Integrating Constraint Satisfaction and Spatial Reasoning

Unmesh Kurup and Nicholas L. Cassimatis
Rensselaer Polytechnic Institute
110 8th Street
Troy, NY 12180, USA
{kurup|cassin}@rpi.edu

Abstract

Many problems in AI, including planning, logical reasoning and probabilistic inference, have been shown to reduce to (weighted) constraint satisfaction. While there are a number of approaches for solving such problems, the recent gains in efficiency of the satisfiability approach have made SAT solvers a popular choice. Modern propositional SAT solvers are efficient for a wide variety of problems. However, particularly in the case of spatial reasoning, conversion to propositional SAT can sometimes result in a large number of variables and/or clauses. Moreover, spatial reasoning problems can often be more efficiently solved if the agent is able to exploit the geometric nature of space to make better choices during search and backtracking. The result of these two drawbacks - larger problem sizes and inefficient search - is that even simple spatial constraint problems are often intractable in the SAT approach. In this paper we propose a spatial reasoning system that provides significant performance improvements in constraint satisfaction problems involving spatial predicates. The key to our approach is to integrate a diagrammatic representation with a DPLL-based backtracking algorithm that is specialized for spatial reasoning. The resulting integrated system can be applied to larger and more complex problems than current approaches and can be adopted to improve performance in a variety of problems ranging from planning to probabilistic inference.

Introduction

The ability to represent and reason about space is a fundamental requirement for many intelligent agents. Not only are spatial reasoning problems ubiquitous, it has also been argued that reasoning in many domains can be reduced to reasoning in the spatial domain (Cassimatis 2006). However, since few domains are purely spatial, the usual method is to use a more general-purpose approach and to subsume spatial reasoning as part of that approach. One such approach, (weighted) constraint satisfaction, has been shown to be particularly effective. Many problems in AI, including planning, logical reasoning and probabilistic inference can be reduced to (weighted) constraint satisfaction problems. With the advent of faster and more efficient SAT solvers, constraint satisfaction via translation to SAT has yielded results that are often as good as or even better than other methods. The drawbacks with such approaches has been reported in (Kurup and Cassimatis 2010).

One way to handle drawbacks in SAT for specific kinds of problems is to integrate the SAT approach with more problem appropriate methods. This integration of domain-specific methods into the satisfiability process is captured under the umbrella of Satisfiability-Modulo Theories or SMT (DeMoura and Rue 2002). In SMT, parts of the formula that refer to the specific theory are handed over to the theory-specific solver while the rest is handled by the SAT solver. Quantifier Free Integer Difference Logic (QF-IDL) is one of the theories commonly used for reasoning about space in the SMT approach. In QF-IDL, spatial relationships between objects are represented as a set of inequalities. For example, the inequality $a_x \leq b_x - 1$, where a_x and b_x are the x-coordinates of objects a and b respectively, represents the fact that object a is to the left of object b. The inequalities can be represented as a graph structure and efficient algorithms exist that can check for satisfiability by checking for the presence of loops in the graph (Cotton 2005). However, while these inequalities are efficient in capturing the relationship between point objects, expanding their use to represent 2-d shapes has at least two drawbacks - One, the number of inequalities needed to represent a shape increases as the complexity of the shape increases since a shape is represented as a set of inequalities between its vertices. Two, if an object (even a simple one such as a rectangle) is allowed to rotate, the relationship between its vertices change and inequalities have to be written for each possible rotation of the object. The number of such sets of inequalities depends on the fineness to which the rotation needs to be captured. In this paper, we propose the use of diagrams as the appropriate theory for representing and reasoning about space. Further, reasoning about spatial constraints can be handled much more efficiently using a specialized DPLL-based backtracking algorithm that utilizes the geometric nature of space.

Representing Space

Consider a 3x3 grid. To encode the information that object a is $Next$ to object b, we would need a propositional SAT formula like the following (in non-CNF form):
$(AT(a,1,1) \wedge (AT(b,1,2) \vee AT(b,2,1) \vee AT(b,2,2))) \vee$
$(AT(a,1,2) \wedge (AT(b,1,1) \vee AT(b,2,1) \vee AT(b,2,2) \vee$

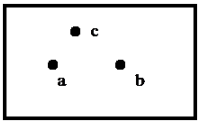

Figure 1: A diagram satisfying the constraint $Left(a, b)$ and $Above(c, a)$

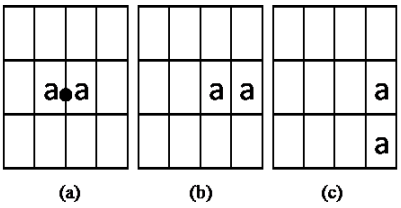

Figure 2: (a) rectangular object a at location (2,2). (b) Maximized possibility space for $LeftOf(a)$ without rotation. (c) Maximized possibility space with rotation

$AT(b, 1, 3) \vee AT(b, 2, 3))\ ) \vee \ldots$ and so on until every location in the grid has been accounted for. Even for simple spatial relations such as $Left$ or $Above$ the number of variables and clauses needed will grow as the size of the grid grows. Propositionalizing space for the purposes of SAT is, thus, an expensive approach. A spatial representation, on the other hand, can represent information more compactly by abstracting individual locations that share constraints into groups.

Diagram

Our spatial representation is based on the Diagrammatic Representation System (DRS) first proposed by (Chandrasekaran et al. 2004). A diagram in DRS consists of three types of objects - points, curves and regions. Point objects have only location (i.e., no spatial extent), line objects have a location and a spatial extent (denoting the axial specification of the curve), and region objects have location and spatial extent. The DRS specification is agnostic as to exactly how the spatial extent of a diagrammatic object is represented in the system. For our purposes, we represents curves as a set of line segments and a region by a curve that represents its perimeter.

The DRS also has a set of perceptual routines, such as $LeftOf$, $RightOf$ etc, that allow information to be extracted from a diagram; and a set of action routines that allow for the creation and modification of diagrams. The diagram forms the basic unit of spatial organization in our reasoning system.

Possibility Space

As mentioned earlier, one of the disadvantages of propositionalizing space is the need to account for the possibility of an object being in every location in the space. However, in qualitative reasoning, it is the relationships that hold between objects that matter rather than their exact locations in space. For example, in satisfying the constraint $Left(b)$ for an object a, we don't care whether a is one spot to the left of b or two spots to the left and so on. This means that we can generalize away from exact locations to location groups where the members of each group satisfy a common set of spatial constraints. Generalization in this manner leads to lesser number of individuals resulting in better performance when converted to propositional SAT. The concept of the possibility space (Wintermute and Laird 2007) allows us to do this generalization. A possibility space is a set of points that satisfy some set of spatial constraints. Every object in a diagram resides in a possibility space and spatial relationships between objects can be computed by finding intersections between these possibility spaces. For example, given a 4x3 diagram, an object a at location $\{(2, 2)\}$, and an object b with constraints $C(b) = \{Left(a), Above(a)\}$, the possibility space satisfying the constraint $Left(a)$ is $P_s(Left(a)) = \{(1, 1), (1, 2), (1, 3)\}$, the possibility space satisfying the constraint $Above(a)$ is $P_s(Above(a)) = \{(1, 1), (2, 1), (3, 1), (4, 1)\}$ and the possibility space of b is $P_s(b) = (P_s(Left(a)) \cap P_s(Above(a))) = \{(1, 1)\}$.

Maximizing a possibility space Given an object a with its possibility space $P_s(a)$ and a constraint $c(a)$, the maximized possibility space that satisfies $c(a)$ ($P_{s_{max}}(c(a))$) is the maximal set of all points that satisfy the constraint assuming a can be placed anywhere in $P_s(a)$. For example, consider the 4x3 diagram from earlier. Let a be a rectangular object of size 2x1. Further, let a occupy the locations $(2, 2)$ and $(3, 2)$ as shown in Figure 2(a). Let a's possibility space be the entire diagram. The usual calculation of possibility space for the constraint $Left(a)$, $P_s(Left(a))$, is the set of points $\{(1, 1), (1, 2), (1, 3)\}$. The maximized possibility space for a constraint $Left(a)$ is the set of points $\{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3)\}$. The configuration that allows this is shown in Figure 2(b). If the rotation of objects is allowed, the maximized space increases to $\{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)\}$ which is shown in 2(c). The angle of rotation that maximizes a possibility space depends on the object, its possibility space (if, for example, the object will not fit rotated 90 degrees, then a partial rotation will maximize the space) and the constraint on which the possibility space is constructed. This maximizing operation is an important part of our algorithm for satisfying spatial constraints.

Satisfying Spatial Constraints

One of the disadvantages of a diagrammatic representation constructed from propositional information is that it is often ambiguous. For example, let the constraints on three objects a, b and c be $\{Left(a, b), Above(c, a)\}$. It is straightforward to construct a diagram such as the one shown in Figure 1 based on these constraints. Now consider that an additional constraint $Below(c, b)$ is given. Based on the current configuration in the figure, it would be impossible to simple move c and satisfy this new constraint, but that does not mean that the constraints are not satisfiable. Given different locations for a or b, $Below(c, b)$ could be satisfied. To further complicate matters, when dealing with curves or regions, it is also possible to rotate these objects in order to

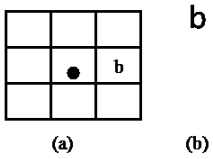

Figure 3: (a) Diagram with object b at $(3, 2)$. (b) The decision tree

satisfy various relationships. Thus, any system that uses diagrams must be capable of manipulating the diagrammatic representation so as to satisfy the given set of constraints (or to accommodate a new constraint).

One way of satisfying a set of constraints is via a depth-first search algorithm such as DPLL. DPLL-based variants have been shown to be some of the most powerful and effective reasoning algorithms in AI (Moskewicz et al. 2001). DPLL-based solvers have also been used to solve constraint satisfaction problems that have been shown to be equivalent to many problems in planning and probabilistic inference (Sang T. 1999). While these algorithms are effective for general constraint satisfaction, even when combined with a diagrammatic or spatial representation, satisfying spatial constraints can lead to repeated backtracking and a much larger search-space. To see how, consider how a DPLL-based algorithm would fare in the following spatial constraint satisfaction problem. There are three objects a, b and c, two constraints $Left(a, b)$ (interpreted as a is to the left of b though it can also be taken as b is to the right of a without any loss in explanatory power) and $Above(c, a)$, and a diagram of size 3x3. Since there are no constraints on b, its possibility space is the entire diagram. Since every point in the possibility space is equivalent, the algorithm picks one of them at random (let's assume $(3, 2)$) and branches on this location for b. Figure 3(a) shows the associated diagram and Figure 3(b) the current DFS tree. Next, the algorithm calculates the possibility space for a for the constraint $Left(a, b)$ [$= \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3)\}$]. Fig 4(a) shows this space (an x in a location marks that location as part of the possibility space). The algorithm picks a location from the space for a (say $(2, 2)$) and branches on it. Figure 4(b) and (c) shows the decision tree so far and the corresponding diagram respectively. Similarly, the algorithm calculates the possibility space for c based on $Above(c, a)$ and branches on a location from the possibility space (say $(2, 1)$). Figure 5 shows the calculated possibility space, decision tree and resulting diagram.

Assume a new constraint $Below(c, b)$ is given. The algorithm calculates the possibility space for $Below(c, b)$ then finds the intersection of this space with c's existing possibility space. Since the two spaces do not intersect, the calculated possibility space is empty and the algorithm has to backtrack. By the intersection of possibility spaces we know that changing the location of c will not satisfy the constraint. The algorithm backtracks further up the decision tree to a. Given a's current location and possibility space (Fig 4) there are 5 other choices for placing a. The algorithm will search through each location (assigning each location to a and backtracking when the choice fails) till it has covered every location. Once it has failed on all choices, it backtracks up another step on the decision tree to b. Again, the algorithm has to randomly search through all the possible locations for b in order to find one that satisfies all the constraints (in this example that location is either $(2, 1)$ or $(3, 1)$). Thus, backtracking over locations in space can quickly lead to exponentially larger search-spaces. Given moderately large diagrams and non-trivial number of objects, the algorithm can quickly become too slow to be of any use.

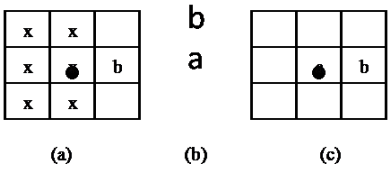

Figure 4: (a) Diagram showing possibility space for a given constraint $Left(b)$. (b) The decision tree. (c) Diagram with b and a.

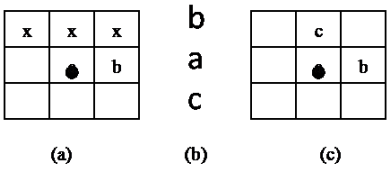

Figure 5: (a) Diagram showing possibility space for c given constraint $Above(a)$. (b) The decision tree. (c) Diagram showing b, a and c.

Reducing search for spatial constraint satisfaction

We present a way to reduce the search space for spatial constraint satisfaction problems by minimizing the number of backtracking operations that have to be performed by a DPLL-based algorithm. The key to this reduction is based on the maximization operation for possibility spaces described earlier. We now describe how this specialized algorithm works.

Let the system be trying to accommodate a new constraint $c(o_m, o_n)$ that is not currently satisfied in the diagram. For explanatory purposes, it is assumed that object o_m is the last decision in the backtracking tree. The systems starts by backtracking up the tree of location decisions. At each node along the backtrack, the system performs the following operation. Let's call the current node of the tree at which the backtrack algorithm is o_k. From o_k, the algorithm works back down the tree maximizing the possibility space of each object in that sub-tree. When it reaches the end of the tree (object o_m), it has found the maximal possibility space for o_m given o_k. The system finds the intersection of this maximal possibility space and the possibility space due to any

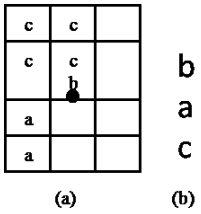

Figure 6: (a) The initial configuration satisfying $Left(a, b)$ and $Above(c, a)$. (b)The decision tree after placing b, a and c

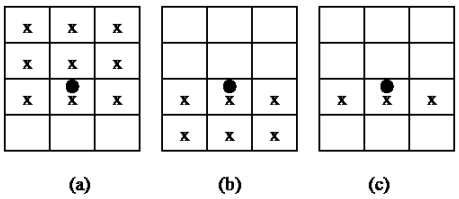

Figure 7: (a)Maximal space for Above(a). (b)Maximal Space for Below(b). (c)Intersection of (a) and (b).

other constraints on o_m (in this case $c(o_m, o_n)$). If this space ($p_{s_{max}}(o_1)$) is non-empty and o_m fits in this space, a solution to the problem can be found by moving object o_k. By maximizing possibility spaces in reverse from $p_{s_{max}}(o_m)$ to the object o_k back up the tree, the system finds the new possibility space for o_k such that $c(o_m, o_n)$ can be satisfied. If $p_{s_{max}}(o_m)$ is empty or o_m does not fit in this space, the algorithm backtracks up the tree and repeats this process. It continues backtracking till a possibility space where o_m fits is found or it has reached the beginning of the tree. In the latter case, it means that the set of constraints is not satisfiable.

To show how this works, consider the following example. Let the size of the diagram be 3x4 and the objects be a, b and c. a is a region object (we'll consider only rectangular regions in this paper) of size 1x2, b is a point object and c is a rectangular object of size 2x2. Let the constraints on the objects be $\{Left(a, b), Above(c, a)\}$. The current configuration looks as shown in Figure 6(a) with each element of the diagram array filled in with the details of the occupying object (overlaps are okay as long as the constraints are satisfied). The system now encounters the new constraint $Below(c, b)$. The system checks to see if this new constraint is satisfied by the diagram. Since it is not, the system starts backtracking up the tree of location decisions (Figure 6(b) shows the current tree) to the decision for object a. From here, it tries to maximize the possibility space of the next element below a in the tree. That object is c and the only constraint between c and a is $Above(c, a)$. Figure 7(a) shows the result of the maximization (each x marks a location that belongs to the possibility space). Note that this maximization is possible only if a is rotated 90 degrees to the right. If there were multiple constraints between c and a (or between c and any object in the sub-tree from a to c in the tree) the

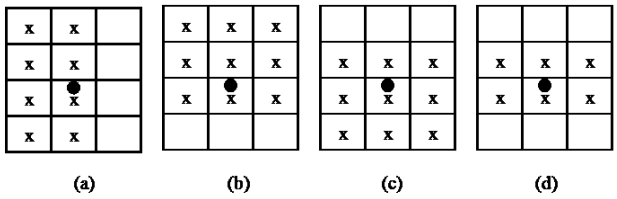

Figure 8: (a)Maximal Space for $Left(a, b)$. (b)Maximal Space for $Above(a)$. (c) Maximal Space for $Below(b)$. (d) Intersection of spaces (b) and (c)

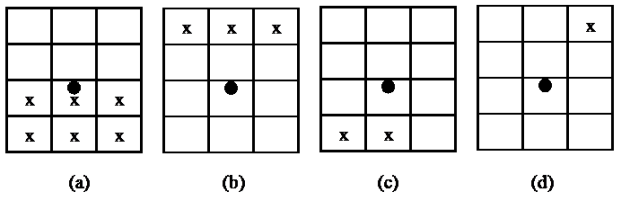

Figure 9: (a)Reverse maximization from c using constraint $Below(c)$. (b) Reverse maximization from c using $Above(c)$. (c)Intersection of (a) and constraint $Left(b)$. (d) Intersection of (b) and constraint $Right(a)$.

maximal space would be the intersection of the individual maximal spaces. Once the maximal space is found, the intersection of this space with the possibility space denoted by any remaining constraints on c that are from objects above a in the tree are calculated. In this example, that object is b and the possibility space due to the constraint $Below(c, b)$ is shown in Figure 7(b). The result of the intersection of these two possibility spaces is shown in Figure 7(c). Since c will not fit in the resulting space, the system backtracks again up the tree from a to b.

From b, it tries to maximize the space available for a (the object below it in the tree). Figure 8(a) shows the maximal space for a given the single constraint $Left(a, b)$. Once this space is calculated, the system moves down the tree to c and calculates its maximal space. c's maximal space is the intersection of the spaces resulting from maximizing a's possibility space with respect to the constraint $Above(c, a)$ (Figure 8(b)) and maximizing b's possibility space with respect to the constraint $Below(c, b)$ (Figure 8(c)). Figure 8(d) shows

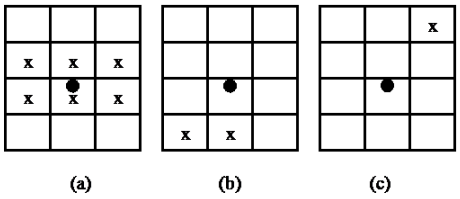

Figure 10: (a)New possibility space for c. (b) New possibility space for a. (c) New possibility space for b.

c's final maximal space. Since c fits in this space, the system concludes that the constraint $Below(c, b)$ can be satisfied. The entire process is now reversed from c through a to b to calculate the new possibility spaces for a and b such that $Below(c, b)$ can be satisfied. Figure 9(a), (b), (c), (d) show the steps in this reverse maximization. The new possibility spaces satisfying all constraints for c, a and b are shown in Figure 10(a), (b) and (c) respectively.

Evaluations

The possibility space approach has a number of points in contact with previous approaches such as Degrees of Freedom Analysis (Kramer 1992) and the accessible volume approach of (Brinkley et al. 1987). However, neither cover quite the same ground. Degrees of Freedom analysis is concerned with problems where objects have at least one fixed point about which they are free to rotate (hence "degrees"). The accessible volume approach specification is closer to the possibility space approach however the non-backtracking solution using this approach appears to derive from the use of abstractions which makes the approach incomplete. Further, while they have multiple heuristics to guide search, we use only a single backtracking algorithm to achieve our goal.

We decided to compare the performance of our system against that of SMT-based solvers due to two reasons - their widespread success and use in constraint satisfaction and the availability of implementations against which we were able to run our tests. The two solvers selected where yices2 and cvc3, two of the top SMT solvers in the QF-IDL section of the 2009 SMT competition. We tested all three solvers in the spatial constraint satisfaction problem described earlier - given a diagram size, a set of rectangles and a set of spatial constraints between these rectangles, find a configuration of these rectangles inside the diagram such that all constraints are satisfied. The objective of the spatial constraint satisfaction problem is to locate a set of objects in a grid space such that they satisfy a set of spatial relations.

Translation of the problem to QF-IDL In order to run these problems on Yices2 and cvc3, the problems need to be translated into QF-IDL problems. Recall that QF-IDL represents constraints between locations in the form of inequalities. There are two sets of inequalities that are needed to capture the spatial constraint satisfaction problem

1. A rectangle can be represented as a set of inequalities between its end-points. For example, consider a rectangle r with sides a and b. Let's denote the end-points of r starting from the top-left and going counter-clockwise as follows $\{(rx1, ry1), (rx2, ry2), (rx3, ry3), (rx4, ry4)\}$. Then, the set of inequalities that capture this rectangle (assuming r is not rotated) will be as follows:

$$\begin{aligned} \{rx1 - rx2 = 0, \quad & ry1 - ry2 = -b, \\ rx2 - rx3 = -a \quad & ry2 - ry3 = 0 \\ rx3 - rx4 = 0 \quad & ry3 - ry4 = b \\ rx4 - rx1 = a \quad & ry4 - ry1 = 0\} \end{aligned}$$

   However, these inequalities change if the rectangle is rotated. So, we need to add the inequalities between these end-points for each rotated version of r. The number of such inequalities depends on the degree of fineness in representing rotated version. We used one degree of separation between each rotated version for a total of 180 variations (Since the objects are all rectangles only 180 degrees of rotation is needed. For more arbitrary objects, inequalities that capture all 360 degrees of variation will be needed).

2. The second set of inequalities capture the constraints themselves. If rectangle a is to the left of rectangle b, the rightmost point of a is to the left of the left-most point of b. And similarly for other constraints such as $Right$, $Above$, $Below$ etc.

Step 1 adds 1440 inequalities for each rectangular object and step 2 adds 128 inequalities for every constraint between two objects.

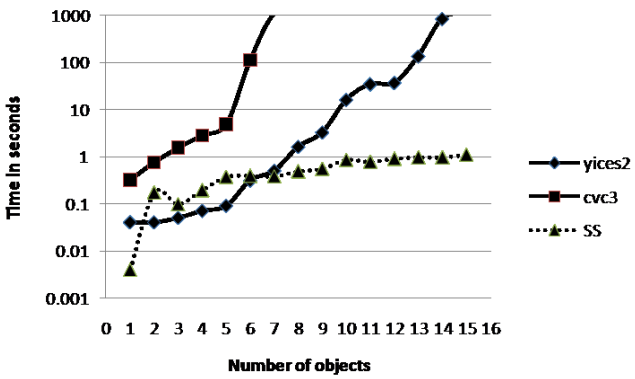

Figure 11: The results of running each solver on a problem with increasing number of objects and constant diagram size.

Results

We ran all three solvers on two sets of evaluation problems. For the first problem set we kept the size of the diagram constant at 350x60 while increasing the number of objects from 1 to 15. Each object was a rectangle of size 50x20. The constraints were kept simple - each new object was to be placed to the left of the previously introduced object. Any solver taking more than 15 minutes for a particular problem was considered to not have finished. Figure 11 shows the results of running each solver on this set. Note that the y-axis (time) is in the logarithmic scale. cvc3 took more than 15 minutes once the number of objects was over 6. Yices2 did better till the object count reached 14. In both cases, the time needed to solve the problem increased exponentially in the number of objects. SS, our spatial reasoning system, did much better with non-exponential growth across the number of objects. This is despite the fact that our implementation is not geared towards efficiency. It was written using convenient but inefficient data structures.The speedup obtained is purely due to the use of the spatial representation and the strategy for spatial constraint satisfaction.

For the second evaluation, we kept the number of objects constant at 8 and increased the size of the diagram from (350,60) to (1000,60) in increments of 50. As before, the relations between objects were limited to $Left$ or $Right$. Figure 12 shows the results of the evaluations. cvc3 was unable to complete the first few problems in the required time limit but once the diagram got bigger it stabilized to around the 17 second mark. Yices2 did better, steadily reducing the time required as the diagram size increased. SS took about a second for every problem in the set thought it did show a slight trend downwards as the diagram size increased.

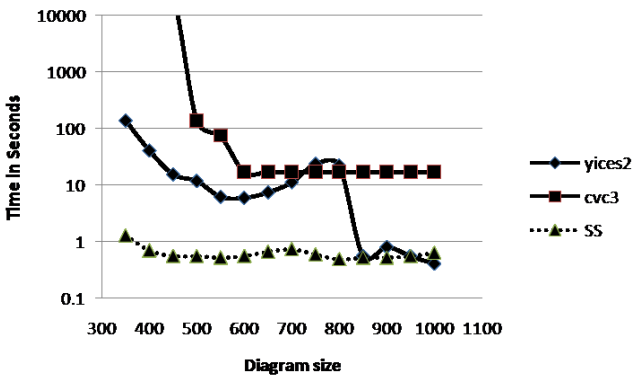

Figure 12: The results of increasing diagram size when the number of objects was kept constant

Discussion and Future Work

From both evaluations, it is clear that the SMT solvers do well with either small number of objects or much larger spaces presumably because any initial assumptions for different values of an object's end-points are equally valid. However, as the number of objects increased without a corresponding increase in space, their performance declined rapidly. In the case of both yices2 and cvc3, as the complexity of a region increases, the number of inequalities required to represent it also increases. Rotating more complex regions and checking if they fit in a possibility space affects the SS as well. Expanding our algorithm to be able to handle this increase in complexity is part of future work. Another area of improvement is in the nature of possibility space. Currently, possibility spaces are limited to being rectangles. Given the kinds of spatial constraints described in this paper ($Left$, $Above$ etc), this assumption is not a limiting factor. With more complicated spatial constraints, the restriction on possibility spaces being rectangles will have to be relaxed. More work needs to done to understand the effects of such complexity for the backtracking algorithm.

Another area for future work is to tease out the individual contributions of the two aspects of our work - the spatial representation and the backtracking algorithm. Previous work (Kurup and Cassimatis 2010) has shown that while spatial representations alone can increase ability of the SAT engine to handle the large numbers of constraints, it also places severe limitations on the number of objects that the system can handle (˜6). The inclusion of the backtracking algorithm is what allows the system to handle large number of objects as well. More evaluations need to be done to understand exactly how much each aspect contributes to the overall success of the approach.

Conclusion

Efficient spatial reasoning is an important component of many intelligent agents. Reasoning as constraint satisfaction has had great success especially with the advent of faster SAT solvers. However, the effects of propositionalizing space makes this approach to spatial reasoning expensive. One possibility is to use the SMT approach where a theory-specific reasoning system is integrated with a general SAT solver. Current SMT theories perform poorly due to the lack of a good representation for space and inefficient backtracking routines for search. In this paper, we have shown how a diagrammatic representation can be beneficial when representing and reasoning about space. When this representation is integrated with a DPLL-based backtracking search that is specialized for spatial problems, it can provide substantial performance improvements in problems ranging from planning to probabilistic inference.

References

Brinkley, J. F.; Buchanan, B. G.; Altman, R. B.; Duncan, B. S.; and Cornelius, C. 1987. A heuristic refinement method for spatial constraint satisfaction problems. Technical Report KSL-87-05, Knowledge Systems, AI Laboratory. STAN-CS-87-1142 15 pages.

Cassimatis, N. L. 2006. A cognitive substrate for human-level intelligence. *AI Magazine* 27(2).

Chandrasekaran, B.; Kurup, U.; Banerjee, B.; Josephson, J. R.; and Winkler, R. 2004. An architecture for problem solving with diagrams. In Alan Blackwell, Kim Marriot, A. S., ed., *Diagrammatic Representation and Inference conference*, 151–165. Springer-Verlag.

Cotton, S. 2005. Satisfiability Checking With Difference Constraints. Master's thesis, IMPRS Computer Science.

DeMoura, L., and Rue, H. 2002. Lemmas on demand for satisfiability solvers. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT)*, 244–251.

Kramer, G. A. 1992. *Solving Geometric Constraint Systems*. Cambridge, MA: MIT Press.

Kurup, U., and Cassimatis, N. L. 2010. Quantitative spatial reasoning for general intelligence. In *proceedings of Third Conference on Artificial General Intelligence*.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient sat solver. In *39th Design Automation Conference*, 530–535.

Sang T., Beame P., K. J. 1999. Solving bayes networks by weighted model counting. In *AAAI-05*, 318–325.

Wintermute, S., and Laird, J. 2007. Predicate projection in a bimodal spatial reasoning system. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*. Vancouver, Canada: Morgan Kaufmann.