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

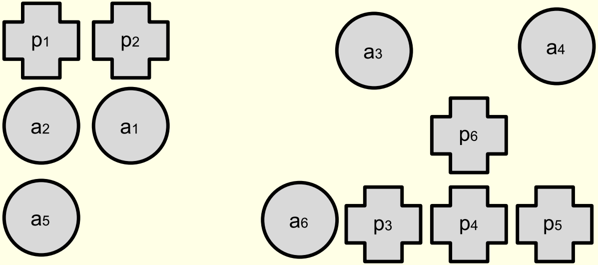

Patrick MacAlpine, Eric Price, and Peter Stone

Department of Computer Science, The University of Texas at Austin


January 30, 2015

Learning Agents Research Group
The University of Texas at Austin

# Role Assignment Problem

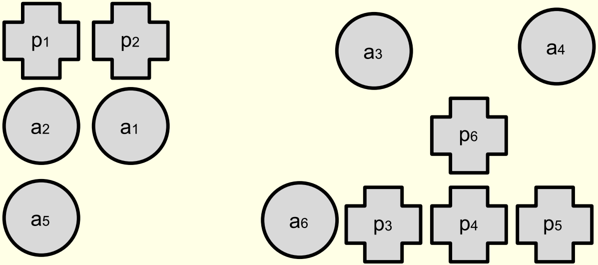



**Problem:**

How to assign *n* interchangeable robots to *n* targets in a one-to-one mapping so that the makespan is minimized and collisions are avoided.

Makespan = time for all robots to reach their assigned target positions (equivalent to the time for the the robot with the longest distance to travel to reach its assigned target position)

# Role Assignment Problem



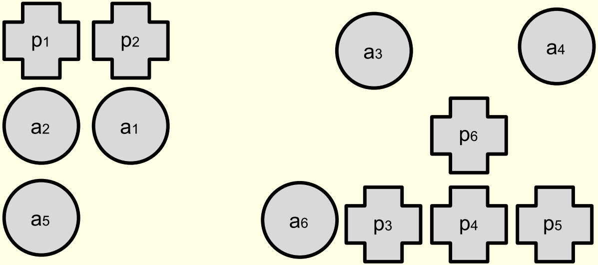

**Problem:**

How to assign *n* interchangeable robots to *n* targets in a one-to-one mapping so that the makespan is minimized and collisions are avoided.

**ASSUMPTIONS:**

- No two robots or targets occupy the same position
- Robots are treated as zero width point masses
- Robots move at same constant speed along straight line paths to targets

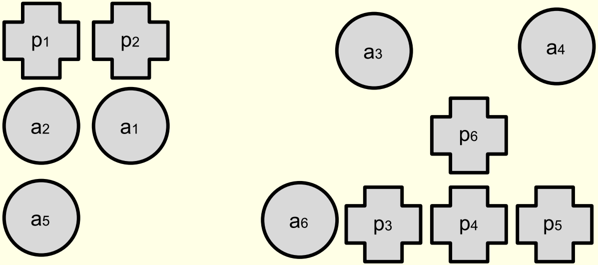
# Role Assignment Problem




**Required properties of a role assignment function to be *CM Valid* (Collision-avoiding with Minimal-makespan):**

1. *Minimizing makespan* - it minimizes the maximum distance from a robot to target, with respect to all possible mappings
2. *Avoiding collisions* - robots do not collide with each other

# Role Assignment Problem



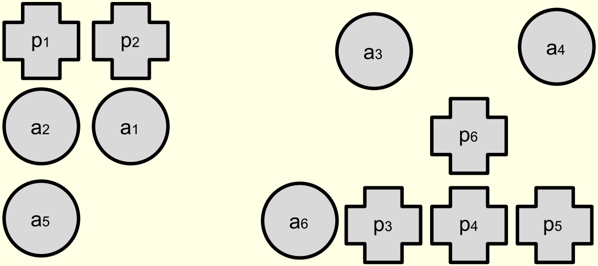

**Required properties of a role assignment function to be *CM Valid* (Collision-avoiding with Minimal-makespan):**

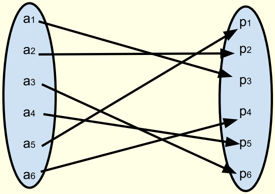
1. *Minimizing makespan* - it minimizes the maximum distance from a robot to target, with respect to all possible mappings
2. *Avoiding collisions* - robots do not collide with each other

**Desirable but not necessary to be *CM Valid*:**

3. *Dynamically consistent* - role assignments don't change or switch as robots move toward target positions

# Role Assignment Problem

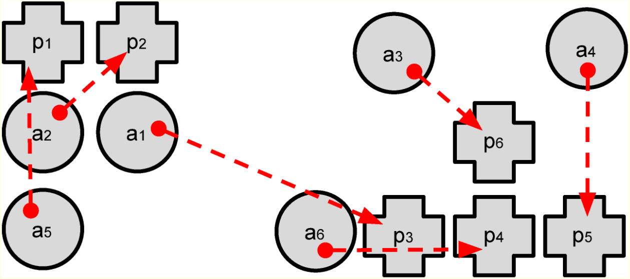



**Bipartite Graph Perfect Matching**

**n! possible mappings**

# Role Assignment Problem



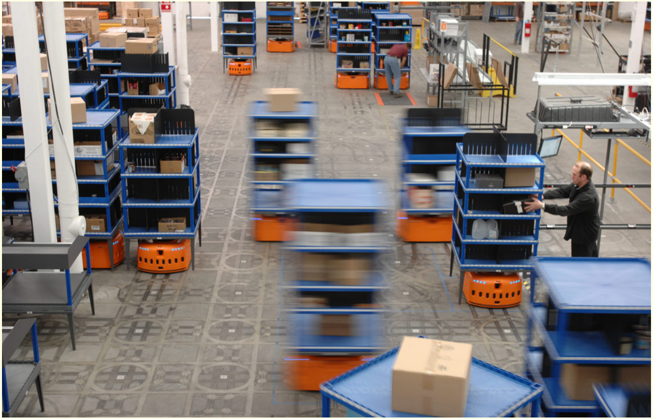

**Required properties of a role assignment function to be *CM Valid* (Collision-avoiding with Minimal-makespan):**

1. *Minimizing makespan* - it minimizes the maximum distance from a robot to target, with respect to all possible mappings
2. *Avoiding collisions* - robots do not collide with each other

Not include $a_2 \rightarrow p_5$ (longest possible distance), instead $a_1 \rightarrow p_3$ (minimal longest distance)

$a_1 \rightarrow p_1$ and $a_2 \rightarrow p_2$ would cause a collision between $a_1$ and $a_2$

# Motivation

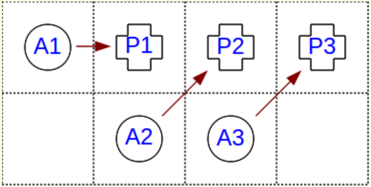


- Scenarios for which the bottleneck is the time it takes for the last robot to get to its target (e.g. robots procuring items for an order to be shipped)
- Tasks requiring robots be synchronized when they start jobs at their target positions (e.g. robots on an assembly line)

# Outline

- SCRAM *CM_Valid* Role Assignment Function and Algorithmic Implementation Analysis
    - Minimum Maximal Distance Recursive (MMDR)
    - Minimum Maximal Distance + Minimum Sum Distance$^2$ (MMD+MSD$^2$)
- RoboCup Robot Soccer Domain Examples
    - 3D Simulation League
    - 2D Simulation League

# Minimum Maximal Distance Recursive (MMDR) Role Assignment Function

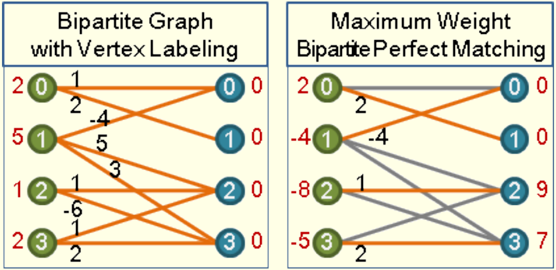



Lowest lexicographical cost (shown with arrows) to highest cost ordering of mappings from agents (A1,A2,A3) to role positions (P1,P2,P3). Each row represents the cost of a single mapping.

| | | | |
|---|---|---|---|
| 1: | $\sqrt{2}$ (A2$\rightarrow$P2), | $\sqrt{2}$ (A3$\rightarrow$P3), | 1 (A1$\rightarrow$P1) |
| 2: | 2 (A1$\rightarrow$P2), | $\sqrt{2}$ (A3$\rightarrow$P3), | 1 (A2$\rightarrow$P1) |
| 3: | $\sqrt{5}$ (A2$\rightarrow$P3), | 1 (A1$\rightarrow$P1), | 1 (A3$\rightarrow$P2) |
| 4: | $\sqrt{5}$ (A2$\rightarrow$P3), | 2 (A1$\rightarrow$P2), | $\sqrt{2}$ (A3$\rightarrow$P1) |
| 5: | 3 (A1$\rightarrow$P3), | 1 (A2$\rightarrow$P1), | 1 (A3$\rightarrow$P2) |
| 6: | 3 (A1$\rightarrow$P3), | $\sqrt{2}$ (A2$\rightarrow$P2), | $\sqrt{2}$ (A3$\rightarrow$P1) |

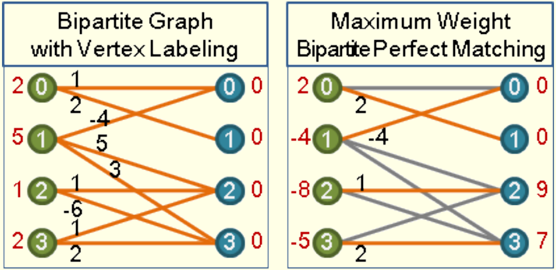
- Mapping cost = vector of distances sorted in decreasing order
- Optimal mapping = lexicographically sorted lowest cost mapping

# Hungarian Algorithm




- Finds a maximum/minimum weight (sum of weights) perfect matching in a bipartite graph (solves the *assignment problem*)
- Runs in O($n^3$) time

# Hungarian Algorithm




- Finds a maximum/minimum weight (sum of weights) perfect matching in a bipartite graph (solves the *assignment problem*)
- Runs in $O(n^3)$ time

**Can we transform MMDR into the *assignment problem*?**

# MMDR $O(n^5)$ Algorithm

**Goal:**

Transform edge distances to be set of weights such that the weight of any edge $e$ is greater than the sum of weights of all edges with distances less than $e$.

1. Transform edge distances to new weights
   - Sort edges in ascending order of distance
   - Set weights to be $2^i$ where $i$ is the index of an edge in this sorted list

2. Run Hungarian algorithm with modified weights returns.
   - Returns MMDR mapping

Time: $O(n^2)$ bits weights X $O(n^3)$ Hungarian algorithm = $O(n^5)$

# MMDR O($n^5$) Algorithm

**Goal:**

Transform edge distances to be set of weights such that the weight of any edge $e$ is greater than the sum of weights of all edges with distances less than $e$.

1. Transform edge distances to new weights: 5 4 6
   - Sort edges in ascending order of distance
   - Set weights to be $2^i$ where $i$ is the index of an edge in this sorted list

2. Run Hungarian algorithm with modified weights returns.
   - Returns MMDR mapping

Time: O($n^2$) bits weights X O($n^3$) Hungarian algorithm = O($n^5$)

# MMDR O($n^5$) Algorithm

**Goal:**

Transform edge distances to be set of weights such that the weight of any edge $e$ is greater than the sum of weights of all edges with distances less than $e$.

1. Transform edge distances to new weights: 5 4 6
   - Sort edges in ascending order of distance: 4 5 6
   - Set weights to be $2^i$ where $i$ is the index of an edge in this sorted list

2. Run Hungarian algorithm with modified weights returns.
   - Returns MMDR mapping

Time: O($n^2$) bits weights X O($n^3$) Hungarian algorithm = O($n^5$)

# MMDR $O(n^5)$ Algorithm

**Goal:**

Transform edge distances to be set of weights such that the weight of any edge $e$ is greater than the sum of weights of all edges with distances less than $e$.

1. Transform edge distances to new weights: 5 4 6
   - Sort edges in ascending order of distance: 4 5 6
   - Set weights to be $2^i$ where $i$ is the index of an edge in this sorted list

     $100_2$ (4) > $010_2$ (2) + $001_2$ (1) = $011_2$ (3)

     6 > 5 + 4
2. Run Hungarian algorithm with modified weights returns.
   - Returns MMDR mapping

Time: $O(n^2)$ bits weights X $O(n^3)$ Hungarian algorithm = $O(n^5)$

# MMDR O($n^5$) Algorithm

**Goal:**

Transform edge distances to be set of weights such that the weight of any edge *e* is greater than the sum of weights of all edges with distances less than *e*.

1. Transform edge distances to new weights: 5 4 6
   - Sort edges in ascending order of distance: 4 5 6
   - Set weights to be $2^i$ where $i$ is the index of an edge in this sorted list

     $100_2$ (4) > $010_2$ (2) + $001_2$ (1) = $011_2$ (3)

     6 > 5 + 4
2. Run Hungarian algorithm with modified weights returns.
   - Returns MMDR mapping

Time: O($n^2$) bits weights X O($n^3$) Hungarian algorithm = O($n^5$)

Scales to 100s of robots

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

Find a perfect matching $M$ that:

1. Has a minimum-maximal edge
2. Minimizes the sum of distances squared

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

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

# Polynomial Time Algorithm for MMD+MSD$^2$

Minimal-maximum Edge Perfect Matching Algorithm: O($n^3$)
breadth-first search using Ford-Fulkerson algorithm to find the minimal maximum length edge in a perfect matching.

1. Find minimal-maximum edge in perfect matching with weight $w$ using Minimal-maximum Edge Perfect Matching Algorithm
2. Remove all edges with weight greater than $w$ from graph
3. Use Hungarian algorithm to compute perfect matching with min sum of distances squared

Time: O($n^3$) Min-max Edge Alg. + O($n^3$) Hung. Alg. = O($n^3$)

# Polynomial Time Algorithm for MMD+$MSD^2$

Minimal-maximum Edge Perfect Matching Algorithm: $O(n^3)$
breadth-first search using Ford-Fulkerson algorithm to find the minimal maximum length edge in a perfect matching.

1. Find minimal-maximum edge in perfect matching with weight $w$ using Minimal-maximum Edge Perfect Matching Algorithm
2. Remove all edges with weight greater than $w$ from graph
3. Use Hungarian algorithm to compute perfect matching with min sum of distances squared

Time: $O(n^3)$ Min-max Edge Alg. + $O(n^3)$ Hung. Alg. = $O(n^3)$

Scales to 1000s of robots

# Polynomial Time Algorithm for MMD+MSD$^2$

Minimal-maximum Edge Perfect Matching Algorithm: O($n^3$)
breadth-first search using Ford-Fulkerson algorithm to find the minimal maximum length edge in a perfect matching.

1. Find minimal-maximum edge in perfect matching with weight $w$ using Minimal-maximum Edge Perfect Matching Algorithm
2. Remove all edges with weight greater than $w$ from graph
3. Use Hungarian algorithm to compute perfect matching with min sum of distances squared

Time: O($n^3$) Min-max Edge Alg. + O($n^3$) Hung. Alg. = O($n^3$)

Scales to 1000s of robots

O($n^4$), Sokkalingam and Aneja

# MMDR vs MMD+MSD$^2$

- Both minimize the makespan (longest distance any agent travels to a target) but use different mesurement values to determine other assignments of agents to targets

- Both avoid collisions among agents

- MMDR is dynamically consistent while
  MMD+MSD$^2$ is not dynamically consistent

Proof sketches of the above three properties are given in the appendix of the paper

- MMD+MSD$^2$ is faster to compute

# Role Assignment Function Properties

Function Properties

| 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 |

Assigning 10 robots to 10 targets on a 100 X 100 grid

| 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 |

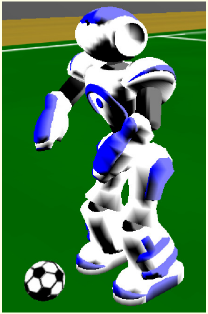
**MSD:** Minimize sum of distances between robots and targets.
**MSD$^2$:** Minimize sum of distances$^2$ between robots and targets.
**Greedy:** Assign robots to targets in order of shortest distances.
**Random:** Random assignment of robots to targets.

# Role Assignment Functions Video


- Yellow robots moving to green targets turn red if they collide
- Robot paths turn light blue if robot switches targets (not dynamically consistent)
- Background turns green when all robots have reached targets (makespan completed)

# 2013 RoboCup 3D Simulation Domain

- Teams of 11 vs 11 autonomous simulated robots play soccer
- Realistic physics using Open Dynamics Engine (ODE)
- Robots modeled after Aldebaran Nao robot
- Robot receives noisy visual information about environment
- Robots can communicate with each other over limited bandwidth channel



# RoboCup 3D Positioning Video


Each position is shown as a color-coded number corresponding to the robots's uniform number assigned to that position. Robots update their role assignments and move to new positions as the ball or a robot is beamed (moved) to a new location.

Key component to winning competition 3 of the past 4 years!

# RoboCup 2D


Yellow team (SCRAM (MMD+MSD$^2$)) vs red team (static)

Modified base Agent2D team using static role assignment to instead use SCRAM role assignment functions. Teams using MMDR and MMD+MSD$^2$ beat the team using static role assignment by an average goal difference of 0.118 (+/- 0.025) and 0.105 (+/- 0.024) respectively over 10,000 games.

# SCRAM Summary

## SCRAM Summary

- SCRAM minimizes the makespan or longest distance any robot has to travel

# SCRAM Summary

- SCRAM minimizes the makespan or longest distance any robot has to travel
- SCRAM avoids collisions between robots

# SCRAM Summary

- SCRAM minimizes the makespan or longest distance any robot has to travel
- SCRAM avoids collisions between robots
- SCRAM role assignment algorithms run in polynomial time and scales to 1000s of robots

# SCRAM Summary

- SCRAM minimizes the makespan or longest distance any robot has to travel
- SCRAM avoids collisions between robots
- SCRAM role assignment algorithms run in polynomial time and scales to 1000s of robots
- SCRAM is effective in complex RoboCup domains

# Future Work

- Task specialization where robots can only be assigned to a subset of target position
- Heterogeneous robots moving at different varying speeds
- Have robots also avoid known fixed obstacles and model robots as having true mass instead of zero width point mass
  - Concurrent Assignment and Planning of Trajectories (CAPT), Turpin et al.
- Make algorithms distributed
  - Auction algorithms

# More Information

More information, C++ implementations of SCRAM role assignment algorithms, and videos at:
http://tinyurl.com/aaai15scram
Email: patmac@cs.utexas.edu


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).

# Role Assignment Algorithm Analysis

Time and space complexities

| 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 | $O(n!n)$ | $O(n)$ |

Running time in milliseconds for different values of $n$

| 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 dyna | 0.555 | 2040 | — | — | — | — |
| brute force | 317.5 | — | — | — | — | — |

# Role Assignment Algorithm Analysis

Time and space complexities

| 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 | $O(n!n)$ | $O(n)$ |

Running time in milliseconds for different values of $n$

| 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 dyna | 0.555 | 2040 | — | — | — | — |
| brute force | 317.5 | — | — | — | — | — |

# Role Assignment Algorithm Analysis

Time and space complexities

| 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 | $O(n!n)$ | $O(n)$ |

Running time in milliseconds for different values of $n$

| 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 dyna | 0.555 | 2040 | — | — | — | — |
| brute force | 317.5 | — | — | — | — | — |

# Role Assignment Algorithm Analysis

Time and space complexities

| 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 | $O(n!n)$ | $O(n)$ |

Running time in milliseconds for different values of $n$

| 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 dyna | 0.555 | 2040 | — | — | — | — |
| brute force | 317.5 | — | — | — | — | — |

# Related Work


- Chaimowicz, L., Campos, M.F., Kumar, V.: Dynamic role assignment for cooperative robots. (ICRA). (2002)
- Ji, M., Azuma, S.i., Egerstedt, M.B.: Role-assignment in multi-agent coordination. (2006)
- Lau, N., Lopes, L., Corrente, G., Filipe, N.: Multi-robot team coordination through roles, positionings and coordinated procedures. (IROS). (2009)
- Michael, N., Zavlanos, M.M., Kumar, V., Pappas, G.J.: Distributed multi-robot task assignment and formation control. (ICRA). (2008)
- Stone, P., Veloso, M.: Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. Artificial Intelligence 110(2) (June 1999)
- Mellinger, D., Kushleyev, A., Kumar, V.: Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. (ICRA). (2012)
- Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent path finding. In: AAAI. (2012)
- Hokayem, P.F., Spong, M.W., Siljak, D.D.: Cooperative avoidance control for multiagent systems. Urbana 51 (2007) 61801
- Richards, A., How, J.: Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In: American Control Conference. (2002)
- Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2(1-2) (1955)
- Sokkalingam, P., Aneja, Y.P.: Lexicographic bottleneck combinatorial problems. Operations Research Letters 23(1) (1998)

# Related Work

- Chaimowicz, L., Campos, M.F., Kumar, V.: Dynamic role assignment for cooperative robots. (ICRA). (2002)
- Ji, M., Azuma, S.i., Egerstedt, M.B.: Role-assignment in multi-agent coordination. (2006)
- Lau, N., Lopes, L., Corrente, G., Filipe, N.: Multi-robot team coordination through roles, positionings and coordinated procedures. (IROS). (2009)
- Michael, N., Zavlanos, M.M., Kumar, V., Pappas, G.J.: Distributed multi-robot task assignment and formation control. (ICRA). (2008)
- Stone, P., Veloso, M.: Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. Artificial Intelligence 110(2) (June 1999)
- Mellinger, D., Kushleyev, A., Kumar, V.: Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. (ICRA). (2012)
- Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent path finding. In: AAAI. (2012)
- Hokayem, P.F., Spong, M.W., Siljak, D.D.: Cooperative avoidance control for multiagent systems. Urbana 51 (2007) 61801
- Richards, A., How, J.: Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In: American Control Conference. (2002)
- Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2(1-2) (1955)
- Sokkalingam, P., Aneja, Y.P.: Lexicographic bottleneck combinatorial problems. Operations Research Letters 23(1) (1998)

# Related Work


- Chaimowicz, L., Campos, M.F., Kumar, V.: Dynamic role assignment for cooperative robots. (ICRA). (2002)
- Ji, M., Azuma, S.i., Egerstedt, M.B.: Role-assignment in multi-agent coordination. (2006)
- Lau, N., Lopes, L., Corrente, G., Filipe, N.: Multi-robot team coordination through roles, positionings and coordinated procedures. (IROS). (2009)
- Michael, N., Zavlanos, M.M., Kumar, V., Pappas, G.J.: Distributed multi-robot task assignment and formation control. (ICRA). (2008)
- Stone, P., Veloso, M.: Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. Artificial Intelligence 110(2) (June 1999)
- Mellinger, D., Kushleyev, A., Kumar, V.: Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. (ICRA). (2012)
- Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent path finding. In: AAAI. (2012)
- Hokayem, P.F., Spong, M.W., Siljak, D.D.: Cooperative avoidance control for multiagent systems. Urbana 51 (2007) 61801
- Richards, A., How, J.: Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In: American Control Conference. (2002)
- Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2(1-2) (1955)
- Sokkalingam, P., Aneja, Y.P.: Lexicographic bottleneck combinatorial problems. Operations Research Letters 23(1) (1998)

# Related Work


- Chaimowicz, L., Campos, M.F., Kumar, V.: Dynamic role assignment for cooperative robots. (ICRA). (2002)
- Ji, M., Azuma, S.i., Egerstedt, M.B.: Role-assignment in multi-agent coordination. (2006)
- Lau, N., Lopes, L., Corrente, G., Filipe, N.: Multi-robot team coordination through roles, positionings and coordinated procedures. (IROS). (2009)
- Michael, N., Zavlanos, M.M., Kumar, V., Pappas, G.J.: Distributed multi-robot task assignment and formation control. (ICRA). (2008)
- Stone, P., Veloso, M.: Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. Artificial Intelligence 110(2) (June 1999)
- Mellinger, D., Kushleyev, A., Kumar, V.: Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. (ICRA). (2012)
- Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent path finding. In: AAAI. (2012)
- Hokayem, P.F., Spong, M.W., Siljak, D.D.: Cooperative avoidance control for multiagent systems. Urbana 51 (2007) 61801
- Richards, A., How, J.: Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In: American Control Conference. (2002)
- Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2(1-2) (1955)
- Sokkalingam, P., Aneja, Y.P.: Lexicographic bottleneck combinatorial problems. Operations Research Letters 23(1) (1998)

# Related Work


- Chaimowicz, L., Campos, M.F., Kumar, V.: Dynamic role assignment for cooperative robots. (ICRA). (2002)
- Ji, M., Azuma, S.i., Egerstedt, M.B.: Role-assignment in multi-agent coordination. (2006)
- Lau, N., Lopes, L., Corrente, G., Filipe, N.: Multi-robot team coordination through roles, positionings and coordinated procedures. (IROS). (2009)
- Michael, N., Zavlanos, M.M., Kumar, V., Pappas, G.J.: Distributed multi-robot task assignment and formation control. (ICRA). (2008)
- Stone, P., Veloso, M.: Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. Artificial Intelligence 110(2) (June 1999)
- Mellinger, D., Kushleyev, A., Kumar, V.: Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. (ICRA). (2012)
- Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent path finding. In: AAAI. (2012)
- Hokayem, P.F., Spong, M.W., Siljak, D.D.: Cooperative avoidance control for multiagent systems. Urbana 51 (2007) 61801
- Richards, A., How, J.: Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In: American Control Conference. (2002)
- Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2(1-2) (1955)
- Sokkalingam, P., Aneja, Y.P.: Lexicographic bottleneck combinatorial problems. Operations Research Letters 23(1) (1998)

# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between:

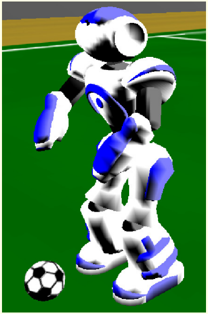
1. Finding minimal-maximum edge in perfect matching
2. Computing minimum sum 0-1 edge weight matchings (using the Hungarian algorithm)

Full details of algorithm are explained in our paper

Time: O($n^4$)

# 2013 RoboCup 3D Simulation Domain

- Teams of 11 vs 11 autonomous agents play soccer
- Realistic physics using Open Dynamics Engine (ODE)
- Agents modeled after Aldebaran Nao robot
- Agent receives noisy visual information about environment
- Agents can communicate with each other over limited bandwidth channel



# RoboCup 3D Role Assignment Function Evaluation

Average goal difference across 1000 games against the top 3 teams at RoboCup 2013

| Function | 1. Apollo3d | 2. UT Austin Villa | 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) |

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

**Greedy:** Assign robots to targets in order of shortest distances.

**Greedy Offense:** Similar to previous work in the 3D sim domain, assign closest robots to roles in order of most offensive positions.

# *CM Validation* of Role Assignment Function MMDR

- Minimizes the longest distance (Property 1) as lexicographical ordering of distance tuples sorted in descending order ensures this.
- Triangle inequality will prevent two agents in a mapping from colliding (Property 2)
- MMDR is dynamically consistent

Proof sketches of the above three properties are given in the appendix of the paper

## *CM Validation* of MMD+MSD$^2$ Role Assignment Function

- MMD+MSD$^2$ minimizes the longest distance traveled by any agent (Property 1) as we are only considering perfect matchings with minimal longest edges

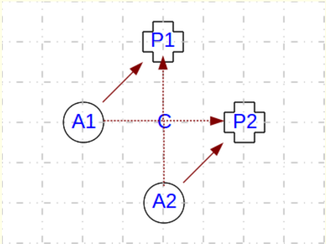
- Triangle inequality will prevent two agents in a mapping from colliding (Property 2)

- MMD+MSD$^2$ is not dynamically consistent (Property 3) as distances squared do not decrease at a constant rate

Proof sketches of the above three properties are given in the appendix of the paper

## *CM Validation* of Role Assignment Function MMDR

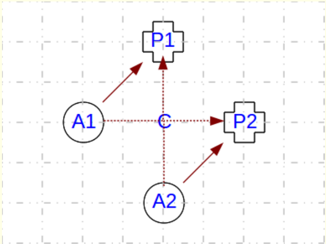
- Minimizes the longest distance (Property 1) as lexicographical ordering of distance tuples sorted in descending order ensures this.
- Triangle inequality will prevent two agents in a mapping from colliding (Property 2), as switching the two agents' targets reduces the maximum distance either must travel.




- MMDR is dynamically consistent (Property 3) as, under assumption all agents move toward their targets at the same constant rate, lowest cost lexicographical ordering of chosen mapping is preserved because distances between any agent and target will not decrease any faster than the distance between an agent and the target it is assigned to.

# *CM Validation* of MMD+MSD$^2$ Role Assignment Function

- MMD+MSD$^2$ minimizes the longest distance traveled by any agent (Property 1) as we are only considering perfect matchings with minimal longest edges
- Triangle inequality will prevent two agents in a mapping from colliding (Property 2), as switching the two agents' targets reduces, but never increases, the distance one or both must travel thereby reducing the sum of distances squared.



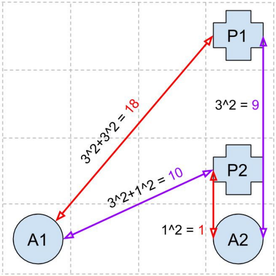
## MMD+MSD$^2$ Dynamic Consistency

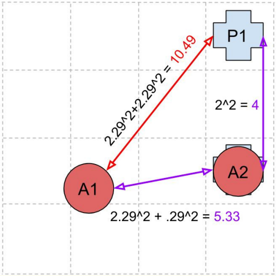
MMD+MSD$^2$ is not dynamically consistent (Property 3) as distances squared do not decrease at a constant rate, but in fact decrease at faster rates for longer distances. This allows for the distance between an agent and target that the agent is not assigned to travel toward to decrease faster than the distance to 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.

Example:

Moving from 5 meters to 4 meters: $5^2 - 4^2 = 9$

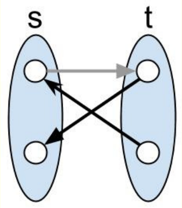
Moving from 4 meters to 3 meters: $4^2 - 3^2 = 7$



## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).




Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: $O(n^2)$ breadth-first search X $n$ edges = $O(n^3)$
Space: Breadth-first search of Ford-Fulkerson = $O(n^2)$

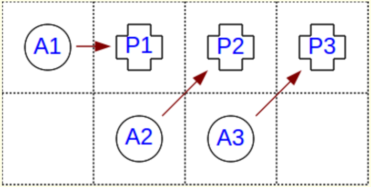
# Recursive Property of Role Assignment Function MMDR

**Theorem**

*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 $MMDR(A_0, P_0)$.*

- *Translation:* Any subset of a lowest cost mapping is itself a lowest cost mapping
- If within any subset of a mapping a lower cost mapping is found, then the cost of the complete mapping can be reduced by augmenting the complete mapping with that of the subset's lower cost mapping

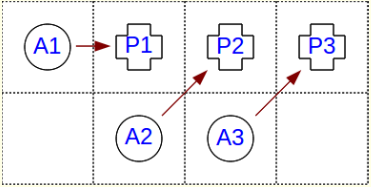
# Dynamic Programming Algorithm for Role Assignment Function MMDR




| {P1} | {P2,P1} | {P3,P2,P1} |
|---|---|---|
| | | |

- Begin evaluating mappings of 1 agent and build up to $n$ agents
- Only evaluate mappings built from subset mappings returned by MMDR
- Evaluates $n2^{n-1}$ mappings
- Time complexity = $O(n^2 2^{(n-1)})$, space complexity = $O(\, n\binom{n}{n/2})$

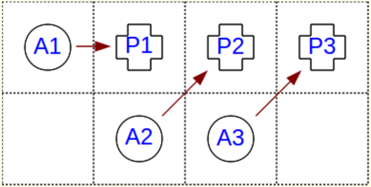
# Dynamic Programming Algorithm for Role Assignment Function MMDR




| {P1} | {P2,P1} | {P3,P2,P1} |
|---|---|---|
| A1→P1<br>A2→P1<br>A3→P1 | | |

- Begin evaluating mappings of 1 agent and build up to $n$ agents
- Only evaluate mappings built from subset mappings returned by MMDR
- Evaluates $n2^{n-1}$ mappings
- Time complexity = $O(n^2 2^{(n-1)})$, space complexity = $O(\, n\binom{n}{n/2})$

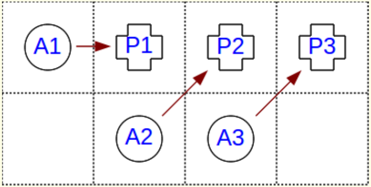
# Dynamic Programming Algorithm for Role Assignment Function MMDR




| {P1} | {P2,P1} | {P3,P2,P1} |
|---|---|---|
| A1→P1 | A1→P2, MMDR(A2→P1) | |
| A2→P1 | A1→P2, MMDR(A3→P1) | |
| A3→P1 | A2→P2, MMDR(A1→P1) | |
| | A2→P2, MMDR(A3→P1) | |
| | A3→P2, MMDR(A1→P1) | |
| | A3→P2, MMDR(A2→P1) | |

- Begin evaluating mappings of 1 agent and build up to $n$ agents
- Only evaluate mappings built from subset mappings returned by MMDR
- Evaluates $n2^{n-1}$ mappings
- Time complexity = $O(n^2 2^{(n-1)})$, space complexity = $O(\, n\binom{n}{n/2})$

# Dynamic Programming Algorithm for Role Assignment Function MMDR




| {P1} | {P2,P1} | {P3,P2,P1} |
|---|---|---|
| A1→P1 | A1→P2, MMDR(A2→P1) | A1→P3, MMDR({A2,A3}→{P1,P2}) |
| A2→P1 | A1→P2, MMDR(A3→P1) | A2→P3, MMDR({A1,A3}→{P1,P2}) |
| A3→P1 | A2→P2, MMDR(A1→P1) | A3→P3, MMDR({A1,A2}→{P1,P2}) |
| | A2→P2, MMDR(A3→P1) | |
| | A3→P2, MMDR(A1→P1) | |
| | A3→P2, MMDR(A2→P1) | |

- Begin evaluating mappings of 1 agent and build up to $n$ agents
- Only evaluate mappings built from subset mappings returned by MMDR
- Evaluates $n2^{n-1}$ mappings
- Time complexity = $O(n^2 2^{(n-1)})$, space complexity = $O(\, n\binom{n}{n/2})$

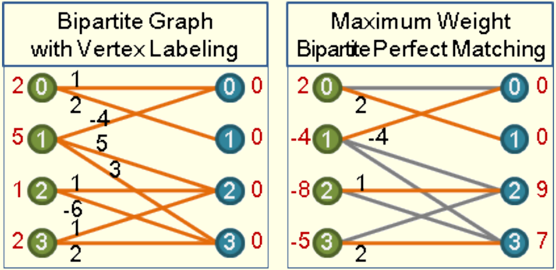
# Dynamic Programming Algorithm for Role Assignment

HashMap $bestRoleMap = \varnothing$
$Agents = \{a_1, ..., a_n\}$
$Positions = \{p_1, ..., p_n\}$
**for** $k = 1$ to $n$ **do**
  **for all** $a$ in $Agents$ **do**
    $S = \binom{n-1}{k-1}$ sets of $k-1$ agents from $Agents - \{a\}$
    **for all** $s$ in $S$ **do**
      Mapping $m_0 = bestRoleMap[s]$
      Mapping $m = (a \rightarrow p_k) \cup m_o$
      $bestRoleMap[\{a\} \cup s] = mincost(m, bestRoleMap[\{a\} \cup s])$
**return** $bestRoleMap[Agents]$

As $\binom{n-1}{k-1}$ agent subset mapping combinations are evaluated for mappings of each agent assigned to the *kth* position, the total number of mappings computed for each of the $n$ agents is thus equivalent to the sum of the $n-1$ binomial coefficients. That is,

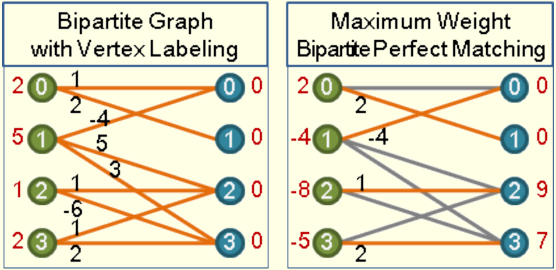
$$\sum_{k=1}^{n} \binom{n-1}{k-1} = \sum_{k=0}^{n-1} \binom{n-1}{k} = 2^{n-1}$$

## Hungarian Algorithm




- Finds a maximum weight perfect matching in a bipartite graph (solves the *assignment problem*)
- Runs in $O(n^3)$ time
- Potential function: $pot(s) + pot(t) \leq cost(e_{s,t})$
- Perfect matching consists of tight edges: $pot(s) + pot(t) = cost(e_{s,t})$
- Perfect matching $M$: $\sum_{v \in M} pot(v) = \sum_{e \in M} cost(e)$

# Hungarian Algorithm




- Finds a maximum weight perfect matching in a bipartite graph (solves the *assignment problem*)
- Runs in O($n^3$) time
- Potential function: $pot(s) + pot(t) \leq cost(e_{s,t})$
- Perfect matching consists of tight edges: $pot(s) + pot(t) = cost(e_{s,t})$
- Perfect matching $M$: $\sum_{v \in M} pot(v) = \sum_{e \in M} cost(e)$

**Can we transform MMDR (*lexicographic bottleneck assignment problem*) into the *assignment problem*?**

## Attempt at MMDR O($n^4$) Algorithm

Use Minimal-maximum Edge Perfect Matching Algorithm to recursively find each maximum edge in a perfect matching of graphs with edges having weight less than the last minimal-maximum edge weight found.

LOOP $n$ times:
1. Find minimal-maximum edge $e$ in perfect matching with weight $w$
2. Save edge $e$ and remove its endpoints from graph 3. Remove all edges with weight greater than $w$ from graph

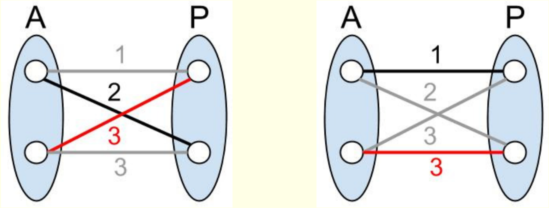
Time: O($n^3$) Min-max Edge Perfect Matching Alg. X $n$ edges = O($n^4$)

## Attempt at MMDR $O(n^4)$ Algorithm

Use Minimal-maximum Edge Perfect Matching Algorithm to recursively find each maximum edge in a perfect matching of graphs with edges having weight less than the last minimal-maximum edge weight found.

LOOP *n* times:
1. Find minimal-maximum edge *e* in perfect matching with weight *w*
2. Save edge *e* and remove its endpoints from graph 3. Remove all edges with weight greater than *w* from graph

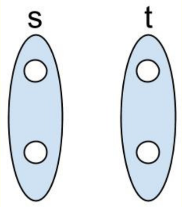
Time: $O(n^3)$ Min-max Edge Perfect Matching Alg. X *n* edges = $O(n^4)$




Can fail when there are multiple perfect matchings with the same maximum edge weight!

## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



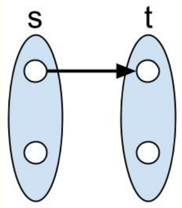

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: O($n^2$) breadth-first search X $n$ edges = O($n^3$)
Space: Breadth-first search of Ford-Fulkerson = O($n^2$)

## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



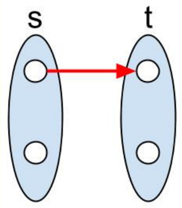

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: $O(n^2)$ breadth-first search X $n$ edges = $O(n^3)$
Space: Breadth-first search of Ford-Fulkerson = $O(n^2)$

# Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



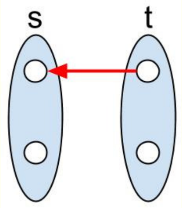

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: $O(n^2)$ breadth-first search X $n$ edges = $O(n^3)$
Space: Breadth-first search of Ford-Fulkerson = $O(n^2)$

## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



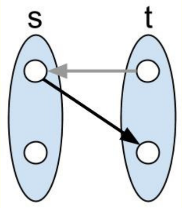

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: O($n^2$) breadth-first search X $n$ edges = O($n^3$)
Space: Breadth-first search of Ford-Fulkerson = O($n^2$)

## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



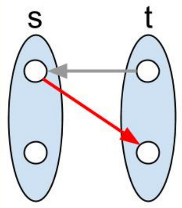

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: $O(n^2)$ breadth-first search X $n$ edges = $O(n^3)$
Space: Breadth-first search of Ford-Fulkerson = $O(n^2)$

# Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



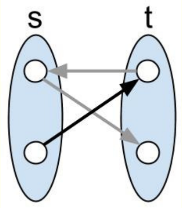

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: O($n^2$) breadth-first search X $n$ edges = O($n^3$)
Space: Breadth-first search of Ford-Fulkerson = O($n^2$)

## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



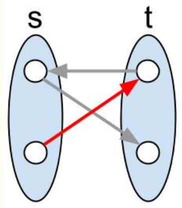

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: O($n^2$) breadth-first search X $n$ edges = O($n^3$)
Space: Breadth-first search of Ford-Fulkerson = O($n^2$)

# Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



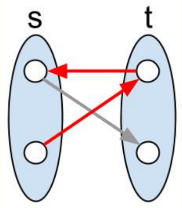

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: $O(n^2)$ breadth-first search X $n$ edges = $O(n^3)$
Space: Breadth-first search of Ford-Fulkerson = $O(n^2)$

## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



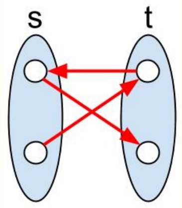

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: O($n^2$) breadth-first search X $n$ edges = O($n^3$)
Space: Breadth-first search of Ford-Fulkerson = O($n^2$)

## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



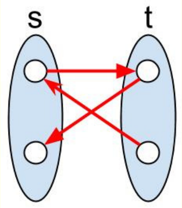

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: $O(n^2)$ breadth-first search X $n$ edges = $O(n^3)$
Space: Breadth-first search of Ford-Fulkerson = $O(n^2)$

## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



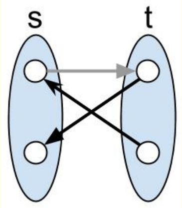

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: $O(n^2)$ breadth-first search X $n$ edges = $O(n^3)$
Space: Breadth-first search of Ford-Fulkerson = $O(n^2)$

## Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).



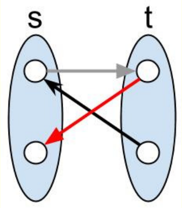

Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: $O(n^2)$ breadth-first search X $n$ edges = $O(n^3)$
Space: Breadth-first search of Ford-Fulkerson = $O(n^2)$

# Minimal-maximum Edge Perfect Matching Algorithm

Find perfect matching with minimum longest edge across all perfect matchings (*bottleneck assignment problem*).




Ford-Fulkerson algorithm finds max cardinality matching with augmenting paths

1. Sort edges in ascending order of distance
2. Do until a perfect matching has been found
   - Add next edge with lowest distance to bipartite graph
   - Run breadth-first search of Ford-Fulkerson

Time: $O(n^2)$ breadth-first search X $n$ edges = $O(n^3)$
Space: Breadth-first search of Ford-Fulkerson = $O(n^2)$

# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{ |e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$$numLongestEdges := \sum_{e \in matching} cost(e)$$

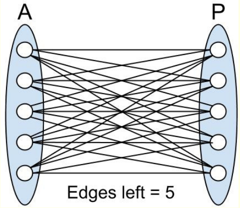
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from *Edges*

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{ |e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$$numLongestEdges := \sum_{e \in matching} cost(e)$$

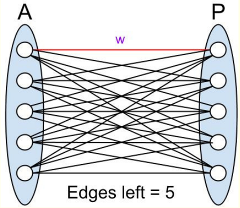
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from *Edges*

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{ |e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$$numLongestEdges := \sum_{e \in matching} cost(e)$$

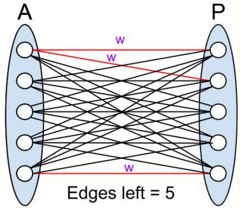
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from $Edges$

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR $O(n^4)$ Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \left\{ |e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty \right.$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$$numLongestEdges := \sum_{e \in matching} cost(e)$$

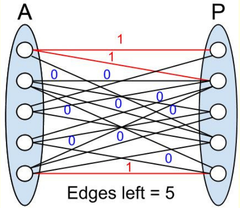
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from *Edges*

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR $O(n^4)$ Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

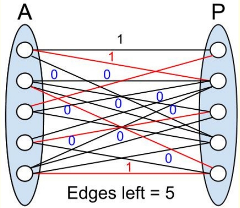
$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where
$\forall e \in Edges \{|e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$
3. Compute number of edges left to find based on number of edges of length $w$ in $M$
$numLongestEdges := \sum_{e \in matching} cost(e)$
$numEdgesLeft := numEdgesLeft - numLongestEdges$
If $numEdgesLeft = 0$ return $M$
4. Remove non-tight edges from *Edges*

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{ |e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$$numLongestEdges := \sum_{e \in matching} cost(e)$$

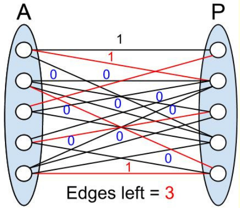
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from *Edges*

Insight: All perfect matchings of tight edges have exactly length $w$ *numLongestEdges*

5. Set weights of new found longest edges to -1



# MMDR $O(n^4)$ Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{ |e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$$numLongestEdges := \sum_{e \in matching} cost(e)$$

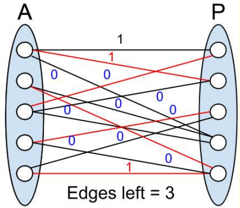
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from *Edges*

Insight: All perfect matchings of tight edges have exactly length $w$ *numLongestEdges*

5. Set weights of new found longest edges to -1



# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where $\forall e \in Edges \{ |e| < \mathrm{w}: \quad cost(e) := 0; |e| = \mathrm{w}: \quad cost(e) := 1; |e| > \mathrm{w}: \quad cost(e) := \infty$
3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$numLongestEdges := \sum_{e \in matching} cost(e)$

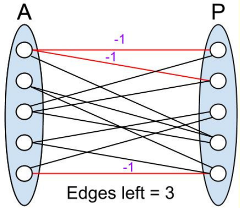
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from $Edges$

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

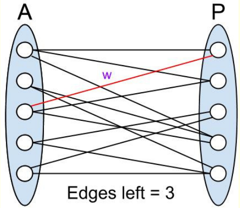
$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where
$\forall e \in Edges \{|e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$
3. Compute number of edges left to find based on number of edges of length $w$ in $M$
$$numLongestEdges := \sum_{e \in matching} cost(e)$$
$numEdgesLeft := numEdgesLeft - numLongestEdges$
If $numEdgesLeft = 0$ return $M$
4. Remove non-tight edges from $Edges$

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{|e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$$numLongestEdges := \sum_{e \in matching} cost(e)$$

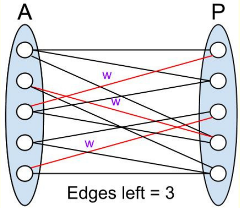
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from $Edges$

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{ |e| < w: \quad cost(e) := 0; |e| = w: \quad cost(e) := 1; |e| > w: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$$numLongestEdges := \sum_{e \in matching} cost(e)$$

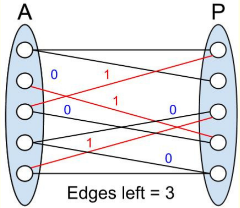
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from $Edges$

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{|e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$$numLongestEdges := \sum_{e \in matching} cost(e)$$

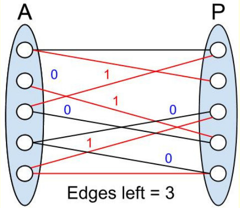
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from *Edges*

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



# MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{|e| <\text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$numLongestEdges := \sum_{e \in matching} cost(e)$

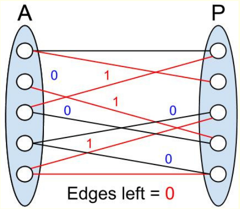
$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from *Edges*

Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1



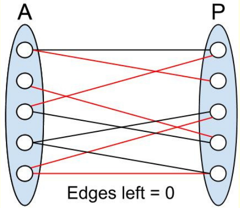
# MMDR $O(n^4)$ Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where
$\forall e \in Edges \{ |e| < \mathrm{w}: \quad cost(e) := 0; |e| = \mathrm{w}: \quad cost(e) := 1; |e| > \mathrm{w}: \quad cost(e) := \infty$
3. Compute number of edges left to find based on number of edges of length $w$ in $M$
$numLongestEdges := \sum_{e \in matching} cost(e)$
$numEdgesLeft := numEdgesLeft - numLongestEdges$
If $numEdgesLeft = 0$ return $M$
4. Remove non-tight edges from $Edges$
Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$
5. Set weights of new found longest edges to -1



## MMDR O($n^4$) Algorithm [Sokkalingam and Aneja 1998]

Alternate between finding minimal-maximum edge in perfect matching and computing minimum sum 0-1 edge weight matchings to find number of minimal-maximum edges.

$numEdgesLeft := n$

LOOP:

1. Find minimal-maximum edge in perfect matching with length $w$
2. Compute 0-1 weight max sum perfect matching $M$ with Hungarian algorithm where

$\forall e \in Edges \{ |e| < \text{w}: \quad cost(e) := 0; |e| = \text{w}: \quad cost(e) := 1; |e| > \text{w}: \quad cost(e) := \infty$

3. Compute number of edges left to find based on number of edges of length $w$ in $M$

$numLongestEdges := \sum_{e \in matching} cost(e)$

$numEdgesLeft := numEdgesLeft - numLongestEdges$

If $numEdgesLeft = 0$ return $M$

4. Remove non-tight edges from *Edges*

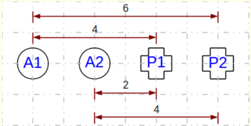
Insight: All perfect matchings of tight edges have exactly length $w$ $numLongestEdges$

5. Set weights of new found longest edges to -1

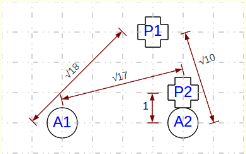
Time: (O($n^3$) Min-max Edge Alg. + O($n^3$) Hung. Alg.) X $n$ edges = O($n^4$)

Space: Breadth-first search of Ford-Fulkerson = O($n^2$)

# Other Role Assignment Functions




**Minimum Sum Distance (MSD)**
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.




**Minimum Sum Distance Squared (MSD$^2$)**
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. MMDR will choose the mapping with the greater sum as its maximum path distance is $\sqrt{17}$ which is less than the other mapping's maximum path distance of $\sqrt{18}$.

# Voting Coordination System

- Each agent broadcasts ball position, own position, and suggested role mapping during allotted time slot
- Sliding window stored of mappings received over last $n$ time slots evaluated and mapping with the most number of votes is chosen
- If two mappings both have greatest number of votes then tie breaker goes to mapping with most recent vote received

# Voting Coordination System

- Each agent broadcasts ball position, own position, and suggested role mapping during allotted time slot
- Sliding window stored of mappings received over last *n* time slots evaluated and mapping with the most number of votes is chosen
- If two mappings both have greatest number of votes then tie breaker goes to mapping with most recent vote received
- Synchronization: With voting system = 100%, without = 36%

# RoboCup 2D Drop-In Player Game


Yellow players 4 and 11 from UTAustinVilla use SCRAM (MMD+MSD$^2$)

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.