Quantitative Spatial Reasoning for General Intelligence

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

Abstract

One of the basic requirements of an intelligent agent is the ability to represent and reason about space. While there are a number of approaches for achieving this goal, the recent gains in efficiency of the Satisfiability approach have made it a popular choice. Modern propositional SAT solvers are efficient for a wide variety of problems. However, conversion to propositional SAT can sometimes result in a large number of variables and/or clauses. Diagrams represent space as collections of points (regions) while preserving their overall geometric character. This representation allows reasoning to be performed over (far fewer number of) regions instead of individual points. In this paper, we show how the standard DPLL algorithm augmented with diagrammatic reasoning can be used to make SAT more efficient when reasoning about space. We present DPLL-S, a complete SAT solver that utilizes diagrammatic representations when reasoning about space, and evaluate its performance against other SAT solvers.

Introduction

One of the fundamental aspects of general intelligence is the ability to represent and reason about space. Successful approaches such as the Region Connection Calculus (Randell, Cui, and Cohn 1992) have concentrated on qualitaive spatial reasoning but an intelligent agent must be capable of reasoning about quantitative (or metric) space as well. With the advent of faster and more efficient SAT solvers, reasoning via translation to SAT has yielded results that are often as good as or even better than standard approaches. However, one drawback of propositionalizing first-order theories is the inherent explosion in variables and clauses. This is particularly true in quantitative spatial reasoning where space is usually represented using a simple Cartesian system. An object in this representation has a (x, y) coordinate that uniquely identifies where it lies in the space. This coordinate is also used to reason about the spatial relationships that the object forms with other objects in the same space. Translating a problem in this metric domain into propositional SAT involves propositions that capture the relationships between each pair of points in the space as well as instantiating propositions that capture the possibility that an object can be located anywhere in the space. As the space grows larger, the number of variables and clauses in the translation increase, making it more difficult to solve the problem.

The inclusion of domain-specific knowledge 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 represented as a graph structure and efficient algorithms exist that can check for satisfiability by checking for the prescence of loops in the graph (Cotton 2005). However, while these inequalities are efficient in capturing the relationship between point objects, expanding their use to capture the representation of 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, when a shape (even a simple one such as rectangle) is allowed to rotate, the relationship between its vertices change and inequalities will have to be written for each possible rotation of the shape. 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 diagrammatic models as the appropriate theory for representing and reasoning about space and show how the SAT approach can be augmented to use diagrammatic models as appropriate during solving. We evaluate our approach against the MiniMaxSat algorithm (Heras, Larrosa, and Oliveras 2008) in a spatial constraint satisfaction problem.

Satisfiability with Spatial Reasoning

Consider a 3x3 grid. To encode the information that object a is $Next$ to object b, we would need a SAT formula like the following (in non-CNF form): $(\ AT(a,1,1) \wedge (AT(b,1,2) \vee AT(b,2,1) \vee AT(b,2,2))\ ) \vee (\ AT(a,1,2) \wedge (AT(b,1,1) \vee AT(b,2,1) \vee AT(b,2,2) \vee AT(b,1,3) \vee AT(b,2,3))\ ) \vee \ldots$ and so on till 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. Diagrams, on the other hand, can represent information more compactly by abstracting individual locations that share constraints into groups. We extend the current satisfiability approach to include diagrammatic models as part of the representation. We first introduce the concept of a diagram.

Diagram

Definition 2.1 (Object Location) *Given a grid of size N_g and an object a, we define*

- *$L(a) = (x, y)|1 \leq x, y \leq N_g$ where (x, y) is the location of object a in the grid.*
- *$L_x(a) = x|1 \leq x \leq N_g$ and $L_y(a) = y|1 \leq y \leq N_g$ where x and y are the x- and y-coordinates of a in the grid*

Definition 2.2 (Relations) Spatial reasoning is based on the constraints that hold between objects in the space. We define five spatial constraint types ($T = \{Left|Right|Above|Below|Near\}$) that we use in this paper. More can be defined and added as necessary.

Given a grid of size N_g, objects a, b and N_e a nearness value, a constraint c holds between objects a and b iff one of the following hold

- *$c = Left$ and $L_x(a) < L_x(b)$*
- *$c = Right$ and $L_x(a) > L_x(b)$*
- *$c = Above$ and $L_y(a) < L_y(b)$*
- *$c = Below$ and $L_y(a) > L_y(b)$*
- *$c = Near$ and $L_x(b) - N_e \leq L_x(a) \leq L_x(b) + N_e, L_y(b) - N_e \leq L_y(a) \leq L_y(b) + N_e$*

Definition 2.3 (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 share certain common 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 3x3 grid, an object a at location (2,2), and an object b with constraints $C(b) = \{Left(a), Above(a)\}$, $P_s(Left(a)) = \{(1,1),(1,2),(1,3)\}$, $P_s(Above(a)) = \{(1,1),(2,1),(3,1)\}$ and $P_s(b) = (P_s(Left(a)) \cap P_s(Above(a))) = \{(1,1)\}$.

We define possibility spaces as follows: *Given a grid of side N_g, an object a and $c \in T$, we define*

1. *$P_s(c(a), I_i)$, the possibility space of a spatial constraint $c(a)$ with truth value I_i as follows*
   - *$c = Left, I_i = true$, $P_s(c(a), I_i) = \{(x_i, y_i)|1 \leq x_i, y_i \leq N_g;\ x_i < L_x(a)\}$*
   - *$c = Left, I_i = false$, $P_s(c(a), I_i) = \{(x_i, y_i)|1 \leq x_i, y_i \leq N_g;\ x_i \geq L_x(a)\}$*
   - *similarly for $c = Right, Above, Below$ and $I_i = true, false$*
   - *$c = Near, I_i = true$, $P_s(c(a), I_i) = \{(x_i, y_i)|1 \leq x_i, y_i \leq N_g;\ L_x(a) - Ne \leq x_i \leq L_x(a) + Ne;\ L_y(a) - Ne \leq y_i \leq L_y(a) + Ne;\ 1 \leq Ne \leq N_g\}$*
   - *$c = Near, I_i = false$, $P_s(c(a), I_i) = \{(x_i, y_i)|1 \leq x_i, y_i \leq N_g;\ (x_i, y_i) \notin P_s(Near(a), true)\}$*
2. *$P_{s_1} \cap P_{s_2}$, the intersection of two possibility spaces as follows*
   - *$P_{s_1} \cap P_{s_2} = \{(x_i, y_i)|(x_i, y_i) \in P_{s_1},\ (x_i, y_i) \in P_{s_2}$*
3. *$P_s(a)$, the possibility space of an object a as follows*
   - *$P_s(a) = \bigcap_{i=1}^{|C(a)|} P_s(c_i(a), I(c_i(a))$ where $C(a) = \{c_1, ..., c_k\}$ is the set of spatial constraints on a and $I(c_i(a)) = true|false$ is the truth value of the constraint $c_i(a)$*

Definition 2.4 (Diagram) Finally, object locations, possibility spaces and relations come together in the diagrammatic representation. A diagram is a collection of objects with their locations and possibility spaces. Reasoning about the spatial relationships between objects in a diagram can be accomplished using the objects' possibility spaces and locations. Formally, we define a diagram as follows:

A diagram d is a 6-tuple $< N_d, O, T, C, I, L >$ where

- *N_d denotes the side of the diagram (for the purposes of this paper, diagrams are considered to be square)*
- *$O = \{a_1, a_2, \ldots, a_k\}$is a set of objects*
- *$T = \{Left|Right|Above|Below|Near\}$ is a set of relation types*
- *C is a set of spatial constraints from T that holds between objects in O*
- *$I : C \rightarrow true|false$ is an assignment of truth values to the constraints in C*
- *$L : O \rightarrow N_d \times N_d$, the location of objects in the diagram*

C is a set of spatial constraints for objects in the diagram. If $C(a_i) = \emptyset$, then a's possibility space is the entire diagram. $L(a_i)$ is the location of a_i in the diagram such that $L(a_i) \in P_s(a_i)$. This location is chosen such that it is at the center of the object's possibility space.

Definition 2.5 *A diagram d satisfies a set of spatial constraints C iff for each $c(a, b) \in C$ and $a, b \in O$, the constraint C holds between a and b in d.*

Satisfiability with Spatial Reasoning (SAT-S)

In order to combine the best of both SAT and diagrammatic reasoning, we introduce a version of SAT called SAT-S, that allows for the representation of spatial information using diagrammatic models. Formally,

A problem specification in SAT-S is given by the 6-tuple $S_s = <\phi, P, O, T, C, M>$ where

- *ϕ is a SAT formula in CNF form*
- *P is the set of variables in ϕ*
- *O is a set of objects*
- *$T = \{Left|Right|Above|Below|Near\}$ is a set of spatial relation types*
- *C is a set of constraints of the form $c(a,b)$ where $c \in T$ and $a, b \in O$*
- *$M : P \rightarrow C$ is a mapping from P to C*

ϕ and P are the same as in SAT. O is a set of objects in the domain. T is the set of spatial relation types that are relevant in the domain. C is a set of constraints from the set $T \times O \times O$ and represents the spatial relations that are important in the domain. Finally, M is a mapping from the set of variables to the set of relations. This mapping allows the SAT-S solver to recognize variables that represent spatial literals. For convenience, in the remainder of this paper, we refer to such variables as spatial variables, even though they are first-order propositions.

Solutions in SAT-S

A solution (model) to a problem in SAT-S is an assignment of truth values, $I_p = \{true, false, neutral\}$ to the variables in P. A valid model in SAT-S is an assignment I_p such that

- *Every clause in the ϕ evaluates to true and*
- *$[\exists P' \subseteq P | (\forall p \in P') I_p(p) \neq neutral \wedge M(p) \neq \emptyset] \Rightarrow |D| > 0 \wedge (\forall d \in D, p \in P') d$ satisfies $M(p)$, where D is a set of diagrams constructed during the solving process.*

The set of diagrams D represents the possible configurations of objects in O given the spatial variables that have been assigned values. If a problem in SAT-S has propositions that denote spatial relations and these propositions have been assigned values in the solution then there must be at least one diagram in D and every diagram in D must satisfy these spatial relations.

DPLL-S - An Algorithm for Solving Problems in SAT-S

Algorithm 1 shows the DPLL-S algorithm. The main difference from the original DPLL algorithm is as follows: When a value is set for a variable, DPLL-S checks to see if it is a spatial variable. If it is not, the algorithm proceeds like in standard DPLL. If it is a spatial variable, say $M(v) = c(a,b)$, DPLL-S constructs a set of diagrams that satisfy the proposition in the following way –

1. If there are no diagrams (this is the first spatial proposition whose value has been set), it creates an empty diagram d and adds b (assigns its possibility space and a location within the possibility space) to this diagram.

Algorithm 1 DPLL-S

```
Procedure DPLL-S(C,P,D,M)
if C is empty, return satisfiable
if C contains an empty clause,
 return unsatisfiable
if C contains a unit clause
 if variable in unit clause is spatial
  D' = Propagate(D,M(v),true);
  if D' is empty, return unsatisfiable
  return DPLL-S(C(v=true),P,D',M)
 return DPLL-S(C(v=true),P,D,M)
pick an unassigned variable v
if v is spatial
 D' = Propagate(D,M(v),true)
 if D is empty, return unsatisfiable
else
 D'=D
 if DPLL-S(C(v=true),P,D',M) is satisfiable,
  return satisfiable
 else
  if v is spatial
   D' = Propagate(D,M(v),false)
   if D is empty, return unsatisfiable
   else
    D'=D
    return DPLL-S(C(v=false),P,D',M)
```

2. If one of the objects is present in the diagrams in D, then from each diagram d_i in D, new diagrams $\{d_{i1}, \ldots, d_{ik}\}$ are constructed such that every d_{ij} satisfies $c(a,b)$. In addition, each d_{ij} also satisfies a different subset of the set of all possible relevant relations between a and O given d_i where O is the set of all objects in d. Thus, the new set of diagrams $D' = \{d_{11} \cup \ldots \cup d_{lk}\}$ where $|D| = l$. If $D' = \emptyset$, the procedure returns unsatisfiable.
3. If both objects are present in D, $D' = \{d_i | d \in D; d_i$ *satisfies* $c(a,b)\}$. If $D' = \emptyset$, the procedure returns unsatisfiable.

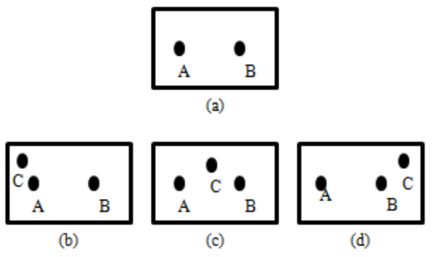

Figure 1: Diagrams created by Propagate for Above(c,a)

The creation of new diagrams once a spatial variable is found is done by the propagate method (shown in Algorithm 2) which takes a spatial constraint $M(v)$, the truth value I_v of this constraint and a set of diagrams D that satisfy a set of constraints C and produces a new set of diagrams D' such that every diagram in D' satisfies $C \cup M(v)$ (if $I_v = true$) or $C \cup \neg M(v)$ (if $I_v = false$). Each individual diagram in D' is constructed such that it captures some subset of the

Algorithm 2 Propagate

```
Procedure Propagate(D,c(a,b),tvalue)
if a and b in d ∈ D,
 eliminate all d from D
 that do not satisfy c(a,b).
 return D
if D is empty,
 create a new diagram d,
 add b to d
D′ = empty set of diagrams
for every d ∈ D
  $C = \{c(a,o) | c \in T; o \in O\}$
  $I$ is the set of all
   possible truth value assignments
   to c ∈ C and $I_i(c_j)$ is the ith
   truth value assignment for $c_j$.
  D′ = D ∪ $d_i^{'}$ where
  $d_i^{'} = d_i \cup a$ where
  $P_s(a) =$
  $P_s(r(b)) \cap \left(\bigcap P_{j=}^{|C|} P_s(c_j, I_i(c_j)\right), P_s(a) \neq \emptyset$
```

set of all possible relevant spatial relationships between the new object being introduced into the diagram and the objects currently in the diagram. Together, these diagrams capture all possible spatial relationships that the new object can be in given the constraint set $C \cup M(v)$.

As an example, consider the diagram in Fig 1(a). It has two objects A and B with $C(A) = Left(B)$. For expository purposes, let us assume that there are only two types of spatial relations that are relevant to our problem – $Left$ and $Above$, i.e., $T = \{Left, Above\}$. Let $M(v) = Above(c, a)$ where c is the new object being introduced into the diagram. Then,

$$\begin{aligned} d_1 &= d + c \text{ where } P(c) = P_s(Above(A) \cap \neg Left(A) \cap \\ &\quad \neg Left(B) \cap \neg Above(b)) \\ &\vdots \\ d_8 &= d + c \text{ where } P_s(c) = P_s(Above(A) \cap Left(A) \cap \\ &\quad Left(B) \cap Above(b)) \end{aligned}$$

In general, given a diagram d, a new object c, a set of constraints C and an assignment of truth values I to elements in C,

$$\begin{aligned} D' &= D + d_i, \text{ where} \\ d_i &= d + c \text{ where } P_s(c) = P_s(r(b)) \cap \\ &\quad \left(\bigcap P_{j=}^{|C|} P_s(c_j, I(c_j)\right), P_s(c) \neq \emptyset \end{aligned}$$

In the worst case, there are $|D| \times 2^{|T| \times |C| - 1}$diagrams produced at each propagate step. On average, only a subset of these diagrams are possible. In the previous example, the diagrams where $P_s(c) = P_s(Above(A) \cap Left(A) \cap \neg Left(B) \cap Above(B))$ is not possible because an object cannot be both to the $Left$ of A and not to the $Left$ of B given $C(a) = \{Left(b)\}$. In such a case, $P_s(c) = \emptyset$ and the diagram is eliminated.

Completeness

Lemma 3.1 - By definition of $Left$, $Right$, $Above$, $Below$ and $Near$, the following are true

1. $Left(a, b) = Right(b, a)$ and vice versa
2. $Above(a, b) = Below(b, a)$ and vice versa
3. $Near(a, b) = Near(b, a)$

Lemma 3.2 - Given a diagram of size N_d, set of spatial constraints $C(b) = \{c_1(a_1), \ldots, c_j(a_k)\}$ on object b, there is a location $L(b)$ for object b that satisfies $C(b)$ iff $\left(\bigcap_{j=1}^{|C(b)|} P_s(c_j, I(c_j))\right) \neq \emptyset$

Proof: By definition of L in diagrams.

Theorem 3.1 Given a problem in SAT-S, $S_s =< \phi, P, O, T, C, I, M >$ for which a solution $< I'D >$ exists, where $|D| > 0$, DPLL-S will find at least one diagram d such that d satisfies the constraints C

Proof: In each execution step DPLL-S handles one of the following cases

1. Not a spatial variable. Continue as in DPLL
2. $|C| = k + 1$ and $|O| = m + 1$. A new object $a_{m+1}$is added to D with the constraint $c_{k+1}(a_j)$ where $a_j \in O$ is an object in D. Then, by construction and by lemma 3.2, all possible relationships between $a_{m+1}$and all objects in D including a_j are satisfied by D'.
3. $|C| = k + 1$ and $|O| = m$. A new constraint $c_{k+1}(a_j)$ is added to an existing object a_i.
   (a) a_i was added after a_j - By construction and lemma 3.2, all possible relationships between a_i and a_j given C are satisfied.
   (b) a_j was added after a_i - By construction, lemma 3.2 and lemma 3.1, all possible relationships between a_i and a_j given C are satisfied.

Hence proved.

Experiments

We compared the performance of DPLL-S to that of MiniMaxSAT (Heras, Larrosa, and Oliveras 2008) in a spatial reasoning problem called the placement problem, a generic version of the 8-queens puzzle. The choice of MiniMaxSAT as the standard for a traditional SAT solver was one of convenience. zChaff or SATzilla may have been marginally better but the reasons why MiniMaxSAT performs so poorly holds for the other solvers as well.

The Placement Problem

The objective of the Placement problem is to locate a set of objects in a grid space such that they satisfy a set of spatial relations. Formally, a placement problem P can be defined as a 4-tuple $P =< N_g, O, T, R >$ where N_g is the side of a square grid , O is a set of objects, $T = \{Left|Right|Above|Below|Near\}$ is a set of spatial relation types, $N_e = N_g/3$ is a nearness value and R is a set

	SAT						SAT-S					
	#Near=1		#Near=2		#Near=3		#Near=1		#Near=2		#Near=3	
Grid Size	#vars	#clauses	#vars	#clauses	#vars	#clauses	#vars	#clauses	#vars	#clauses	#vars	#clauses
25	153	128893	153	161952	153	377511	3	3	3	3	3	3
35	213	461408	213	765482	213	1217816	3	3	3	3	3	3
45	273	770949	273	2103939	273	2282724	3	3	3	3	3	3
55	333	1795243	333	5752032	333	5340666	3	3	3	3	3	3
65	393	3609053	393	10913087	393	16353831	3	3	3	3	3	3
75	453	10119709	453	16119434	453	22119159	3	3	3	3	3	3
85	513	10239193	513	20424402	513	36685921	3	3	3	3	3	3

Table 1: Number of variables and clauses for SAT and SAT-S

Grid Size	MiniMaxSAT times			DPLL-S times			MiniMaxSAT(SAT-PP) times		
	#Near=1	#Near=2	#Near=3	#Near=1	#Near=2	#Near=3	#Near=1	#Near=2	#Near=3
25	0.34s	0.43s	1.31s	0.016s	0.0s	0.0s	0.82s	0.25s	0.02s
35	1.590s	2.46s	3.87s	0.0s	0.016s	0.0s	0.06s	0.02s	0.63s
45	2.334s	6.61s	6.93s	0.0s	0.0s	0.0s	0.5s	0.27s	0.06s
55	5.36s	17.46s	15.83s	0.0s	0.0s	0.0s	1.13s	0.18s	0.19s
65	10.97s	1m22.7s	2m26.4s	0.0s	0.0s	0.0s	1.09s	0.17s	.07s
75	1m54.5s	2m31.6s	10m39s	0.0s	0.0s	0.0s	0.23s	0.26s	0.32s
85	1m59.6s	5m31.2s	-	0.0s	0.0s	0.0s	0.36s	0.03s	0.21s
90	9m38.8s	-	-	0.0s	0.0s	0.0s	5.22s	0.03s	0.25s

Table 2: Comparison of MiniMaxSat and DPLL-S on Placement problems of increasing grid size

of spatial relations that hold between objects in O. A solution to the placement problem P is given by an assignment $S : O \rightarrow N_g \times N_g$ that identifies a unique location for each object $a \in$O such that all constraints in R are satisfied.

Translation from P to SAT

Literals For every object $a_i \in O$ and location (k, l) in the grid, we assign two variables of the form $AT_{x_k}(a_i)$ and $AT_{y_l}(a_i)$, corresponding to the object's x and y-coordinates. A pair of variables $AT_{x_k}(a_i)$ and $AT_{y_l}(a_i)$ are true iff the object a_i is at location (k, l) in the grid. For every relation $r(a_i, a_j)$ in R, we add a variable r_m such that r_m is true iff $r(a_i, a_j) \in R$.

Clauses For every object a_i, we add two clauses $\Big((AT_{x_1}(a_i)) \vee \ldots \vee AT_{x_{N_g}}(a_i)\Big)$ and $\Big((AT_{y_1}(a_i)) \vee \ldots \vee AT_{y_{N_g}}(a_i)\Big)$

For each $Left$ relation $Left(a_i, a_j)$, we add clauses that capture the following constraint $Left(a_i, a_j) \Rightarrow$

$$\Big[\Big((AT_{x_1}(a_i) \wedge AT_{x_2}(a_j)) \vee \ldots \vee (AT_{x_1}(a_i) \wedge AT_{x_{N_g}}(a_j))\Big)$$

$$\vee \ldots \vee \Big((AT_{x_{N_g-1}}(a_i) \wedge AT_{x_{N_g}}(a_j))\Big) \ ]$$

We add similar constraints for each $Above, Right,$ $and\ Below$ relations.

For each $Near$ relation $Near(a_i, a_j)$, we add clauses that capture the following constraint $Near(a_i, a_j) \Rightarrow \bigvee_{k=1,l=1}^{N_g,N_g} (\ AT_{x_k}(a_i) \wedge AT_{y_l}(a_i) \wedge (\bigvee_{m=k-N_e,n=l-N_e}^{k+N_e,l+N_e}(AT_{x_m}(a_j) \wedge AT_{y_n}(a_j)))\ )$ where $1 \leq m, n \leq N_g$

For every object a_i, we add clauses that capture the constraint that an object can be only at one location in the grid [$(AT_{x_1}(a_i) \wedge AT_{y_1}(a_i) \wedge \neg(AT_{x_1}(a_i) \wedge AT_{y_2}(a_i)) \wedge \ldots \wedge \neg(AT_{x_{N_g}}(a_i) \wedge AT_{y_{N_g}}(a_i))) \vee \ldots \vee (AT_{x_{N_g}}(a_i) \wedge AT_{y_{N_g}}(a_i))$]

For every location (k, l), we add clauses that capture the constraint that there can be only object at that location [$((A(T_{x_k}(a_1) \wedge AT_{y_l}(a_1)) \wedge \neg(A(T_{x_k}(a_2) \wedge AT_{y_l}(a_2)) \wedge \ldots \wedge \neg(A(T_{x_k}(a_{|O|}) \wedge AT_{y_l}(a_{|O|}))) \vee \ldots \vee (A(T_{x_k}(a_{|O|}) \wedge AT_{y_l}(a_{|O|}))$]

All constraints were converted to CNF form without an exponenetial increase in the number of clauses. Due to space constraints, we do not provide a proof of this translation but a proof by contradiction is straightforward.

Translation from P to SAT-S

Literals For every relation $r(a_i, a_j)$ in R, we add a variable r_m, and set $M(r_m) = r(a_i, a_j)$

Clauses For every relation $r(a_i, a_j)$ that is true, we add a clause containing the single literal r_m. We add a clause with the single literal $\neg r_m$ otherwise.

Results

We ran MiniMaxSAT and DPLL-S on a set of problems that varied based on the grid size N_g, the number of relations R and the number of objects O. All problems in the set were solvable and were created by randomly placing the requisite number of objects in a grid of the relevant size. Relations were then randomly chosen. Table 1 shows the number of variables and clauses for SAT and SAT-S problems for dif-

#relations	MiniMaxSAT	DPLL-S
5	0.9s	3.89s
10	2.49s	5.8s
15	3.49s	5.69s
20	4.19s	6.12
25	5.37s	6.19s
30	23.01s	1.46s
35	8.09s	6.52s
40	13.59s	4.76s

Table 3: MiniMaxSAT and DPLL-S runtimes for increasing number of relations

#objects	MiniMaxSAT	DPLL-S
3	0.43s	0.02s
4	1.46s	0.02s
5	1.33s	0.76s
6	1.95s	-

Table 4: MiniMaxSat and DPLL-S runtimes for increasing number of objects

ferent grid sizes. The size of a SAT-S problem depends on the number of relations that have to be satisfied. Problem sizes in SAT vary on the grid size, number of objects, number of relations and relation type. $Near$ relations are more costlier as they result in many more clauses than the other four relation types. For this reason, we have broken down Tables 1 and 2 based on the number of $Near$ relations as well. Table 2 shows the execution times for MiniMaxSAT and DPLL-S for increasing grid size. We stopped a run if it did not produce a result within 15 minutes. For comparison purposes, we used the propagate method to pre-processes a SAT-S problem into a SAT problem whose runtimes are shown in the table as SAT-PP. The runtime for a single problem in SAT-PP is the sum of the times required to pre-process it into SAT and solve using MiniMaxSAT. From the table it is clear that a SAT-S problem requires virtually no time to run. This is despite the fact that the DPLL-S implementation is not geared towards efficiency. It was written using convenient but inefficient data structures and run in parallel with other processes on the system.The speedup obtained is purely due to the use of diagrammtic models.

Table 3 shows the comparison between MiniMaxSAT and DPLL-S for increasing number of relations given a standard grid size of 25x25 and 5 objects. For MiniMaxSAT, as the number of relations increase, the runtime increases. For DPLL-S, the increase is minimal because given a fixed number of objects, as the relations increase, the propagate method merely has to eliminate diagrams that do not satisfy relations instead of having to generate new diagrams.

Table 4 shows the comparison between MiniMaxSAT and DPLL-S for increasing number of objects with the grid size fixed at 25. This is where DPLL-S underperforms MiniMaxSAT. By the time the number of objects gets to six, there are so many diagrams being generated by the propagate method that it becomes impossible for the algorithm to finish within the 15 minute limit. Since the number of diagrams generated increases exponentially, even an efficient implementation of the current DPLL-S algorithm would not be able to solve problems with more than 7 or 8 objects. There are, however, approaches that could allow us to effectively deal with this problem. Our current work is focused on maintaining a single diagram rather than the entire set of diagrams. As spatial constraints arise, this diagram is manipulated to satisfy these constraints. This strategy, while preserving completeness will provide better overall scalability at the expense of run times with small number of objects.

Conclusion

The satisfiability approach to problem solving has shown great promise in recent years. However, the effects of propositionalizing space, makes the satisfiability approach to spatial reasoning expensive. In this work, we have shown how the satisfiability approach can be augmented with the use of diagrammatic models to reduce the problem space. Our approach utilizes diagrams to represent space as discrete regions instead of individual points. This leads to savings in both problem size and solution times. We introduced a complete solver called DPLL-S, a variation of DPLL, and evaluated it against a current SAT solver MiniMaxSAT in a spatial reasoning problem. Our approach greatly reduced the number of variables and clauses in the formula and led to nearly instantaneous runtimes in many cases for solving the placement problem. One of the problems with the current DPLL-S approach is the explosions in the number of diagrams as a function of the number of objects and relations. In future work, we address these concerns and compare our algorithm against SMTs such as QF-IDL.

References

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 *In Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT)*, 244–251.

Heras, F.; Larrosa, J.; and Oliveras, A. 2008. MiniMaxSAT: An efficient Weighted Max-SAT Solver. *Journal of Artificial Intelligence Research* 31:1–32.

Randell, D. A.; Cui, Z.; and Cohn, A. G. 1992. A spatial logic based on regions and connection. In *Proceedings 3rd International Conference on Knowledge Representation and Reasoning*.

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.