Introduction to Multi-Agent Programming

9. Working together

Coalitions, Voting, and Roles

Alexander Kleiner, Bernhard Nebel

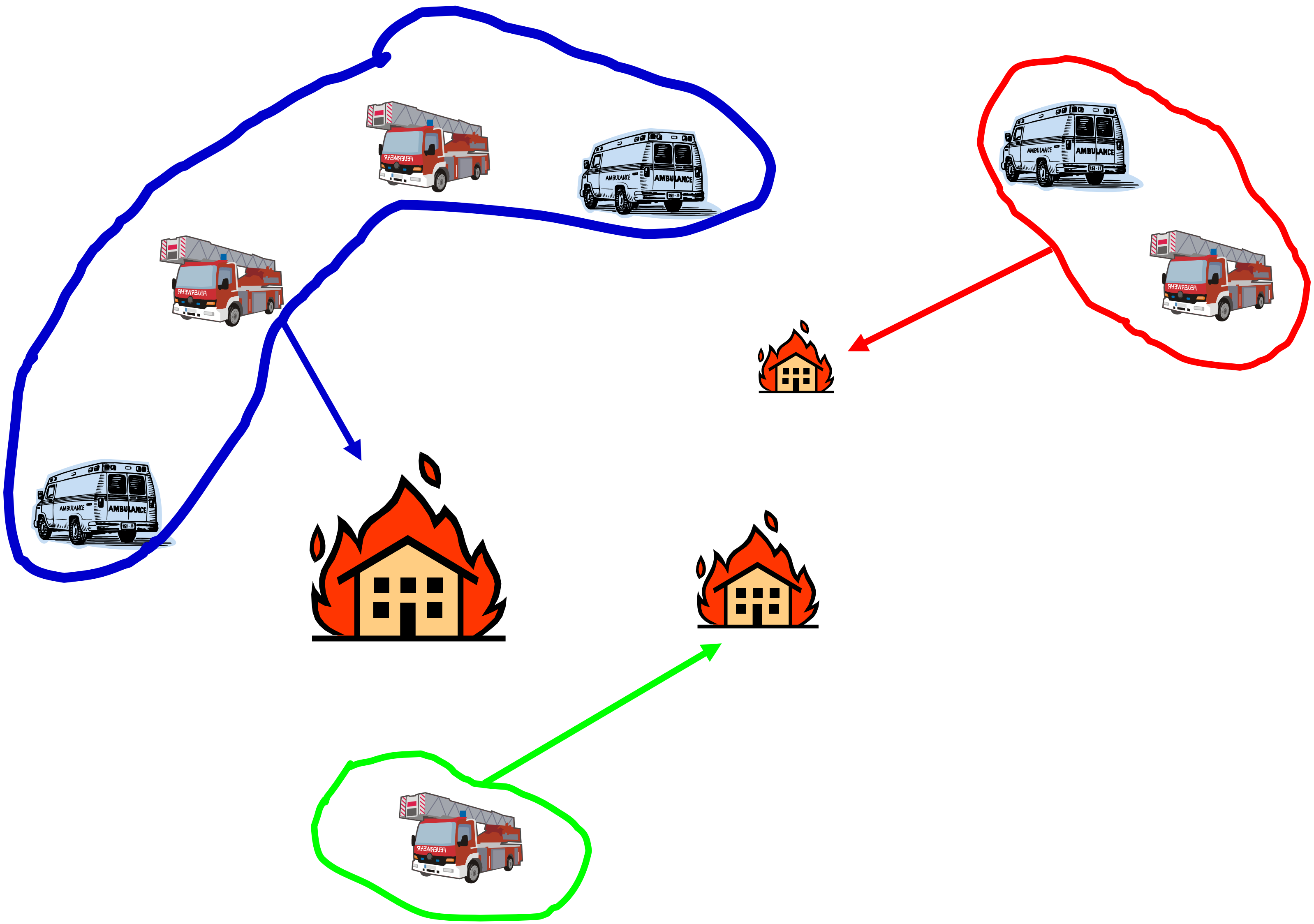
Contents

- Introduction
- Coalition formation
  - Case study: ResQ Freiburg task allocation
- Voting
- Dynamic Role Assignment
  - Case study: CS-Freiburg
- Summary

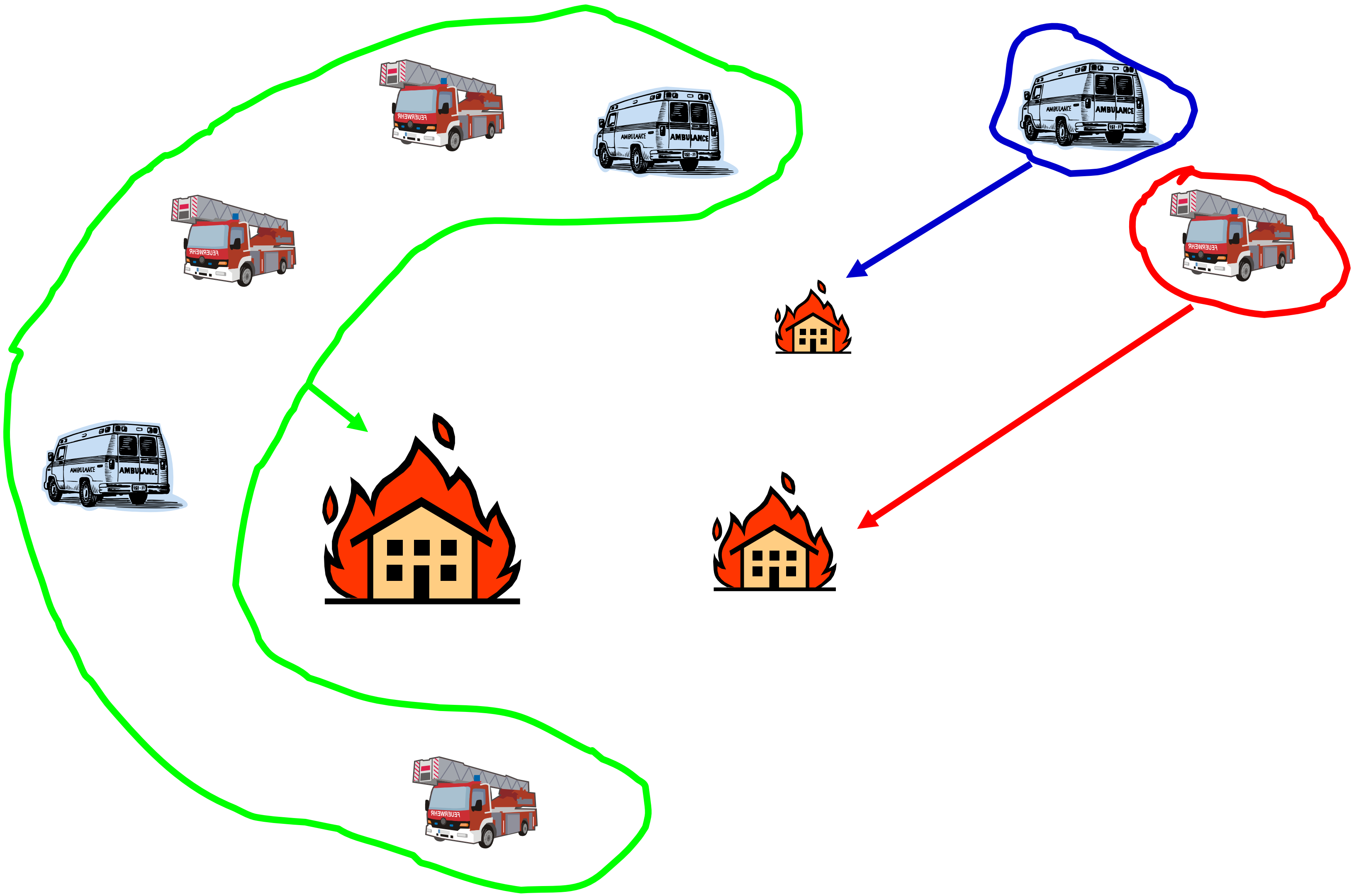
Coalition Formation
Introduction

- Useful when a group's performance is more efficient than the performance of a single agent
  - E.g. ambulances can faster (more likely) rescue a victim if they are in a bigger group
- Allocation of tasks to groups is necessary when tasks cannot be performed by a single agent
  - E.g. a single fire brigade cannot extinguish a large fire
- A group of agents is called a coalition
- A *coalition structure* is a partitioning of the set of agents into disjoint coalitions
- An agent participates in only one coalition
- A coalition may consist of only a single agent
- Generally, coalitions consist of heterogeneous agents

Coalition Formation
Example



Coalition Formation
Example



Applications for coalition formation

- In e-commerce, buyers can form coalitions to purchase a product in bulk and take advantage of price discounts (Tsvetovat et al., 2000)

- For information gathering, several information servers can form coalitions for answering queries (Klusch and Shehory, 1996)

- Distributed vehicle routing among delivery companies with their own delivery tasks and vehicles (Sandholm 1997)

- Wide-area surveillance by autonomous sensor networks (Dang 2006)

- In Rescue, team formation to solve particular sub-problems, e.g. larger robots deploy smaller robots within confined spaces

Coalition Formation
Definition I

- Coalition formation includes three activities:
  - Coalition structure generation
    - Partitioning of the agents into exhaustive and disjoint coalitions
    - Inside the coalitions, agents will coordinate their activities, but agents will not coordinate between coalitions
  - Solving the optimization problem in each coalition:
    - pooling the tasks and resources of the agents in the coalition and solving the joint problem
    - The coalition objective could be to maximize the monetary value, or the overall expected utility
  - Dividing the value of the generated solution:
    - In the end, each agent will receive a value (money or utility) as a result of participating in the coalition
    - In some problems, the coalition value the agents have to share is negative, being a shared cost

Discussed in this lecture

Coalition Formation
Definition II

- A group of agents $S \subseteq A$ is called a coalition, where A denotes the set of all agents and $S \neq \emptyset$
  - The coalition of all the agents is called grand coalition
- A coalition structure (CS) partitions the set of agents into coalitions
  - CS^* is the social welfare maximizing coalition structure
- The value of each coalition S is given by a function v_S
  - Each coalition value is independent of non-members actions

Coalition structure generation

- The value of a coalition structure is given by:

$$\mathrm{V}(CS) = \sum_{S \in CS} v_S$$

- The goal is to maximize the social welfare of the agents *A* by finding a coalition structure:

$$CS^* = \operatorname*{argmax}_{CS \in Partitions(A)} \mathrm{V}(CS)$$

Special Coalition Values

- The coalition values are *super-additive* iff for every pair of disjoint coalitions $S, T \subseteq A$: $v_{S \cup T} \geq v_S + v_T$
  - If coalition values are super-additive, then the coalition structure containing the *grand coalition* gives the highest value
- The coalition values are *sub-additive* iff for every pair of disjoint coalitions $S, T \subseteq A$: $v_{S \cup T} < v_S + v_T$
  - If coalition values are sub-additive, then the coalition structure $\{\{a\} \mid a \in A\}$ in which no agent cooperates gives the highest value
- Is the ambulance rescue task in the RoboCup Rescue domain super-additive, sub-additive, or none of both?

Coalition structure generation
Example

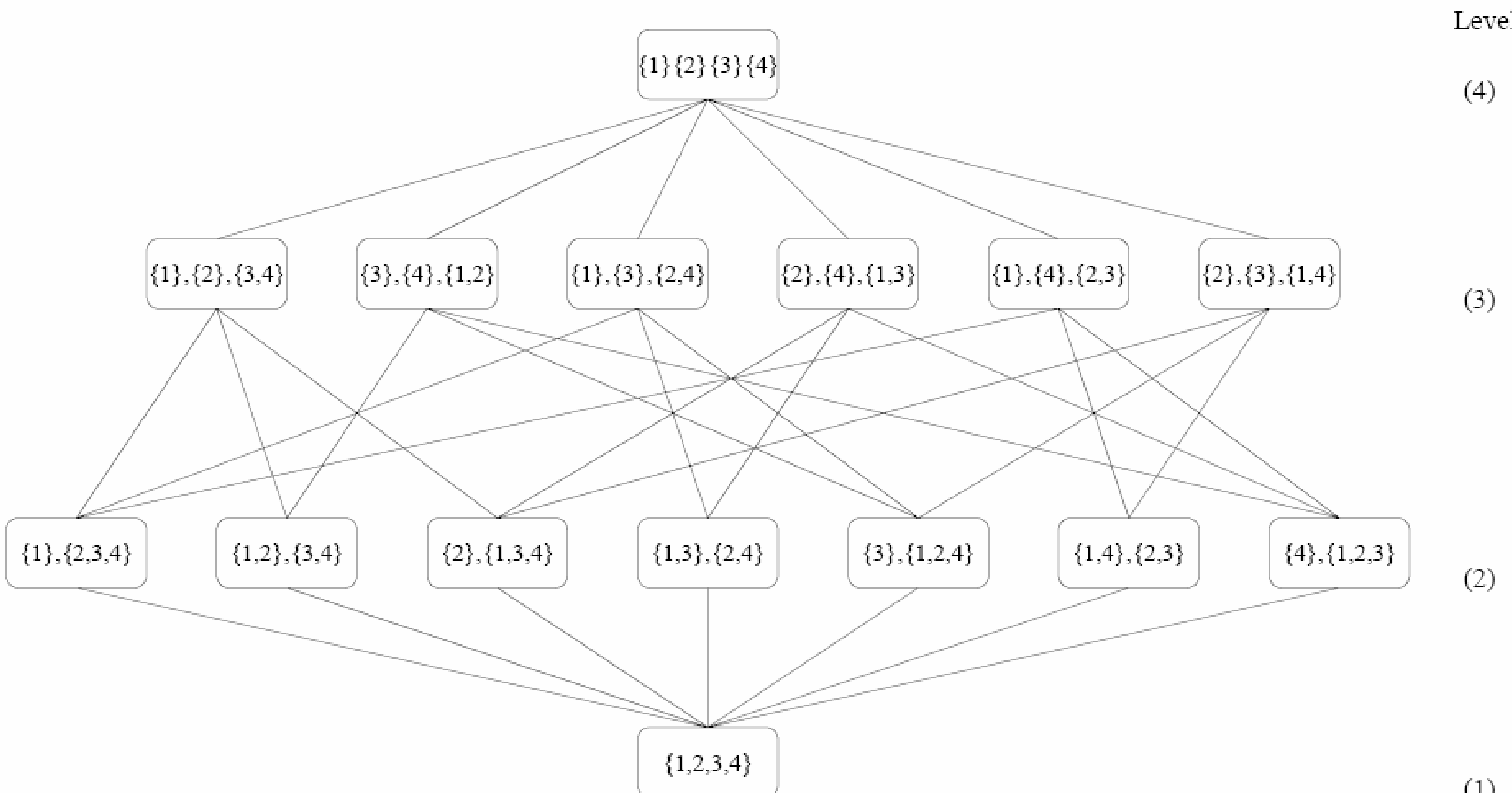
The input is the possible coalitions and their values:

A = { 1 , 2 , 3 , 4 }

CL1	v_s	*CL2*	v_s	*CL3*	v_s	*CL4*	v_s
{1}	92	{1, 2}	189	{1, 2, 3}	316	{1, 2, 3, 4}	395
{2}	96	{1, 3}	210	{1, 2, 4}	297		
{3}	87	{1, 4}	203	{1, 3, 4}	335		
{4}	105	{2, 3}	171	{2, 3, 4}	272		
		{2, 4}	215				
		{3, 4}	182				

For N agents the number of possible coalitions is 2^N-1
but the number of possible coalition structures is $N^{N/2}$

Coalition graph



- For 4 agents: A = { 1 , 2 , 3 , 4 }
- Nodes represent coalition structures
- Arcs represent either merges (downwards) or splits (upwards)

Coalition Structure Search I

- To search the whole coalition graph for the optimal coalition is intractable (in practice up from |A|>15)
- Can we approximate the search by visiting only a subset of L nodes?

$$CS_L^* = \underset{CS \in L}{argmax} \; V(CS)$$

- One requirement is to guarantee that the found coalition structure is within a worst case bound from optimal:

$$k * V(CS_L^*) \geq V(CS^*)$$

Coalition Structure Search II

- Theorem: to bound k for some subset N of the coalition structures, it suffices to search the lowest two levels of the coalition structure graph
  - With this search, the bound is $k = |A|$, this bound is tight, and the number of nodes searched is $n = 2^{|A|-1}$
  - No other search algorithm (than the one that searches the bottom two levels) can establish a bound k while searching only $n = 2^{|A|-1}$ nodes or fewer

- Intuition:
  - The lowest two levels of the coalition graph are the only two levels in which all possible coalitions occur
    - A level l consists of coalition structures containing l coalitions
    - Hence, if $l > 2$, the largest coalition in the level contains $|A| - l + 1$ agents since the smallest possible coalition contains 1 agent

Coalition Structure Search III

- Algorithm:
  - Search the bottom two levels of the coalition structure graph
  - Continue with breadth-first search from the top of the graph as long as there is time left, or until the entire graph has been searched
  - Return the coalition structure that has the highest welfare among those seen so far
- Note the search can be distributed among self-interested agents

Case study: ResQ Freiburg task allocation

- Problem description:
  - N ambulance teams have to rescue M civilians after an earthquake
  - Civilians are characterized by *Buriedness*, *Damage* and *Hit-points*
    - *Buridness* is proportional to the required resources (ambulance cycles)
    - As more *hit-points* as more likely the civilian dies
    - The amount of *damage* increases the growth of hit-points, i.e. accelerates the time of death
  - Costs are the time to rescue a civilian, composed of the coalition's joint travel time to reach the victim, and the time needed for the rescue
  - The joint utility is the number of rescued civilians (the civilians brought to a refuge)
- We considered the ambulance rescue task as super-additive
  - The rescue operation itself is super-additive
  - Assumption: travel costs are the same for every agent
  - However, consider the situation of 2 victims at two different locations that could both be rescued by a single agent but will die within a short amount of time
  - Maybe not the optimal solution!

ResQ Freiburg task allocation
Task allocation

- The problem reduces to assign a sequence R of rescue tasks to the entire set of agents A (here the ambulances):
  – R = <r_1, r_2, …, r_N> where r_i denotes a rescue task and i the position in the sequence
- $U(R)$ denotes the predicted utility (the number of survivors) when executing sequence R
- Hence, the problem is find the optimal sequence from the set of all possible sequences
  – $R^* = \arg\max U(R)$
- Enumerating all possible sequences is impossible within limited time (the world model changes frequently, altering the current sequence)
- Greedy solutions
  – Prefer victims that can be rescued fast (small buridness)
  – Prefer urgent victims (high damage)

ResQ Freiburg task allocation
Implementation

- Non-allocated agents (e.g. police & fire brigades) continuously search unexplored locations and update information (e.g. buridness, health) about known victims
- The ambulance station (agent)
  - predicts for each known victim the lifetime and costs for rescue
  - simulates rescue sequences, selected by a genetic algorithm, over the set of known victims
  - When a better sequence has been found, the rescue sequence of agents in the field is altered
- Life time prediction
  - Learning of a decision tree for the classification of victims into *will die* and *will survive*
  - Adaptive Boosting (Ada Boost) for the regression learning of the life time prediction (previously on data sets)
  - Calculation of confidence values with respect to the age of information (e.g. as older the information as more unreliable the prediction)

ResQ Freiburg task allocation
Genetic Optimization

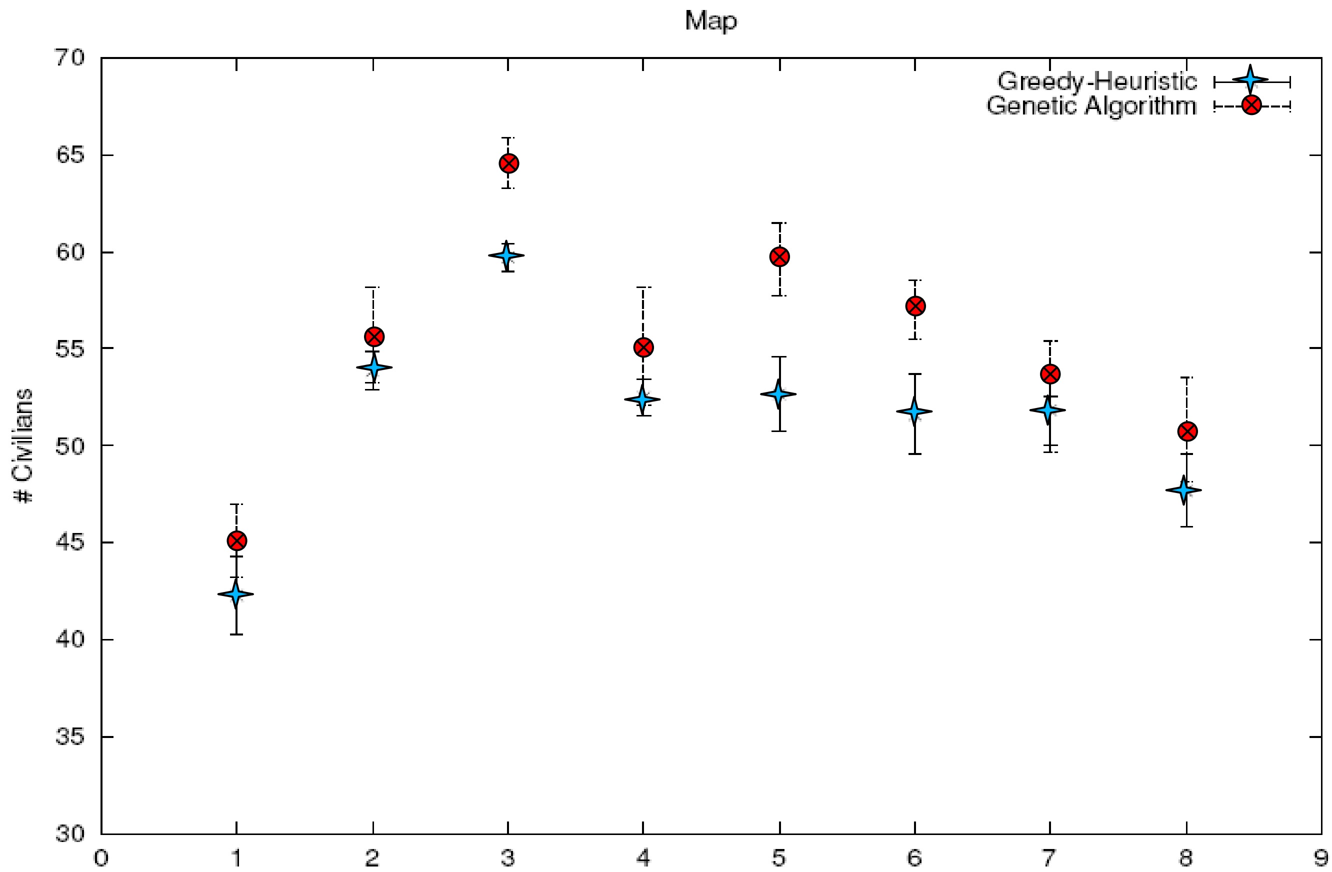
- Local search, i.e. hill climbing, that continuously improves the current best solution (selection)
- Solutions are represented by strings (DNA) that are locally modified for finding better outcomes (mutation)
  - For example 543261 → 534261
- Offsprings are generated by a crossing operation
  - For example “one-point crossover”
- Genetic pool is initialized with greedy solutions (e.g. prefer urgent victims or prefer victims that can be rescued fast)
- Elitism: Keep best two solutions in the genetic pool
- Anytime execution:
  - Number of genetic pool generations can be adjusted according to CPU usage
  - Optimization can anytime be stopped at current best solution

ResQ Freiburg task allocation
Results RoboCup 2004

	ResQ	Damas	Caspian	BAM	SOS	SBC	ARK	B.Sheep
Final-VC	42	43	**52**	34	N/A	N/A	N/A	N/A
Final-Random	**32**	25	29	16	N/A	N/A	N/A	N/A
Final-Kobe	**46**	45	**46**	30	N/A	N/A	N/A	N/A
Final-Foligno	**66**	54	50	29	N/A	N/A	N/A	N/A
Semi-VC	**18**	15	17	12	11	12	12	14
Semi-Random	22	**26**	16	14	20	14	15	15
Semi-Kobe	**57**	47	54	52	20	39	34	44
Semi-Foligno	37	**46**	44	43	42	28	29	24
Round2-Kobe	**57**	37	43	50	43	35	28	43
Round2-Random	**52**	48	39	45	47	44	50	37
Round2-VC	31	33	32	24	37	**51**	N/A	34
Round1-Kobe	45	**51**	47	43	47	31	25	34
Round1-VC	**62**	**62**	55	57	N/A	51	54	44
Round1-Foligno	**53**	**53**	37	33	37	41	30	23
# wins:	**9**	5	2	0	0	1	0	0
$\sum$ TOTAL:	**620**	585	561	482	304	346	277	312
$\sum$ SEMI+PREM	**434**	418	384	373	304	346	277	312

Number of saved civilians

ResQ Freiburg task allocation
Results RoboCup 2004 cont.

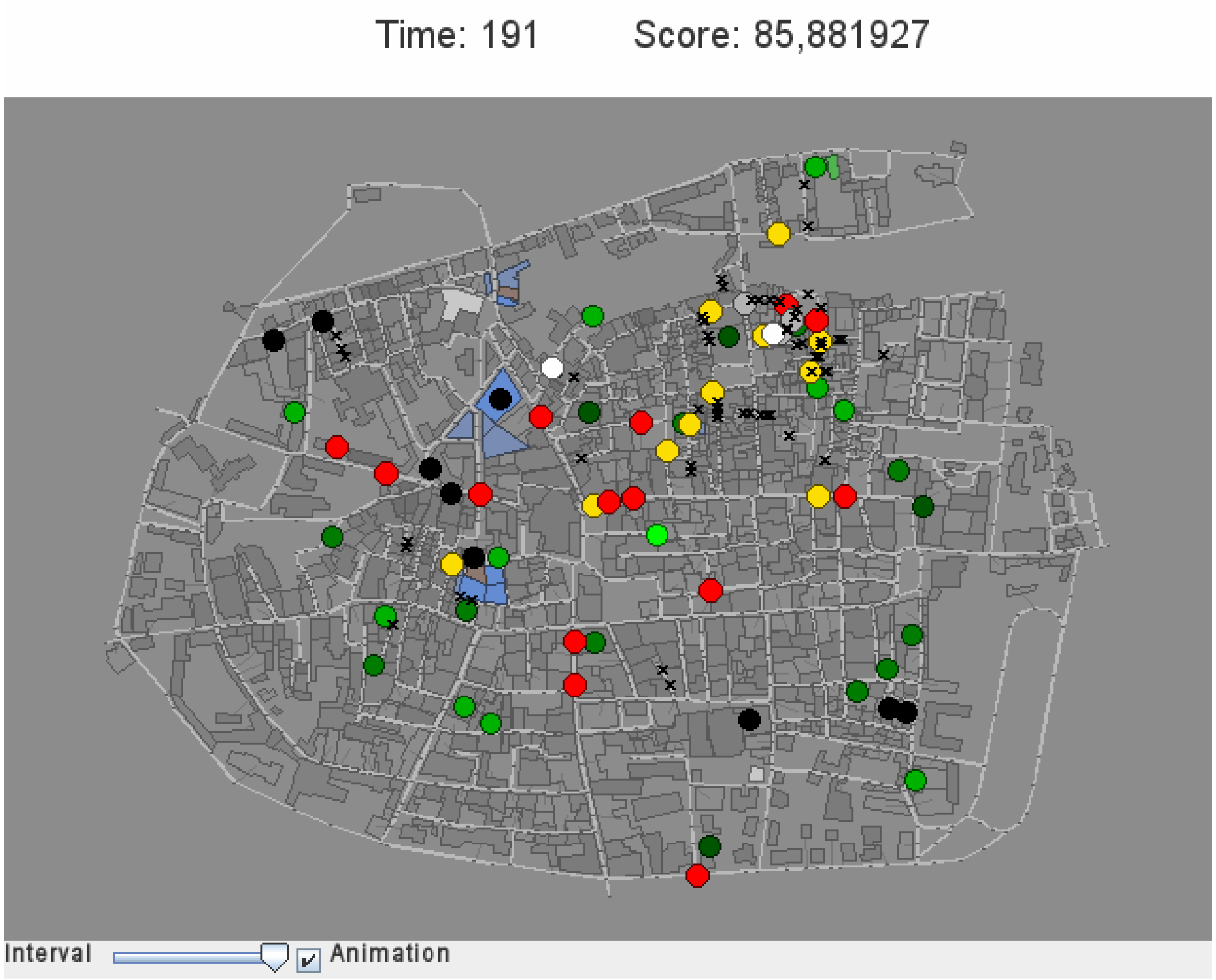


Number of saved civilians by greedy and genetic sequence optimization on different maps

Task Allocation For Fire Brigades

- Fires have to be clustered in order to define tasks
  - For each cluster a utility has to be computed, e.g. # of victims nearby, # of neighboring houses
  - For each cluster the # of needed fire brigades has to be computed
- Problem: How to assign fire brigades to fire clusters efficiently?
  - Auctions are problematic due to communication constraints of the domain
  - Coalition formation
    - Is the problem is super additive?
    - Plays the sequence an important role?
- Some more problems:
  - Some fires are more dangerous than others due to their firyness
  - Some fires can be much faster extinguished than others due to size and material of the building
  - It is advantageous to prefer “border fires” in order to stop fire spread
  - Logistics: How to optimally place fire brigades around fires in order to avoid that they block each other?
- Maybe a “task” for the exercises

ResQ Freiburg task allocation
Example Animation

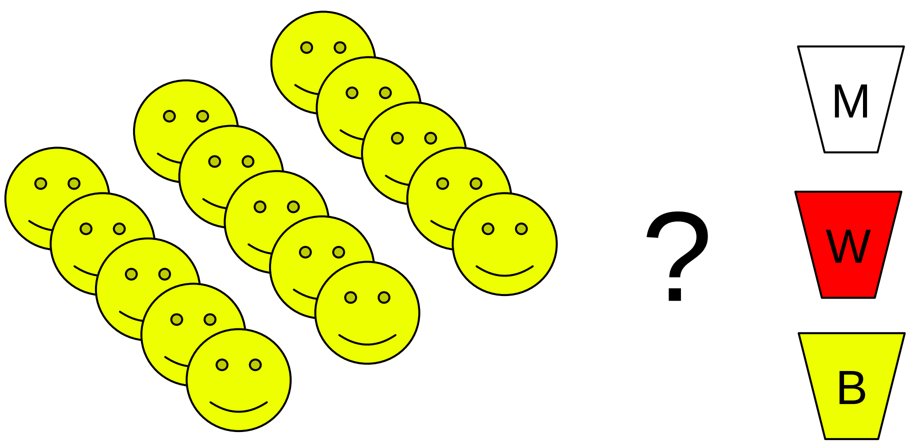


Voting
Introduction

- In open systems agents have their *individual preferences* an agreement can be reached by *voting*
  - Applicable for both *benevolent* and *self-interested* agents
- A voting system derives a social preference form all agents` individual preferences
- How to find a fair solution? What means a fair solution?
- One way to approach the fairness problem is to require:
  - If one agent prefers A to B and another one prefers B to A then their votes should cancel each other out
  - If one agent’s preferences are A,B,C and another one’s are B,C,A and a third one prefers C,A,B then their votes should cancel out

Voting
Example

15 mathematicians are planning to throw a party. They must first decide which beverage the department will serve at this party. There are three choices available to them: beer, wine, and milk.



6 x Milk $\succ$ Wine $\succ$ Beer

5 x Beer $\succ$ Wine $\succ$ Milk

4 x Wine $\succ$ Beer $\succ$ Milk

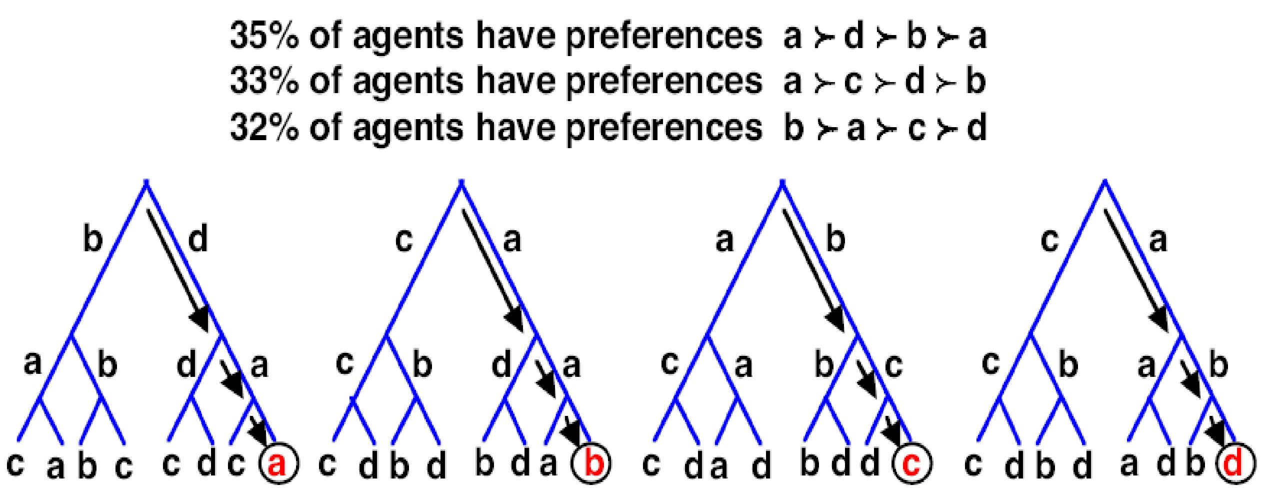
Voting
Plurality protocol

- Majority voting protocol where alternatives are compared simultaneously

- In the example:
  - Each one votes for their favorite drink, the votes and the drink with the most votes is the winner
  - Beer would get 5 votes, wine 4, and milk 6 → Milk wins!
  - Problem: there are 8 agents that prefer beer over milk and wine over milk, but only 6 that have the opposite preferences, and yet milk wins?

Voting
Binary Voting

- Alternatives are voted on pairwise, the winner stays to challenge further alternatives while the looser is eliminated
- For example:
  - beer & wine: wine wins, wine & milk: wine wins
- Problem: The order of the considered pairings can totally change the outcome. For example:



Voting
Borda Protocol

- Takes into account all agents' knowledge equally
- Assigns |O| points to an alternative whenever it is highest in some agent's preference, assigns |O-1| whenever it is second, …
- Counts are summed across voters, alternative with highest count becomes the social choice

- In the example:
  - Milk: 6*3 + 5*1 + 4*1 = 27
  - Wine: 6*2 + 5*2 + 4*3 = 34
  - Beer: 6*1 + 5*3 + 4*2 = 29
  - Wine wins!

Voting
Definition

- Given a set of agents A and a set of outcomes O, each agent $i \in A$ has a strict, asymmetric, and transitive preference relation $\succ_i$ on O
- A voting system derives a social preference $\succ_*$ form all agents` individual preferences $(\succ_i, \ldots, \succ_{|A|})$
- Desired properties of a voting system are:
  1. $\succ_*$ exists for all possible inputs $\succ_i$
  2. $\succ_*$ should be defined for every pair $o, o' \in O$
  3. $\succ_*$ should be asymmetric and transitive over O
  4. The outcome should be Pareto efficient: if $\forall i \in A$, $o \succ_i o'$ then $o \succ_* o'$, e.g., if all agents prefer beer over milk then $\succ_*$ should also prefer beer over milk
  5. The scheme should be independent of irrelevant alternatives, i.e. when adding another alternative the ranking should be same
  6. No dictatorship: if $o \succ_i o'$ implies $o \succ_* o'$ for all preferences of the other agents

Voting
Arrow's impossibility Theorem

- There is no voting mechanism that satisfies all six conditions (Arrow, 1951)
  - For example, in the Borda protocol, irrelevant alternatives can lead to paradox results (violating (5)):

Agent	Preferences
1	$a \succ b \succ c \succ d$
2	$b \succ c \succ d \succ a$
3	$c \succ d \succ a \succ b$
4	$a \succ b \succ c \succ d$
5	$b \succ c \succ d \succ a$
6	$c \succ d \succ a \succ b$
7	$a \succ b \succ c \succ d$
Borda count	c wins with 20, b has 19, a has 18, d loses with 13
Borda count with d removed	a wins with 15, b has 14, c loses with 13

Winner turns loser and loser turns winner paradox in the Borda protocol

Dynamic Role Assignment
Introduction

- Role assignment is a computational cheap mechanism to efficiently coordinate omniscient agents
  - Individual roles are assign according to a team formation
  - Can be applied in domains with n pre-defined tasks and m robots that can potentially be assigned to each task
  - Particularly suited in dynamic domains, such as robot soccer, where the optimal assignment depends on the current world state
- Example domain robot soccer:
  - The goal is to avoid *swarm behavior* and inference
    - do not attack your own team mates
    - do not get into the way of an attacking or defending robot
  - Task decomposition and task (re-)allocation
    - the player which is closest to the ball should go to the ball
    - If one player cannot do his task, another should take over
  - Joint execution: passing the ball

Dynamic Role Assignment
General Algorithm

- Assumptions:
  - There are n available roles (not necessarily distinct)
  - The state is fully observable to the agents
  - There is a fixed ordering {1, 2, …, n} of the roles. Role 1 must be assigned first, followed by role 2, etc.
  - Each agent can be assigned only one role
  - The utility u_{ij} reflects how appropriate agent i is for role j given the current state
- Role assignment:

```
for all agents in parallel
        I := ∅;
        for each role j = 1,...,n
            for each agent i = 1,...,n with n ∉ I
                    compute potential r_{i,j};
            end;
            assign role j to agent i* = arg max_{i∉I} {r_{i,j}};
            I := I ∪ { i* };
        end;
    end.
```

Case Study: CS-Freiburg
Dynamic roles

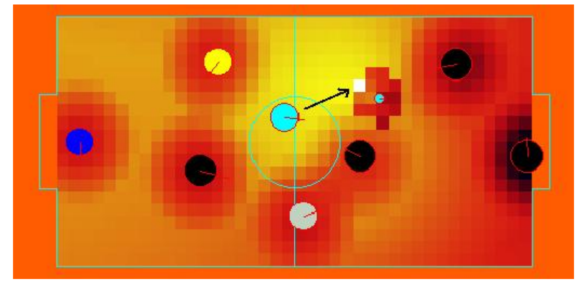
- Each player has one of four roles:
  - *goalie* (fixed)
    - special hardware setup → unable to change its role
  - *active player*: in charge of dealing with the ball
    - can approach the ball or to bring the ball forward towards the opponent goal
  - *strategic* player: defender
    - maintains a position back in its own half
  - *supporter*: serves the team
    - in defensive play it complements the team's defensive formation
    - in offensive play it presents itself to receive a pass close to the opponents goal

Case Study: CS-Freiburg

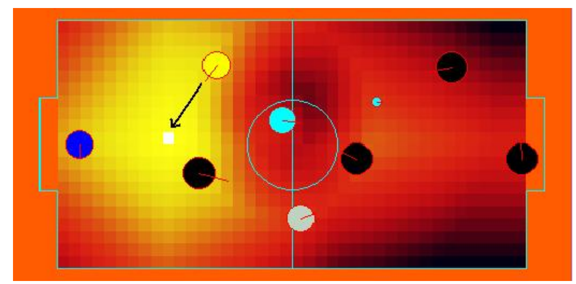
Role Utilities

- Placement: each role has a preferred location, which depends on the situation:
  - ball position, position of team mates and opponents
  - defensive situation or attack
  - computed by potential fields
- Utility for each role:
  - "Negative costs" for reaching the preferred location of the role
  - Summed-up from partial utilities computed from distance, turn angle, objects on the path, …

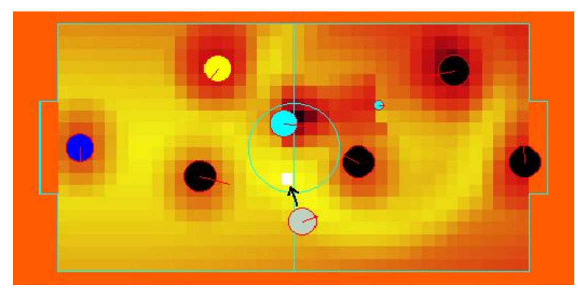
active Role:



strategic role:



supporter role:



Case Study: CS-Freiburg
Dynamic Role Assignment

- Each player computes the utility for each role and broadcasts it to the other players
- Each player tries to maximize the group utility
  - under the assumption that all team members do the same
- Group utility:
  - Consider all possible n! assignments and compute the summed utility from each agents' individual utility for its assigned role
  - Take the assignment with the highest utility sum as solution
- Roles are reassigned only when
  - the role change is significant, i.e. the new utility >> old utility (hysteresis factor to avoid oscillation)
  - two players agree (by communication)
- Note that opinion about global position can differ (even with a global world model)
  - Agents might "lie" without intention

Case Study: CS-Freiburg
Example for Role Switching I

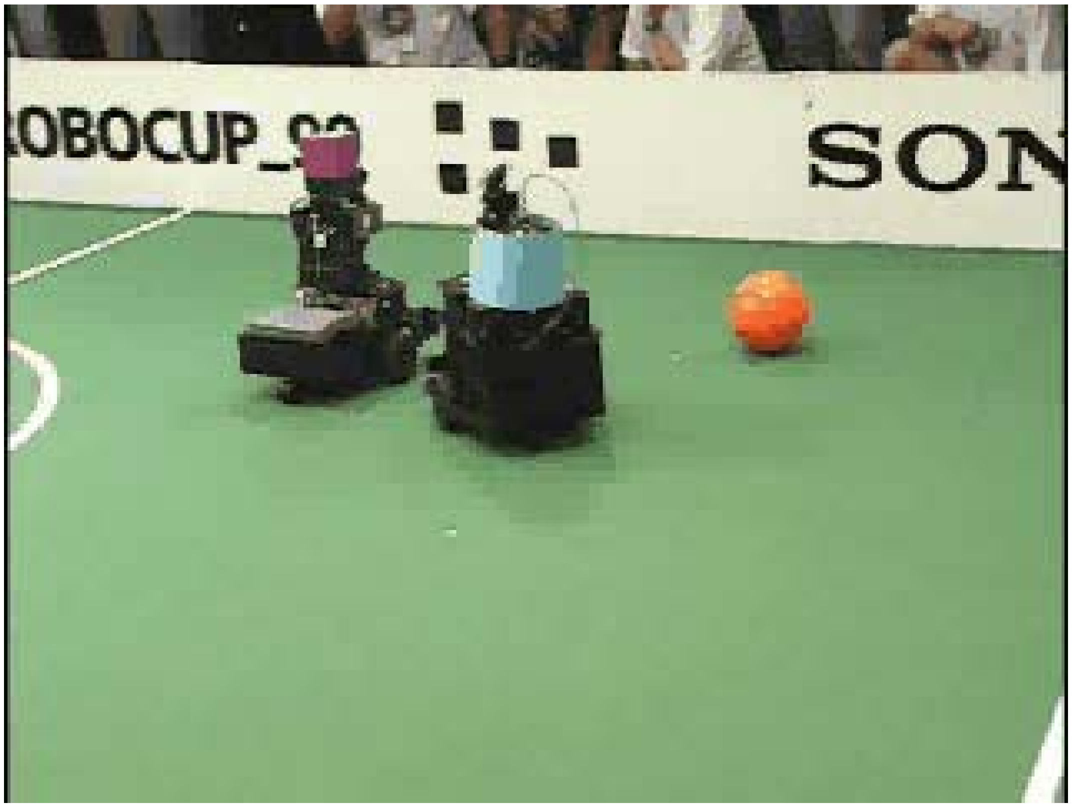
Attack against Osaka (Japan). The attacking robot is blocked by a defender and consequently replaced by an unblocked player.

Case Study: CS-Freiburg
Example for Role Switching II

Defense against *Artisti Veneti* (Italy).
The roles *active* and *strategic player* are switched a couple of times

Case Study: CS-Freiburg

Joint Execution: A Pass . . . that was Unsuccessful



A pass in the semi-final against the Italian *ART Italy* team (RoboCup 1999). This was based on standard plan: “if it is not possible to score directly, wait until supporter arrives, then make the pass”

Case Study: CS-Freiburg

Demo Webplayer

See www.cs-freiburg.de

Summary

- Action selection and coordination are essential when acting in groups
  - If done right, you can win a robotic soccer or rescue agent world championship
- Coalition formation is the process of finding the “social welfare” coalition structure among a set of agents
  - The search can be computational expensive when dealing with heterogeneous agents
  - In practice, domain dependent heuristics are necessary for pruning the search tree (i.e. constraining the split and merge arcs)
- Voting methods have to be implemented carefully with respect to the desired outcome
  - In practice, the plurality protocol is often used in multi-agent systems
  - However, the Borda protocol should be the preferred as it can effectively aggregate multiple disparate opinions
- Dynamic role assignment is an efficient and cheap method for team coordination
  - However, the protocol requires truthful participants
  - Due to world model inconsistencies, this assumption can be violated

Literature

- M. Woolridge: **An Introduction to Multi-Agent-Systems**, *Wiley*, 2001, 294 pages
- Gerhard Weiss **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligenc,** *The MIT Press*, pages 201-258
- A. Kleiner, M. Brenner, T. BrÃ¤uer, C. Dornhege, M. Göbelbecker, M. Luber, J. Prediger, J. Stückler, and B. Nebel **Successful Search and Rescue in Simulated Disaster Areas** *Robocup 2005: Robot Soccer World Cup IX* pp. 323-334, 2005
- T. Weigel, J.-S. Gutmann, M. Dietl, A. Kleiner and B. Nebel **CS-Freiburg: Coordinating Robots for Successful Soccer Playing** *IEEE Transactions on Robotics and Automation* 18(5):685-699, 2002