To appear in *AAMAS Autonomous Robots and Multirobot Systems Workshop* (ARMS 2014), Paris, France, May 2014.

SCRAM: Scalable Collision-avoiding Role Assignment with Minimal-makespan for Formational Positioning

Patrick MacAlpine†, Eric Price‡, and Peter Stone†

†Department of Computer Science, The University of Texas at Austin
{patmac,pstone}@cs.utexas.edu
‡Computer Science and Artificial Intelligence Laboratory, MIT
ecprice@mit.edu

Abstract. Teams of mobile robots often need to divide up subtasks efficiently. In spatial domains, a key criterion for doing so may depend on distances between robots and the subtasks' locations. This paper considers a specific such criterion, namely how to assign interchangeable robots to a set of target goal locations such that the makespan (time for all robots to reach their target locations) is minimized while also preventing collisions among robots. We present a scalable multiagent dynamic role assignment system, used for formational positioning of mobile robots, known as SCRAM (Scalable Collision-avoiding Role Assignment with Minimal-makespan). SCRAM uses a graph theoretic approach to map robots to target goal locations such that our objectives for both minimizing the makespan and avoiding robot collisions are met. The system was originally designed to allow for decentralized coordination among physically realistic simulated humanoid soccer playing robots in the partially observable, non-deterministic, noisy, dynamic, and limited communication setting of the RoboCup 3D simulation league. In its current form, SCRAM generalizes well to many realistic and real-world multiagent systems, and scales to thousands of robots, as role assignment algorithms run in polynomial time.

1 Introduction

Coordinated movement among autonomous mobile robots is an important research area with many applications such as search and rescue and warehouse operations. Research within this space spans multiple topics including role assignment (deciding which robot should move to which position or role) [7, 12, 16, 21, 27], path planning [20, 25], and collision avoidance [11, 24].

The work in this paper focuses on role assignment—specifically tackling the problem of assigning homogeneous mobile robots to move to a set of fixed target positions such that a robot is present at every target position in as little time as possible. Path planning and collision avoidance issues are addressed during role assignment, as mappings of robots to target positions operate under the constraint that no robots collide.

Previous work on assigning robots to target positions has focused on minimizing the sum of distances all robots must travel which is the well known *assignment problem* [23]. Work related to our own [5] has added the constraint of avoiding collisions among agents to the *assignment problem* by requiring that all paths from agents to target positions be disjoint. Our work differs as we minimize the *makespan* (time for all robots to reach goal positions) instead of the sum of distances traveled.

Minimizing the makespan is a decisive factor in performance when robots are moving to target positions to complete a shared task where all robots must be in place before the task can be completed and/or started. Such tasks include those requiring robots be synchronized when they start jobs at their target positions (e.g. mobile robots assuming necessary positions on an assembly line) and scenarios for which the bottleneck is the time it takes for the last robot to get to its target position (e.g. warehouse robots delivering items for an order to be shipped and mobile robots being used as pixels to display an image [3]).

We refer to our role assignment and positioning system as SCRAM (Scalable Collision-avoiding Role Assignment with Minimal-makespan). It provides a collision free mapping of robots to target positions, minimizes the makespan, and scales to thousands of robots. The primary contributions of this paper are a complete specification of SCRAM,[1] as well as a thorough theoretical and empirical analysis of the role assignment problem, with application to the RoboCup robot soccer domain and potentially far beyond.

The remainder of the paper is organized as follows. Section 2 provides a formulation of the role assignment problem we are solving. Two role assignment functions, as well as algorithms implementing them, are presented in Section 3, with an empirical evaluation of them given in Section 4. Section 5 provides case studies of complete SCRAM role assignment positioning systems used within the RoboCup 2D and 3D simulation domains, and Section 6 concludes.

2 Role Assignment Problem

Let there be n homogeneous mobile robots with current positions $A := \{a_1, ..., a_n\}$, and we want to assign the robots to move to n specified target goal positions or roles $P := \{p_1, ..., p_n\}$ such that the time for robots to have reached every goal position is minimized under the constraint that no robots collide with each other. Figure 1 illustrates an example problem with six robots and target positions. This problem can be thought of as finding a perfect matching M^* within the set of perfect matchings $\mathbb{M}$ of a weighted bipartite graph $G := (A, P, E)$ that meets the above criteria with the weight for each edge in E being the Euclidean distance between associated robot and target positions.

Similar to work by Broucke [5], we model robots as point masses with zero width. Additionally, we make two more assumptions. First, no two robots and no

[1] Videos of SCRAM in action, as well as C++ implementations of the role assignment algorithms, can be found at `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2013/html/scram.html`

two target positions occupy the same position. Second, we assume that all robots move toward fixed target positions along a straight line at the same constant speed.

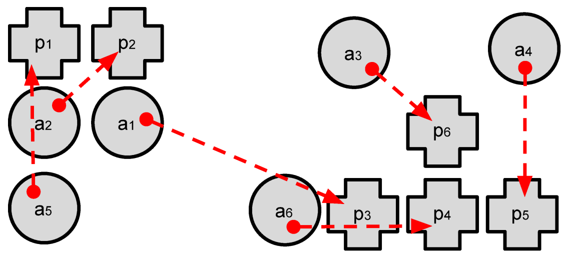

Fig. 1: Role assignment problem where we want to assign robots (circles) $\{a_1,\ldots,a_6\}$ to target positions (crosses) $\{p_1,\ldots,p_6\}$. Dashed arrows show solution.

We call a role assignment *CM valid* (Collision-avoiding with Minimal-makespan) if it satisfies two properties:

1. *Minimizing longest distance* - M^* minimizes the longest distance from a robot to target, with respect to all possible mappings. A valid mapping for the problem shown in Figure 1 would not include $a_2 \rightarrow p_5$ (the longest distance between a robot and target). Instead a valid mapping includes $a_1 \rightarrow p_3$ (the minimal longest distance any robot travels across all mappings).
2. *Avoiding collisions* - robots do not collide as they move to their assigned positions. In Figure 1 a mapping including both $a_1 \rightarrow p_1$ and $a_2 \rightarrow p_2$ would be invalid as it would cause robots a_1 and a_2 to collide.

A third desirable property, although not necessary for a role assignment function f to be *CM valid*, is the following:

3. *Dynamically consistent* - Given a *fixed* set of target positions, if f outputs a mapping M of robots to targets at time T, then f outputs M for every time $t > T$ as robots move to the targets specified by M.

The first two properties come directly from the definition of the role assignment problem. The third property guarantees that once a role assignment function f outputs a mapping, f will always output that same mapping as long as there is no change in the target positions. This guarantee is desirable as otherwise robots might unduly thrash between roles thus impeding progress. In the following section we construct *CM valid* role assignment functions.

3 Role Assignment Functions

The following subsections present two *CM valid* assignment functions for the role assignment problem detailed in Section 2. Algorithmic implementations of the functions and analysis of their time and space complexities are also given.

3.1 Minimum Maximal Distance Recursive (MMDR) Role Assignment Function

One potential role assignment function is to find a mapping of robots to target positions which recursively minimizes the maximum distance that any robot travels. We refer to this as this the Minimum Maximal Distance Recursive (MMDR) role assignment function. It is also known as the *lexicographic bottleneck assignment problem* [23]. In this section we first analyze properties of MMDR, and then identify algorithms to compute MMDR.

Let $\mathbb{M}$ be the set of all one-to-one mappings between robots and roles. If there are n robots and n target role positions, then there are $n!$ possible mappings $M \in \mathbb{M}$. Let the *cost* of a mapping M be the n-tuple of distances from each robot to its target, sorted in decreasing order. We can then sort all the $n!$ possible mappings based on their costs, where comparing two costs is done lexicographically. Sorted costs of mappings for a small example are shown in Figure 2.

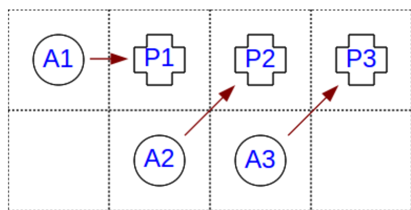

Fig. 2: Lowest lexicographical cost (shown with arrows) to highest cost ordering of mappings from robots (A1,A2,A3) to role positions (P1,P2,P3). Each row represents the cost of a single mapping.
1: $\sqrt{2}$ (A2→P2), $\sqrt{2}$ (A3→P3), 1 (A1→P1)
2: 2 (A1→P2), $\sqrt{2}$ (A3→P3), 1 (A2→P1)
3: $\sqrt{5}$ (A2→P3), 1 (A1→P1), 1 (A3→P2)
4: $\sqrt{5}$ (A2→P3), 2 (A1→P2), $\sqrt{2}$ (A3→P1)
5: 3 (A1→P3), 1 (A2→P1), 1 (A3→P2)
6: 3 (A1→P3), $\sqrt{2}$ (A2→P2), $\sqrt{2}$ (A3→P1)

Denote the role assignment function that always outputs the lexicographically smallest cost mapping as MMDR. Here we provide an informal proof sketch that MMDR is *CM valid* and is also dynamically consistent; we provide a longer, more thorough derivation in Appendix A and B.

Theorem 1. *MMDR is CM valid and dynamically consistent.*

MMDR minimizes the longest distance (Property 1) as the lexicographical ordering of distance tuples sorted in descending order ensures this. If two robots in a mapping are to collide (Property 2) it can be shown, through the triangle inequality, that MMDR will find a lower cost mapping as switching the two robots' targets reduces the maximum distance either must travel. Finally, as we assume all robots move toward their targets at the same constant rate, the distance between a robot and any target will not decrease any faster than the distance between any robot and the target that robot is assigned to. This observation provides dynamic consistency (Property 3) by preserving the lowest lexicographical cost ordering of a MMDR mapping across all timesteps.

Dynamic Programming Algorithm for MMDR

Theorem 2. *Let A and P be sets of n agents and positions respectively. Denote the mapping $M := MMDR(A, P)$. Let M_0 be a subset of M that maps a subset of agents $A_0 \subset A$ to a subset of positions $P_0 \subset P$. Then M_0 is also the mapping returned by $MMDR(A_0, P_0)$.*

A key recursive property of MMDR that allows us to exploit dynamic programming is expressed in Theorem 2. This property stems from the fact that if a subset $M_0 \subset M$ can be replaced by a lower cost mapping $M_0{}'$, then we can replace M_0 with $M_0{}'$ in M to obtain a lower cost MMDR mapping M'. The savings from using dynamic programming comes from only evaluating mappings whose subset mappings are also returned by MMDR. This is accomplished in Algorithm 1 by iteratively building up optimal mappings for position sets from $\{p_1\}$ to $\{p_1, ..., p_n\}$ and using optimal mappings of $k-1$ agents to positions $\{p_1, ..., p_{k-1}\}$ (line 6) as a base when constructing each new mapping of k agents to positions $\{p_1, ..., p_k\}$ (line 7). Then, we save the lowest cost mapping for the current set of k agents to positions $\{p_1, ..., p_k\}$ (line 8).

Algorithm 1 MMDR Dynamic Programming Implementation

Input:
$Robots := \{a_1, ..., a_n\}$; $Positions := \{p_1, ..., p_n\}$

```
1: HashMap bestRoleMap := ∅
2: for k := 1 to n do
3:     for each a ∈ Robots do
4:         S := C(n−1, k−1) sets of k − 1 agents from Robots \ {a}
5:         for each s ∈ S do
6:             Mapping m_0 := bestRoleMap[s]
7:             Mapping m := (a → p_k) ∪ m_o
8:             bestRoleMap[a ∪ s] := mincost(m, bestRoleMap[a ∪ s])
9: return bestRoleMap[Agents]
```

During the kth iteration of the dynamic programming process to find a mapping for n agents, where k is the current number of positions that agents are being mapped to, each agent is sequentially assigned to the kth position and then all possible subsets of the other $n-1$ agents are assigned to positions 1 to $k-1$ based on computed optimal mappings to the first $k-1$ positions from the previous iteration of the algorithm. These assignments result in a total of $\binom{n-1}{k-1}$ agent subset mapping combinations to be evaluated for mappings of each agent assigned to the kth position. The total number of mappings evaluated by the dynamic programming algorithm for each of the n agents across all n iterations of dynamic programming can be written as

$$\sum_{k=1}^{n} n \binom{n-1}{k-1} = n \sum_{k=0}^{n-1} \binom{n-1}{k} = n2^{n-1}$$

As it takes $O(n)$ time to compare the cost of two mappings the time complexity of the algorithm is $O(n^2 2^{n-1})$. Additionally, for the kth iteration of the dynamic programming process we must store the $\binom{n}{k}$ best mappings for the next iteration to use. As the maximum number of mappings will be stored at the $\frac{n}{2}th$ iteration, and each mapping takes $\mathrm{O}(n)$ space, the space complexity of the algorithm is $\mathrm{O}(n\binom{n}{n/2})$.

$\mathbf{O(n^5)}$ Polynomial Time Algorithm for MMDR An issue with the dynamic programming algorithm presented in Section 3.1 is that its time complexity is exponential and thus does not scale well to large values of n. We can compute the MMDR role assignment function in polynomial time, however, by transforming MMDR into the *assignment problem* (finding a perfect matching in a bipartite graph that minimizes the sum of edge weights) which is solvable by the Hungarian algorithm [15] in $\mathrm{O}(n^3)$ time.

Lemma 1. *Denote* $W_n := \{w_0, ..., w_n\}$ *where* $w_i := 2^i$. *Then* $\forall W \in \mathcal{P}(W_{n-1}) : w_n > \sum W$.

To transform MMDR into the *assignment problem* we modify the weights of the edges of our bipartite graph to be a set of values such that the weight of any edge e is greater than the sum of weights of all edges with weight values less than that of e. A key insight into this transformation is expressed in Lemma 1. By sorting all edges in ascending order by distance, and then relabeling edge weights to be the value 2^i where i is the index of an edge in this sorted list, the sum of all edge weights of shorter distance edges will be less than any sum of edge weights with a longer edge. Solutions to the *assignment problem* return lowest cost MMDR mappings as the sum of modified weights of any mapping with a higher cost is greater than that of a lower cost mapping.

Algorithm 2 gives a polynomial time solution for computing MMDR. First, weights are sorted in ascending order of distance (line 1). Next, edge weights are transformed into appropriate values for the assignment problem as expressed in Lemma 1 (line 8). Finally, the re-weighted edges are given as input into the Hungarian algorithm which returns the lowest cost MMDR mapping (line 10). Time complexity is dominated by the $\mathrm{O}(n^3)$ Hungarian algorithm. Note that our transformed edge weights, represented as bit vectors with the *ith* bit of a 2^i value turned on, are of size n^2. The Hungarian algorithm must do comparisons of these weights and thus the time complexity of Algorithm 2 is $\mathrm{O}(n^5)$. As our implementation of the Hungarian algorithm requires us to store length n lists of size n^2 transformed weights, Algorithm 2 has a space complexity of $\mathrm{O}(n^3)$.

There exists previous work in modifying edge weights to transform the *lexicographic bottleneck assignment problem* into the *assignment problem*. For cases in which there are n^2 edges, with each having a unique cost, a higher complexity $\mathrm{O}(n^5 logn)$ algorithm exists [6]. Work by Croce et al. [9] changes edge weights into weight vectors of length n before solving the *assignment problem* and has the same time complexity as our method of $\mathrm{O}(n^5)$. However, on modern computer architectures Algorithm 2 is more efficient as we represent edge weights as bit vectors instead of vectors of integers. The compact format of bit vectors allows for integer operations to be performed on w bits in parallel where w is the

Algorithm 2 MMDR O(n^5) Polynomial Time Implementation

Input:
$Robots := \{a_1, ..., a_n\}$; $Positions := \{p_1, ..., p_n\}$
$Edges := \{\overline{a_1p_1}, \overline{a_1p_2}, ..., \overline{a_np_n}\}$; $|\overline{a_ip_j}| :=$ `euclideanDist`(a_i, p_j)

$edgesSorted :=$ `sortAscendingDist`$(Edges)$
$lastDistance := -1$
$rank, currentIndex := 0$
for each $e \in edgesSorted$ **do**
  if $|e| > lastDistance$ **then**
    $rank := currentIndex$
  $lastDistance := |e|$
  $|e| := 2^{rank}$
  $currentIndex := currentIndex + 1$
return `hungarianAlg`$(edgesSorted)$

size of a processor's maximum word length. This parallelism reduces the running time of Algorithm 2 by a factor of w (e.g. a factor of 64 on a 64-bit architecture).

O(n^4) Polynomial Time Algorithm for MMDR Another approach to compute MMDR, presented by Sokkalingam and Aneja [26], and detailed in Algorithm 3, alternates between solving the *bottleneck assignment problem* [23] (finding the smallest maximum edge in a perfect matching) and a 0-1 cost version of the *assignment problem*.

Algorithm 3 MMDR O(n^4) Polynomial Time Implementation

Input:
$Robots := \{a_1, ..., a_n\}$; $Positions := \{p_1, ..., p_n\}$
$Edges := \{\overrightarrow{a_1p_1}, \overrightarrow{a_1p_2}, ..., \overrightarrow{a_np_n}\}$; $|\overrightarrow{a_ip_j}| :=$ `euclideanDist`(a_i, p_j)

1: **function** GETTIGHTEDGES($poten$)
2:   **return** $e_{a,p} \in Edges$, s.t. $poten(a) + poten(p) = cost(e_{a,p})$

3: $numEdgesLeft := n$
4: **loop**
5:   $minLongestEdge :=$ `getMinimalMaxEdgeInPerfectMatching`$(Edges)$
6:   $\forall e \in Edges \begin{cases} |e| < |minLongestEdge| : & cost(e) := 0 \\ |e| = |minLongestEdge| : & cost(e) := 1 \\ |e| > |minLongestEdge| : & cost(e) := \infty \end{cases}$
7:   $\{matching, poten\} :=$ `hungarianAlgWithEdgeCosts`$(Edges)$
8:   $numLongestEdges := \sum_{e \in matching} cost(e)$
9:   $numEdgesLeft := numEdgesLeft - numLongestEdges$
10:   **if** $numEdgesLeft = 0$ **then**
11:     **return** $matching$
12:   $Edges :=$ `getTightEdges`$(poten)$
13:   $\forall e \in Edges$, s.t. $|e| = |minLongestEdge| : |e| := -1$

At every iteration of Algorithm 3 solving the *bottleneck assignment problem* (line 5 which is implemented by Algorithm 4 discussed later in this section) returns the current largest edge weight value in the MMDR mapping. Next solving the *assignment problem* using the Hungarian algorithm (line 7), with 0-1 edge costs as specified in line 6, returns a mapping whose sum of costs (line 8) reveals the number of edges of this weight in the MMDR mapping.

At the same time the Hungarian algorithm naturally computes a potential function *poten* over the set of vertices in the bipartite graph such that $\forall e_{a,p} \in Edges : poten(a) + poten(p) <= cost(e_{a,p})$. It is revealed by Sokkalingam and Aneja [26] that all perfect matchings of the subset of tight edges (defined as edges for which $poten(a) + poten(p) = cost(e_{a,p})$) contain exactly *numLongestEdges* edges of length $|minLongestEdge|$. Given this knowledge we remove all non-tight edges from consideration in the MMDR mapping (line 12). The reduction to tight edges, and reducing the weight of edges of length $|minLongestEdge|$ (line 13), results in subsequent solutions of the *bottleneck assignment problem* revealing the next largest edge weight value in the MMDR mapping as every perfect matching will have exactly *numLongestEdges* edges of length $|minLongestEdge|$. We learn the weight of *numLongestEdges* edges in the MMDR mapping during every iteration of Algorithm 3, and after determining the weights for n edges, the solution returned by the Hungarian algorithm is the MMDR mapping (line 11).

Algorithm 4 finds the minimal maximum edge in a perfect matching by incrementally adding edges to the graph in order of increasing distance from the list of edges sorted in ascending order of weight (line 23). It interleaves adding edges (line 30) with running the Ford-Fulkerson algorithm [10] for finding a maximum cardinality (number of edges) matching. Ford-Fulkerson (implemented with the `flood`, `resetFlood`, and `reversePath` functions) works by using a breath-first search to find augmenting paths from a robot to a target.

Algorithm 4 starts with a graph with the empty set of edges *allowedEdges* (line 1), and whenever the breadth-first search of the Ford-Fulkerson algorithm is unable to find a path from a robot to a target, we add an edge to the graph (line 30) and continue the breadth-first search. At the point when we find n paths from robots to target, the last edge we added is the minimal maximum edge for a perfect matching.

An important factor for performance in Algorithm 4 is that we can pick up the Ford-Fulkerson breadth-first search where it left off after adding an edge as any nodes previously reachable in the graph remain reachable. Because we do not lose state in each breadth-first search, each breadth-first search takes $O(E)$ time. Thus the total time for running Algorithm 4 to find a perfect matching with the minimum maximal edge length is $O(nE)$ which is less than the $O(n^3)$ time complexity of the Hungarian algorithm.

We determine at least one new minimal maximum edge in a perfect matching during every iteration of the loop in Algorithm 3. Thus no more than n instances of both the Hungarian algorithm and Algorithm 4 need to be computed. As the $O(n^3)$ time complexity of the Hungarian algorithm dominates Algorithm 3's loop, the time complexity of Algorithm 3 is $O(n^4)$. The breadth-first search of Ford-Fulkerson in Algorithm 4 gives a space complexity of $O(n^2)$.

3.2 Minimum Maximal Distance + Minimum Sum Distance^2 (MMD+MSD^2) Role Assignment Function

Another role assignment function to map robots to target goal positions is one which minimizes the maximum distance any robot has to travel (but not re-

Algorithm 4 Minimal-maximum Edge Perfect Matching

```
Input:
    Robots := {a_1, ..., a_n}; Positions := {p_1, ..., p_n}
    Edges := {a_1p_1→, a_1p_2→, ..., a_np_n→}; |a_ip_j→| := euclideanDist(a_i,p_j)

matchedRobots, allowedEdges := {}

function FLOOD(curNode, prevNode)
    curNode.visited := true
    curNode.previous := prevNode
    if curNode ∈ Positions and ∄ e ∈ allowedEdges, s.t. e.start = curNode then
        return currentNode
    for each e ∈ allowedEdges, s.t. (e.start = curNode andnot e.end.visited) do
        val := flood(e.end, e.start)
        if val ≠ ∅ then
            return val
    return ∅

function RESETFLOOD
    for each node ∈ {Robots ∪ Positions} do
        node.visited := false
        node.previous := ∅
    for each a ∈ {Robots \ matchedRobots} do
        flood(a, ∅)

function REVERSEPATH(node)
    while node.previous ≠ ∅ do
        reverseEdgeDirection((node, node.previous)→)
        node := node.previous
    return node

edgeQ := sortAscendingDist(Edges)
longestEdge := ∅
for match := 1 to n do
    resetFlood()
    matchedPosition := ∅
    while matchedPosition = ∅ do
        longestEdge := edgeQ.pop()
        allowedEdges ← longestEdge
        matchedPosition := flood(longestEdge.end, longestEdge.start)
    matchedRobot := reversePath(matchedPosition)
    matchedRobots ← matchedRobot
return longestEdge
```

cursively as done by MMDR in Section 3.1), after which it minimizes the sum of distances squared that all robots travel. We call this the Minimum Maximal Distance + Minimum Sum Distance^2 (MMD+MSD^2) role assignment function. Specifically we want to find a perfect matching M^* such that

$$\mathbb{M}'' := \{X \in \mathbb{M} \mid \|X\|_\infty = \min_{M \in \mathbb{M}}(\|M\|_\infty)\} \tag{1}$$

$$M^* := \operatorname*{argmin}_{M \in \mathbb{M}''}(\|M\|_2^2) \tag{2}$$

Here we provide an informal proof sketch that MMD+MSD^2 is a *CM valid* role assignment; we provide a longer, more thorough derivation in Appendix A.

Theorem 3. *MMD+MSD^2 is CM valid.*

By only considering the set of perfect matchings $\mathbb{M}''$ with minimal longest edges (equation 1) we are minimizing the longest distance any robot must travel

(Property 1). If two robots in a mapping are to collide (Property 2), it can be shown, through the triangle inequality, that MMD+MSD^2 will find a lower cost mapping as switching the two robots' targets reduces, but never increases, the distance that one or both must travel thereby reducing the sum of distances squared (equation 2) and the longest distance (equation 1).

Unlike MMDR, MMD+MSD^2 is not dynamically consistent because distances squared do not decrease at a constant rate, but in fact decrease at faster rates for larger distances, as robots move toward targets (e.g. the difference in distance squared as a robot moves from 5 meters to 4 meters from a target ($5^2 - 4^2 = 9$) is greater than the difference moving from 4 meters to 3 meters ($4^2 - 3^2 = 7$)). This lack of a constant rate of decrease for distances squared allows for squared distances between a robot and targets it is not assigned to travel toward to decrease faster than the squared distance between a robot and the target it is assigned to. The sum of distances squared for non-MMD+MSD^2 mappings can thus become less than the current MMD+MSD^2 mapping as robots travel to their targets. An example where MMD+MSD^2 is not dynamically consistent is provided in Appendix B.

Polynomial Time Algorithm for MMD+MSD^2 Algorithm 5 implements MMD+MSD^2 by first finding a perfect matching with the smallest maximum edge (line 1) which is computed by Algorithm 4 presented earlier in Section 3.1. We then create a set of *minimalEdges* consisting of all edges with length less than or equal to the longest edge in our perfect matching (line 2) and use it as input to the Hungarian algorithm (line 3). Note that edge weights are their distances squared and thus the Hungarian algorithm minimizes the sum of distances squared. As all edges greater in length than the minimal maximum edge in a perfect matching are removed before running the Hungarian algorithm, the maximum distance any robot travels is also minimized.

Algorithm 5 MMD+MSD^2 O(n^3) Polynomial Time Implementation

Input:
$Robots := \{a_1, ..., a_n\}$; $Positions := \{p_1, ..., p_n\}$
$Edges := \{\overrightarrow{a_1p_1}, \overrightarrow{a_1p_2}, ..., \overrightarrow{a_np_n}\}$; $|\overrightarrow{a_ip_j}| :=$ `euclideanDist`$(a_i,p_j)^2$

1: $longestEdge :=$ `getMinimalMaxEdgeInPerfectMatching`$(Edges)$
2: $minimalEdges := e \in Edges$, s.t. $|e| \leq |longestEdge|$
3: **return** `hungarianAlg`$(minimalEdges)$

The O(n^3) time complexity of the Hungarian algorithm dominates Algorithm 4 and thus the time complexity of Algorithm 5 is O(n^3). The breadth-first search of Ford-Fulkerson in Algorithm 4 gives a space complexity of O(n^2).

Table 1: Time and space complexities of role assignment algorithms.

Algorithm	Time Complexity	Space Complexity
MMD+MSD^2	O(n^3)	O(n^2)
MMDR O(n^4)	O(n^4)	O(n^2)
MMDR O(n^5)	O(n^5)	O(n^3)
MMDR dyna	O($n^2 2^{(n-1)}$)	O($n\binom{n}{n/2}$)
brute force	O($n!n$)	O(n)

Table 2: Average running time in milliseconds for role assignment algorithms and different values of n as measured on an Intel(R) Xeon(R) CPU E31270 @ 3.40GHz.

Algorithm	$n = 10$	$n = 20$	$n = 100$	$n = 300$	$n = 10^3$	$n = 10^4$
MMD+MSD^2	0.016	0.062	1.82	21.2	351.3	115006
MMDR O(n^4)	0.049	0.262	17.95	403.0	14483	—
MMDR O(n^5)	0.022	0.214	306.4	40502	—	—
MMDR dyn.	0.555	2040	—	—	—	—
brute force	317.5	—	—	—	—	—

4 Role Assignment Function and Algorithm Analysis

To evaluate role assignment algorithms, we generated mapping scenarios for n robots and targets. Both robots and targets were assigned random integer value positions on a two dimensional square grid with sides of length n^2. Table 2 shows the average run-time of the role assignment algorithms for different values of n. By far the slowest was the brute force method of evaluating all $n!$ possible mapping permutations of robots to targets. Not surprisingly the fastest was MMD+MSD^2 which has the lowest time complexity as shown in Table 1. MMD+MSD^2's relatively low time and space complexities allow it to scale well such that it is able to compute the mapping for 1000 robots in less than half a second, and a mapping for 10,000 robots in less than two minutes. The polynomial time implementations of MMDR scale well to 100s of robots and are much faster than the dynamic programming implementation of MMDR. The O(n^4) implementation of MMDR scales to 1000 robots and is faster than the O(n^5) implementation except for smaller ($n <= 20$) inputs where it takes a little longer due to the extra computations needed in its main loop.

Table 3: Role assignment function properties discussed in Section 2.

Assignment Function	Minimizes Makespan	Avoids Collisions	Dynamically Consistent
MMD+MSD^2	Yes	Yes	No
MMDR	Yes	Yes	Yes
MSD^2	No	Yes	No
MSD	No	No	No
Random	No	No	No
Greedy	No	No	No

In Table 4 we compare MMDR and MMD+MSD^2 against the following role assignment functions when assigning 10 robots to targets on a 100 X 100 grid.

MSD Minimize sum of distances between robots and targets.
MSD^2 Minimize sum of distances squared between robots and targets.
Greedy Assign robots to targets in order of shortest distances.
Random Random assignment of robots to targets.

Table 4: Average makespan, average distance, and distance standard deviation computed across a million assignments of 10 robots to 10 targets on a 100 X 100 grid.

Assignment Function	Average Makespan	Average Distance	Distance StdDev
MMD+MSD^2	45.79	27.38	10.00
MMDR	45.79	28.02	9.30
MSD^2	48.42	26.33	10.38
MSD	55.63	25.86	12.67
Random	90.78	52.14	19.38
Greedy	81.73	28.66	18.95

Both MMDR and MMD+MSD^2 have the same lowest average makespan for they are defined so as to minimize the makespan. As can be seen in Table 3 none of the other role assignment functions are *CM valid* as they fail to minimize the makespan (further analysis of how other role assignment functions fail to hold properties necessary for SCRAM assignment functions is provided in Appendix C). MMDR is the only dynamically consistent function of the ones we compare.

Average distance is not something SCRAM role assignment functions explicitly attempt to minimize. However, this metric can be useful if robots exhaust a shared resource such as fuel when moving. MSD by definition minimizes the average distance and thus represents the best possible value for this metric. MMDR and MMD+MSD^2 both have average distance values close to that of MSD with MMD+MSD^2 being slightly better than MMDR. A third metric is distance standard deviation which is useful if there is a preference for having robots travel similar distances (e.g. wanting to have equal wear and tear across robots). MMDR has the best value for this with MMD+MSD^2 being second best. While they are the simplest to implement and compute, the Random and Greedy role assignment functions do poorly across all metrics. MMDR and MMD+MSD^2, on the other hand, do well across all metrics in Table 4.

5 RoboCup Case Studies

While the empirical analysis presented in Section 4 of grid-based role assignment instances is good for isolating different properties of role assignment functions, our original motivation is the performance seen in richer domains involving dynamic role assignment such as robot soccer. RoboCup[2] robot soccer has served as an excellent research domain for autonomous robots and multiagent systems over the past decade and a half. In this domain, teams of autonomous robots compete with each other in complex, real-time, noisy and dynamic environments. One often thinks of the soccer teamwork challenge as being about where the player with the ball should pass or dribble, but at least as important is where the robots position themselves when they *do not* have the ball [14]. Positioning the players in a formation requires the robots to coordinate with each other and determine where each robot should position itself on the field. As RoboCup necessitates that robots coordinate movement as a team to be successful, it provides an ideal testbed for SCRAM. In the following sections, we provide case studies and analysis of SCRAM systems employed in two different RoboCup leagues and

[2] `http://www.robocup.org/`

used by our champion RoboCup team UTAustinVilla. In Section 5.1, we focus on the deployment of a SCRAM system in the RoboCup 3D simulation league. Section 5.2 evaluates SCRAM's usage in the RoboCup 2D simulation league.

5.1 RoboCup 3D Simulation

The RoboCup 3D simulation environment is based on SimSpark,[3] a generic physical multiagent system simulator. SimSpark uses the Open Dynamics Engine[4] (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. ODE also provides support for the modeling of advanced motorized hinge joints used in the humanoid agents.

The robot agents in the simulation are modeled after the Aldebaran Nao robot.[5] Visual information about the environment is given to robots through noisy measurements of the distance and angle to objects within a restricted vision cone (120°). Additionally, robots can communicate with each other every other simulation cycle (40 ms) by sending messages limited to 20 bytes. Games consist of two five minute halves of 11vs11 robot teams playing soccer.

In UTAustinVilla's positioning system players' positions are determined in three steps. First, a full team formation is computed using Delaunay triangulation [2] based on set offset positions from the ball (formations used are provided in [18]). Second, each player computes an assignment of players to positions in this formation according to its own view of the world using the MMD+MSD^2 role assignment function. An important factor in any SCRAM-based system is that robots have reasonably accurate knowledge of where all robots are currently located. We use robot communication to share and synchronize robot world models as discussed in [17]. For the third and final step a voting coordination mechanism detailed in [17] synchronizes players' computed assignments. While outside the scope of this paper, others have used auction algorithms [4] for distributed computation and synchronization. Additionally, if needed, robots employ a local collision avoidance system discussed in [17].

Table 5: Average goal difference across 1000 games when playing against the top three teams at the RoboCup 2013 competition (with standard error shown in parentheses).

Function	1. Apollo3d	2. UTAustinVilla	3. FCPortugal
MMDR	0.710 (0.027)	0.007 (0.013)	0.469 (0.024)
MMD+MSD^2	0.698 (0.027)	0.000 (self)	0.465 (0.023)
Static	0.604 (0.027)	-0.012 (0.016)	0.356 (0.024)
Greedy	0.530 (0.028)	-0.044 (0.016)	0.315 (0.024)
Greedy Offense	0.670 (0.027)	-0.039 (0.016)	0.435 (0.024)

A SCRAM positioning system using the MMDR role assignment function was a key component in winning the RoboCup 3D simulation world championship in 2011 [19] and 2012 [18]. SCRAM using the MMD+MSD^2 role assignment function was also an important factor in achieving 2nd place at the 2013 competition. In Table 5 we show how our team's performance is affected by using

[3] `http://simspark.sourceforge.net/`

[4] `http://www.ode.org/`

[5] `http://www.aldebaran-robotics.com/eng/`

the following alternative role assignment functions when playing against released binaries of the top three teams at the 2013 RoboCup competition.

Static Role assignments are fixed based on a player's uniform number.
Greedy Greedily assign robots to targets in order of shortest distances.
Greedy Offensive Similar to previously reported work in the RoboCup 3D simulation domain [8], assign closest robots to role positions in order of most offensive to least offensive role positions.

The SCRAM role assignment functions are superior to the other functions as they perform better against all opponents.

5.2 RoboCup 2D Simulation

As one of the oldest RoboCup leagues, 2D simulation soccer has been well explored, both in competition and in research. The domain consists of two teams of eleven autonomous agents playing soccer on a simulated 2D field. Agents receive sensory information, including the position of the ball and other robots, from a central game server. After processing this information, robot agents tell the server what actions they want to take such as dashing, kicking, and turning. 2D soccer abstracts away many of the low-level behaviors required for humanoid robot soccer in the 3D simulation league, including walking, and thus affords the chance to focus on higher-level aspects of playing soccer such as multirobot coordination and strategic play.

To test SCRAM in the RoboCup 2D simulation league we used the Agent2D [1] base code release which provides a fully functional soccer playing agent team. Agent2D includes default formation files using Delaunay triangulation [2] to specify robot role positions. In the Agent2D base code, robots are statically assigned to roles based on their uniform numbers. Agent2D teams only modified to use the MMDR and MMD+MSD[2] assignment functions beat the default Agent2D team by an average goal difference of 0.118 (+/- 0.025) and 0.105 (+/- 0.024) respectively over 10,000 games.

New at RoboCup 2013 was the addition of a drop-in player challenge[6] where robot agent teams consisting of different players randomly chosen from participants in the competition play against each other. This event is also known as an ad hoc teamwork challenge. Performance in the challenge was measured by a robot's average goal difference across all games played. An important aspect of the challenge is for a robot to be able to adapt to the behaviors of its teammates: for instance if most of a robot's teammates are assuming offensive roles, that robot might better serve the team by taking on a defensive role. SCRAM implicitly allows for this adaptation to occur as it naturally chooses roles for a robot that do not currently have another robot nearby.

Using released binaries from the RoboCup 2013 drop-in player challenge, we played 2800 drop-in player matches with both the default version of Agent2D

[6] Full rules of the challenge can be found at `http://www.cs.utexas.edu/~AustinVilla/sim/2dsimulation/2013_dropin_challenge/2D_DropInPlayerChallenge.pdf`

and a version of Agent2D with SCRAM (MMD+MSD^2). Empirically we found most robots used static role assignment thus underscoring the need for adapting to teammates' fixed roles as it was unlikely that teammates would adapt to roles assumed by you. Adding SCRAM to Agent2D improved performance in the challenge from an average goal difference of 1.473 (+/-0.157) with static role assignments to 1.659 (+/-0.153) with SCRAM. This result shows promise for SCRAM as not only a way to coordinate motion among one's own teammates, but also for adapting to unknown teammates in an ad hoc teamwork setting.

6 Summary and Discussion

This paper introduces SCRAM, a dynamic role assignment system for formational positioning of autonomous mobile robots, and provides theoretical and empirical analysis of the role assignment problem. SCRAM minimizes the makespan for robots to reach target goal positions while also avoiding collisions among robots. As role assignment algorithms run in polynomial time SCRAM scales to thousands of robots.

Our ongoing research agenda in the area of role assignment includes extending SCRAM to role assignment problems for heterogeneous robots. Example problem instances include robots specialized for different tasks such that particular robots are restricted to going to only a certain subset of target positions, as well as robots moving at different varying speeds. Another scenario to consider is when there are fixed obstacles that robots must avoid. Algorithms reported to be faster than the Hungarian algorithm for solving the *assignment problem*, such as the Jonker-Volgenant algorithm [13], and the dynamic Hungarian algorithm [22] for the case when most robots have reached their targets and few distances are changing, can also be explored to speed up role assignment algorithms.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by grants from the National Science Foundation (CNS-1330072, CNS-1305287) and ONR (21C184-01). Patrick MacAlpine is supported by a NDSEG fellowship.

References

1. Akiyama, H.: Agent2d base code (2010)
2. Akiyama, H., Noda, I.: Multi-agent positioning mechanism in the dynamic environment. In: RoboCup 2007: Robot Soccer World Cup XI. Springer (2008)
3. Alonso-Mora, J., Breitenmoser, A., Rufli, M., Siegwart, R., Beardsley, P.: Image and animation display with multiple mobile robots. The International Journal of Robotics Research **31**(6) (2012) 753–773
4. Bertsekas, D.P.: The auction algorithm: A distributed relaxation method for the assignment problem. Annals of operations research **14**(1) (1988) 105–123
5. Broucke, M.: Disjoint path algorithms for planar reconfiguration of identical vehicles. In: American Control Conference. (2003)
6. Burkard, R.E., Rendl, F.: Lexicographic bottleneck problems. Operations Research Letters **10**(5) (1991)

7. Chaimowicz, L., Campos, M.F., Kumar, V.: Dynamic role assignment for cooperative robots. In: Int. Conf. on Robotics and Automation (ICRA). (2002)
8. Chen, W., Chen, T.: Multi-robot dynamic role assignment based on path cost. In: Chinese Control and Decision Conf. (CCDC). (2011)
9. Croce, F.D., Paschos, V.T., Tsoukias, A.: An improved general procedure for lexicographic bottleneck problems. Operations research letters **24**(4) (1999) 187–194
10. Ford, D., Fulkerson, D.R.: Flows in networks. Princeton university press (2010)
11. Hokayem, P.F., Spong, M.W., Siljak, D.D.: Cooperative avoidance control for multiagent systems. Urbana **51** (2007) 61801
12. Ji, M., Azuma, S.i., Egerstedt, M.B.: Role-assignment in multi-agent coordination. (2006)
13. Jonker, R., Volgenant, A.: A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing **38**(4) (1987) 325–340
14. Kalyanakrishnan, S., Stone, P.: Learning complementary multiagent behaviors: A case study. In: RoboCup 2009: Robot Soccer World Cup XIII. (2010)
15. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly **2**(1-2) (1955) 83–97
16. Lau, N., Lopes, L., Corrente, G., Filipe, N.: Multi-robot team coordination through roles, positionings and coordinated procedures. In: Int. Conf. on Intelligent Robots and Systems (IROS). (2009)
17. MacAlpine, P., Barrera, F., Stone, P.: Positioning to win: A dynamic role assignment and formation positioning system. In: RoboCup-2012: Robot Soccer World Cup XVI. Springer Verlag, Berlin (2013)
18. MacAlpine, P., Collins, N., Lopez-Mobilia, A., Stone, P.: UT Austin Villa: RoboCup 2012 3D simulation league champion. In: RoboCup-2012: Robot Soccer World Cup XVI. Springer Verlag, Berlin (2013)
19. MacAlpine, P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., Stone, P.: UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In: Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS). (June 2012)
20. Mellinger, D., Kushleyev, A., Kumar, V.: Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In: Int. Conf. on Robotics and Automation (ICRA). (2012)
21. Michael, N., Zavlanos, M.M., Kumar, V., Pappas, G.J.: Distributed multi-robot task assignment and formation control. In: IEEE Int. Conf. on Robotics and Automation (ICRA). (2008)
22. Mills-Tettey, G.A., Stentz, A., Dias, M.B.: The dynamic hungarian algorithm for the assignment problem with changing costs. (2007)
23. Pentico, D.W.: Assignment problems: A golden anniversary survey. European Journal of Operational Research **176**(2) (2007) 774 – 793
24. Richards, A., How, J.: Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In: American Control Conference. (2002)
25. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent path finding. In: AAAI. (2012)
26. Sokkalingam, P., Aneja, Y.P.: Lexicographic bottleneck combinatorial problems. Operations Research Letters **23**(1) (1998) 27–33
27. Stone, P., Veloso, M.: Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. Artificial Intelligence **110**(2) (June 1999) 241–273

Appendix

A Role Assignment Function CM Validity

The following is a more in depth analysis of the *CM validity* of the role assignment functions MMDR and MMD+MSD^2 described in Section 3.

A.1 Minimizing Longest Distance

It is trivial to determine that both MMDR and MMD+MSD^2 select a mapping of robots to role positions that minimizes the time for all robots to have reached their target destinations. The total time it takes for all robots to move to their desired positions is determined by the time it takes for the last robot to reach its target position. As the first comparison between mapping costs for both role assignment functions is the maximum distance that any single robot in a mapping must travel, and it is assumed that all robots move toward their targets at the same constant rate, the property of minimizing the longest distance holds for both MMDR and MMD+MSD^2.

A.2 Avoiding Collisions

Given the assumptions that no two robots and no two role positions occupy the same position on the field, and that all robots move toward role positions along a straight line at the same constant speed, if two robots collide it means that they both started moving from positions that are the same distance away from the collision point. Furthermore if either robot were to move to the collision point, and then move to the target of the other robot, its total path distance to reach that target would be the same as the path distance of the other robot to that same target. Considering that we are working in a Euclidean space, by the triangle inequality we know that the straight path from the first robot to the second robot's target will be less than the path distance of the first robot moving to the collision point and then moving on to the second robot's target (which is equal to the distance of the second robot moving on a straight line to its target). Thus if the two colliding robots were to switch targets the maximum distance either is traveling will be reduced (along with the sum of the squared distances traveled), thereby reducing the cost of the mapping for both MMDR and MMD+MSD^2, and the collision will be avoided. Figure 3 illustrates an example of this scenario.

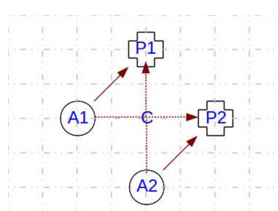

Fig. 3: Example collision scenario. If the mapping (A1→P2,A2→P1) is chosen the robots will follow the dotted paths and collide at the point marked with a C. Instead both MMDR and MMD+MSD^2 will choose the mapping (A1→P1,A2→P2), as this minimizes both maximum path distance and sum of distances squared, and the robots will follow the paths denoted by the solid arrows thereby avoiding the collision.

The following is a proof sketch related to Figure 3 that no collisions will occur.

Assumption. *Robots A1 and A2 move at constant velocity v on straight line paths to static positions P2 and P1 respectively. A1 $\neq$ A2 and P1 $\neq$ P2. Robots collide at point C at time t.*

Claim. A1→P2 and A2→P1 is an optimal mapping returned by MMDR.

Case 1. P1 and P2 $\neq$ C.
By assumption:
$\overline{A_1C} = \overline{A_2C} = vt$
$\overline{A_1P_2} = \overline{A_1C} + \overline{CP_2} = \overline{A_2C} + \overline{CP_2}$
$\overline{A_2P_1} = \overline{A_2C} + \overline{CP_1} = \overline{A_1C} + \overline{CP_1}$

By triangle inequality:
$\overline{A_1P_1} < \overline{A_1C} + \overline{CP_1} = \overline{A_2P_1}$
$\overline{A_2P_2} < \overline{A_2C} + \overline{CP_2} = \overline{A_1P_2}$

$\max(\overline{A_1P_1}, \overline{A_2P_2}) < \max(\overline{A_1P_2}, \overline{A_2P_1})$
$\overline{A_1P_1}^2 + \overline{A_2P_2}^2 < \overline{A_1P_2}^2 + \overline{A_2P_1}^2$
$\therefore \texttt{cost}(A1 \rightarrow P1, A2 \rightarrow P2) < \texttt{cost}(A1 \rightarrow P2, A2 \rightarrow P1)$ and claim is False.

Case 2. P1 = C, P2 $\neq$ C.
By assumption:
$\overline{CP_2} > \overline{CP_1} = 0$
$\overline{A_2C} \leq \overline{A_1C} = vt$
$\overline{A_1P_1} = \overline{A_1C} < \overline{A_1C} + \overline{CP_2} = \overline{A_1P_2}$

By triangle inequality:
if $\overline{A_1C} = \overline{A_2C}$
$\quad \overline{A_2P_2} < \overline{A_2C} + \overline{CP_2} = \overline{A_1C} + \overline{CP_2} = \overline{A_1P_2}$
otherwise $\overline{A_2C} < \overline{A_1C}$
$\quad \overline{A_2P_2} \leq \overline{A_2C} + \overline{CP_2} < \overline{A_1C} + \overline{CP_2} = \overline{A_1P_2}$

$\max(\overline{A_1P_1}, \overline{A_2P_2}) < \max(\overline{A_1P_2}, \overline{A_2P_1})$
$\overline{A_1P_1}^2 + \overline{A_2P_2}^2 < \overline{A_1P_2}^2 + \overline{A_2P_1}^2$
$\therefore \texttt{cost}(A1 \rightarrow P1, A2 \rightarrow P2) < \texttt{cost}(A1 \rightarrow P2, A2 \rightarrow P1)$ and claim is False

Case 3. P2 = C, P1 $\neq$ C.
Claim False by corollary to Case 2.

Case 4. P1, P2 = C.
Claim False by assumption.

As claim is False for all cases MMDR does not return mappings with collisions. □

B Dynamic Consistency

Dynamic consistency is important such that as robots move toward fixed target role positions they do not continually switch or thrash between roles thus impeding their progress in reaching target positions. Given the assumption that all robots move toward target positions at the same constant rate, all distances to targets in a MMDR mapping of robots to role positions will decrease at the same constant rate as the robots move until becoming 0 when a robot reaches its destination. Considering that robots move toward their target positions on straight line paths, it is not possible for the distance between any robot and any role position to decrease faster than the distance between a robot and the role position it is assigned to move toward. This means that the cost of any MMDR mapping can not improve over time any faster than the lowest cost MMDR mapping being followed, and thus dynamic consistency is preserved. Note that it is possible for two mappings of robots to role positions to have the same MMDR cost as the case of two robots being equidistant to two role positions. In this case one of the mappings may be arbitrarily selected and followed by the robots. As soon as the robots start moving the selected mapping will acquire and maintain a lower cost than the unselected mapping. The only way that the mappings could continue to have the same MMDR cost would be if the two role positions occupy the same place on the field, however, as stated in the given assumptions, this is not allowed.

MMD+MSD^2 is not dynamically consistent as minimizing the sum of distances squared (MSD^2) is not dynamically consistent. (MSD^2) is shown to be not dynamically consistent in Appendix C.

C Other Role Assignment Functions

Other functions for mapping robots to target positions include minimizing the sum of all distances traveled (MSD), minimizing the sum of all path distances squared (MSD^2), and assigning robots to targets in order of shortest distances (Greedy). None of these functions preserve both required properties listed in Section 2 for *CM validity*. Also none of them are dynamically consistent.

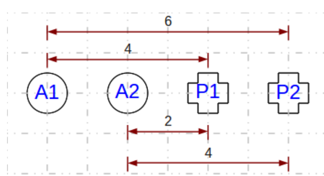

Fig. 4: Example where minimizing the sum of path distances fails to hold desired properties. Both mappings of (A1→P1,A2→P2) and (A1→P2,A2→P1) have a sum of distances value of 8. The mapping (A1→P2,A2→P1) will result in a collision and has a longer maximum distance of 6 than the mapping (A1→P1,A2→P2) whose maximum distance is 4. Once a mapping is chosen and the robots start moving the sum of distances of the two mappings will remain equal which could result in thrashing between the two.

As can be seen in Figure 4, none of the properties necessarily hold for MSD.

The first property of all robots having reached their target destinations in as little time as possible is not always true for MSD^2 as shown in Figure 5. MSD^2 does avoid collisions as explained in Appendix A.2. The following is an example in which MSD^2 is not dynamically consistent:

At time $t = 0$:
$A_1 = (3, 0)$
$A_2 = (2, 999)$
$P_1 = (0, 0)$
$P_2 = (1, 0)$

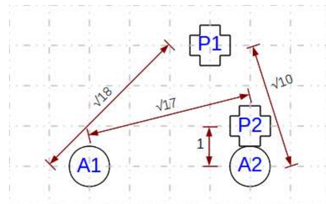

Fig. 5: Example where minimizing the sum of path distances squared fails to hold desired property of minimizing the time for all robots to have reached their targets. The mapping (A1→P1,A2→P2) has a path distance squared sum of 19 which is less than the mapping (A1→P2,A2→P1) for which this sum is 27. Both MMDR and MMD+MSD^2 will choose the mapping with the greater sum as its maximum path distance (proportional to the time for all robots to have reached their targets) is $\sqrt{17}$ which is less than the other mapping's maximum path distance of $\sqrt{18}$.

$A1 \rightarrow P1, A2 \rightarrow P2$
$\overline{A_1P_1} = 3$, $\overline{A_2P_2} = \sqrt{998002}$; $\overline{A_1P_1}^2 + \overline{A_2P_2}^2 = 998011$

$A1 \rightarrow P2, A2 \rightarrow P1$
$\overline{A_1P_2} = 2$, $\overline{A_2P_1} = \sqrt{998005}$; $\overline{A_1P_2}^2 + \overline{A_2P_1}^2 = 998009$

MSD^2 mapping $(A1 \rightarrow P2, A2 \rightarrow P1) \because 998009 < 998011$

At time $t = 2$:
$A_1 = (1, 0)$
$A_2 = (\sim 2, \sim 997)$
$P_1 = (0, 0)$
$P_2 = (1, 0)$

$A1 \rightarrow P1, A2 \rightarrow P2$
$\overline{A_1P_1} = 1$, $\overline{A_2P_2} = \sqrt{994010}$; $\overline{A_1P_1}^2 + \overline{A_2P_2}^2 = 994011$

$A1 \rightarrow P2, A2 \rightarrow P1$
$\overline{A_1P_2} = 0$, $\overline{A_2P_1} = \sqrt{994013}$; $\overline{A_1P_2}^2 + \overline{A_2P_1}^2 = 994013$

MSD^2 mapping $(A1 \rightarrow P1, A2 \rightarrow P2) \because 994011 < 994013$

As the mapping switched MSD^2 is not dynamically consistent.

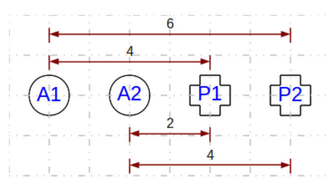

Fig. 6: Example where greedily choosing shortest paths fails to hold desired properties. The shortest distance is from A2→P1 resulting in a mapping of (A2→P1,A1→P2) to be chosen. The mapping (A2→P1,A1→P2) will result in a collision and has a longer maximum distance of 6 than the mapping (A1→P1,A2→P2) whose maximum distance is 4. Once the robots collide it is possible that A1 will move on top of P1 thus pushing A2 off of P1 and towards P2. This displacement of A2 may result in a switch between mappings and potential thrashing.

As can be seen in Figure 6, none of the properties necessarily hold for Greedy.