SAT Based Motion Planning of a Multi-Robot System

Manish Kumar Bera
Mentor: Dr. Indranil Saha

July 8, 2017

1 Introduction

1.1 Motivation

Swarm robotics is a field of multi-robotics in which large number of 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.

1.2 Objective

Robots need to move simultaneosly, to go from one position to another, to collectively form a new configuration. Describing complex spatial specifications for swarms is a non-trivial task. Our goal is to optimise the cost of 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. We will form boolean constraints that describe the workspace and the motion primitives, and feed it to the SAT solver, whose output will be processed further to get the final result.

2 Problem Statement

We are give 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 time and incurs minimum cost.

3 Methodology

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, and a robot will occupy only a single cell of the grid. In a timestep, each robot can either move to an adjacent cell, or may remain in the same cell. We will refer to a *move* as the

movement of a robot. A robot can move only to an adjacent cell upon executing a *move*. 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*. A robot may or may not execute a move on a *timestep*.

Let L be the total number of *timesteps* which the *swarm* takes to go from initial configuration to final configuration. 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 :

$$X_t = (x_t^1, x_t^2, x_t^3, ..., x_t^R)$$

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 $X_0 = (x_0^1, x_0^2, x_0^3, ..., x_0^R)$ represents the initial configuration of the *swarm*, and $X_L = (x_L^1, x_L^2, x_L^3, ..., x_L^R)$ 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 formulated into boolean expressions. For this we will have to define boolean variables.

4.1 Defining boolean variables:

I define $X(t, k, x, y)$ as a boolean variable s.t. $X(t, k, x, y)$ is $TRUE$ iff at time step t, the kth robot is at position (x, y). The number of boolean variables will be $(L+1) \times R \times n^2$

4.2 Constraints for initial and final states:

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

$$X(0, k, x, y) = TRUE;\ if\ x = x_0^k\ and\ y = y_0^k$$
$$= FALSE;\ otherwise$$

$$X(L,k,x,y) = TRUE;\ if\ x = x_L^k\ and\ y = y_L^k$$
$$= FALSE;\ otherwise$$

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:
$\forall t \in \{0,...,L\}; \forall k \in \{0,...,R-1\};$

$$X(t+1,k,x,y) \Rightarrow X(t,k,x,y) \vee X(t,k,x-1,y) \vee X(t,k,x,y-1) \vee X(t,k,x+1,y) \vee X(t,k,x,y+1)$$

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_{obs}^g, y_{obs}^g) 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:
$\forall t \in \{0,...,L\}; \forall k \in \{0,...,R-1\};$

$$X(t,k,x,y) = FALSE;\ if\ \exists g \in \{0,...,G-1\}\ s.t.\ x = x_{obs}^g\ and\ y = y_{obs}^g$$

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

This refers to the collision when two robots try to occupy the same cell of the grid. The constraints for this type of collision avoidance will be:
$\forall t \in \{0,1,...,L\};\ \forall k_1, k_2 \in \{0,1,...,R-1\}\ s.t.\ k_1 \neq k_2;\ \forall x, y \in \{0,1,...,n-1\};$

$$\neg(X(t,k_1,x,y) \wedge X(t,k_2,x,y))$$

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:
$\forall t \in \{0, 1, ..., L\};\ \forall k_1, k_2 \in \{0, 1, ..., R-1\}\ s.t.\ k_1 \neq k_2;\ \forall x, y \in \{0, 1, ..., n-1\};$

$$\neg((X(t, k_1, x, y) \wedge X(t+1, k_1, x+1, y)) \wedge (X(t, k_2, x+1, y) \wedge X(t+1, k_2, x, y)))\wedge$$

$$\neg((X(t, k_1, x, y) \wedge X(t+1, k_1, x, y+1)) \wedge (X(t, k_2, x, y+1) \wedge X(t+1, k_2, x, y)))$$

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

4.6 Soft constraints for cost optimization

For the optimisation of cost, we will need some additional constraints. Instead fomulating hard constraints, we formulate soft constraints, such that when the maximum number of those soft constraints are true, we will get the optimum cost. We will formulate each constraint such that it represents the negation of a particular motion. We assume that if the robot chooses not to move from his position in a time step then it incurs a cost of 0. But if he moves, in any of the four directions, then it incurs a cost of 1. The soft constraints for cost optimization will be:
$\forall t \in \{0, ..., L-1\};\ \forall k \in \{0, 1, ..., R-1\}\ \forall x_0, y_0 \in \{0, ..., n-1\}; x_1, y_1 \in \{0, ..., n-1\};\ x_1 = x_0 \pm 1\ OR\ y_1 = y_0 \pm 1$ *but not both*;

$$\neg(X(t, k, x_0, y_0) \wedge X(t+1, k, x_1, y_1))$$

Time complexity for forming these constraints is $O(L \times R \times n^2)$. These constraints are given to Open-WBO, along with the rest of the constraints, while solving for the solution with optimum cost. These constraints are not fed to the MiniSAT while computing the optimum value of *time length*.

5 Implementation

For the implementation we use two SAT solver tools, namely, MiniSAT[4] and Open-WBO[6]. MiniSat is a minimalistic, open-source SAT solver. Open-WBO is an open source MaxSAT solver initially based on WBO.
First we find the optimum *time length* for the problem using MiniSAT and then we find Open-WBO to find the optimal cost. First we generate the constraints using C++ executables. Then, we transform the constraints into CNF form 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(*time length*) 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.

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

Following are the timings of Strategy 1 v/s Strategy 2 in zigzag workspace(6.2.1):

Grid size($n \times n$)	Number of robots(R)	Timing for Strategy 1	Timing for Strategy 2
5×5	3	0.10 s	0.07 s
9×9	5	2.04 s	0.77 s
13×13	7	30.38 s	6.21 s
17×17	9	3 m 33 s	33.74 s

TABLE: Timings of Strategy 1 v/s Startegy 2 on zigzag workspace

Following are the timings of Strategy 1 v/s Strategy 2 in straight workspace(6.2.2):

Grid size($n \times n$)	Number of robots(R)	Timing for Strategy 1	Timing for Strategy 2
4×4	2	0.03 s	0.04 s
8×8	4	0.14 s	0.13 s
12×12	6	0.78 s	0.74 s
16×16	8	4.06 s	2.22 s
20×20	10	14.18 s	6.09 s
24×24	12	40.06 s	21.63 s

TABLE: Timings of SMT based v/s SAT based tool on straight workspace

5.2 Finding the solution with optimum cost

Once we find the minmum value of L for which the constraints are solvable, we need to find the solution with the optimum cost. This is achieved by feeding the Open-WBO MAX-SAT solver with the soft constraints we discussed earlier(4.6), along with the rest of the constraints. The result thus obtained will be optimised, both in terms of *time length* and cost.

5.3 Optimising constraint generation

It must be 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.

5.3.1 Optimised constraint generation v/s non-optimised generation

Following are the timings of Optimised constraint generation v/s non-optimised generation in zigzag workspace(6.2.1):

Grid size($n \times n$)	Number of robots(R)	Non-optimised generation	Optimised generation
5×5	3	0.07 s	0.13 s
9×9	5	0.77 s	0.44 s
13×13	7	6.21 s	2.55 s
17×17	9	33.74 s	13.24s

TABLE: Timings of Strategy 1 v/s Startegy 2 on zigzag workspace

Following are the timings of Optimised constraint generation v/s non-optimised generation in straight workspace(6.2.2):

Grid size($n \times n$)	Number of robots(R)	Non-optimised generation	Optimised generation
4×4	2	0.04 s	0.09 s
8×8	4	0.13 s	0.14 s
12×12	6	0.74 s	0.43 s
16×16	8	2.22 s	0.99 s
20×20	10	6.09 s	2.31 s
24×24	12	21.63 s	8.56 s

TABLE: Timings of SMT based v/s SAT based tool on straight workspace

6 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.

6.1 A brief discussion on the implementation of the SMT based tool

The SMT tool originally accomodated 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.

6.2 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.

6.2.1 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	.	.

FIGURE: An example grid: 5×5; with 3 robots

For the above mentioned type of workspace, following are the timings of the SMT based tool and SAT based tool.

Grid size($n \times n$)	Number of robots(R)	Timing for SMT based solver	Timing for SAT based solver
5×5	3	10.95 s	0.29 s
9×9	5	18 m 16 s	1.5 s
13×13	7	1 hr +	11.8 s
17×17	9	-	1 m 19 s

TABLE: Timings of SMT based v/s SAT based tool on zigzag workspace

6.2.2 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

For the above mentioned type of workspace, following are the timings of the SMT based tool and SAT based tool.

Grid size($n \times n$)	Number of robots(R)	Timing for SMT based solver	Timing for SAT based solver
4×4	2	0.97 s	0.17 s
8×8	4	6.25 s	0.3 s
12×12	6	2 m 49 s	1.5 s
16×16	8	-	4.9 s
20×20	10	-	12.9 s
24×24	12	-	53.54 s

TABLE: Timings of SMT based v/s SAT based tool on straight workspace

6.2.3 Workspace used in the aforementioned paper[7]

This is the workspace used in the paper in [7] to demonstrate the working of the tool. It is a 19×19 grid with 4 robots and 92 obstacles. The following are the timings for both the tools

Grid size($n \times n$)	Number of robots(R)	Timing for SMT based solver	Timing for SAT based solver
19×19	4	2 m 13 s	1.66 s

TABLE: Timings of SMT based v/s SAT based tool on example workspace in [7]

7 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.

References

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