# Predator Prey

Robert Lass
May 18th, 2011
Robot Lab

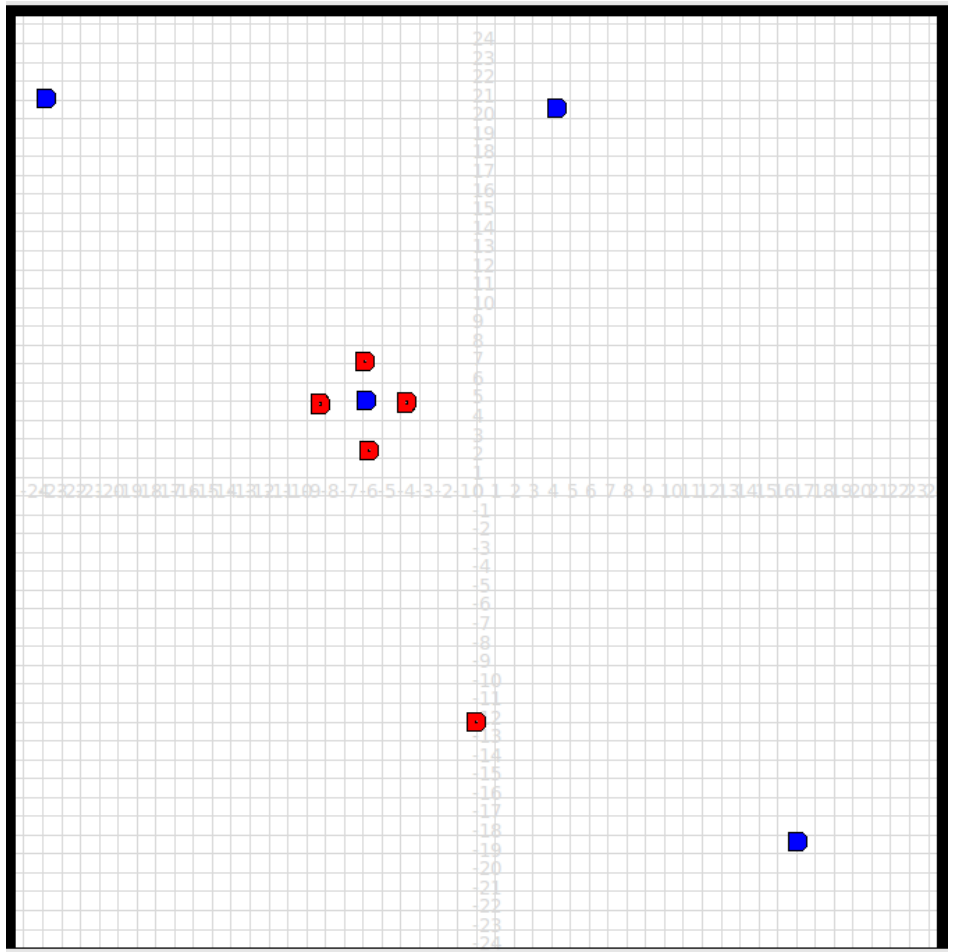
# The Predator vs Prey Problem

- There are predators.
- There are prey.
- The are situated in some world.
- The predators want to capture or kill the prey.
- Example: a pair of coyotes hunting antelope.

# Predator vs Prey (continued)

- The variation of this problem we will study is as follows:
  - Four predators are required to capture one prey.
  - One predator each must be north, south, east, and west of the prey to capture it.
  - The predators must be within a certain distance of the prey.
  - Each predators sensors can only “see” the part of the world nearby.

# Example



# Group Behavior

- As you learned last week, this is more complex than single robot problems.
- This problem requires teamwork between predators to capture prey.
  - Find the prey (individual);
  - Convey prey location to teammates (group)
  - Role allocation: which predator will stand in each capture position? (group)

# Uncertainty

- The predators cannot see the prey unless they are close by.
- Prey can move.
- A predator who is told where a prey is may have been given stale information.
- Lossy communications environment.

# What are the pieces of this problem?

- Finding prey;
- Tracking prey;
- Coordinating tasks among predators;

# Assignment Four: Predator Prey

- What is the problem;
- What are the steps to solving the problem;
- What are some ways to solve each of the steps;
- Where to look for more help.

# Finding Prey

- Coordinated Search;
- Robots can divide up the map to search more quickly;
- When four robots are capturing a prey, perhaps the fifth could search for more prey?

# Tracking Prey

- Prey has been located:
  - Prey move;
  - Need to track prey or they will need to be found again;
  - Need to follow them in order to capture them.
- Suggestion:
  - Track the prey, moving towards the assigned location (NSEW).
  - When they prey stop, should be able to capture them.

# Problems in Common

- Both finding and capturing prey have the same properties:
  - Multiple predators;
  - Problem can be divided into pieces;
- One way to handle this is a *task assignment* algorithm;
- Many ways to do this, I will cover two of them: the Hungarian algorithm and the distributed stochastic algorithm.

# Hungarian Algorithm

- Also called Munkres algorithm;
- Was first published in 1890s, but was ignored and independently rediscovered in the 1950s
- Can run as fast as O(N^3), depending on implementation.

# Hungarian Algorithm Example

- We have three robots.
- We have three tasks.
- Robot 1 has a cost of 3, 2, and 3 to do jobs 1, 2, and 3 respectively.
- Robot 2 has a cost of 1, 2, and 0  to do  jobs 1, 2, and 3 respectively.
- Robot 3 has a cost of 3, 2, and 1 to do jobs 1, 2, and 3 respectively.
- Construct a cost matrix.

| | | |
|---|---|---|
| 3 | 1 | 3 |
| 2 | 2 | 2 |
| 3 | 0 | 1 |

# Hungarian Algorithm: Step Five

- Step Five:
  - Do
    - Go to the starred zero in the column of the uncovered primed zero from step four.
    - Find the primed zero in the row of this starred zero.
  - While there is no starred zero in the primed zero's column.
  - Unstar all the starred zeros you visited in the previous loop, star all the primed zeros you visited, erase all the primes, and uncover every line in the matrix.

# Hungarian Algorithm Description

- Step 1:
  - Find each row's minimum element.
  - Subtract it from all row elements.
- Step 2:
  - Star every zero in the matrix with no starred zeros in the same row or column.
- Step 3:
  - Cover all columns with a starred zero.
  - If all columns have a starred zero, you are done.
- Step 4:
  - While uncovered zeros exist:
    - Prime a non-covered zero.
    - If no starred zero in this row, go to Step 5.
    - Cover row & uncover column with the starred 0
  - Go to Step 6.
- Step 5:
  - *see next slide*
- Step 6:
  - Add the smallest uncovered value to all covered rows, and subtract it from all uncovered columns.

- Adapted from: http://www.cs.duke.edu/brd/Teaching/Bio/asmb/current/Handouts/munkres.html

# Hungarian Algorithm Example

| | | |
|---|---|---|
| 3 | 1 | 3 |
| 2 | 2 | 2 |
| 3 | 0 | 1 |

Step One

| | | |
|---|---|---|
| 2 | 0 | 2 |
| 0 | 0 | 0 |
| 3 | 0 | 1 |

Step Two

| | | |
|---|---|---|
| 2 | 0* | 2 |
| 0* | 0 | 0 |
| 3 | 0 | 1 |

Step Three

| | | |
|---|---|---|
| 2 | 0* | 2 |
| 0* | 0 | 0 |
| 3 | 0 | 1 |

Step Four

| | | |
|---|---|---|
| 2 | 0* | 2 |
| 0* | 0 | 0' |
| 3 | 0 | 1 |

Step Four

| | | |
|---|---|---|
| 1 | 0* | 1 |
| 0* | 1 | 0' |
| 2 | 0 | 0' |

Step Five

| | | |
|---|---|---|
| 1 | 0* | 1 |
| 0* | 1 | 0' |
| 2 | 0 | 0* |

Step Six

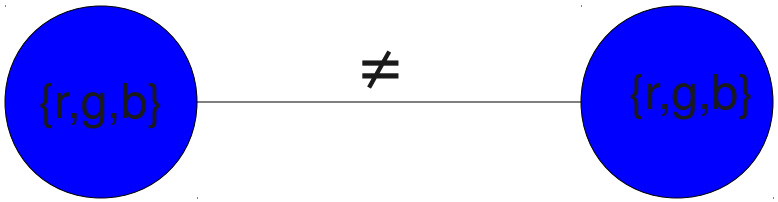
| | | |
|---|---|---|
| 1 | 0* | 1 |
| 0* | 1 | 0' |
| 2 | 0 | 0 |

# Implementation Issues

- Number of rows and columns is not equal:
  - Add dummy rows or columns.
- Decentralized, who should compute the assignments:
  - Centralize the problem;
  - Everyone computes the solution.
- Problem is changing;
  - Recompute “frequently.”
- Robots may have disjoint views.
  - Recomputing and “recommunicating” may help here too.

# Constraint Satisfaction

- A set of *variables* which must be assigned *values* that satisfy *constraints*.
- In general, NP-Complete.
- Goal of CS algorithm is to assign values that satisfy all constraints.

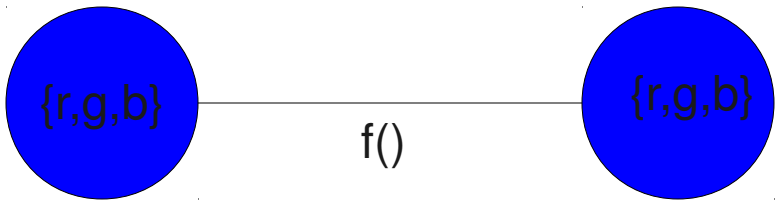


# Constraint Satisfaction

- Examples of constraints are “equals” or “not equals”.
- Any set of valid states over a constraint relation:
- e.g.: {(r,g), (r,b), (g,r), (g,b), (b,r), (b,g)}

# Constraint Optimization

- In constraint satisfaction, constraints specify valid joint value assignments.
- In constraint optimization, constraints specify a *cost* for joint value assignments.
- Goal is to minimize the cost.

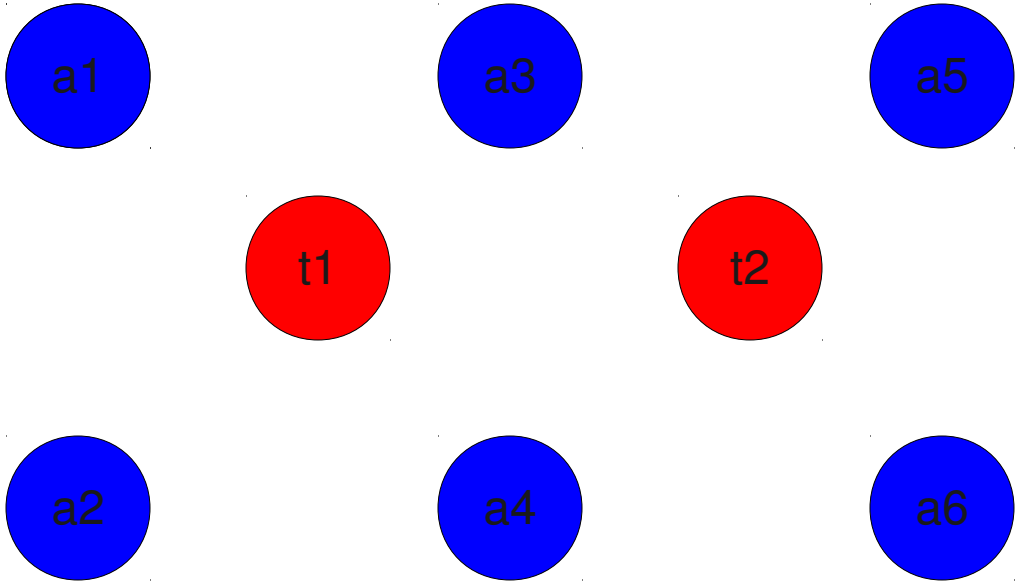



| | r | g | b |
|---|---|---|---|
| r | 0 | 19 | 2 |
| g | 1 | 12 | 0 |
| b | 17 | 4 | 1 |

# Constraint Optimization

- Many problems in robotics can be represented as constraint reasoning problems:
  - Sensors networks and tracking,
  - Task allocation,
  - Scheduling,
  - Resource Allocation.

# Example: Target Tracking




a1, a2, a3, a4: {(t1, t1, t1, *), (t1, t1, *, t1), (t1, *, t1, t1), (*, t1, t1, t1)}

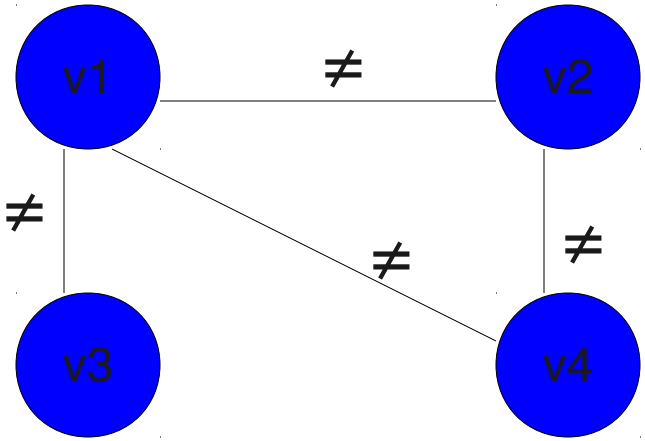
a3, a4, a5, a6: {(t2, t2, t2, *), (t2, t2, *, t2), (t2, *, t2, t2), (*, t2, t2, t2)}

# Constraint Reasoning

- Many problems in robotics can be represented as constraint reasoning problems:
  - Sensors networks and tracking,
  - Task allocation,
  - Scheduling,
  - Resource Allocation.

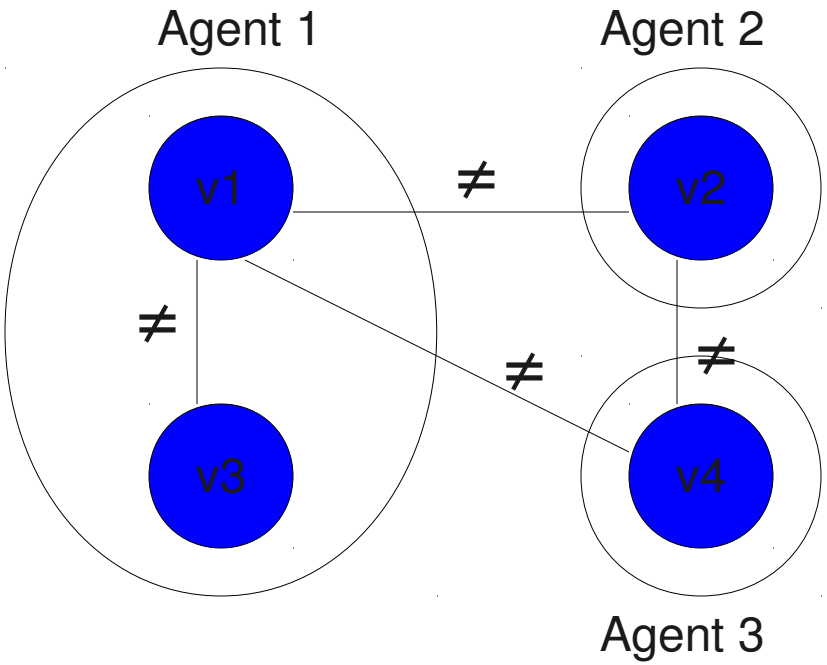
# Distributed Constraint Reasoning

- Like before, we have variables and constraints.



# Distributed Constraint Reasoning

- We add the concept of agents, who control value assignment and may be distributed across multiple variables.



# Why Use DCR?

- Well defined theoretical framework
- Algorithms that are complete or have performance bounds
- Can borrow many examples from constraint reasoning community
- Assumes that agents are cooperative

# Algorithms

- Complete Algorithms:
  - Asynchrounous Distributed OPTimization (ADOPT)
  - Dynamic Programming Optimization Protocol (DPOP)
  - Optimal Asynchronous Partial Overlay (OptAPO)
  - No-Commit Branch and Bound (NCBB)
- Why not use these for everything?

# Distributed Stochastic Search Algorithm

- Local search technique.
- Hill climbing.
- No quality guarantees.
- Can be used as an anytime algorithm.

# DSA

**Algorithm 1** Sketch of DSA, executed by all agents.

```
Randomly choose a value
while (no termination condition is met) do
    if (a new value is assigned) then
        send the new value to neighbors
    end if
    collect neighbors' new values, if any
    select and assign the next value (See Table 1)
end while
```

| Algo. | $\Delta > 0$ | C, $\Delta = 0$ | no C, $\Delta = 0$ |
|---|---|---|---|
| DSA-A | $v$ with $p$ | - | - |
| DSA-B | $v$ with $p$ | $v$ with $p$ | - |
| DSA-C | $v$ with $p$ | $v$ with $p$ | $v$ with $p$ |
| DSA-D | $v$ | $v$ with $p$ | - |
| DSA-E | $v$ | $v$ with $p$ | $v$ with $p$ |

en from “Distributed Stochastic Search for Constraint Satisfaction and Optimizat
allelism, Phase Transitions, and Performance” by Zhang, et al

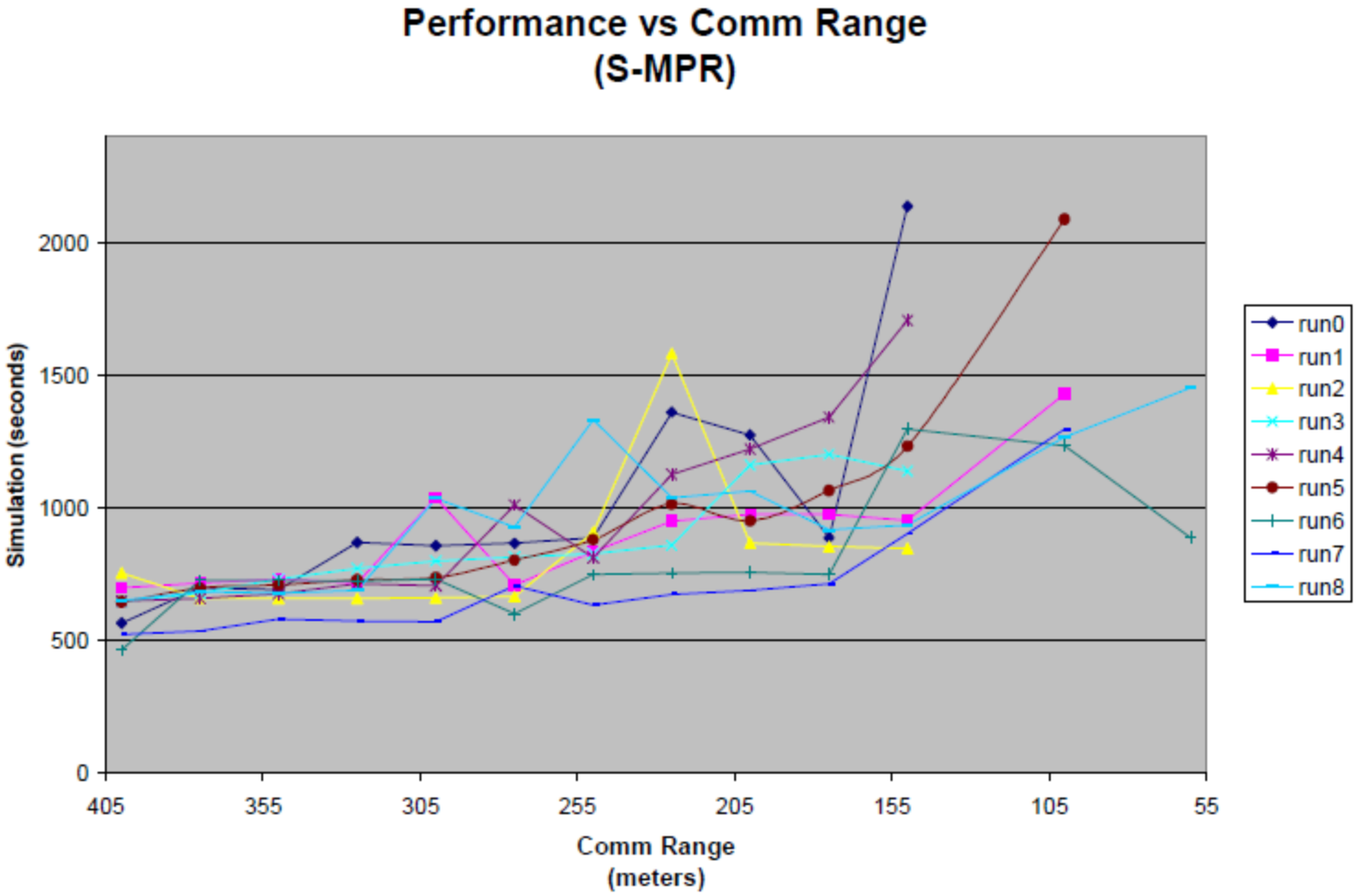
# DSA

- The problem is really a dynamic series of problems:
  - DSA can keep solving new problems, if the agents constantly evaluate their value functions.
  - Because of this, DSA can adapt to changing environments.

# Communication

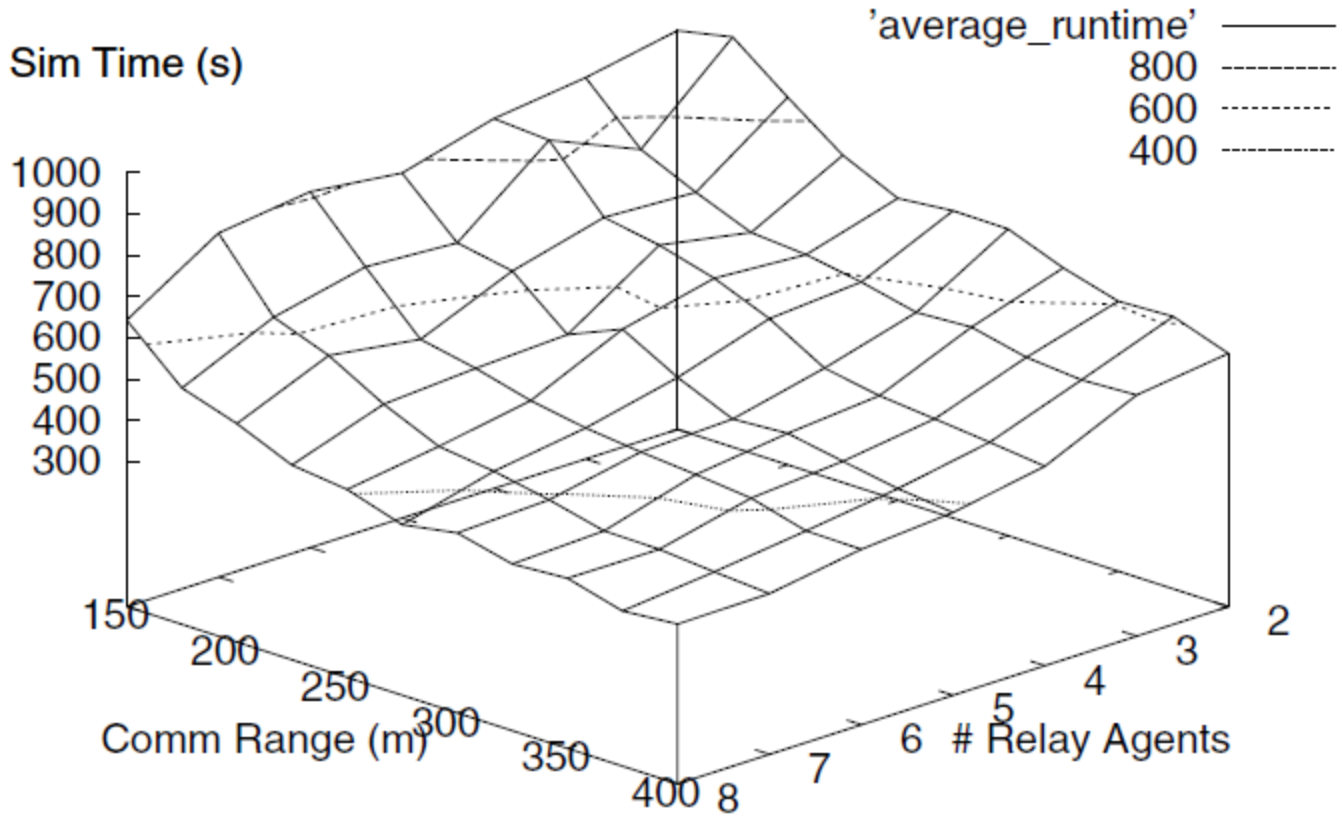
- Hungarian and DSA both require communication.
- How does communication affect the solution quality?
- How well could an algorithm with no communications perform?

# Communication's Influence on Performance




- Chart source: “A Study of Mulitagent System Operation within Dynamical Ad-hoc Networks” by Dean, et al, Milcom 2008

# Communication's Influence on Performance




- Chart source: “A Study of Mulitagent System Operation within Dynamical Ad-hoc Networks” by Dean, et al, Milcom 2008

THE END