SAT Based Motion Planning of a Multi-Robot System

Manish Kumar Bera
Advisor: Dr. Indranil Saha

November 23, 2017

1 Introduction

1.1 Motivation

Swarm robotics is a field of multi-robotics in which large number of identical robots are coordinated in a centralized[1] or distributed[3] way. It is supposed that a desired collective behavior emerges from the interactions between the robots and interactions of robots with the environment. With recent technological developments, producing a large number of inexpensive robots that are equipped with sophisticated sensing, computation and communication tools has become a reality[5]. Swarm robots can be used to form various spatial arrangements for any required purpose. To plan a path for all the robots in a system is a difficult task. Multi-robot motion planning has in fact been shown to be NP-complete[9].

1.2 Objective

Robots need to move simultaneously, to go from one position to another, to collectively form a new configuration. Motion Planning problems are categorized into 2 classes:

- sum of costs
- makespan

Further, the solvers are also divided into 2 types:

- search based
- reduction based

In this project we work towards developing a reduction based solver for the motion planning problem using a SAT-solving tool. Our goal is to optimise the *makespan* for the collective motion of the robots from initial configuration to final configuration. We will be doing this with the help of a SAT based tool.

2 Problem Statement

Given a $N \times N$ grid with R number of robots and G number of obstacles. The initial and final positions of the robots along with the positions of the obstacles are given. Our task is to generate optimal trajectory for all the robots such that it takes minimum makespan.

3 Methodology

The idea is to reduce the problem of motion planning into a SAT problem. A SAT-solver is used to solve the SAT instance. The output of the SAT solver is then processed to get meaningful information, in our case, the path of the robots.

Note: As most SAT solvers accept input in DIMACS format, our formulation has to be in Conjunctive Normal Form(CNF). All the following constraint disjunctions will be connected by conjunctions. In our case we have used the MINISAT[4].

We consider a grid A of size $N \times N$. We will refer the collection of all the robots in the grid as *swarm*. We will have R number of robots such that a robot will occupy a cell of the grid. No two robots can occupy the same cell of the grid simultaneously, and a robot will occupy only a single cell of the grid at any time. We define a *timestep* as the time period after which a robot can execute a *move*. A robot can execute a *move* on each *timestep*. Each move will be selected from a set of *Motion Primitives*.

Let L be the total number of *timesteps* which the *swarm* takes to go from initial configuration to final configuration, i.e., L is the *makespan*. Let x_t^i be a variable that describes the position of ith robot after t time steps. We define *swarm position at timestep* t, X_t as :

$$X_t = (x_t^0, x_t^1, x_t^2, x_t^3, ..., x_t^{R-1})$$

i.e., X_t is an ordered collection of tuples, with the ith tuple describing the position of the ith robot after t *timesteps*. Thus, X_t describes the position of the *swarm* in the grid after t *timesteps*. So, we can say that $X_0 = (x_0^0, x_0^1, x_0^2, x_0^3, ..., x_0^{R-1})$ represents the initial configuration of the *swarm*, and $X_L = (x_L^0, x_L^1, x_L^2, x_L^3, ..., x_L^{R-1})$ represents the final configuration of the *swarm*.

The movement of the *swarm* will obey a set of rules. The progress from one *swarm position* to another will always follow certain conditions. Our aim is to develop a set of constraints, whose solution will give an optimal solution for the *swarm*. We will frame these constraints using a set of boolean variables, which will be encorporated into a formula ϕ. We define a *solution* S:

$$S = [X_0, X_1, X_2, ..., X_L]$$

where S is a sequence of *swarm positions* which occur in the chronological order of *timesteps*. The symbol S represents the sequence in which the *swarm* movement can occur from initial to final configuration. A *solution* S is said to be valid iff:

$$S \models \phi \tag{1}$$

$$\Leftrightarrow [X_0, X_1, X_2, ..., X_L] \models \phi \tag{2}$$

The problem will thus be reduced from an optimal path finding problem to a problem of finding satisfiability(SAT)[2]. We will use a SAT solver (like the MiniSAT[4]) to compute the solution to the above model.

4 Defining the constraints

The rules for the robots to move have to be formulated into boolean expressions. For this we will have to define boolean variables. Most of the constraints mentioned in this section are similar to the ones mentioned in [10]. The constraints for headon collision are additional to those in the paper.

4.1 Defining boolean variables:

I define $X(t, r, x, y)$ as a boolean variable s.t. $X(t, r, x, y)$ is *TRUE* iff at time step t, the rth robot is at position (x, y). We know that in DIMACS format, each variable is written as a number. So, we define X(t,r,x,y) in the following way:

$$\begin{aligned} X(t, r, x, y) &= t \times R \times N^2 + r \times N^2 + x \times N + y + 1 \\ & t \in \{0, ..., L\} \\ & r \in \{0, ..., R-1\} \\ & x \in \{0, ..., N-1\} \\ & y \in \{0, ..., N-1\} \end{aligned}$$

So, the number of variables $= (L+1) \times R \times N^2$.

4.2 Constraints for initial and final states:

I define (x_0^r, y_0^r) to be the initial position of the rth robot. Similarly, (x_L^r, y_L^r) is the final position of the rth robot. The constraints for initial and final positions will be: $\forall r \in \{0, ..., R-1\}; \forall x \in \{0, ..., N-1\}; \forall y \in \{0, ..., N-1\}$:

$$\begin{aligned} & STARTPOSITION: \\ & X(0, r, x, y);\ if\ (x, y) = (x_0^r, y_0^r) \\ & \neg X(0, r, x, y);\ otherwise \end{aligned}$$

Number of clauses $= R \times N^2$

$$\begin{aligned} & FINALPOSITION: \\ & X(L, r, x, y);\ if\ (x, y) = (x_L^r, y_L^r) \\ & \neg X(L, r, x, y);\ otherwise \end{aligned}$$

Number of clauses $= R \times N^2$

The time complexity for forming the constraints for the initial and final positions will be $O(R \times n^2)$.

4.3 Constraints for motion primitives:

I assume that the robots will have five simple motion primitives. each robot can:
1. Move up by one cell
2. Move down by one cell
3. Move right by one cell
4. Move left by one cell
5. Stay in that cell
Each robot has to perform one of the above mentioned motion primitives at each time step. Whether the robots are allowed to perform a particular primitive at a give timestep also depends on other constraints.

The constraints for motion primitives will be:
$\forall r \in \{0, ..., R-1\}; \forall t \in \{0, ..., L-1\}; \forall x \in \{0, ..., N-1\}; \forall y \in \{0, ..., N-1\}$:

$$\begin{aligned} X(t+1, r, x, y) \Rightarrow & X(t, r, x, y) \vee X(t, r, x+1, y) \\ & \vee X(t, r, x, y+1) \vee X(t, r, x-1, y) \vee X(t, r, x, y-1) \end{aligned}$$

which in CNF is:

$$\neg X(t+1,r,x,y) \vee X(t,r,x,y) \vee X(t,r,x+1,y)$$
$$\vee X(t,r,x,y+1) \vee X(t,r,x-1,y) \vee X(t,r,x,y-1)$$

Number of clauses $= L \times R \times N^2$

The time complexity for forming the constraints for the motion primitives will be $O(L \times R \times n^2)$.

4.4 Constraints for obstacle avoidance:

Let the number of obstacles be G. Let (x^g_{obs}, y^g_{obs}) be the position of the gth obstacle where $g \in \{0, 1, ..., G-1\}$. Each robot has to avoid each obstacle at every timestep. The constraints for obstacle avoidance will be:
$\forall r \in \{0, ..., R-1\}; \forall t \in \{0, ..., L\}; \forall g \in \{0, ..., G-1\}$:

$$\neg X(t, r, x^g_{obs}, y^g_{obs})$$

Number of clauses $= (L+1) \times R \times G$ The time complexity for forming the constraints for obstacle avoidance will be $O(L \times R \times G)$.

4.5 Constraints for collision avoidance:

Each robot must avoid getting hit by another robot. There will be two types of collision.

4.5.1 Constraints for same space collision avoidance

SAME SPACE COLISSION AVOIDANCE:
$\forall r_1, r_2 \in \{0, ..., R-1\}; r_1 \neq r_2; \forall t \in \{0, ..., L\}; \forall x, y \in \{0, ..., N-1\}$:

$$\neg(X(t, r_1, x, y) \wedge X(t, r_2, x, y))$$

which in CNF is:

$$\neg X(t, r_1, x, y) \ \vee \ \neg X(t, r_2, x, y)$$

Number of clauses $= (L+1) \times \frac{R \times (R-1)}{2} \times N^2$

The time complexity for forming the constraints for same space collision avoidance will be $O(L \times R^2 \times n^2)$.

4.5.2 Constraints for headon collision avoidance

This refers to the collision type where two robots move into each other. The constraints for this type of collision avoidance will be: HEADON COLLISSION AVOIDANCE:
Horizontal:
$\forall r_1, r_2 \in \{0, ..., R-1\}; r_1 \neq r_2; \forall t \in \{0, ..., L-1\}; \forall x \in \{0, ..., N-2\}; y \in \{0, ..., N-1\}$:

$$\neg(X(t, r_1, x, y) \wedge X(t, r_2, x+1, y) \wedge X(t, r_1, x+1, y) \wedge X(t, r_2, x, y+1))$$

which in CNF is:

$$\neg X(t, r_1, x, y) \vee \neg X(t, r_2, x+1, y) \vee \neg X(t, r_1, x+1, y) \vee \neg X(t, r_2, x, y)$$

Number of clauses $= L \times (R^2 - R) \times (N^2 - N)$

Vertical:

$\forall r_1, r_2 \in \{0, ..., R-1\}; r_1 \neq r_2; \forall t \in \{0, ..., L-1\}; \forall y \in \{0, ..., N-2\}; x \in \{0, ..., N-1\}$:

$$\neg(X(t, r_1, x, y) \wedge X(t, r_2, x, y+1) \wedge X(t, r_1, x, y+1) \wedge X(t, r_2, x, y))$$

which in CNF is:

$$\neg X(t, r_1, x, y) \vee \neg X(t, r_2, x, y+1) \vee \neg X(t, r_1, x, y+1) \vee \neg X(t, r_2, x, y)$$

Number of clauses $= L \times (R^2 - R) \times (N^2 - N)$

The time complexity for forming the constraints for headon collision avoidance will be $O(L \times R^2 \times n^2)$.

5 Implementation

For the implementation we use the SAT solver tool called MiniSAT[4]. MiniSat is a minimalistic, open-source SAT solver. First we generate the constraints.Then, we transform the constraints into DIMACS format and feed it as input to the SAT solver. The output of SAT solver is processed to get the final result.

5.1 Finding the optimal *time length*

Initially we don't know the optimal value of $L(makespan)$ for which the given problem can be solved. But in order to generate the constraints we need to give a specific value of L to the constraint generator. To find the optimal *makespan*, the solver has to "guess" a value for the *makespan* and check whether the constraints are satisfiable. This is done iteratively until all the minimum makespan for which constraints are satisfiable is found.

5.1.1 Strategy 1: Linear Jumps

In this strategy we iterate the value of L over $1, 2, 3, ...$, linearly increasing the value, till we get the value of L for which the constraints are satisfiable.

5.1.2 Strategy 2: Exponential Jumps with Binary Search

Here, we increase the value of L exponentially$(1, 2, 4, 8, ...)$ each time and check if the constraints are satisfiable. Say we get satisfiability at $L = 2^a$. But, 2^a may not be the minimum value for which the constraints are satisfiable because we did not check the values between 2^{a-1} and 2^a. So, we execute a binary search between 2^{a-1} and 2^a till we reach the minimum value for which we get satisfiability for the constraints.

5.1.3 Strategy 1 v/s Strategy 2

We observe that Strategy 2 exceeds Strategy 1. The exact observations are mentioned in the Result section.

5.2 Optimizing constraint generation

We have noticed that while computing the optimal *time length*, in the iterations of L the constraint generation time is substantial as compared to the tool call time. So, in order to improve upon the total time we need to optimise the constraint generation. The idea is to reuse the constraints generated for the smaller values of L in the constraints for the larger values of L.

What we do is we store the constraints generated for each value of L ,say upto t, in a file. When we generate the constraints for $t+1$, we escape the long iterations of generating the constraints for $L = 1$ to $L = t$, and instead just catinate them from the stored files.

5.2.1 Optimised constraint generation v/s non-optimised generation

It is observed that optimized constraint generation facilitates in reducing the amount of tie taken for computation. The exact observations are in the Result section.

5.3 C++ Implementation

The later part of the project duration was mainly devoted to compiling the entire code(which was just a bash script invoking various executables), into a single C++ code. It is observed that the C++ implementation does orders of magnitude better than the bash-implementation.

6 Soundness

If we look into our formulation, we find that it is theoretically not sound. But the MiniSAT(core) tool's property of producing an assignment with least number of TRUTH assignments helps make our implementation work in practice. Nonetheless, we give the formulation for soundness too.

6.1 Strategy 1: Robot can be present only at a single place

These constraints assert that a robot can be present only in one of the positions in the grid at a given *timestep*. This constraint is mentioned in [10]
$\forall t; \forall r; \forall x_1, y_1, x_2, y_2; (x_1, y_1) \neg (x_2, y_2)$

$$\neg(X(t, r, x_1, y_1) \wedge X(t, r, x_2, y_2))$$

6.2 Strategy 2: Robot is allowed to choose only one motion primitive

Here we allow the robot to go for only one motion primitive. This is done by changing the motion primitive constraints by replacing the OR by XOR.

$$X(t+1, r, x, y) \Rightarrow X(t, r, x, y) \oplus X(t, r, x+1, y) \oplus X(t, r, x, y+1) \oplus X(t, r, x, y-1) \oplus X(t, r, x-1, y)$$

6.3 Strategy 1 v/s Strategy 2

It is observed that Strategy 2 beats Strategy 1, but both of them are beat by the "unsound" encoding.

7 Complex Motion Primitives

With the SAT-Based encoding it is now possible complex motion primitives into the sysytem. The description of the complex motion system can be found in [7].

7.1 Variables

$$X(t,r,x,y) = t \times R \times (N^2 + P + N^2) + r \times (N^2 + P + N^2) + x \times N + y + 1$$
$$Y(t,r,p) = t \times R \times (N^2 + P + N^2) + r \times (N^2 + P + N^2) + N^2 + p + 1$$
$$W(t,r,x,y) = t \times R \times (N^2 + P + N^2) + r \times (N^2 + P + N^2) + (N^2 + p) + x \times N + y + 1$$

$X(t,r,x,y)$ is TRUE if at t-th timestep r-th robot is at (x,y), else FALSE. Y(t,r,p) is true if at t-th timestep r-th robot chooses primitive p. W(t,r,x,y) is TRUE if at t-th timestep r-th robot has a swath on (x,y).

7.2 Initial and Final Configuration

The initial configuration and final configuration constraints are similar to those in the simple encoding discussed earlier. The only difference is we also have to assert some initial motion primitive(here we have taken that to be 0). $\forall r \in \{0,...,R-1\}; \forall x \in \{0,...,N-1\}; \forall y \in \{0,...,N-1\}$:

$$\begin{aligned}
&STARTPOSITION: \\
&X(0,r,x,y);\ if\ (x,y) = (x_0^r, y_0^r) \\
&\neg X(0,r,x,y);\ otherwise \\
&Y(0,r,0);
\end{aligned}$$

$$\begin{aligned}
&FINALPOSITION: \\
&X(L,r,x,y);\ if\ (x,y) = (x_L^r, y_L^r) \\
&\neg X(L,r,x,y);\ otherwise \\
&Y(L,r,p);\ if\ q_f(p) = 0
\end{aligned}$$

7.3 Primitive Selection

Only one motion primitive should be selected at a given *timestep*. $\forall t; \forall r; \forall p_1, p_2; p_1 \neq p_2$

$$\neg(Y(t,r,p_1) \wedge Y(t,r,p_2))$$

At least one motion primitive has to be selected at a given *timestep*. $\forall t; \forall r$

$$Y(t,r,0) \vee Y(t,r,1) \vee ... \vee Y(t,r,P-1)$$

The initial velocity of the selected motion primitive must agree with the final velocity of the motion primitive selected at the previous *timestep*. $\forall t; \forall r; \forall p$

$$Y(t+1,r,p) \Rightarrow \bigvee_{j:q_f(p_j)=q_i(p)} Y(t,r,p_j)$$

7.4 Obstacle Avoid

A robot must not move through any position that has an obstacle. $\forall t; \forall r$

$$\neg W(t, r, x_{obs}, y_{obs})$$

7.5 Collision Avoidance

A robot must not move trough a position that another robot is moving through at that *timestep*. $\forall t; \forall x; \forall y; \forall r_1; \forall r_2; r_1 \neq r_2$

$$\neg(W(t, r_1, x, y) \wedge W(t, r_2, x, y))$$

7.6 Motion Primitive

The next position of the robot should be decided by current position and the primitive that is selected. $\forall t; \forall r; \forall x; \forall y; \forall p$

$$X(t-1, r, x, y) \wedge Y(t, r, p) \Rightarrow X(t, r, x + posf.x(p), y + posf.y(p))$$

The swath of the next *timestep* also has to be updated. $\forall t; \forall r; \forall x; \forall y; \forall p; \forall swath$

$$X(t-1, r, x, y) \wedge Y(t, r, p) \Rightarrow W(t, r, x + swath.x(p), y + swath.y(p))$$

7.7 Soundness

The soundness constraints are the same as that in the case of the simple formulation. $\forall t; \forall r; \forall x_1, y_1, x_2, y_2; (x_1, y_1) \neq (x_2, y_2)$

$$\neg(X(t, r, x_1, y_1) \wedge X(t, r, x_2, y_2))$$

8 Cost Optimization

Upon investigating we find that ensuring miniumum *makespan* does not always guarantee minimum cost. It may so happen that the robot which can reach its final position in less than the time of optimal *makespan* selects a trajectory which is costlier. Thus we need to give some kind of incentive to the robot to not choose such trajectories. We can do this by adding soft constraints.

8.1 Soft Constraints for Cost Optimization

$\forall t \in \{0, ..., L-1\}; \forall r \in \{0, ..., R-1\}; \forall x_1, x_2, y_1, y_2 \in \{0, ..., N-1\}; x_2 = x_1 \pm 1\ OR\ y_2 = y_1 \pm 1$ but not both:

$$\neg(X(t, r, x_1, y_1) \wedge X(t+1, r, x_2, y_2))$$

which in CNF is:

$$\neg X(t, r, x_1, y_1) \vee \neg X(t+1, r, x_2, y_2)$$

Number of clauses $= 4 \times L \times R \times n^2$

These soft constraints can be solved using a MAX-SAT solver like [6]. We still search for the optimal *makespan* using the SAT-Based tool, but the final iteration will be carried out using the MAX-SAT Based tool.

9 Results

In this section we will be discussing the results that we obtain after experimenting. We will be comparing the results with the tool discussed in the paper *Automated composition of motion primitives for multi-robot systems from safe LTL specifications*[7]. The paper uses a SMT based tool.

9.1 Testing Workspace

We will be comparing our various implementations on the following workpaces. The figures are just scaled down representatives of the actual testing environment. The grid size, number of robots and number of obstacles can be changed accordingly.

9.1.1 Zigzag patterned workpace

In this workspace the obstacles are placed in such a way that the robots have to move in a zigzag path to reach their final positions. In the below grid, G represents an obsacle; k_i and k_f indicates the initial and final positions of the kth robot respectively.

.	.	1_f	2_f	3_f
.	G	G	G	G
.	.	.	.	.
G	G	G	G	.
1_i	2_i	3_i	.	.

9.1.2 Straight patterned workspace

In this workspace the obstacles are placed in such a way that the robots move in a relatively straight paths to reach their destinations, without having to cross the paths of other robots.

1_f	G	2_f	G	3_f	G
.	.	.	.	.	.
G	.	G	.	G	.
.	.	.	.	.	.
.	G	.	G	.	G
1_i	.	2_i	.	3_i	.

FIGURE: An example grid: 6×6; with 3 robots

9.2 Strategy for Finding Optimal *makespan*

Here we compare 2 strategies for finding optimal *makespan*.

1. linear search
2. exponential jumps followed by binary search

Grid Size(NN)	Number of Robots(R)	Strategy 1	Strategy 2
Zigzag patterned Workspace			
5×5	3	0.10s	0.07s
9×9	5	2.04s	0.77s
13×13	7	30.38s	6.21s
17×17	9	3m33s	33.74s
Straight patterned Workspace			
12×12	6	0.78s	0.74s
16×16	8	4.06s	2.22s
20×20	10	14.18s	6.09s
24×24	12	40.06s	21.63s

9.3 Optimized Constraint Generation

Here we compare optimized v/s non-optimized constraint generation.

Grid Size(NN)	Number of Robots(R)	Non-Optimized	Optimized
Zigzag patterned Workspace			
5×5	3	0.07s	0.13s
9×9	5	0.77s	0.44s
13×13	7	6.21s	2.55s
17×17	9	33.74s	13.24s
Straight patterned Workspace			
12×12	6	0.74s	0.43s
16×16	8	2.22s	0.99s
20×20	10	6.09s	2.31s
24×24	12	21.63s	8.56s

9.4 Comparision with SMT-Based tool

The SMT tool originally accommodated complex motion primitives. We modified the source files so that the tool produces results with the basic motion primitives that we use currently in this paper.
The tool needs to be given the *time length* and the upper bound of the total cost, along with the workspace, in order to produce output.
So, first we compute the optimal *time length* using exponential jumps followed by binary search for the value of *time length*. During this computation, we set the cost to be a safe upper bound. For *time length* L and total number of robots R, we set the total cost to be $(L+1) \times R$. This is followed by a binary search for cost between L and $(L+1) \times R$.
The final output of the SMT solver is finally processed to give the result.

9.5 SMT based tool v/s SAT based tool

We will be comparing the timings of the SMT based tool and the SAT based tool for three kinds of workspaces.

Grid Size ($N \times N$)	Number of Robots(R)	SMT-Based Solver	SAT-Based Solver
Zigzag patterned Workspace			
5×5	3	10.95s	0.29s
9×9	5	18m16s	1.5s
13×13	7	1hr+	11.8s
17×17	9	-	1m19s
Straight patterned Workspace			
4×4	2	0.97s	0.17
8×8	4	6.25s	0.3s
12×12	6	2m49s	1.5s
16×16	8	-	4.9s
Workspace mentioned in [7]			
19×19	4	2m13s	1.66s

The workspace used in the paper in [7] to demonstrate the working of the tool is a 19×19 grid with 4 robots and 92 obstacles.

9.6 Comparison of bash-implementation v/s C++-implementation

Grid Size ($N \times N$)	Number of Robots(R)	bash implementation	C++ implementation
Zigzag patterned Workspace			
5×5	3	0.147s	0.004s
9×9	5	0.976s	0.125s
13×13	7	8.526s	0.728s
17×17	9	1m21s	4.084s
Straight patterned Workspace			
8×8	4	0.260s	0.022s
12×12	6	1.117s	0.089s
16×16	8	3.060s	0.262s
20×20	10	10.948s	0.689s
24×24	12	51.234s	3.101s

9.7 Comparison of soundness strategies

Grid Size ($N \times N$)	Number of Robots(R)	simple encoding	sound encoding Strategy 1	sound encoding Strategy 2
Zigzag patterned Workspace				
5×5	3	0.004s	0.067s	0.006s
9×9	5	0.125s	0.222s	0.127s
13×13	7	0.728s	2.072s	0.972s
17×17	9	4.084s	18.849s	5.170s
Straight patterned Workspace				
8×8	4	0.022s	0.031s	0.015s
12×12	6	0.089s	0.255s	0.130s
16×16	8	0.262s	0.778s	0.359
20×20	10	0.689s	2.271s	0.909
24×24	12	3.101s	14.140s	3.931

9.8 Comparison of simple encoding v/s complex encoding

. Here we compare the simple encoding of the solver(in which one can encode only 5 simple motion primitives) against the complex encoding, in which one can encode complex encoding. It is found that the complex encoding solver does not scale up when the number of primitives increases. Nevertheless, we give the comparison in which simple primitives are encoded in the complex solver.

Grid Size ($N \times N$)	Number of Robots(R)	simple encoding	simple-primitives in complex encoding
Zigzag patterned Workspace			
5×5	3	0.004s	0.202s
9×9	5	0.125s	3.982s
13×13	7	0.728s	2m19s
Straight patterned Workspace			
8×8	4	0.022s	0.155s
12×12	6	0.089s	3.322s
16×16	8	0.262s	9.133s
20×20	10	0.689s	25.359s

9.9 Comparison of SAT v/s MAX-SAT

Grid Size ($N \times N$)	Number of Robots(R)	MAX-SAT-based solver	SAT-based solver
Zigzag patterned Workspace			
5×5	3	0.187s	0.147s
9×9	5	1.377s	0.976s
13×13	7	13.928s	8.526s
17×17	9	1m59s	1m21s
Straight patterned Workspace			
8×8	4	0.410s	0.260s
12×12	6	1.393s	1.117s
16×16	8	4.791s	3.060s
20×20	10	12.915s	10.948s
24×24	12	1m15s	51.234s

10 Further Work

We wish to extend the developed implementation to include more complex motion primitives for smoother trajectories. There are many suggested methods to make the constraint generation more efficient. One of them is multi threading. As the variables in the loops can be made independent of each other, multi-threading seems to be a viable choice for reducing the constraint generation time. We further hope to enhance the compuation speed using pre-computation of constraints. Further, the use of incremental SAT-Solving can be used for making the solver faster.

References

[1] Iman Haghighi, Sadra Sadraddini, and Calin Belta. *Robotic Swarm Control from Spatio-Temporal Specifications*. 2016 IEEE 55th Conference on Decision and Con-

trol (CDC), ARIA Resort & Casino, December 12-14, 2016, Las Vegas, USA.
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7799146

[2] Wikipedia *Boolean satisfiability problem.*
https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

[3] Caroline Perry. *A self-organizing thousand-robot swarm.*
https://www.seas.harvard.edu/news/2014/08/self-organizing-thousand-robot-swarm

[4] Niklas En, Niklas Srensson *The MiniSAT Page.*
http://minisat.se/Main.html

[5] Self Organizing Systems Research Group. *The Kilobot Project.*
https://www.eecs.harvard.edu/ssr/projects/progSA/kilobot.html

[6] Ruben Martins, Vasco Manquinho, Ins Lynce. *Open-WBO: An open source version of the MaxSAT solver WBO*
http://sat.inesc-id.pt/open-wbo/index.html

[7] Indranil Saha, Rattanachai Ramaithitima, Vijay Kumar *Automated composition of motion primitives for multi-robot systems from safe LTL specifications*
http://ieeexplore.ieee.org/document/6942758/

[8] Vasco Manquinho, Ruben Martins, Ins Lynce, Joo Marques-Silva, Jordi Planes *WBO - Weighted Boolean Optimization Solver*
http://sat.inesc-id.pt/wbo/

[9] Pavel Surynek. *An Optimization Variant of Multi-Robot Path Planning is Intractable*
https://pdfs.semanticscholar.org/6314/2e9b9dacd9e7552972374001210a215f666a.pdf

[10] Pavel Surynek. *A Simple Approach to Solving Cooperative Path-Finding as Propositional Satisfiability Works Well*
http://surynek.com/publications/files/SurynekPavel_Simple-Direct_PRICAI-2014.pdf

[11] Pavel Surynek, Eli Boyarski, Ariel Felner and Roni Stern. *Efficient SAT Approach to Multi-Agent Path Finding under the Sum of Costs Objective*
http://ktiml.mff.cuni.cz/ surynek/publications/Surynek-Felner-Stern-Boyarski_Efficient-Cost-Encoding_DMAP-2016.pdf