Scalable Path and Time Coordination for Robot Formation[†]

Hoon Sung Chwa, Andrii Shyshkalov, Kilho Lee, and Insik Shin

Dept. of Computer Science, KAIST, South Korea
insik.shin@cs.kaist.ac.kr

***Abstract*—In this paper, we consider several CPS challenges (e.g., responsiveness, scalability, adaptability) in multi-robot formation. In general, the response time of multi-robot formation task involves two parts: the computation time for path and time coordination to avoid any collision among robots and the actuation time for the control of the robots to actually move to their destinations. In terms of responsiveness, a shorter response time provides a higher quality of responsiveness. However, it is complicated to reduce the response time since reducing computation time and reducing robot actuation time are conflicting objectives, and such a trade-off varies over environment. We present a scalable optimization framework that explores such a trade-off dynamically and exploits it in a feedback manner to find efficient trajectory schedules. Our simulation results show that our framework successfully finds a shorter response time by adapting to various environments compared to a commercial optimization tool, and it is scalable for a large number of robots.**

I. Introduction

In the last decade, multi-robot formation [2], [3] has experienced a rise of attention along with the advances in multi-robot systems [4]. As the applications of the multi-robot formation become broader, several CPS challenges (e.g., responsiveness, scalability, adaptability) arise let alone typical challenges (e.g., collision avoidance, formation transformation). For example, multiple navigation robots with rescue missions not only need to arrive at target location safely but also need to arrive as soon as possible. In this case, it is necessary to reduce the response time to improve responsiveness. Generally, the response time of multi-robot formation task involves two parts: the computation time for the assignment of robots to goal locations (i.e., trajectory scheduling for given paths) to visualize given input patterns and the actuation time for the control of the robots to actually establish the goal locations (i.e., actual robot movement). It is straightforward that the better trajectory scheduling can introduce the shorter robots traveling time. However, it often takes longer to find the better trajectory scheduling, so it is complicated to reduce the response time by taking into consideration of the trade-off between the computation time and the actuation time.

In this paper, we present a "scalable" optimization framework that supports "responsiveness" for a multi-robot formation. Generally, *trajectory scheduling* problem is to find timing synchronizations for robots to avoid collision when paths are given. In the framework, trajectory scheduling aiming at minimizing the actuation time is the key problem since it significantly affects the response time. However, finding an optimal solution (i.e., minimizing the maximum robot traveling time in our case) to trajectory scheduling is known as NP-hard [5]. Hence, it is necessary to develop an approximation algorithm that can explore a trade-off between reducing computation time and reducing traveling time effectively for sub-optimal solutions.

Figure 1 shows an example of such a trade-off from an iterative approximation algorithm. It is shown in the figure that the computation time is accumulated as the number of iteration steps grows, but the actuation time becomes decreased since the algorithm works better with repetitions. Here, an optimal point, which minimizes the response time, can be found around 60 iterations. It is generally complicated to find such an optimal point in advance since it is often sophisticated to predict how the robot actuation time is varied by environmental parameters (i.e., the number of robots, robot kinematics, robot deployment) on which the maximum robot traveling time significantly depends.

Several existing studies [6], [7], [8], [9] have been focusing on computation of optimal solution or sub-optimal solution with reduced complexity. However, little works have been considering on the trade-off between the computation time and the actuation time for response time minimization. This paper presents a scalable optimization framework that explores such a trade-off dynamically and exploits it in a feedback manner to find an optimal trajectory scheduling which minimizes response time.

Our framework consists of two components: (1) *scalable priority assignment policy*, and (2) *priority-based trajectory scheduling* for collision-free trajectory generation. A priority ordering is defined as a priority ordered list of all participating robots, and it determines the collision-free trajectories of individual robots in a decreasing order of priority. Hence, the priority ordering directly indicates which robots to have higher priorities (i.e., which robots to move first) in potential collision resolution. Similar to prioritized planning [8], [9], the proposed priority-based trajectory scheduling is *scalable*. Different priority orderings generally produce different actuation times, and thus many trials of different priority orderings increase a chance to

[†]A preliminary version of this paper was published in the proceedings of the 1st International Workshop on Large-Scale Cyber-Physical Systems (LCPS 2011) [1].

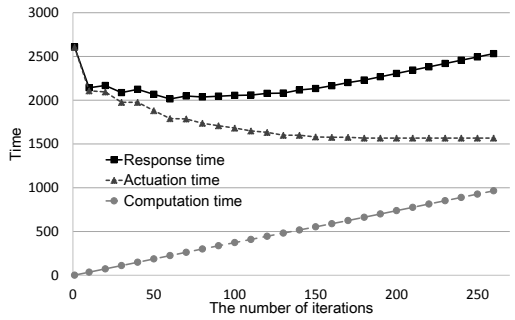

Figure 1. A trade-off between the actuation time and the computation time according to the number of iterations

decrease actuation time at the expense of increasing computation time. Aiming at minimizing the response time, the proposed framework effectively balances such a trade-off in a feedback manner. By the use of a priority ordering as a unit of computation in trajectory scheduling, the framework *adaptively* controls the trade-off by deciding whether to try a new priority ordering or not.

We evaluate the proposed framework from the viewpoints of scalability, responsiveness and adaptability. Our simulation results show that the commercial optimization tool is not scalable, then the computation time increases rapidly with a growing number of robots. However, the proposed framework successfully finds a sub-optimal response time in a scalable manner by dynamically adjusting the number of iterations in consideration of the trade-off between actuation and computation time.

Organization. The reminder of this paper is organized as follows. Section II introduces our system model, assumptions and notations. Section III proposes a scalable response time minimization framework. Section IV develops a priority-based trajectory scheduling algorithm that efficiently finds an actuation time under a given priority ordering of robots. Section V discusses a priority assignment policy that determines how many priority orderings and which priority orderings should be tried for minimizing the response time. Section VI evaluates the proposed framework. Section VII finally concludes this paper.

II. System Model

This paper considers N robots on a two-dimensional plane $\mathbb{Z}$. Each robot B_i is configured as

- Robot B_i has the shape of a circle with *radius* r_i.
- $\mathcal{P}_i$, a path of B_i, is specified as a continuous sequence of two-dimensional points from its starting location $L_{i,S}$ to its ending location $L_{i,E}$.
- There are two kinematic constraints which limit B_i's maximum speed (by V_i^{max}) and acceleration (by A_i^{max}) respectively. In addition, all robots are not allowed to move backwards.

Each point on a path $\mathcal{P}_i$ can be uniquely determined by *traveling distance* d which B_i has traveled along its path from $L_{i,S}$ to the point. For example, when B_i has the same traveling distance with its *path length* (denoted by $\mathbf{D}_i$), which is the maximum traveling distance for B_i, it is located on the ending location $L_{i,E}$. Then, we can represent the path $\mathcal{P}_i$ as a function of the traveling distance $\mathbf{l}_i(d)$, where $d \in [0, \mathbf{D}_i]$.

The *traveling time* of each robot B_i (denoted by $\mathbf{R}_i$) is defined as the time at which the robot arrives at its ending location $L_{i,E}$. For convenience, we assume that all robots start their movement at a time instant 0. Then, for each time t between 0 and $\mathbf{R}_i$, we define $\mathbf{d}_i(t)$ as the distance robot B_i has traveled during the time interval $[0, t]$.

Based on the above two functions, a trajectory of each robot B_i, i.e., its location at a given time t (denoted by $\mathbf{L}_i(t)$), can be obtained as follows:

$$\mathbf{L}_i = \mathbf{l}_i \circ \mathbf{d}_i : [0, R_i] \rightarrow \mathcal{P}_i.$$

III. Scalable Response Time Minimization Framework

In this section, we formally describe the major problems of this paper. We then propose the scalable trajectory scheduling framework, which is designed to address them.

A. Problem Statement

A key goal of this paper is to determine collision-free trajectories of robots for doing user-interactive functionalities such as multi-robot formation. To enhance user experience, it is required to perform the interactive tasks as fast as possible. In other words, it is essential to minimize the system response time $\mathbf{R}_T$. From user's perspective, $\mathbf{R}_T$ includes the following two kinds of time; 1) the computation time $\mathbf{R}_C$ for determining trajectories of all robots, 2) the actuation time $\mathbf{R}_A$ for moving all robots to their destination. We then define the system response time minimization problem as the problem of determining the location of each robot for every time instant as follows:

> When paths $\{\mathcal{P}_i\}_{1 \le i \le N}$ are given, we seek to determine trajectories $\{\mathbf{L}_i(t)\}_{1 \le i \le N}$, such that $\mathbf{R}_C + \mathbf{R}_A$ is minimized, and there is no collision, subject to $\{V_i^{max}, A_i^{max}\}_{1 \le i \le N}$.

It is difficult to solve the above problem analytically since it considers both $\mathbf{R}_C$ and $\mathbf{R}_A$ simultaneously, while $\mathbf{R}_C$ is not well defined as a closed-form formula. Furthermore, it is challenging to define an accurate model for a trade-off between $\mathbf{R}_C$ and $\mathbf{R}_A$ at design time. This entails run-time estimation for such a trade-off.

B. The Proposed Framework

We propose a framework to dynamically make a decision for trajectory scheduling by exploiting the trade-off between

$\mathbf{R}_C$ and $\mathbf{R}_A$ at run-time. It achieves to minimize the system response time based on two key techniques; *the priority-based trajectory scheduling (PTS) algorithm* and *the scalable priority assignment policy*.

The PTS algorithm effectively reduces $\mathbf{R}_A$ for a given priority ordering, which is a basic unit of computation, by scheduling the trajectory of individual robots one by one in the order of their priorities. For such an individual schedule, we develop *an individual robot trajectory generation algorithm* that produces the minimum traveling time of a robot, when the higher-priority robots' trajectories are given, within polynomial time of the number of robots.

Once the PTS algorithm is finished for a given priority ordering, the scalable priority assignment policy estimates an impact of additional scheduling computation on $\mathbf{R}_T$ and determines how many priority orderings to try. If the additional computation is expected to reduce $\mathbf{R}_T$, it re-schedules trajectories of robots with a new priority ordering. We propose a heuristic to choose a "good" priority ordering that produces shorter $\mathbf{R}_A$.

IV. Priority-based Trajectory Scheduling

This section presents a Priority-based Trajectory Scheduling (PTS) algorithm to compute the minimum system traveling time ($\mathbf{R}_A$) for a given priority ordering. We present a structure of the PTS algorithm in Section IV-A and an individual robot trajectory generation algorithm in Section IV-B as a sub-routine of the PTS algorithm.

A. Priority-based Trajectory Scheduling Algorithm

Algorithm 1 shows the PTS algorithm to decide trajectories of all robots with the minimum system traveling time for a given priority ordering of robots. According to the priority ordering, it sequentially generates trajectory of an individual robot when higher-priority robots are determined (Lines 2-7). In Line 5, a sub-routine generates collision-free trajectory for an individual robot. Its details are explained in Section IV-B.

Algorithm 1 PTS algorithm

Input: a priority ordering of robots Q
Output: trajectory for the priority ordering
1: $W := \emptyset$
2: **while** $Q \neq \emptyset$ **do**
3: $B_i :=$ the head of Q
4: Remove B_i from Q
5: Individual-robot-trajectory-algorithm(B_i, W)
6: Insert B_i into W
7: **end while**

B. Individual Robot Trajectory Generation Algorithm

As a sub-routine of the PTS algorithm, we present an algorithm to generate collision-free trajectory with the minimum traveling time for an individual robot when the trajectories of higher-priority robots are given.

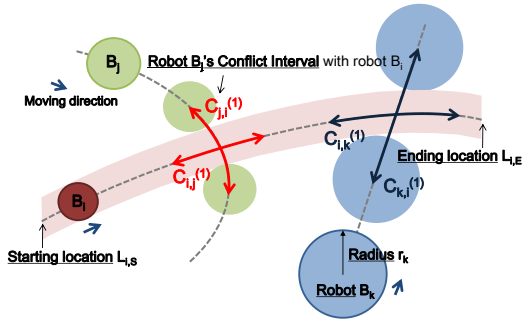

Figure 2. Conflict zone: a dotted line is a robot's path, and a solid one is a conflict zone

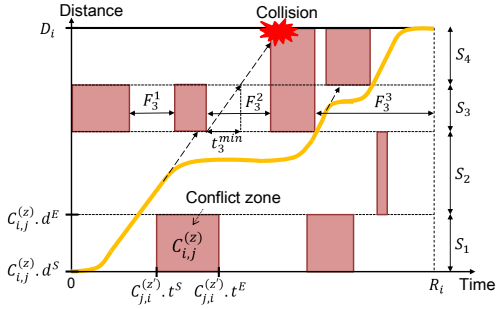

Figure 3. Finding collision-free trajectory for robot B_i with minimum $\mathbf{R}_i$

To analyze collision-free trajectory, we introduce a notion of *potential conflict zone* of robots, which is the minimum geometric region to be checked for collision occurrence (Figure 2).

We can consider robot B_i's traveling distance $\mathbf{d}_i(t)$ for time t on two-dimensional plane with time as X-axis and traveling distance as Y-axis, as shown in Figure 3. The function $\mathbf{d}_i(t)$ is a continuous curve from $(0, 0)$ to $(\mathbf{R}_i, \mathbf{D}_i)$.

We now analyze collision-free trajectory of robot B_i. For a given $\mathbf{D}_i$, naturally, the steeper the function $\mathbf{d}_i(t)$ is, the lower $\mathbf{R}_i$ is. However, we need to consider some constraints. First, the trajectory from this curve must satisfy the kinematic constraints of the robot (Figure 4). Second, the curve must be monotonically non-decreasing from the constraint that any robot cannot move backwards. Finally, the trajectory from this curve must be collision-free.

Consider the third constraint with conflict zone. Conflict

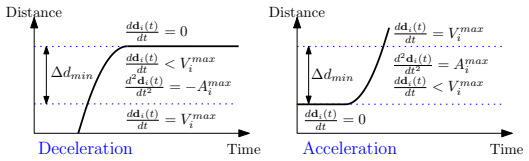

Figure 4. Kinematic constraints impose a limit on the minimal traveling distance Δd_{min} that robot must travel to accelerate from 0 to the maximum speed or to decelerate back to 0.

zone of robot B_i corresponding to robot B_j (denoted by $\mathsf{C}_{i,j}$) is formally defined as a set of traveling distances d_i in $[0, \mathbf{D}_i]$ at which Euclidean distance between B_i and B_j is less than or equal to the sum of two robots' radius as follows:

$$\begin{aligned}\mathsf{C}_{i,j} =&\{d_i \in [0, \mathbf{D}_i] \mid \exists d_j \in [0, \mathbf{D}_j] \text{ such that} \\ &\|\mathbf{l}_i(d_i) - \mathbf{l}_j(d_j)\| \leq r_i + r_j\}. \end{aligned} \quad (1)$$

We can find each member of conflict zone $\mathsf{C}_{i,j}$. We denote the z-th member of $\mathsf{C}_{i,j}$ by $\mathsf{C}_{i,j}^{(z)}$, which is represented by a rectangle. Its projection onto the traveling distance axis is an interval $\left[\mathsf{C}_{i,j}^{(z)}.d^S, \mathsf{C}_{i,j}^{(z)}.d^E\right]$, and that onto the time axis is an interval of time (denoted by $\left[\mathsf{C}_{j,i}^{(z')}.t^S, \mathsf{C}_{j,i}^{(z')}.t^E\right]$) during which the other robot B_j may collide with robot B_i. Once the trajectory of the robot B_j is known, collision zone $\mathsf{C}_{i,j}$ can be computed as shown in Figure 2.

Now, we can generate collision-free trajectory for the robot. We assume that conflict zones are given. If two robots collide, then it must satisfy that one robot's $\mathbf{d}_i(t)$ must intersect at least one conflict zone's rectangle. Therefore, in order to avoid collision for robot or B_i, it suffices to make sure that the curve $\mathbf{d}_i(t)$ does not intersect any of collision zone rectangles. Then, finding a collision-free trajectory for a robot becomes a geometric problem of finding $\mathbf{d}_i(t)$ with minimum $\mathbf{R_i}$ among all continuous curves subject to the limitations of no intersection with its collision zone rectangles and kinematic constraints as shown in Figure 3.

Algorithm 2 describes the individual robot trajectory algorithm, which takes robot B_i and a set of other robots with given trajectories (denoted as W) as input parameters and returns collision-free trajectory of robot B_i. In Line 1, it divides the robot's path into a set of segments (denoted as S) as shown in Figure 3 to make search for the curve efficient. For each segment $S_m \in S$, the minimum traveling time (denoted as t_m^{min}) is pre-computed according to kinematic constraints and a set of free time intervals (denoted by F_m) consisting of intervals F_m^k between collision zones is computed (Line 2-4). In Line 5, we execute a depth first search (DFS) on the first segment's earliest free time interval. DFS recursively tries to find the earliest free interval in the next segment that the curve can "reach" from the current free interval until the last segment's end is reached. At the end of the DFS process, we can find the time it takes for the robot B_i to pass through each segment S_m, which can compute the whole curve $\mathbf{d}_i(t)$.

Algorithm 2 Individual Robot Trajectory Algorithm

Input: B_i, W
Output: collision-free trajectory for robot B_i
1: $\mathbf{S} \leftarrow$ Split $[0, \mathbf{D}_i]$ into segments
2: **for all** $S_m \in \mathbf{S}$ **do**
3: $F_m \leftarrow$ FreeIntervals(S_m, W)
4: **end for**
5: DFS(F_1^1, 0, S_1)

Complexity. The worst-case time complexity of the PTS algorithm depends on two parameters: the number of robots N and the maximum number of conflict zones per robot M. In the individual robot trajectory optimization, computation of free time intervals takes $O(M^2 \cdot logM)$. Therefore, time complexity of PTS algorithm (Algorithm 1) is $O(N \cdot M^2 \cdot \log M)$.

V. Scalable Priority Assignment Policy

As explained in the previous section the PTS algorithm produces the possible minimum value of $\mathbf{R}_A$ for a given single priority ordering. To minimize $\mathbf{R}_A$ with the PTS algorithm, we have to iterate all the priority orderings which can induce large computation time, $\mathbf{R}_C$. As the number of priority orderings increases, $\mathbf{R}_C$ also increases in proportion. Therefore, to achieve our goal of minimizing $\mathbf{R}_T$, we should selectively deal with a subset of priority orderings only. In order to construct the subset properly, we need to address two questions: (i) how many priority orderings are included in the subset, and (ii) which priority orderings are added to the subset. This section explains how to resolve those two questions to adaptively produce the priority orderings resulting in the least $\mathbf{R}_T$.

A. The number of priority orderings to consider

To accomplish our goal of minimizing $\mathbf{R}_T$, we have to keep trying an additional priority ordering until the new iteration is not beneficial to the goal anymore. In other words, we should attempt a next priority ordering if the additional computation time is less than the expected decrement of $\mathbf{R}_A$. We arrange this scheme in the following lemma.

Lemma 1: When the k-th iteration step is done, the system response time for k+1-th step can be reduced compared to that for k-th step if the following inequality holds:

$$\mathbf{R}_C^{(k+1)} - \mathbf{R}_C^{(k)} < \mathbf{R}_A^{(k)} - \mathbf{R}_A^{(k+1)}, \quad (2)$$

where $\mathbf{R}_C^{(k)}$ indicates the total computation time for k iterations, and $\mathbf{R}_A^{(k)}$ indicates the minimum system traveling time among k iterations. With the definitions, $\mathbf{R}_T^{(k)}$ is then defined to be $\mathbf{R}_C^{(k)} + \mathbf{R}_A^{(k)}$.

To apply Lemma 1, estimations of $\mathbf{R}_C^{(k+1)}$ and $\mathbf{R}_A^{(k+1)}$ are required. It is relatively easy to approximate $\mathbf{R}_C^{(k+1)}$, since the computation time of the PTS algorithm is stable over different priority orderings. Hence, $\mathbf{R}_C^{(k+1)}$ is estimated to be k+1 times as much as the average computation time of the PTS algorithm. On the other hand, estimation of $\mathbf{R}_A^{(k+1)}$ is more difficult since it can vary over different input priority orderings. Since Fig 1 shows that $\mathbf{R}_A^{(k)}$ has an exponentially decreasing trend as the number of iteration steps increases, we hypothesize that the $\mathbf{R}_A^{(k)}$ can be approximated as an exponential function of the variable k, the number of steps.

We apply a linear regression model to approximate such an exponential form of $\mathbf{R}_A^{(k)}$ as follows:

$$\mathbf{x} = a + b\mathbf{k}, \tag{3}$$

where $\mathbf{x}$ is $\log(\mathbf{R}_A^{(k)})$, and both a and b are unknown coefficients that we should find the estimation formula. We have to determine such a formula with applying linear regression model based on $\{\mathbf{R}_A^{(j)}\}$ for all $j \leq k$. Then, from estimated values of $\mathbf{R}_A^{(k+1)}$ and $\mathbf{R}_C^{(k+1)}$, Lemma 1 decides whether we should try one more priority ordering or not.

B. Selection of priority ordering in each iteration

In the previous sub-section we introduce a method that finds the number of iterations in order to minimize $\mathbf{R}_T$. However, the output value $\mathbf{R}_T$ for a given number of iterations (or equivalently given $\mathbf{R}_C$) depends on a set of tried priority orderings. Hence, we propose a heuristic to choose better priority orderings that produce a shorter $\mathbf{R}_A$. First, we explain how to choose an initial priority ordering and then how to evolve the initial priority ordering based on the result of the initial priority ordering.

First step. The system traveling time $\mathbf{R}_A$ depends on the largest traveling time among all robots, and therefore we can reduce $\mathbf{R}_A$ by decreasing the traveling time of the robot with the maximum $\mathbf{R}_i$. However, we cannot identify which robot has the maximum $\mathbf{R}_i$ unless a specific priority ordering is given. This entails the need of expecting the traveling time of each robot without any priority ordering. As a heuristic, we estimate traveling time of robot B_i (denoted by R_i) as the summation of the static traveling time and the waiting time. The static traveling time is defined as the amount of time consumed when robot B_i travels along its path at earliest without delay for collision avoidance, and the waiting time is the sum of the amount of time consumed by all the other robots $\{B_j\}$ to pass the conflict zones $\{\mathsf{C}_{j,i}\}$ with their maximum velocity.

Considering the kinematic constraints, the static traveling time of robot B_i (denoted by R_i^s) can be simply calculated as follows:

$$R_i^s = \frac{\mathbf{D}_i}{V_i^{max}} + \frac{V_i^{max}}{A_i^{max}}, \tag{4}$$

where $\mathbf{D}_i$ is the total traveling distance of B_i, and V_i^{max} and A_i^{max} are the maximum velocity and the maximum acceleration of robot R_i, respectively.

The waiting time of robot B_i at $\mathsf{C}_{i,j}^{(z)}$ when robot B_j passes a corresponding element of its collision zone $\mathsf{C}_{j,i}^{(z')}$ (denoted by $R_i^w(z)$) is simply calculated as follows:

$$R_i^w(z) = \frac{\mathsf{C}_{j,i}^{(z')}.d^E - \mathsf{C}_{j,i}^{(z')}.d^S}{V_j^{max}}. \tag{5}$$

Then, the estimated traveling time of robot R_i is given as follows:

$$R_i = R_i^s + \sum_{j=1}^{N} \sum_{\mathsf{C}_{j,i}^{(z')} \in \mathsf{C}_{j,i}} R_i^w(z). \tag{6}$$

We sort robots by descending order of the estimated traveling time, and we use this priority ordering as an initial priority ordering.

Iteration. We first run the PTS algorithm, then we can find the robot that determines the system traveling time. Here, we denote such a robot by B_{max}, and B_{max} decides $\mathbf{R}_A$. So, B_{max} needs to be assigned a new priority to reduce its traveling time. We first identify a set of robots (denoted as $H(B_{max})$) that have higher priority than B_{max} and share the conflict zones with B_{max}. After that, we set the priority of B_{max} to be the highest priority among robots in the set, and decrease the priority of robots in the set by one. By iterating this priority ordering evolution, we can have successive priority orderings, and two characteristics: (i) if the size of $H(B_{max})$ is zero, the current priority ordering is one of the best priority orderings in terms of minimizing $\mathbf{R}_A$, and (ii) if the current priority ordering is equal to one of the priority orderings that appear before, our priority ordering evolution cannot find shorter $\mathbf{R}_A$.

With Lemma 1, there properties can be also the halting condition of next iteration. We immediately stop the next if any of the properties is satisfied.

VI. Evaluation

In this section, we present performance and scalability evaluation of our priority-based trajectory scheduling framework. We first present simulation environment and then discuss simulation results.

A. Simulation Environment

We compare two algorithms with our framework: mixed integer linear programming, the priority-based trajectory scheduling with an initial priority ordering only. These algorithms are respectively annotated as MILP and STSF(1). Our scalable trajectory scheduling framework is annotated as STSF(n).

For MILP implementation, we use MILP formulation presented in [7]. For the MILP solver, we use a commercial optimization tool, CPLEX [10] in JAVA.

We generate 50 test sets for each combination of the number of robots. We use a commercial computer with 4 core CPU with 4 GB memory. Each computation has been done in one thread.

B. Simulation Results

For evaluating the benefits of using our framework, STSF(n), we conducted a rigorous simulation by varying the number of robots. Figure 5 shows the results of this simulations. As the number of robots is increasing, the response times of all algorithms are also increasing. One of the most noticeable feature is that response time of

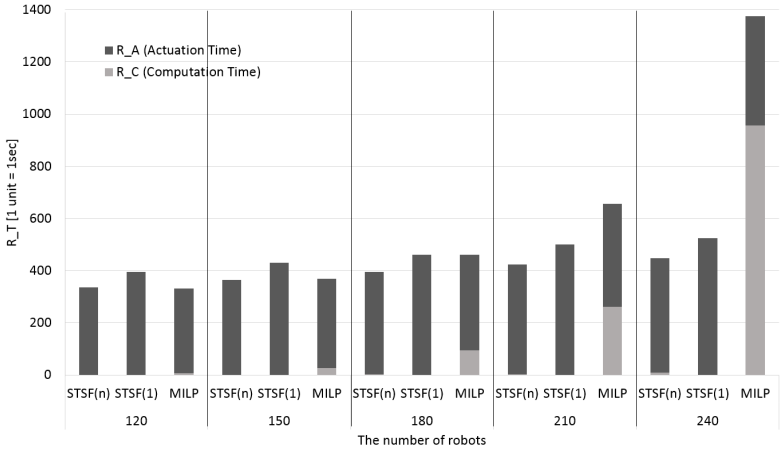

Figure 5. The system response time. For these test cases, environment size is 50m $\times$ 50m, average path length is 25m, average radius is 30cm, average speed is 15cm/sec.

MILP increases sharply from 150 robots because computation time becomes the major part of the system response time. For example, the portion of computation time in the system response time increases from 2% with 120 robots to 70% with 240 robots. This indicates that when the system response time is important as in multi-robot formation, MILP may not be a desirable solution though it produces an optimal actuation time. On the other hand, our framework (STSF(1) and STSF(n)) yields the system response time linearly proportional to the number of robots. This is because the total computation time takes a small part of response time. For example, the total computation time of STSF(n) is only 8 seconds (2% of the system response time) for 240 robots which is less than 1% of MILP.

Now we compare the response times of STSF(1), and STSF(n). Compared STSF(n) to STSF(1), response time of STSF(n) is much smaller than STSF(1). This shows that even though STSF(n) takes more time for computation, STSF(n) can find more efficient way for trajectory scheduling. Therefore, STSF(n) is suitable for a large number of robots; it is scalable in terms of response time.

These simulation results indicate that our STSF(n) algorithm can be successfully used for multi-robot formation since it approximately minimizes system response time adapting to environments, and it is scalable for a large number of robots.

VII. Conclusion

In this paper, we develop the scalable trajectory scheduling framework that minimizes the system response time to support multi-robot formation with responsiveness, scalability, and adaptability. Our aim is to get a smaller response time by using the trade-off between the system traveling time and the total computation time at run time. Our framework dynamically iterates the priority-based trajectory scheduling algorithm for different priority orderings to reduce the response time efficiently. Our simulation results show that the framework is suitable for multi-robot formation in that it can produce a near optimal solution in a scalable manner by adapting well to various environments.

Acknowledgement

This work was supported in part by BSRP (NRF-2010-0006650, NRF-2012R1A1A1014930), NCRC (2012-0000980), KEIT (2011-10041313), RCPSR (14-824-09-013) and KIAT (M002300089) funded by the Korea Government (MEST/MSIP/MOTIE).

References

[1] H. S. Chwa, A. Shyshkalov, J. Lee, H. Back, and I. Shin, "Adaptive trajectory coordination for scalable multiple robot control," in *LCPS*, 2011.

[2] M. M. Zavlanos and G. J. Pappas, "Dynamic assignment in distributed motion planning with local coordination," *IEEE Transaction on Robotics*, vol. 24, no. 1, pp. 1173–1178, 2008.

[3] *The Flyfire Project at MIT*. - http://senseable.mit.edu/flyfire/.

[4] J. Alonso-Mora, A. Breitenmoser, M. Rufli, R. Siegwart, and P. Beardsley, "Multi-robot system for artistic pattern formation," in *ICRA*, 2011.

[5] S. Akella and S. Hutchinson, "Coordinating the motions of multiple robots with specified trajectories," in *ICRA*, 2002.

[6] T. Simeon, S. Leroy, and J.-P. Laumond, "Path coordination for multiple mobile robots: A resolution-complete algorithm," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 1, pp. 42–49, 2002.

[7] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," *The International Journal of Robotics Research*, vol. 24, no. 4, pp. 295–310, 2005.

[8] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 4, pp. 477–521, 1987.

[9] J. P. van den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *IROS*, 2005.

[10] *IBM ILOG CPLEX V12.1 User's Manual for CPLEX*. IBM ILOG CPLEX, 2009.