation. Here, instead of directly considering assignments to the variables x_i, x_j, y_i, and y_j, we create a meta-variable $C_{i,j}$ for each non-overlap constraint between any pair of rectangles i and j. The domain $D(C_{i,j})$ is simply the set $\{d_{iLj}, d_{iRj}, d_{iAj}, d_{iBj}\}$, representing the various alternatives (corresponding to *left*, *right*, *above*, *below*) one has for satisfying that non-overlap constraint. A complete assignment in the meta-CSP thus involves a selection of a single disjunct for each non-overlap constraint.[3] Figure 2 provides pseudocode for a simple backtracking algorithm that will solve this meta-CSP. The input variable A is the set of assignments to meta-variables, and initially contains all containment constraints with their singleton assignments; input variable U is the set of unassigned meta-variables, and initially contains all non-overlap constraints.

The idea of constructing a meta-CSP was first proposed to solve the Binary Temporal Constraint Satisfaction Problem (or Binary TCSP) (Dechter, Meiri, & Pearl 1991), and has since also been applied to the Disjunctive Temporal Problem (DTP) (Stergiou & Koubarakis 1998). DTPs permit arbitrary disjunctions of temporal constraints, i.e., linear inequalities of the form $x - y \le b$, where x and y are real- or integer-valued variables, and b is a constant. Since each of the constraints in our meta-CSP formulation can be rearranged to fit this form, our construction is actually a special

---

[3] The containment constraints can be regarded as meta-variables as well, though since they can take only a single value, we omit them in our discussion.
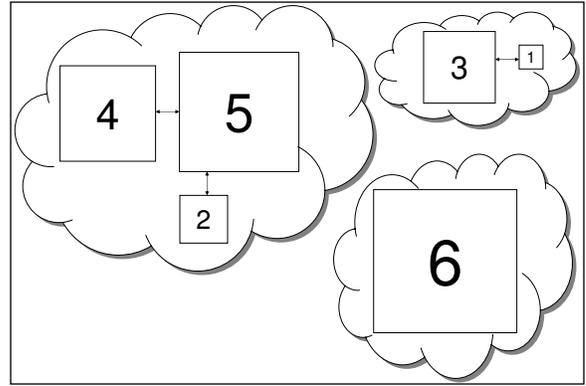

Figure 3: The meta-CSP search space

case of a DTP.[4]

Within the meta-CSP formulation, the constraints are implicitly defined by the underlying semantics of the disjuncts: in particular, the values (disjuncts) assigned to each meta-variable must be mutually consistent. Recall that each value is a linear inequality (e.g., $x_i + w_i \le x_j$). The consistency of a set $\mathcal{S}$ of such inequalities can be determined by first constructing its *distance graph*, a graph that includes a node for each (object-level) variable (e.g., x_i, x_j, etc.) and an arc with weight $-b$ from x_j to x_i whenever $x_i + b \le x_j$ is in $\mathcal{S}$. Then $\mathcal{S}$ is consistent if and only if its distance graph has no negative cycles, which can be determined in polynomial time by computing its all-pairs shortest path (APSP) matrix and checking that there are no negative values along the main diagonal. A fixed-placement solution can be extracted directly from the rows of this matrix (Dechter, Meiri, & Pearl 1991). This approach bears considerable resemblance to the single-source shortest path techniques used in some previous approaches to rectangle packing (Liao & Wong 1983).

The meta-CSP search space has several advantages over the original CSP formulation – for instance, the size of neither the rectangles nor the enclosing space has an effect on the runtime of our algorithm. In contrast, these are likely to greatly impact the performance of the approach in (Korf 2003; 2004), not only because they determine the number of possible locations for each rectangle, but also because the original CSP requires maintenance of a bitmap representation of the enclosing space when placing rectangles and performing wasted space calculations. Secondly, several types of additional shapes and constraints beyond those of the original formulation (Young, Ho, & Chu 2002) can be encoded and imposed effortlessly in our meta-CSP as edges in the distance-graph, whereas additional consistency checks would be required to handle such constraints in the original CSP.

One disadvantage to this alternative search space is that it is significantly harder to visualize than Korf's. We have made an attempt to depict it in Figure 3, which again as-

---

[4] Simple algebra suffices to convert the constraints into the appropriate form: for example, $x_i + w_i \le x_j$ is equivalent to $x_i - x_j \le -w_i$.

95

sumes square blocks with side lengths of 1, 2, ..., 6. None of the rectangles shown has been given a fixed location – in effect, one can imagine that they are floating freely in the enclosing space. However, in this particular assignment, there are three pairs of rectangles whose relative spatial relationships have been chosen; namely, $\{(4,5),(5,2),(3,1)\}$. This corresponds to the following partial assignment:

$$C_{4,5} \leftarrow \{d_{4L5} : x_4 + 4 \le x_5\}$$
$$C_{2,5} \leftarrow \{d_{2B5} : y_5 + 5 \le y_2\}$$
$$C_{1,3} \leftarrow \{d_{1R3} : x_3 + 3 \le x_1\}$$

As our objective is to determine whether the set of rectangles will fit into the enclosing space, this partial assignment will lead to one of two things: either (1) a consistent solution, with all $N(N-1)/2$ non-overlap constraints receiving an assignment, or (2) a dead-end, where all extensions of this partial assignment induce a negative cycle in underlying distance graph. In the latter case, the backtracking search will then systematically attempt other partial assignments (perhaps with $C_{1,3}$ receiving the disjunct $\{d_{1B3} : y_3 + 3 \le y_1\}$) until consistency is achieved or search is exhausted.

Traditional Meta-CSP Pruning Techniques

As noted earlier, there have been previous attempts at using a graph-based approach to rectangle packing, but they were largely abandoned in the early 1990's because of their inability to scale; in particular, they could not tractably solve problems containing more than six rectangles.[5] However, in the past decade, a great deal of effort has been devoted to the development of efficient meta-CSP pruning techniques for temporal constraint satisfaction, and this work can be applied to the meta-CSP formulation of rectangle packing. Some of these pruning techniques resemble methods originally developed for finite-domain CSPs, while others are unique to the meta-CSP. In this section, we present three powerful techniques that have been recently employed by the DTP solver Epilitis (Tsamardinos & Pollack 2003), and apply them to our meta-CSP formulation of rectangle packing. As in (Korf 2004), we will typically use examples involving squares to illustrate these techniques, although (non-rotatable) rectangles can be handled in an identical fashion.

Forward Checking

Forward checking is one of the simplest and most effective pruning mechanisms for dead-end detection in CSPs, and it can be applied to our meta-CSP as well. It works by examining each as-yet unassigned meta-variable, removing values that are inconsistent with the current partial assignment. Whenever the domain of a variable is reduced to $\oslash$, backtracking may be invoked. Previous graph-based algorithms for rectangle packing did not make use of this technique, although it is today used in virtually every CSP system.

As an example, suppose that we have an enclosing rectangle with a width of 12 and a height of 8, as in Figure 1. We are given the task of packing squares of dimensions 1×1 through 6×6. Note that even before any non-overlap constraints have been given assignments, the vertical disjuncts

belonging to $C_{4,6}$ are in conflict with the containment constraints. This is because there is no way to stack the 4×4 square either above or below the 6×6 square within a space of height 8. Thus, these vertical disjuncts can be removed from the domain of $C_{4,6}$.

Now, suppose we require the 5×5 square to be placed somewhere to the right of both the 4×4 and 6×6 squares. This corresponds to the following partial assignment:

$$C_{4,5} \leftarrow \{d_{5R4} : x_4 + 4 \le x_5\}$$
$$C_{5,6} \leftarrow \{d_{5R6} : x_6 + 6 \le x_5\}$$

This renders both of the horizontal disjuncts for constraint $C_{4,6}$ unavailable, since a width of at least 15 would be needed to displace these three blocks horizontally. Since the domain of $C_{4,6}$ has now been obliterated, this partial assignment can be abandoned entirely.

To check whether the current partial assignment can be extended by a particular disjunct, one could simply apply an all-pairs shortest path algorithm (such as Floyd-Warshall) in $O(|X|^3)$ time, where $|X|$ is the number of variables. Fortunately, the presence of a precomputed APSP matrix allows a disjunct $v_i + b \le v_j$ to be tested for consistency in $O(1)$ time by ensuring that the length of the shortest path from v_i to v_j is no less than b.

Removal of Subsumed Variables

Consider once again a rectangle packing problem that involves the placement of squares having dimensions 1×1 through 6×6. Suppose we require the 5×5 square to be placed below the 4×4 square and above the 6×6 square, which involves the following assignments:

$$C_{4,5} \leftarrow \{d_{4A5} : y_4 + 4 \le y_5\}$$
$$C_{5,6} \leftarrow \{d_{5A6} : y_5 + 5 \le y_6\}$$

If we ignore the dimensions of the enclosing space, there are exactly three disjuncts available for the constraint $C_{4,6}$.[6] For instance, we could place the 4×4 square to the left of the 6×6 square, a decision reflected by the assignment $C_{4,6} \leftarrow \{d_{4L6} : x_4 + 4 \le x_6\}$.

However, it so happens that the current partial assignment already satisfies the non-overlap constraint between the 4×4 and 6×6 squares. Specifically, the expressions $y_4 + 4 \le y_5$ and $y_5 + 5 \le y_6$ can be composed transitively to obtain the expression $y_4 + 9 \le y_6$.[7] Surely if such a condition holds, then so does the (weaker) vertical disjunct $\{d_{4A6} : y_4 + 4 \le y_6\}$. In other words, our current partial assignment has already placed the 4×4 above the 6×6 square. As a result, we can immediately make the assignment $C_{4,6} \leftarrow \{d_{4A6} : y_4 + 4 \le y_6\}$. Furthermore, we can safely prune all other potential disjuncts for constraint $C_{4,6}$ when we return to this decision point, as they only serve to constrain the problem further.

The aforementioned technique is an example of what has been called *removal of subsumed variables* in temporal constraint literature (Oddi & Cesta 2000), although the technique was discovered in the domain of rectangle packing almost a full decade earlier (Onodera, Taniguchi, & Tamaru 1991). As in the case of forward-checking, the presence
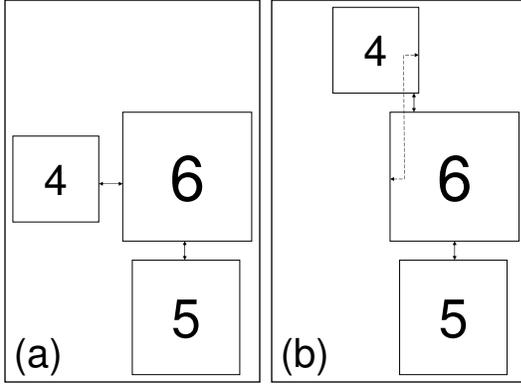

Figure 4: An illustration of semantic branching

of an all-pairs shortest path distance matrix makes it possible to check in $O(1)$ whether a particular meta-variable is subsumed (i.e., already satisfied, as a result of the assignments made so far). Specifically, if there exists a disjunct $v_i + b \leq v_j$ such that the shortest path from v_j to v_i has length less than $-b$, the disjunct can be assigned to its respective constraint immediately.

Semantic Branching

Consider, as before, a rectangle packing problem that involves the placement of squares having dimensions 1×1 through 6×6. Suppose we are faced with the situation shown in Figure 4(a), which maps to the partial assignment below:

$$C_{5,6} \leftarrow \{d_{5B6} : y_6 + 6 \leq y_5\}$$
$$C_{4,6} \leftarrow \{d_{4L6} : x_4 + 4 \leq x_6\}$$

Imagine that we attempt all possible extensions to this partial assignment and are unable to find a solution. At this point, a new disjunct will be attempted to satisfy constraint $C_{4,6}$. For instance, we might require the 4×4 square to be placed above the 6×6 square, corresponding to the assignment $C_{4,6} \leftarrow \{d_{4A6} : y_4 + 4 \leq y_6\}$. This partial assignment may or may not lead to a feasible solution, but one thing is known for certain – if there *is* a consistent extension to this partial assignment, it will not be one where the 4×4 square is both above and entirely to the left of the 6×6 square. How do we know this? Suppose the contrary is true, and that the 4×4 *could* be placed to the left of the 6×6 square. If this were the case, then such a solution would have been found already when the disjunct $\{d_{4L6} : x_4 + 4 \leq x_6\}$ had been attempted.

Since we know that the disjunct $\{d_{4L6} : x_4 + 4 \leq x_6\}$ will never hold in any solution extending our new partial assignment, we can explicitly add its negation (i.e., $x_4 + 4 > x_6$) to the set of constraints.[8] The geometrical interpretation of this additional constraint is shown as a dashed arrow in Figure 4(b); essentially, we are requiring the right-hand side of the 4×4 square to be placed beyond the left-hand side of the

6×6 square. The benefit of adding such a constraint is that it will tighten the path lengths stored in the distance graph and thus aid in pruning dead-ends earlier. In general, if an extension $A \cup \{C \leftarrow d\}$ of a partial assignment A fails, one can enforce the negation of d (i.e., $\neg d$) for any other extension of A, such as $A \cup \{C \leftarrow d'\}$. If this second extension fails, both $\neg d$ and $\neg d'$ can be enforced, and so on. This technique is referred to as *semantic branching* in temporal constraint literature in (Armando, Castellini, & Giunchiglia 1999). To our knowledge, no optimal rectangle packing solver has yet made use of this powerful innovation.

Domain-Specific Techniques

The techniques presented thus far can be applied to a broad range of meta-CSP formulations. However, the geometry and symmetry present in rectangle packing permit the use of additional domain-specific techniques. In this section, we develop two new such techniques, and also improve a previously developed domain-specific ordering heuristic.

Dynamic Symmetry Breaking

The problem of exploiting symmetry in constraint satisfaction problems has gained increased attention over the past few years. Its application to the domain of rectangle packing has been examined only recently (Chan & Markov 2003); however, the majority of techniques developed in that line of research are instance-specific, requiring some rectangles to share a common width or height. The original CSP search space (Korf 2004) is able to perform a very simple operation to exploit symmetry while preserving optimality, by ensuring that the center of the largest rectangle is never placed outside the upper-left quadrant of the enclosing space.

Not surprisingly, issues of symmetry arise in our search space as well. For instance, suppose that our first meta-CSP assignment places the 6×6 square above the 5×5 square, and the subsequent search space is fully explored recursively. Afterward, the algorithm might attempt the opposite relationship, placing the 6×6 square below the 5×5 square. Such an attempt is clearly useless, as every partial assignment in this subproblem can be mapped to a corresponding equivalent assignment in the previous subproblem. The same redundancy would exist if one were to try both horizontal relationships. Note that this symmetry holds even if the blocks in our example were not squares but rather strict rectangles. Thus, one option to combat symmetry is to remove one horizontal disjunct and one vertical disjunct from the domain of a *single* constraint – the first one to be examined – before executing the meta-CSP search.[9]

However, it so happens that the symmetry we have just described is only a special case of a more general phenomenon. Consider the scenario in Figure 5(a), which reflects the partial assignment:

$$C_{5,6} \leftarrow \{d_{5B6} : y_6 + 6 \leq y_5\}$$
$$C_{4,5} \leftarrow \{d_{4B5} : y_5 + 5 \leq y_4\}$$
$$C_{3,6} \leftarrow \{d_{3L6} : x_3 + 3 \leq x_6\}$$

After exploring this subproblem, the constraint $C_{3,6}$ might instead receive the assignment $\{d_{3R6} : x_6 + 6 \leq x_3\}$, as

---

[8]To keep all constraints of the same form, we would actually add the slightly tighter constraint $x_4 + 3 \geq x_6$ (or $x_6 - 3 \leq x_4$) instead. Since the coordinates are all required to take integral values, the soundness and completeness of the procedure are preserved.

[9]Though it is not required that this constraint be the first to receive an assignment, it typically will help in reducing the search space visited.
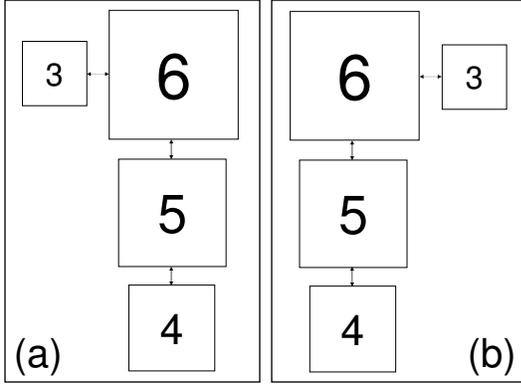
Figure 5: An illustration of symmetric assignments

Figure 6: Dead ends encountered in the meta-CSP search

shown in Figure 5(b). By visual inspection, these two assignments are isomorphic, but note that such symmetry will not be pruned with the simple technique discussed previously. That preprocessing trick only serves to prune alternate assignments for $C_{5,6}$.

As a result, we introduce a new technique that performs a test *during* search to check whether an assignment is symmetric to one previously considered, similar in spirit to the approach proposed in (Gent & Smith 2000) for general CSPs. Specifically, suppose that a given partial assignment A exclusively contains assignments $\{C \leftarrow d\}$ where each disjunct d is a vertical disjunct. If the next extension of this partial assignment $A \cup \{C' \leftarrow d'\}$ includes a horizontal disjunct d', then that disjunct's horizontal "sibling" d'' may be safely pruned once the algorithm has backtracked to this decision point. In other words, the extension $A \cup \{C' \leftarrow d''\}$ will not be explored. This same technique can also be applied when a vertical assignment is attempted after a series of horizontal decisions. In our example, the disjunct $\{d_{3L6} : x_3 + 3 \leq x_6\}$ is the first horizontal disjunct to be chosen, so its partner $\{d_{3R6} : x_6 + 6 \leq x_3\}$ can be pruned. Such a procedure is sound, since all pruned partial assignments are mirrored by isomorphic partial assignments that have been previously considered.

Detecting Cliques of Displacement

One of the disadvantages of the meta-CSP formulation of rectangle packing is that some of the geometric information is lost in the encoding. As an example, consider the problem of packing squares of dimensions 1×1 through 10×10 into an enclosing space of size 10×44. The area of this enclosing space (440 square units) appears to be sufficiently large, as the sum of the areas of the blocks is only 385. The algorithm can make several assignments without reaching a dead-end – for instance, consider Figure 6(a), where squares 6×6 through 10×10 are given pairwise relationships that place them in decreasing order of size. However, there is no way to relate the 5×5 square with the 6×6 square (or any other larger square) that would allow it to fit. As a result, this partial assignment must be abandoned. Backtracking will then continue to find similar partial assignments, such as those depicted in Figure 6(b-c). Even if symmetric assignments are pruned, 60 such permutations of blocks will
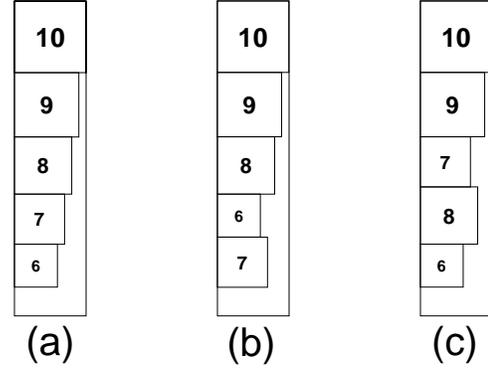
be reached before failure is ultimately discovered. The problem here is that no pair of squares having dimensions 5×5 through 10×10 can be displaced horizontally within a width of 10, and thus they must all be laid out in a sequence, requiring a height of at least 45.

To discover conflicts such as this earlier within the meta-CSP search, we introduce a new technique that allows our solver to exploit the geometry of the rectangle packing domain. This method requires the existence of a structure we refer to as a horizontal (or vertical) displacement graph, which we maintain incrementally at each step in the search.

Definition: A horizontal (or vertical) *displacement graph* G_D^H (or G_D^V) contains N vertices, one for each rectangle. Furthermore, it contains an undirected edge $E_{i,j}$ between any pair of vertices i and j if either of the following conditions holds:

- The meta-variable $C_{i,j}$ has been assigned one of its horizontal (or vertical) disjuncts.

- Forward checking has removed both vertical (or horizontal) disjuncts from the domain of $C_{i,j}$. $\square$

In essence, each edge in the horizontal (or vertical) displacement graph represents a pair of rectangles that will be displaced horizontally (or vertically) in any subsequent solution. In fact, a *clique* in the displacement graph represents an entire set of such rectangles, all of which must be placed in some total ordering along a single dimension. As a result, a lower bound on the height of the enclosing space needed to pack the N rectangles can be obtained by finding the maximum weighted clique C in the vertical displacement graph – that is, a clique that maximizes the expression:

$$\sum_i^{i \in C} h_i$$

where h_i is the height of rectangle i. A lower bound on the width can be calculated in a similar way. In our example of 10 squares, the containment constraints render the horizontal disjuncts for the following meta-variables inapplicable:

$C_{1,10}$	$C_{2,10}$	$C_{3,10}$	$C_{4,10}$	$C_{5,10}$	$C_{6,10}$	$C_{7,10}$
$C_{8,10}$	$C_{9,10}$	$C_{2,9}$	$C_{3,9}$	$C_{4,9}$	$C_{5,9}$	$C_{6,9}$
$C_{7,9}$	$C_{8,9}$	$C_{3,8}$	$C_{4,8}$	$C_{5,8}$	$C_{6,8}$	$C_{7,8}$
$C_{4,7}$	$C_{5,7}$	$C_{6,7}$	$C_{5,6}$			

and thus forward checking removes them before search begins. Once the corresponding edges are inserted into the vertical displacement graph, a clique is found containing the vertices $\{5, 6, 7, 8, 9, 10\}$. The heights of these rectangles sum to 45, and since this is larger than the height of the enclosing space, failure is detected without expanding even a single search node.

Since maximal clique detection is itself an NP-complete problem (Garey & Johnson 1979), we instead implement a greedy procedure that finds (potentially suboptimal) cliques in polynomial time. To find a clique C in a displacement graph (let us assume the horizontal displacement graph, G_D^H), the procedure makes two passes; the first pass begins by adding the rectangle with the largest width to C. It then examines the remaining rectangles in decreasing order of width, adding rectangle i to C provided that there exists an edge $E_{i,j}$ between i and every rectangle j in C. The second pass does the same, except that it begins with the rectangle with the smallest width that is connected to an edge in G_D^H, and then works upwards. Of the two cliques that are generated, the one with the higher total weight is used to estimate the lower bound. This procedure takes advantage of the fact that the optimal clique often contains rectangles with neighboring widths.

Our clique detection mechanism subsumes a related technique used in the fixed placement formulation (Korf 2004). In that work, the rectangles are sorted in decreasing order by height. The list is then scanned in order, and the widths of the rectangles are summed until a rectangle is reached that can be placed above the previous rectangle. A lower bound on the width required is the smaller of this sum and the maximum width of any rectangle. This procedure is performed once for a particular enclosing space, and is not repeated after search begins. The advantage to our approach is that it can compute more accurate lower bounds that arise from spatial arrangements induced during search, and is therefore not limited to estimations based solely on the dimensions of the enclosing space.

Variable and Value Ordering Heuristics

It is well known that a constraint satisfaction engine can perform quite poorly in the absence of good heuristics. There are generally two heuristics that one is concerned with; namely, the *variable ordering* heuristic (sometimes referred to as the *branching schedule*) and the *value ordering* heuristic. For the case of rectangle packing, (Onodera, Taniguchi, & Tamaru 1991) hints at a simple, static variable ordering heuristic that imposes pairwise relationships between large blocks early on. Since the manner in which these constraints are satisfied is likely to have a larger impact on the resulting placement than constraints involving pairs of smaller rectangles, the heuristic is essentially a variation on the traditional *most constrained variable first* heuristic. We formalize their heuristic as follows:

- Select the meta-variables that maximize $min(w_i \times h_i, w_j \times h_j)$, where i and j are the rectangles in the variable's scope.

- Of these, select randomly from the set of meta-variables that maximize $max(w_i \times h_i, w_j \times h_j)$.

An example of this heuristic is shown in Figure 7, which illustrates the order in which meta-variables would be se-

	5×5	4×4	3×3	2×2	1×1
6×6	1	2	4	7	11
5×5		3	5	8	12
4×4			6	9	13
3×3				10	14
2×2					15

Figure 7: A simple, static meta-variable ordering heuristic for a 6 instance square packing problem

lected for a problem involving squares of dimensions 1×1, ..., 6×6. Here, the first meta-variable that would receive an assignment is $C_{5,6}$, followed by $C_{4,6}$ and $C_{4,5}$, and so on.

However, suppose that the domain of some uninstantiated constraint $C_{i,j}$ has been reduced to a single disjunct d as a result of forward checking. In such a case, there is no benefit in considering other constraints earlier than $C_{i,j}$, since any consistent solution extending this partial assignment (if one exists) must necessarily include the assignment $\{C_{i,j} \leftarrow d\}$. Consequently, we propose a small but important modification to this heuristic; specifically, we take the approach of immediately making an assignment to any meta-variable whose domain has been reduced to a singleton, making our heuristic a *dynamic* one. This resembles the *unit clause propagation* technique commonly used in modern SAT solvers (Moskewicz *et al.* 2001).

As for the value ordering heuristic, we randomly choose among those disjuncts that require the minimal increase in area. In the event of a tie, we choose the disjunct with the least amount of *slack*[10] as to produce tighter packings.

Minimizing Area

At this point, we have given a formulation of the Containment Problem, and have developed a variety of techniques to make search within this space efficient. As with Korf's approach, one of the options we have for extending our algorithm to solve the Minimal Bounding Box Problem is to iteratively attempt enclosing spaces of smaller and smaller area, using this constraint satisfaction engine as a subroutine. However, the search trees for the resulting sequence of meta-CSPs will share a significant amount of structure, which we would like to exploit to improve efficiency.

Thus, we will instead perform a single branch-and-bound backtracking search through the space of partial assignments, ensuring that the area of the enclosing space not exceed that of the minimum area found thus far. This is the approach taken in previous graph-based methods (Onodera, Taniguchi, & Tamaru 1991). Specifically, suppose that the current lower bounds on the width and height of the enclosing space are w_l and h_l. If the area of the current best solution is A, we can enforce upper bounds on the width and height by introducing these constraints:

$$W \leq \lfloor A/h_l \rfloor$$
$$H \leq \lfloor A/w_l \rfloor$$

If either of these two constraints cannot be imposed, the current partial assignment may be abandoned, as it cannot lead

---

[10]The slack of a disjunct $v_i + b \leq v_j$ is calculated by subtracting b from the length of the shortest path from v_i to v_j in the distance graph.

# Rect. (N)	Optimal Dimen.	BloBB '04 Implementation	Clautiaux '04 Implementation	Korf '04 Implementation	BlueBlocker '06 Implementation
6	9×11	0	0	0	0
7	7×22	0	0	0	0
8	14×15	0	0	0	0
9	15×20	23	1	0	0
10	15×27	5:12	3	0	0
11	19×27	25:16	4	0	0
12	23×29	7:45:49	13	0	0
13	22×38	79:21:46	23	0	0
14	23×45		1:03	0	0
15	23×55		1:57	1	1
16	27×56		10:05	2	3
17	39×46		14:49	10	10
18	31×69		31:33	1:08	1:29
19	47×53		72:53:18	8:15	4:11
20	34×85			13:32	15:03
21	38×88			1:35:08	1:32:01
22	39×98			6:46:15	4:51:23
23	64×68			36:54:50	29:03:49
24	56×88			213:33:00	146:38:48

Figure 8: Experimental results for minimum-area rectangles than contain all consecutive squares from 1×1 up to $N \times N$

to a solution of smaller area. One can also backtrack whenever the product of these upper bounds (which is equal to the upper bound of the area) is less than the total combined area of the rectangles.

It should be noted that the branch-and-bound approach is an *anytime* algorithm, since search can be interrupted at any time to extract the best solution that has been found so far.

Experimental Results

To test our algorithm, we begin with a set of square packing benchmarks proposed in (Korf 2004). Specifically, we consider the task of packing a set of N squares of sizes 1×1 up to $N \times N$ into a rectangle of minimum area. One benefit of this suite of test cases is that it provides a set of increasingly difficult instances, each easily specified by a single parameter. However, we also use them because they facilitate comparison with Korf's solver, which is hardcoded to handle these particular test cases, and cannot solve problems that contain either rectangles or squares of nonconsecutive sizes.[11]

It should be noted that there is nothing about the techniques we have developed that makes the problem of square packing any easier than that of packing oriented rectangles. However, since all blocks are squares, the enclosing space of any consistent solution can be rotated 90° to obtain an identical enclosing space of opposite dimensions. As a result, we can exploit this final dimension of symmetry by considering only one pairwise relationship between the largest pair of squares. Korf performs an equivalent operation by requiring the height of any enclosing space considered to be no greater than its width.

---

[11]As of the time of this writing, only the compiled version of Korf's solver was made available to the authors, and hence we could not make modifications to handle other cases.

Comparison of Solvers

In Figure 8 we present the runtimes of four solvers that are capable of optimal rectangle packing, including our own. The leftmost column reports the number of rectangles in the instance, and the neighboring column reports the optimal dimensions for that instance. Runtime is reported in hours, minutes, and seconds. Blank cells indicate instances where the computation time exceeded a time-out limit of ten days. Our experiments were conducted in Linux on a 2.2GHz Opteron processor with 8GB of RAM.

The first package (named BloBB) is a recent development in the VLSI community (Chan & Markov 2004) that makes use of an O-Tree data structure as its representation of a partial assignment. BloBB, like our solver, is a *topological* packer, in that its search space is one where a layout is described using relative relationships between rectangles rather than fixed positions. Among topological packers, BloBB was the first to optimally pack up to nine rectangles, surpassing the previous record of six. It also has several additional features that are not showcased in our experiments, such as the ability to handle soft blocks (whose dimensions are not necessarily fixed), as well as a hierarchical mode that can find high quality sub-optimal solutions rather quickly. In our experiments, we confirm that BloBB can indeed find optimal solutions to problems containing up to nine rectangles tractably, after which it begins to run out of steam.

The second algorithm is also a rather new technique (Clautiaux, Carlier, & Moukrim 2004) that builds on ideas in operations research literature, where this problem is known as the *two-dimensional orthogonal packing problem (2OPP)*. The approach is best described as a two stage branch-and-bound procedure, where an outer loop computes assignments to the x-coordinates of each rectangle, and an inner loop attempts to find consistent assignments to the y-coordinates. The algorithm is limited to solving the Containment Problem, and so to minimize area, we repeatedly call this algorithm for enclosing spaces of smaller and smaller area, using the same techniques as Korf's outer solver (Korf

# Rect.	Increment Parameter		
(N)	M = 10	M = 50	M = 200
15	0	0	0
16	2	1	1
17	8	7	7
18	42	25	25
19	3:25	2:58	2:58
20	17:07	11:05	11:05
21	55:11	44:56	44:56

Figure 9: Experimental results for BLUEBLOCKER on squares of nonconsecutive sizes

Techniques				Number of Rectangles (N)						
R	B	S	C	11	12	13	**14**	15	16	17
✓	✓	✓	✓	0.00	0.05	0.11	**0.19**	0.85	2.80	9.74
✓	✓	✓		0.01	0.04	0.12	**0.22**	0.97	3.25	11.44
	✓	✓	✓	0.02	0.04	0.14	**0.24**	1.06	3.85	14.11
	✓	✓		0.01	0.06	0.15	**0.26**	1.21	4.51	16.43
✓	✓		✓	0.01	0.05	0.20	**0.36**	1.54	5.03	18.72
✓	✓			0.01	0.07	0.25	**0.42**	1.77	5.88	22.12
	✓		✓	0.01	0.07	0.26	**0.44**	1.93	7.09	27.43
	✓			0.02	0.08	0.28	**0.46**	2.23	8.24	32.16
✓		✓	✓	0.01	0.06	0.31	**1.01**	7.07	40.10	260.01
✓		✓		0.02	0.08	0.40	**1.40**	10.95	61.09	380.87
✓			✓	0.02	0.12	0.58	**1.99**	13.84	77.43	
✓				0.02	0.14	0.77	**2.79**	21.58		
		✓	✓	0.40	0.81	2.54	**37.48**			
		✓		0.51	1.08	3.47	**46.14**			
			✓	0.76	1.46	4.91	**71.30**			
				0.99	1.97	6.78	**86.30**			

Figure 10: Contributions of techniques to BLUEBLOCKER's performance

2004). The approach fares reasonably well on problems as large as eighteen squares, but suffers on larger instances.

The third solver is the latest refinement to the original CSP formulation, and was used to generate the most recently published results (Korf 2004). As we alluded to earlier, these runtimes are unprecedented. As many as eighteen blocks can be packed in under two minutes using this algorithm, twice the number that can be handled by BloBB, and three times the number that the original graph-based method can manage. It has generally been believed that Korf's unique fixed-placement search space, in combination with a collection of powerful wasted space calculations, is what makes such results possible.

The final solver is ours, which we give the name BLUEBLOCKER. As explained in the preceding sections, our algorithm employs an entirely different search space than does Korf's, and thus it does not make use of his wasted-space calculations. Yet, as shown in the table, ours can also handle problems with eighteen rectangles or less in under two minutes. For problems larger than this, the performance of our solver is consistently competitive with Korf's, achieving superior results on five of the six remaining instances.

Ability to Scale with Rectangle Size

The benchmarks reported on in Figure 8 represent a very small and specific family of rectangle packing problems. In fact, we contend that the nature of these test cases makes them particularly amenable to the approach taken in Korf's solver. Recall that it uses a bitmap representation of the enclosing space when placing rectangles and computing wasted cells. Thus, its performance should be highly dependent on the size of the rectangles and of the dimensions of the enclosing spaces attempted. For instance, the placement of the 2×2 square requires 4 updates to this bitmap, whereas a square of size 200×200 would require 796 such updates (since only the pixels along the perimeter of the square need to be drawn). As the squares in the current benchmarks never approach this size, the original CSP search space is saved a great deal of bookkeeping.

This seemingly small detail can become a rather important issue when the application in question requires a high degree of precision. In scheduling applications, one may need to express that job j_1 requires 2 units of processing time, whereas job j_2 requires 2.01 units (perhaps due to a small setup time). To accurately represent these sizes using integral dimensions, one must encode the corresponding rectangles with larger widths of 200 and 201 units respectively, increasing their size by a significant amount. Similar complications arise often in VLSI design, where modules can easily differ slightly in size.

In contrast, our meta-CSP approach operates independently of rectangle size. To demonstrate this, we employ a new set of benchmarks, where in addition to the number of squares N, we define an increment parameter M to be the amount by which consecutive squares differ in size. For instance, the setting $N = 3$ and $M = 10$ corresponds to the set of squares $\{1 \times 1, 11 \times 11, 21 \times 21\}$.[12] In Figure 9 we report the performance of our solver with $N \in [15, 21]$ and $M \in \{10, 50, 200\}$. While the number of rectangles still has a dramatic impact on runtime, the presence of larger squares clearly has no negative effect. In fact, performance actually improves when M is increased from 10 to 50; this is likely due to the fact that it becomes easier to find room for the 1×1 square as the remaining squares grow in size. When M jumps from 50 to 200, no change in performance is observed at all. We suspect that to solve these test cases within the original CSP search space would require runtimes that are at least an order of magnitude slower. Unfortunately, we cannot verify this empirically since, as mentioned earlier, the existing implementation of Korf's solver is hardcoded to solve only the first set of benchmarks.

Relative Contributions of Techniques

In our final experiment, we study the contributions of our various meta-CSP pruning techniques to the performance of the BLUEBLOCKER solver. Specifically, we allow removal of subsumed variables (R), semantic branching (B), symmetry breaking (S), and displacement clique detection (C) to be enabled or disabled individually. In Figure 10 we present

---

[12]Note that the original benchmarks are a special case of this generalization with $M = 1$.

the runtimes (in seconds) of each of the resulting sixteen possible configurations on a subset of the original benchmarks. These configurations are sorted according to their performance on the $N = 14$ test case, which has been bold-faced for reference.

The fastest configuration requires all techniques to be enabled (as one would hope) while the slowest makes use of none of them. Of the four techniques, semantic branching has by far the greatest effect, as it is enabled in each of the top eight configurations. When semantic branching is enabled, symmetry breaking has the next largest impact; however, this is not the case when semantic branching is disabled, where instead removal of subsumed variables becomes more important. Detection of cliques in the displacement graphs has the smallest effect, although its impact is certainly not negligible. For instance, it consistently reduces runtime by at least 15% for the $N = 17$ test case. In summary, we find that each technique makes a substantial and positive individual contribution, and that best performance is obtained when these techniques are combined collectively.

Future Work

Our approach opens the door to several promising avenues of continued research. For instance, our current formulation cannot handle unoriented rectangles, which can be rotated $90°$ in any solution. Although the standard meta-CSP is unable to capture such flexibility, we suspect that our recent addition of conditional bounds and finite-domain constraints to DTPs (Moffitt, Peintner, & Pollack 2005) can be used to allow for this extension. Furthermore, the application of soft constraints to DTPs (Peintner & Pollack 2004) might prove to be useful in this domain, especially if objectives other than area are to be considered. Above all else, we hope that this newly discovered connection between temporal reasoning and rectangle packing will serve to stimulate discussion between researchers across neighboring disciplines.

Conclusion

In this paper, we have presented a new approach to optimal rectangle packing. Specifically, we have cast the problem of packing a set of rectangles with fixed orientations as a *meta-CSP*, in which partial assignments are subsets of relative pairwise relationships between rectangles. In doing so, we have resurrected a graph-based technique that was previously capable of dealing with only a small handful of rectangles. By combining existing pruning techniques found in constraint-based temporal reasoning with a suite of new domain-specific methods, we have created the BLUEBLOCKER solver, which has been shown to perform competitively compared to the previous state-of-the-art CSP formulation.

Acknowledgements

References

Armando, A.; Castellini, C.; and Giunchiglia, E. 1999. SAT-based procedures for temporal reasoning. In *Proceedings of the 5th European Conference on Planning (ECP-1999)*, 97–108.

Chan, H., and Markov, I. L. 2003. Symmetries in rectangular block-packing. In *Notes of the 3rd International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon '03)*.

Chan, H. H., and Markov, I. L. 2004. Practical slicing and non-slicing block-packing without simulated annealing. In *ACM Great Lakes Symposium on VLSI (GLSVLSI '04)*, 282–287.

Clautiaux, F.; Carlier, J.; and Moukrim, A. 2004. A new exact method for the orthogonal packing problem. *http://www.hds.utc.fr/˜fclautia/clautiaux_et_al-orthogonal*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Company.

Gent, I. P., and Smith, B. M. 2000. Symmetry Breaking in Constraint Programming. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, 599–603.

Guo, P.-N.; Cheng, C.-K.; and Yoshimura, T. 1999. An O-tree representation of non-slicing floorplan and its applications. In *Proceedings of the 36th Design Automation Conference (DAC 1999)*, 268–273.

Korf, R. E. 2003. Optimal rectangle packing: Initial results. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-2003)*, 287–295.

Korf, R. E. 2004. Optimal rectangle packing: New results. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-2004)*, 142–149.

Liao, Y., and Wong, C. K. 1983. An algorithm to compact a VLSI symbolic layout with mixed constraints. In *Proceedings of IEEE Transactions on CAD, Vol. 2, No. 2*.

Moffitt, M. D.; Peintner, B.; and Pollack, M. E. 2005. Augmenting disjunctive temporal problems with finite-domain constraints. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, 1187–1192.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*.

Murata, H.; Fujiyoshi, K.; Nakatake, S.; and Kajitani, Y. 1995. Rectangle-packing-based module placement. In *Proceedings of the 1995 International Conference on Computer-Aided Design (ICCAD '95)*, 472–479.

Nakatake, S.; Fujiyoshi, K.; Murata, H.; and Kajitani, Y. 1996. Module placement on BSG-structure and IC layout applications. In *Proceedings of the 1996 International Conference on Computer-Aided Design (ICCAD '96)*, 484–491.

Oddi, A., and Cesta, A. 2000. Incremental forward checking for the disjunctive temporal problem. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, 108–112.

Onodera, H.; Taniguchi, Y.; and Tamaru, K. 1991. Branch-and-bound placement for building block layout. In *Proceedings of the 28th Design Automation Conference (DAC '91)*, 433–439.

Peintner, B., and Pollack, M. E. 2004. Low-cost addition of preferences to DTPs and TCSPs. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-2004)*, 723–728.

Stergiou, K., and Koubarakis, M. 1998. Backtracking algorithms for disjunctions of temporal constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, 248–253.

Tsamardinos, I., and Pollack, M. E. 2003. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151(1-2):43–90.

Young, E. F. Y.; Ho, M. L.; and Chu, C. C. N. 2002. A unified method to handle different kinds of placement constraints in floorplan design. In *2002 Asia and South Pacific Design Automation Conference (ASP-DAC)*, 661–670.