To appear in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (AAAI-15), Austin, Texas, USA, January 2015.

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

**Patrick MacAlpine** and **Eric Price** and **Peter Stone**
Department of Computer Science
The University of Texas at Austin
Austin, TX 78701, USA
{patmac, ecprice, pstone}@cs.utexas.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, represented as point masses, to a set of target goal locations within an open two dimensional space such that the makespan (time for all robots to reach their target locations) is minimized while also preventing collisions among robots. We present scaleable (computable in polynomial time) role assignment algorithms that we classify as being SCRAM (Scalable Collision-avoiding Role Assignment with Minimal-makespan). SCRAM role assignment algorithms use a graph theoretic approach to map agents to target goal locations such that our objectives for both minimizing the makespan and avoiding agent collisions are met. A system using SCRAM role assignment 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 role assignment generalizes well to many realistic and real-world multiagent systems, and scales to thousands of agents.

## 1 Introduction

Coordinated movement among mobile agents 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 agent moves to which position or role) (Chaimowicz, Campos, and Kumar 2002; Michael et al. 2008; Ji, Azuma, and Egerstedt 2006), path planning (paths agents take to assigned positions) (Mellinger, Kushleyev, and Kumar 2012; Sharon et al. 2012), and collision avoidance (Hokayem, Spong, and Siljak 2007; Richards and How 2002).

The work in this paper focuses on role assignment—specifically tackling the problem of assigning homogeneous mobile agents to move to a set of fixed target positions such that an agent 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 agents to target positions operate under the constraint that no agents collide.

Previous work on assigning agents to target positions has focused on minimizing the sum of distances all agents must travel which is the well known *assignment problem* (Pentico 2007). Our work differs as we minimize the makespan (time for all agents to reach goal positions) instead of the sum of distances traveled. Minimizing the makespan is a decisive factor in performance when agents are moving to target positions to complete a shared task where all agents must be in place before the task can be completed and/or started. Such tasks include those requiring agents 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 agent to get to its target position (e.g. warehouse robots delivering items for an order to be shipped and mobile robots used as pixels to display images (Alonso-Mora et al. 2012)).

We refer to our role assignment as SCRAM (Scalable Collision-avoiding Role Assignment with Minimal-makespan). It provides a collision free mapping of agents to target positions, minimizes the makespan, and scales to thousands of agents. Primary contributions of this paper include a complete specification of SCRAM, the presentation of role assignment functions for assigning agents to target positions, algorithms (both new and existing) for computing the role assignment functions,[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 positioning systems incorporating SCRAM role assignment used within the RoboCup 2D and 3D simulation domains, Section 6 discusses related work and extensions, and Section 7 concludes.



[1] Videos of SCRAM role assignment 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`

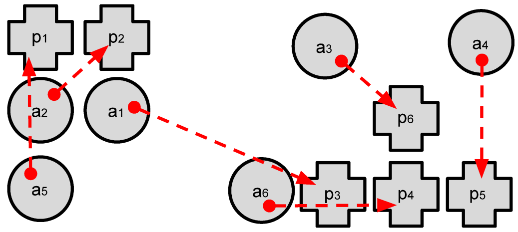


Figure 1: Role assignment problem where we want to assign agents (circles) $\{a_1,...,a_6\}$ to target positions (crosses) $\{p_1,...,p_6\}$. Dashed arrows show solution.

## 2 Role Assignment Problem

Let there be $n$ homogeneous mobile agents with current positions $A := \{a_1, ..., a_n\}$, and we want to assign the agents to move to $n$ specified target goal positions or roles $P := \{p_1, ..., p_n\}$ such that the time for agents to have reached every goal position is minimized under the constraint that no agents collide with each other. Figure 1 illustrates an example problem with six agents 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 agent and target positions.

Similar to path planning work by Broucke (2003), we model agents as point masses with zero width. Additionally, we make two more assumptions. First, no two agents and no two target positions occupy the same position. Second, we assume that all agents move toward fixed target positions along a straight line at the same constant speed. While the assumptions may not hold in practice, they are necessary for theoretical analysis of the role assignment problem and are often often good enough approximations, as corroborated by our successful empirical results in RoboCup domains presented in Section 5. We mention a potential extension of our work in Section 6 that includes allowing for non-point masses as well as obstacles in the environment.

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 an agent to target, with respect to all possible mappings. A valid mapping for the problem shown in Figure 1 would not include $a_2 \rightarrow p_5$ as that is the longest distance between an agent and target. Instead a valid assignment includes $a_1 \rightarrow p_3$ which is the minimal longest distance any agent travels across all possible assignments.
2. *Avoiding collisions* - agents do not collide with each other 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 agents $a_1$ 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 agents to targets at time $T$, then $f$ continues to output $M$ for every time $t > T$ as the agents move to the targets specified by $M$.

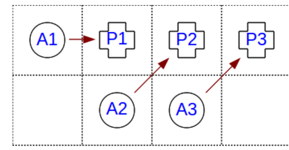


Figure 2: Lowest lexicographical cost (shown with arrows) to highest cost ordering of mappings from agents (A1,A2,A3) to role positions (P1,P2,P3). Each row represents the cost of a single mapping.

| | | | |
|---|---|---|---|
| 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) |

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 agents 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* role 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) Function

One potential role assignment function is to find a mapping of agents to target positions which recursively minimizes the maximum distance that any agent travels. We refer to this as this the Minimum Maximal Distance Recursive (MMDR) function. It is also known as the *lexicographic bottleneck assignment problem* (Pentico 2007). In previous work we presented an exponential time dynamic programming implementation of MMDR (MacAlpine, Barrera, and Stone 2013). In this section we first analyze properties of MMDR, and then identify efficient polynomial time algorithms to compute MMDR.

Let $\mathbb{M}$ be the set of all one-to-one mappings between agents and roles. If there are $n$ agents 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 agent 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.

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 an online appendix.[2]

[2] http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/AustinVilla3DSimulationFiles/2013/files/scram/scram-appendix.pdf

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

MMDR minimizes the longest distance (Property 1) as the lexicographical ordering of distance tuples sorted in descending order ensures this. If two agents 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 agents' targets reduces the maximum distance either must travel. Finally, as we assume all agents move toward their targets at the same constant rate, the distance between an agent and any target will not decrease any faster than the distance between any agent and the target that agent is assigned to. This observation provides dynamic consistency (Property 3) by preserving the lowest lexicographical cost ordering of a MMDR mapping across all timesteps.

**O($n^5$) Polynomial Time Algorithm for MMDR** We can compute the MMDR role assignment function in polynomial time 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 (Kuhn 1955) in O($n^3$) time.

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

In order 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 1 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 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 1 is O($n^5$). As our implementation of the Hungarian algorithm requires us to store length $n$ lists of size $n^2$ transformed weights, Algorithm 1 has a space complexity of 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 O($n^5 logn$) algorithm exists (Burkard and Rendl 1991).

**Algorithm 1** MMDR O($n^5$) Polynomial Time Impl.

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

1: *edgesSorted* $:=$ `sortAscendingDist`(*Edges*)
2: *lastDistance* $:= -1$
3: *rank*, *currentIndex* $:= 0$
4: **for each** $e \in$ *edgesSorted* **do**
5: $\quad$ **if** $|e| >$ *lastDistance* **then**
6: $\quad\quad$ *rank* $:=$ *currentIndex*
7: $\quad$ *lastDistance* $:= |e|$
8: $\quad$ $|e| := 2^{rank}$
9: $\quad$ *currentIndex* $:=$ *currentIndex* $+ 1$
10: **return** `hungarianAlg`(*edgesSorted*)

Work by Croce et al. (1999) changes edge weights into weight vectors of length $n$ before solving the *assignment problem* and has the same time complexity as our method of O($n^5$). However, on modern computer architectures Algorithm 1 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 size of a processor's maximum word length. This parallelism reduces the running time 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 (1998), and detailed in Algorithm 2, alternates between solving the *bottleneck assignment problem* (Pentico 2007) (finding the smallest maximum edge in a perfect matching) and a 0-1 cost version of the *assignment problem*.

**Algorithm 2** MMDR O($n^4$) Polynomial Time Impl.

**Input:**
$\quad$ *Agents* $:= \{a_1, ..., a_n\}$; *Positions* $:= \{p_1, ..., p_n\}$
$\quad$ *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: $\quad$ **return** $e_{a,p} \in Edges$, s.t. *poten*$(a)$ + *poten*$(p)$ = *cost*$(e_{a,p})$

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

At every iteration of Algorithm 2 solving the *bottleneck assignment problem* (line 5 which is implemented by Algorithm 3 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 (1998) that all perfect matchings of the subset of tight edges (defined as edges for which $\textit{poten}(a) + \textit{poten}(p) = \textit{cost}(e_{a,p})$) contain exactly *numLongestEdges* edges of length $|\textit{minLongestEdge}|$. 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 $|\textit{minLongestEdge}|$ (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 $|\textit{minLongestEdge}|$. We learn the weight of *numLongestEdges* edges in the MMDR mapping during every iteration of Algorithm 2, and after determining the weights for $n$ edges, the solution returned by the Hungarian algorithm is the MMDR mapping (line 11).

**Algorithm 3** Minimal-maximum Edge Perfect Matching

**Input:**
    $\textit{Agents} := \{a_1, ..., a_n\}$; $\textit{Positions} := \{p_1, ..., p_n\}$
    $\textit{Edges} := \{\overrightarrow{a_1 p_1}, \overrightarrow{a_1 p_2}, ..., \overrightarrow{a_n p_n}\}$; $|\overrightarrow{a_i p_j}| :=$ `euclideanDist`$(a_i, p_j)$

1: $\textit{matchedAgents}, \textit{allowedEdges} := \{\}$

2: **function** FLOOD(*curNode*, *prevNode*)
3:     *curNode.visited* := **true**
4:     *curNode.previous* := *prevNode*
5:     **if** *curNode* $\in$ *Positions* **and** $\nexists\, e \in$ *allowedEdges*, s.t. *e.start* = *curNode* **then**
6:         **return** *currentNode*
7:     **for each** $e \in$ *allowedEdges*, s.t. (*e.start* = *curNode* **and** **not** *e.end.visited*) **do**
8:         *val* := `flood`(*e.end*, *e.start*)
9:         **if** *val* $\neq \varnothing$ **then**
10:             **return** *val*
11:     **return** $\varnothing$

12: **function** RESETFLOOD
13:     **for each** *node* $\in$ {*Agents* $\cup$ *Positions*} **do**
14:         *node.visited* := **false**
15:         *node.previous* := $\varnothing$
16:     **for each** $a \in$ {*Agents* $\setminus$ *matchedAgents*} **do**
17:         `flood`$(a, \varnothing)$

18: **function** REVERSEPATH(*node*)
19:     **while** *node.previous* $\neq \varnothing$ **do**
20:         `reverseEdgeDirection`$(\overrightarrow{\textit{node}, \textit{node.previous}})$
21:         *node* := *node.previous*
22:     **return** *node*

23: *edgeQ* := `sortAscendingDist`$(\textit{Edges})$
24: *longestEdge* := $\varnothing$
25: **for** *match* := 1 **to** $n$ **do**
26:     `resetFlood()`
27:     *matchedPosition* := $\varnothing$
28:     **while** *matchedPosition* = $\varnothing$ **do**
29:         *longestEdge* := *edgeQ*.`pop`()
30:         *allowedEdges* $\leftarrow$ *longestEdge*
31:         *matchedPosition* := `flood`(*longestEdge.end*, *longestEdge.start*)
32:     *matchedAgent* := `reversePath`(*matchedPosition*)
33:     *matchedAgents* $\leftarrow$ *matchedAgent*
34: **return** *longestEdge*

Algorithm 3 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 (Ford and Fulkerson 2010) 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 an agent to a target.

Algorithm 3 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 an agent 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 agents to target, the last edge we added is the minimal maximum edge for a perfect matching.

An important factor for performance in Algorithm 3 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 3 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 2. Thus no more than $n$ instances of both the Hungarian algorithm and Algorithm 3 need to be computed. As the O($n^3$) time complexity of the Hungarian algorithm dominates Algorithm 2's loop, the time complexity of Algorithm 2 is O($n^4$). The breadth-first search of Ford-Fulkerson in Algorithm 3 gives a space complexity of O($n^2$).

### 3.2 Minimum Maximal Distance + Minimum Sum Distance$^2$ (MMD+MSD$^2$) Function

Another role assignment function to map agents to target goal positions is one which minimizes the maximum distance any agent has to travel (but not recursively as done by MMDR in Section 3.1), after which it minimizes the sum of distances squared that all agents 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

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

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

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 an online appendix.[2]

**Theorem 2.** *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 agent must travel (Property 1). If two agents 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 agents' 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 agents move toward targets (e.g. the difference in distance

Table 1: Time and space complexities of algorithms. SCRAM algorithms are show in bold.

| 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 (ms) of algorithms for values of $n$ on an Intel(R) Xeon(R) CPU E31270 @ 3.40GHz. SCRAM algorithms are show in bold.

| 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 | — | — | — | — | — |

squared as an agent 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 ($4^2 - 3^2 = 5$)). This lack of a constant rate of decrease for distances squared allows for squared distances between an agent and targets it is not assigned to travel toward to decrease faster than the squared distance between an agent 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 agents travel to their targets. An example of where MMD+MSD$^2$ is shown to not be dynamically consistent is provided in an online appendix.[2]

**Polynomial Time Algorithm for MMD+MSD$^2$** Algorithm 4 implements MMD+MSD$^2$ by first finding a perfect matching with the smallest maximum edge (line 1) which is computed by Algorithm 3 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 agent travels is also minimized.

**Algorithm 4** MMD+MSD$^2$ O($n^3$) Polynomial Time Impl.

**Input:**
*Agents* $:= \{a_1, ..., a_n\}$; *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 3 and thus the time complexity of Algorithm 4 is O($n^3$). The breadth-first search of Ford-Fulkerson in Algorithm 3 gives a space complexity of O($n^2$).

## 4 Assign. Function & Algorithm Analysis

To evaluate role assignment algorithms, we generated mapping scenarios for $n$ agents and targets. Both agents 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 algorithms for different values of $n$. The slowest was the brute force method evaluating all $n!$ possible mappings. The fastest was MMD+MSD$^2$ which has the lowest time complexity and a relatively low space complexity as shown in Table 1. MMD+MSD$^2$ took less than half a second for 1000 agents and less than two minutes for 10,000 agents. The polynomial time implementations of MMDR scale well to 100s of agents and are much faster than the dynamic programming implementation of MMDR. The O($n^4$) implementation of MMDR scales to 1000 agents and is faster than the O($n^5$) implementation except for smaller ($n <= 20$) inputs (our use case for RoboCup) where it takes longer due to the extra computations needed in its main loop.

Table 3: Role assignment function properties from Section 2. *CM valid* functions are shown in bold.

| Function | Min. Makespan | No Collisions | Dyn. 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 |

Table 4: Avg. makespan, avg. distance, and distance std. dev. over $10^6$ assignments of 10 agents to targets on a $100^2$ grid. *CM valid* role assignment functions are shown in bold.

| Function | Avg. Makespan | Avg. 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 |

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

**MSD** Minimize sum of distances between agents and targets.
**MSD$^2$** Minimize sum of distances$^2$ between agents and targets.
**Greedy** Assign agents to targets in order of shortest distances.
**Random** Random assignment of agents to targets.

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 functions are *CM valid* as they fail to minimize the makespan (further analysis of how other functions fail to hold properties is provided in an online appendix[2]). MMDR is the only dynamically consistent function of the ones we compare.

Average distance is not something *CM valid* role assignment functions explicitly attempt to minimize. However, this metric can be useful if agents 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. A third metric is distance standard deviation which is useful if there is a preference for having agents 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. MMDR and MMD+MSD$^2$ do well across all metrics in Table 4.

Table 5: Avg. goal difference (std. error) over 1000 games when playing against the top three teams at RoboCup 2013. *CM valid* role assignment functions are shown in bold.

| Function | 1. Apollo3D | 2. AustinVilla | 3. FCPortugal |
|---|---|---|---|
| **MMDR** | 0.710 (0.027) | 0.007 (0.013) | 0.469 (0.024) |
| **MMD+MSD$^2$** | 0.698 (0.027) | 0 (self) | 0.499 (0.024) |
| 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) |

# 5 RoboCup Case Studies

RoboCup robot soccer has served as an excellent research domain for autonomous agents 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. As RoboCup necessitates that agents 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 employed in two different RoboCup leagues.

## 5.1 RoboCup 3D Simulation

The RoboCup 3D simulation environment is based on SimSpark, a generic physical multiagent system simulator. SimSpark uses the Open Dynamics Engine (ODE) library for its realistic simulation of rigid body dynamics. Agents in the simulation are modeled after the Aldebaran Nao robot. Visual information about the environment is given to agents through noisy measurements of the distance and angle to objects within a restricted view cone (120°). Games consist of two five minute halves of 11vs11 teams playing soccer.

In UT Austin Villa's positioning system players' positions are determined in three steps. First, a full team formation is computed using Delaunay triangulation (Akiyama and Noda 2008) based on set offset positions from the ball (formations used are provided in (MacAlpine et al. 2013)). 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 agents have reasonably accurate knowledge of where all agents are currently located. We use agent communication to share and synchronize agent world models as discussed in (MacAlpine, Barrera, and Stone 2013). For the third and final step a voting coordination mechanism detailed in (MacAlpine, Barrera, and Stone 2013) synchronizes players' computed assignments.

A SCRAM positioning system using MMDR was a key component in winning the RoboCup 3D simulation world championship in 2011 (MacAlpine et al. 2012), 2012 (MacAlpine et al. 2013), and 2014 (MacAlpine, Depinet, and Stone 2015). SCRAM using MMD+MSD$^2$ was also an important factor in achieving 2nd place in 2013. In Table 5 we show our team's performance against the top three teams at the 2013 RoboCup competition using both SCRAM and the following alternative assignment functions.

**Static** Role assignments fixed based on player's uniform number.

**Greedy** Assign agents to targets in order of shortest distances.

**Greedy Offense** Similar to previously reported work in the RoboCup 3D simulation domain (Chen and Chen 2011), assign closest agents to roles in order of most offensive positions.

The *CM valid* 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 soccer field. Agents receive sensory information, including the position of the ball and other agents, from a central game server. After processing this information, agents then tell the server what actions they want to take such as dashing, kicking, and turning.

To test SCRAM in the RoboCup 2D simulation league we used the Agent2D (Akiyama 2010) base code release which provides a fully functional soccer playing agent team. Agent2D includes default formation files using Delaunay triangulation (Akiyama and Noda 2008) to specify agent role positions. In the Agent2D base code, agents are statically assigned to roles based on their uniform numbers. Agent2D teams only modified to use the MMDR and MMD+MSD$^2$ role assignment functions beat the default Agent2D team by an average goal difference of 0.118 (+/- 0.025) and 0.105 (+/- 0.024) respectively across 10,000 games.

New at RoboCup 2013 was the addition of a drop-in player challenge where agent teams consisting of different players randomly chosen from participants in the competition play against each other. An important aspect of the challenge is for an agent to be able to adapt to the behaviors of its teammates: for instance if most of an agent's teammates are assuming offensive roles, that agent 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 an agent that do not currently have another agent nearby.

Using agents from the drop-in player challenge, we played 2800 drop-in player matches with both the default version of Agent2D and a version of Agent2D with SCRAM (MMD+MSD$^2$). Empirically we found most agents used static role assignment thus underscoring the need for adapting to teammates' fixed roles. 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 adapting to unknown teammates' behaviors.

# 6 Related Work and Extensions

While there exists previous work on role assignment in RoboCup simulated soccer domains (Lau et al. 2009; Stone and Veloso 1999; Chen and Chen 2011), it has not focused on collision avoidance or formation completion time.

Work very related to ours (Turpin, Michael, and Kumar 2014) assigns interchangeable robots to goal positions where robots have non-point masses and exist in environments containing obstacles. Additional work by Turpin et al. (2014) suggests using the Hungarian algorithm to minimize the sum of distances raised to large powers as a proxy for MMDR, however such an approach is not guaranteed to return the same result as MMDR and minimize the makespan. We believe the above work could be augmented with the

MMD+MSD$^2$ role assignment algorithm to allow for minimizing the makespan of robots traveling in cluttered environments in O($n^3$) time.

Another possible extension to SCRAM includes role assignment problems where there are unequal numbers of agents and targets. To extend SCRAM to the case when there are $m$ agents and $n$ target locations, and $m > n$, is trivial. All that must done is to add $m - n$ dummy target locations with all agents being assigned a distance of 0 to each of the dummy locations. As the Hungarian algorithm minimizes the sum of edge weights, the excess $m - n$ agents (those not in a minimum makespan matching to real targets) are assigned to dummy locations. Conversely, if there are more target locations than agents ($n > m$), and we desire agents to travel from one target to another such that every target location is eventually visited by an agent, role assignment becomes a vehicle routing problem (Toth and Vigo 2001).

## 7 Summary and Discussion

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

Our ongoing research agenda includes extending SCRAM to heterogeneous agents for cases such as agent task specialization, where agents are restricted to only certain subsets of role positions, as well as agents moving at varying speeds. Additional work includes looking into decentralized ways of computing SCRAM role assignment algorithms.

## References


Akiyama, H., and Noda, I. 2008. Multi-agent positioning mechanism in the dynamic environment. In *RoboCup 2007: Robot Soccer World Cup XI*. Springer.

Akiyama, H. 2010. Agent2d base code.

Alonso-Mora, J.; Breitenmoser, A.; Rufli, M.; Siegwart, R.; and Beardsley, P. 2012. Image and animation display with multiple mobile robots. *The International Journal of Robotics Research* 31(6):753–773.

Broucke, M. 2003. Disjoint path algorithms for planar reconfiguration of identical vehicles. In *American Control Conference*.

Burkard, R. E., and Rendl, F. 1991. Lexicographic bottleneck problems. *Operations Research Letters* 10(5).

Chaimowicz, L.; Campos, M. F.; and Kumar, V. 2002. Dynamic role assignment for cooperative robots. In *Robotics and Automation. Proceedings. ICRA'02. IEEE International Conference on*.

Chen, W., and Chen, T. 2011. Multi-robot dynamic role assignment based on path cost. In *2011 Chinese Control and Decision Conference (CCDC)*.

Croce, F. D.; Paschos, V. T.; and Tsoukias, A. 1999. An improved general procedure for lexicographic bottleneck problems. *Operations research letters* 24(4):187–194.

Ford, D., and Fulkerson, D. R. 2010. *Flows in networks*. Princeton university press.

Hokayem, P. F.; Spong, M. W.; and Siljak, D. D. 2007. Cooperative avoidance control for multiagent systems. *Urbana* 51:61801.

Ji, M.; Azuma, S.-i.; and Egerstedt, M. B. 2006. Role-assignment in multi-agent coordination.

Kuhn, H. W. 1955. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2(1-2):83–97.

Lau, N.; Lopes, L.; Corrente, G.; and Filipe, N. 2009. Multi-robot team coordination through roles, positionings and coordinated procedures. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*.

MacAlpine, P.; Barrera, F.; and Stone, P. 2013. Positioning to win: A dynamic role assignment and formation positioning system. In *RoboCup-2012: Robot Soccer World Cup XVI*, Lecture Notes in Artificial Intelligence. Berlin: Springer Verlag.

MacAlpine, P.; Urieli, D.; Barrett, S.; Kalyanakrishnan, S.; Barrera, F.; Lopez-Mobilia, A.; Ştiurcă, N.; Vu, V.; and Stone, P. 2012. UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In *Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*.

MacAlpine, P.; Collins, N.; Lopez-Mobilia, A.; and Stone, P. 2013. UT Austin Villa: RoboCup 2012 3D simulation league champion. In *RoboCup-2012: Robot Soccer World Cup XVI*, Lecture Notes in Artificial Intelligence. Berlin: Springer Verlag.

MacAlpine, P.; Depinet, M.; and Stone, P. 2015. UT Austin Villa 2014: RoboCup 3D simulation league champion via overlapping layered learning. In *Proc. of the Twenty-Ninth AAAI Conf. on Artificial Intelligence (AAAI)*.

Mellinger, D.; Kushleyev, A.; and Kumar, V. 2012. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *Robotics and Automation (ICRA)*.

Michael, N.; Zavlanos, M. M.; Kumar, V.; and Pappas, G. J. 2008. Distributed multi-robot task assignment and formation control. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 128–133. IEEE.

Pentico, D. W. 2007. Assignment problems: A golden anniversary survey. *European Journal of Operational Research* 176(2):774 – 793.

Richards, A., and How, J. 2002. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *American Control Conference*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2012. Conflict-based search for optimal multi-agent path finding. In *AAAI*.

Sokkalingam, P., and Aneja, Y. P. 1998. Lexicographic bottleneck combinatorial problems. *Operations Research Letters* 23(1):27–33.

Stone, P., and Veloso, M. 1999. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence* 110(2):241–273.

Toth, P., and Vigo, D. 2001. *The vehicle routing problem*. Siam.

Turpin, M.; Mohta, K.; Michael, N.; and Kumar, V. 2014. Goal assignment and trajectory planning for large teams of interchangeable robots. *Autonomous Robots* 37(4):401–415.

Turpin, M.; Michael, N.; and Kumar, V. 2014. Capt: Concurrent assignment and planning of trajectories for multiple robots. *The International Journal of Robotics Research* 33(1):98–112.

# 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 agents to role positions that minimizes the time for all agents to have reached their target destinations. The total time it takes for all agents to move to their desired positions is determined by the time it takes for the last agent to reach its target position. As the first comparison between mapping costs for both role assignment functions is the maximum distance that any single agent must travel, and it is assumed that all agents 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 agents and no two role positions occupy the same position on the field, and that all agents move toward role positions along a straight line at the same constant speed, if two agents collide it means that they both started moving from positions that are the same distance away from the collision point. Furthermore if either agent were to move to the collision point, and then move to the target of the other agent, its total path distance to reach that target would be the same as the path distance of the other agent 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 agent to the second agent's target will be less than the path distance of the first agent moving to the collision point and then moving on to the second agent's target (which is equal to the distance of the second agent moving on a straight line to its target). Thus if the two colliding agents 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.

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

**Assumption.** *Agents 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. Agents 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:*
$\overline{A_1P_1} < \overline{A_1C} + \overline{CP_1} = \overline{A_2P_1}$
$\overline{A_2P_2} < \overline{A_2C} + \overline{CP_2} = \overline{A_1P_2}$

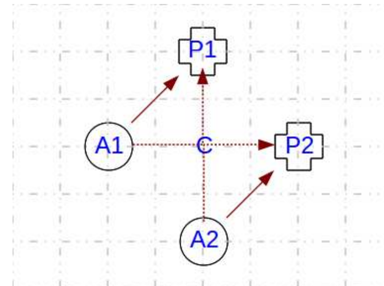


Figure 3: Example collision scenario. If the mapping (A1→P2,A2→P1) is chosen the agents will follow the dotted paths and collide at the point marked with a C. Instead both MMDR and MMD+MSD$^2$ will choose the mapping (A1→P1,A2→P2), as this minimizes both maximum path distance and sum of distances squared,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

$\max(\overline{A_1P_1}, \overline{A_2P_2}) < \max(\overline{A_1P_2}, \overline{A_2P_1})$
$\overline{A_1P_1}^2 + \overline{A_2P_2}^2 < \overline{A_1P_2}^2 + \overline{A_2P_1}^2$
$\therefore\ \mathtt{cost}(A1 \to P1, A2 \to P2) < \mathtt{cost}(A1 \to P2, A2 \to P1)$
*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* $\overline{A_1C} = \overline{A_2C}$
$\overline{A_2P_2} < \overline{A_2C} + \overline{CP_2} = \overline{A_1C} + \overline{CP_2} = \overline{A_1P_2}$
*otherwise* $\overline{A_2C} < \overline{A_1C}$
$\overline{A_2P_2} \leq \overline{A_2C} + \overline{CP_2} < \overline{A_1C} + \overline{CP_2} = \overline{A_1P_2}$

$\max(\overline{A_1P_1}, \overline{A_2P_2}) < \max(\overline{A_1P_2}, \overline{A_2P_1})$
$\overline{A_1P_1}^2 + \overline{A_2P_2}^2 < \overline{A_1P_2}^2 + \overline{A_2P_1}^2$
$\therefore\ \mathtt{cost}(A1 \to P1, A2 \to P2) < \mathtt{cost}(A1 \to P2, A2 \to P1)$
*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 agents 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 agents move toward target positions at the same constant rate, all distances to targets in a MMDR mapping of agents to role positions will decrease at the same constant rate as the agents move until becoming 0 when an agent reaches its destination. Considering that agents move toward their target positions on straight line paths, it is not possible for the distance between any agent and any role position to decrease faster than the distance between an agent and the role position it is assigned to move toward. This means that the cost of any MMDR mapping can not improve over time

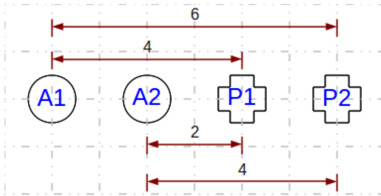


Figure 4: Example where minimizing the sum of path distances fails to hold desired properties. Both mappings of (A1→P1,A2→P2) and (A1→P2,A2→P1) have a sum of distances value of 8. The mapping (A1→P2,A2→P1) will result in a collision and has a longer maximum distance of 6 than the mapping (A1→P1,A2→P2)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

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 agents to role positions to have the same MMDR cost as the case of two agents being equidistant to two role positions. In this case one of the mappings may be arbitrarily selected and followed by the agents. As soon as the agents 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 potential ordering heuristics for mappings of agents to target positions include minimizing the sum of all distances traveled (MSD), minimizing the sum of all path distances squared (MSD$^2$), and assigning agents to targets in order of shortest distances (Greedy). None of these heuristics preserve both required properties listed in Section 2 for *CM validity* which are true for both MMDR and MMD+MSD$^2$. Also none of them are dynamically consistent.

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

The first property of all agents having reached their target destinations in as little time as possible is not always true for MSD$^2$ as shown in the example 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 $t = 0$:
$A_1 = (3, 0)$
$A_2 = (2, 999)$
$P_1 = (0, 0)$
$P_2 = (1, 0)$

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

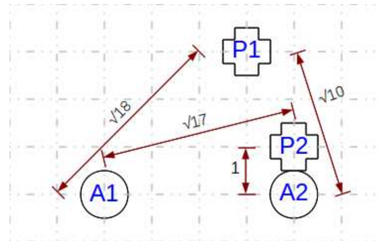


Figure 5: Example where minimizing the sum of path distances squared fails to hold desired property of minimizing the time for all agents to have reached their target destinations. The mapping (A1→P1,A2→P2) has a path distance squared sum of 19 which is less than the mapping (A1→P2,A2→P1) for which this 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 agents to have reached their targets) is $\sqrt{17}$ which is less than the other mapping's maximum path distance of $\sqrt{18}$.

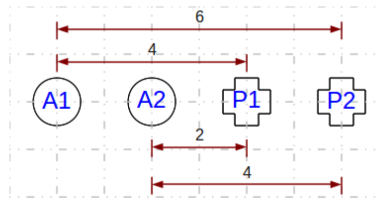


Figure 6: Example where greedily choosing shortest paths fails to hold desired properties. The shortest distance is from A2→P1 resulting in a mapping of (A2→P1,A1→P2) to be chosen. The mapping (A2→P1,A1→P2) will result in a collision and has a longer maximum distance of 6 than the mapping (A1→P1,A2→P2) whose maximum distance is 4. Once the agents 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.

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

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.

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