Optimization in Multi-Agent Systems

Jesús Cerquides

Alessandro Farinelli

Pedro Meseguer

Sarvapali D. (Gopal) Ramchurn

Juan A. Rodríguez-Aguilar

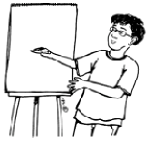
ARTIFICIAL INTELLIGENCE RESEARCH INSTITUTE - IIIA

UNIVERSITY OF Southampton

University of Verona

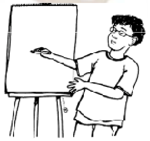
Schedule

- 9:00 – 9:15 – Intro



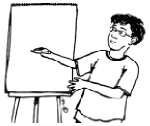
- 9:15 - 11:00 Distributed Constraint Optimization

- **11:00 - 11:30 Coffee break**



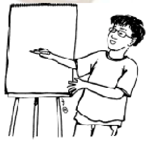
- 11:30 - 13:00 Tutorial -- Distributed Constraint Optimization (continued)

- **13:00 - 14:30 Lunch break**



- 14:30 - 16:30 Tutorial -- Market Based Allocation

- **16:30 - 17:00 Coffee break**

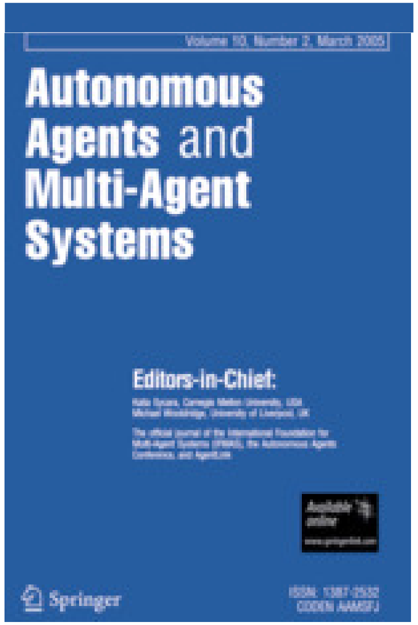


- 17:00 - 18:30 Tutorial -- Coalition Formation

OPTMAS in different forms…

- AAMAS 08,09, 10, 11
  - With Nick Jennings, Alex Rogers, Alessandro Farinelli, Gopal Ramchurn, Juan Antonio Rodriguez

- JAAMAS Special issue (Vol. 22, No. 3)



Volume 10, Number 2, March 2005

Autonomous Agents and Multi-Agent Systems

Editors-in-Chief:

[illegible]

[illegible]

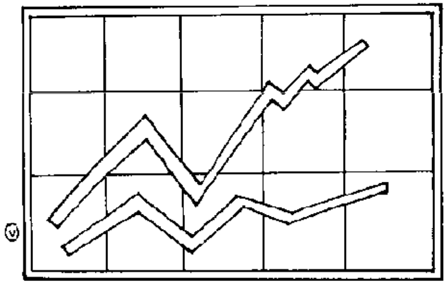
Available online

Springer

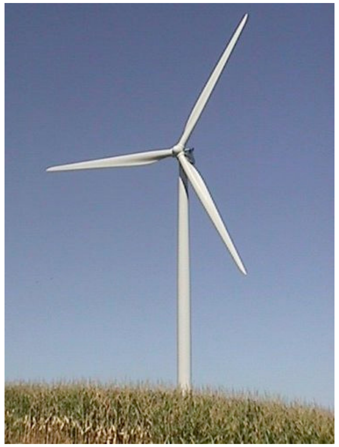
ISSN: 1387-2532
CODEN AAMSFJ

Why OPTMAS?

- How do we optimise (for) multi-agent coordination?



- Can benchmarks and modelling techniques be shared across problem spaces?



- What are interesting application domains for multi-agent coordination where optimisation is important?

Outline

- DCOPs for Decentralized Decision Making
  - Pedro Meseguer (DCSP)
  - Alessandro Farinelli (Approximate Algorithms)
- Market Based Resource Allocation
  - Juan A. Rodriguez-Aguilar (Introduction to Auctions)
  - Jesus Cerquides (Combinatorial Auctions/Supply Chain Formation)
- Coalition Formation
  - Sarvapali Ramchurn

What you will learn from this tutorial

- MAS representations for optimisation
- Using existing techniques and algorithms and developing new ones
- Optimisation for a purpose (research is not just about theorems)

Distributed Constraint Optimization Problem for Decentralized Decision Making

Optimization in Multi-Agent Systems

Alessandro Farinelli and Pedro Meseguer

At the end of this talk, you will be able to:

1. Model decision making problems with DCOPs
   - Motivations for using DCOP
   - Modeling practical problems using DCOPs
2. Understand main exact techniques for DCOPs
   - Main ideas, benefits and limitations of each technique
3. Understand approximate techniques for DCOPs
   - Motivations for approximate techniques
   - Types of quality guarantees
   - Benefits and limitations of main approximate techniques

Outline

- Introduction
  - DCOP for Dec. Decision Making
  - how to model problems in the DCOP framework
- Solution Techniques for DCOPs
  - Exact algorithms (DCSP, DCOP)
    - ABT, ADOPT, DPOP
  - Approximate Algorithms (without/with quality guarantees)
    - DSA, MGM, Max-Sum, k-optimality, bounded max-sum
- Future challenges in DCOPs

Cooperative Decentralized Decision Making

- Decentralised Decision Making
  - Agents have to coordinate to perform best actions
- Cooperative settings
  - Agents form a team -> best actions for the team
- Why ddm in cooperative settings is important
  - Surveillance (target tracking, coverage)
  - Robotics (cooperative exploration)
  - Scheduling (meeting scheduling)
  - Rescue Operation (task assignment)

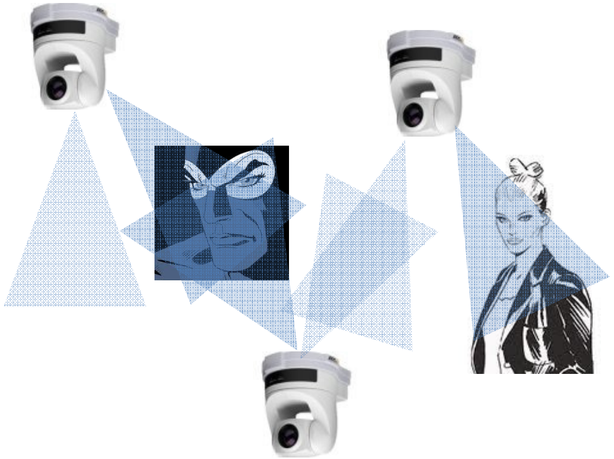
DCOPs for DDM

Why DCOPs for Coop. DDM ?

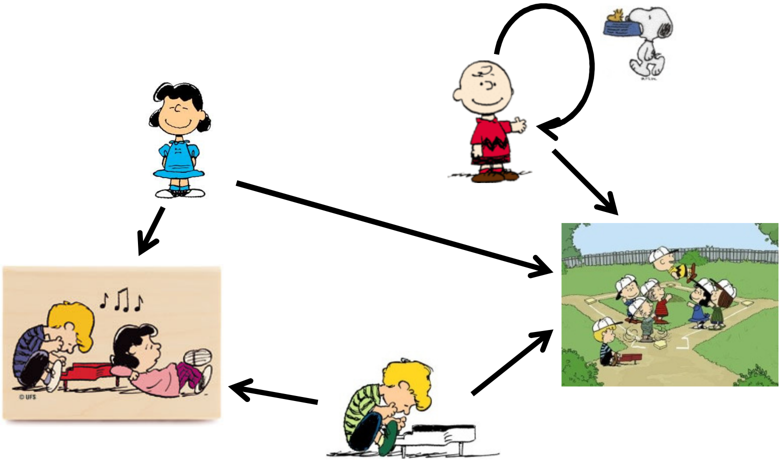
- Well defined problem
  - Clear formulation that captures most important aspects
  - Many solution techniques
    - Optimal: ABT, ADOPT, DPOP, ...
    - Approximate: DSA, MGM, Max-Sum, ...
- Solution techniques can handle large problems
  - compared for example to sequential dec. Making (MDP, POMDP)

Modeling Problems as DCOP

- Target Tracking



- Meeting Scheduling

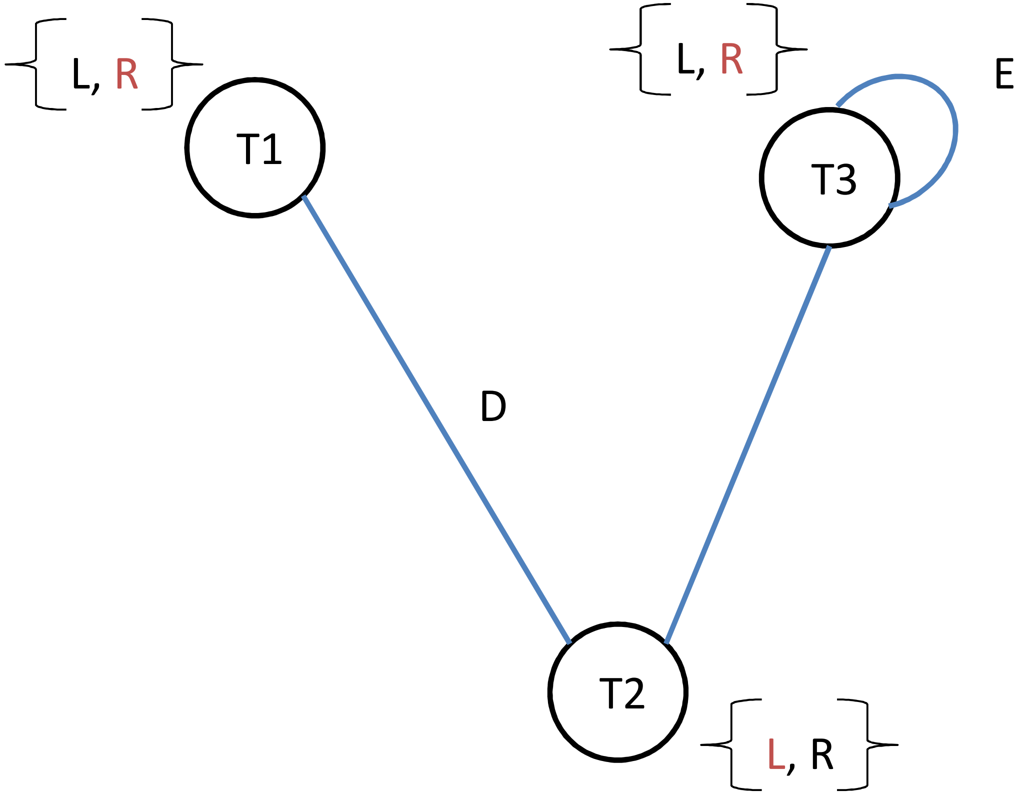


Target Tracking

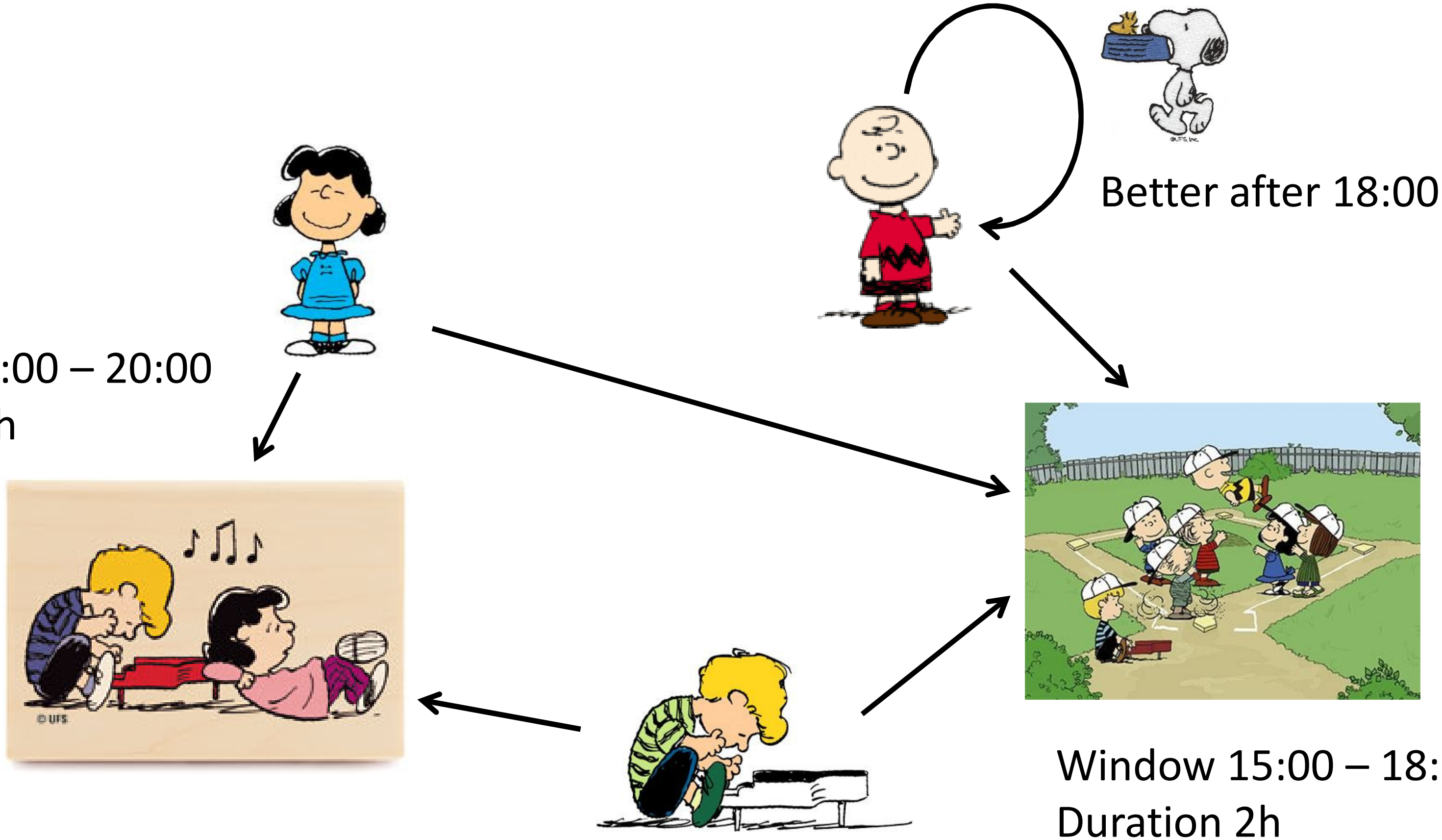
- Why decentralize
  - Robustness to failure and message loss

Target Tracking - DCOP

- Variables -> Cameras
- Domains -> Camera actions
  - look left, look right
- Constraints
  - Overlapping cameras
  - Related to targets
    - Diabolik, Eva
- Maximise sum of constraints



Meeting Scheduling



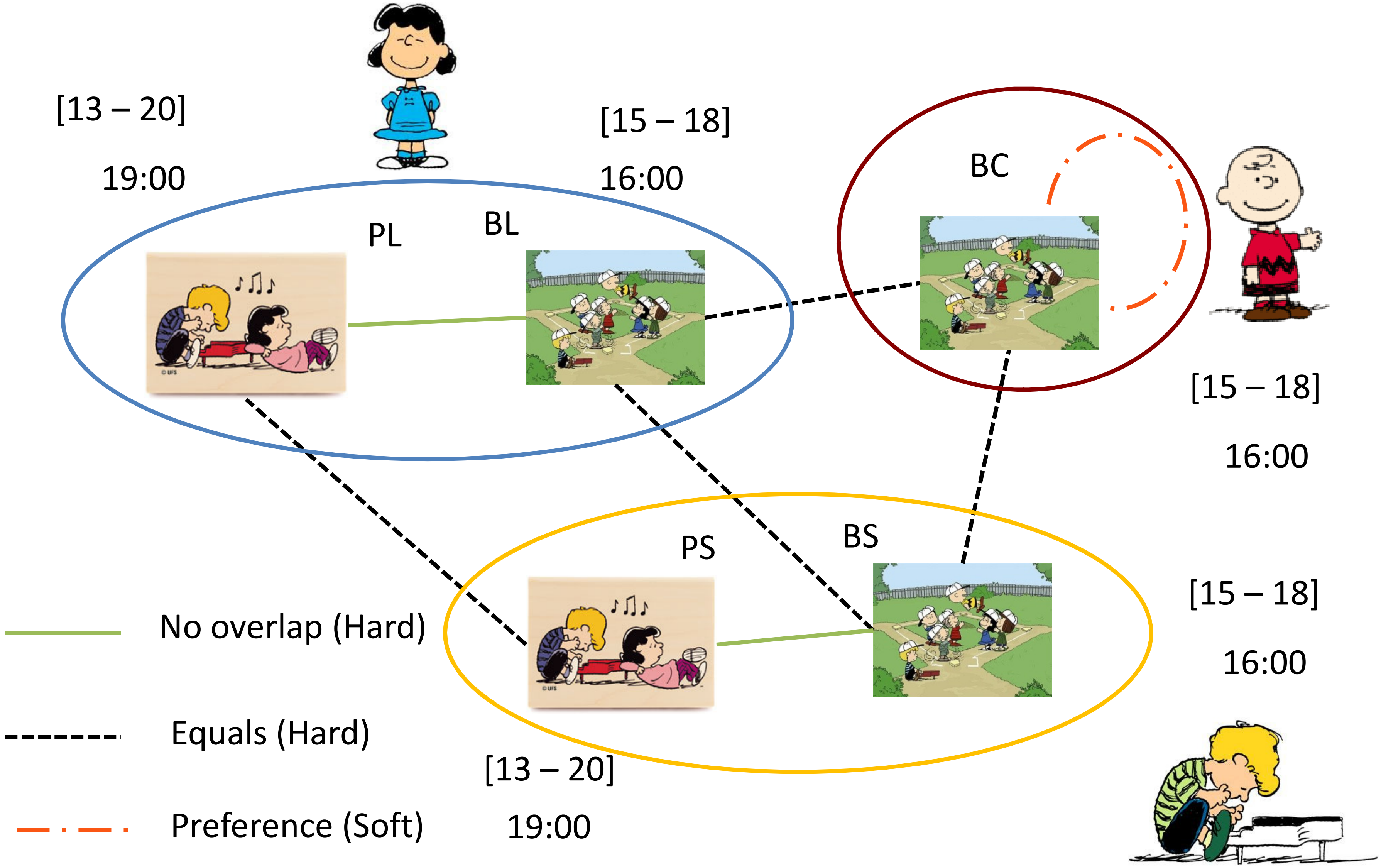
Window13:00 – 20:00
Duration 1h

Better after 18:00

Window 15:00 – 18:00
Duration 2h

- Why decentralize
  - Privacy

Meeting Scheduling - DCOP



[13 – 20]

19:00

[15 – 18]

16:00

BC

PL

BL

[15 – 18]

16:00

PS

BS

[15 – 18]

16:00

No overlap (Hard)

Equals (Hard)

[13 – 20]

19:00

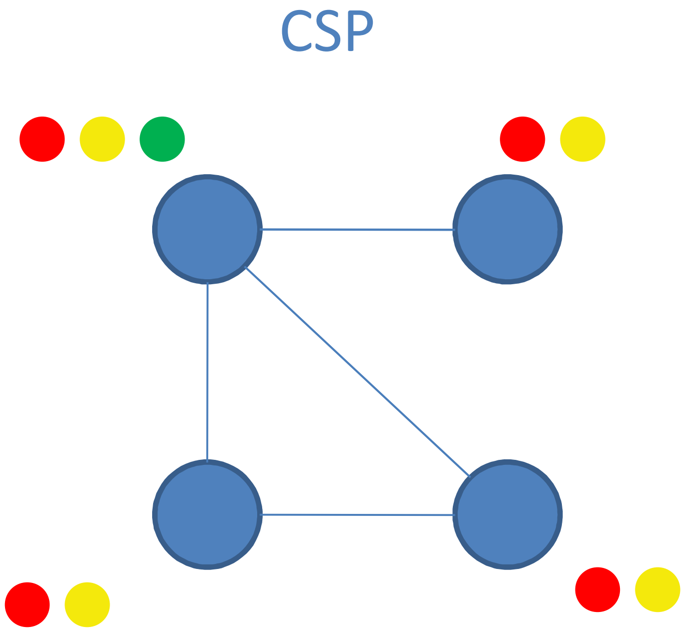
Preference (Soft)

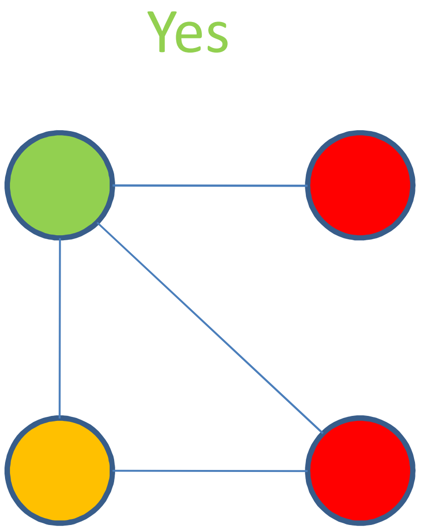
Benchmarking problems

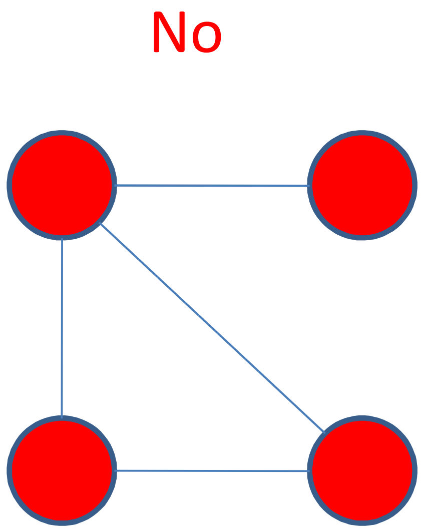
- Motivations
  - Analysis of complexity and optimality is not enough
  - Need to empirically evaluate algorithms on the same problem
- Graph coloring
  - Simple to formalise very hard to solve
  - Well known parameters that influence complexity
    - Number of nodes, number of colors, density (number of link/number of nodes)
  - Many versions of the problem
    - CSP, MaxCSP, COP

Graph Coloring

- Network of nodes
- Nodes can take on various colors
- Adjacent nodes should not have the same color
  - If it happens this is a conflict

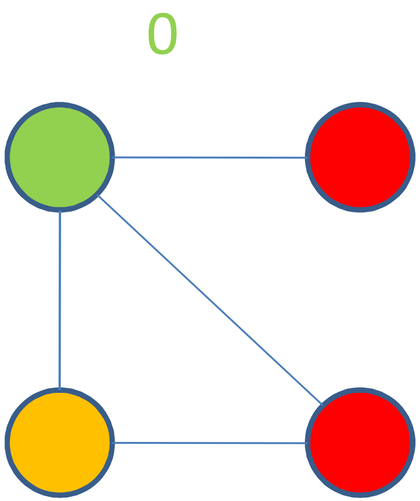


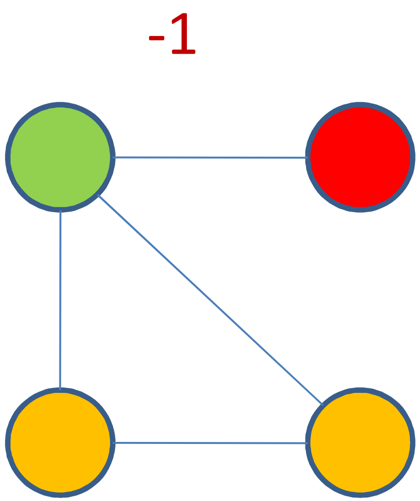


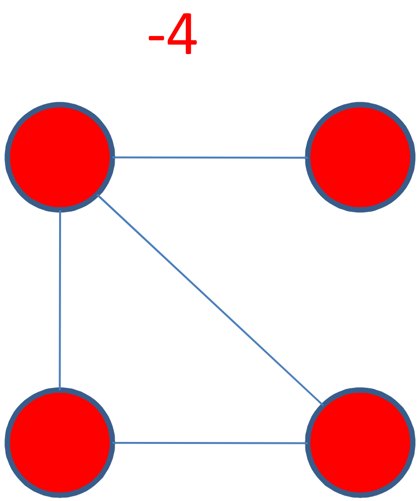


Graph Coloring - MaxCSP

- Optimization Problem
- Natural extension of CSP
- Minimise number of conflicts

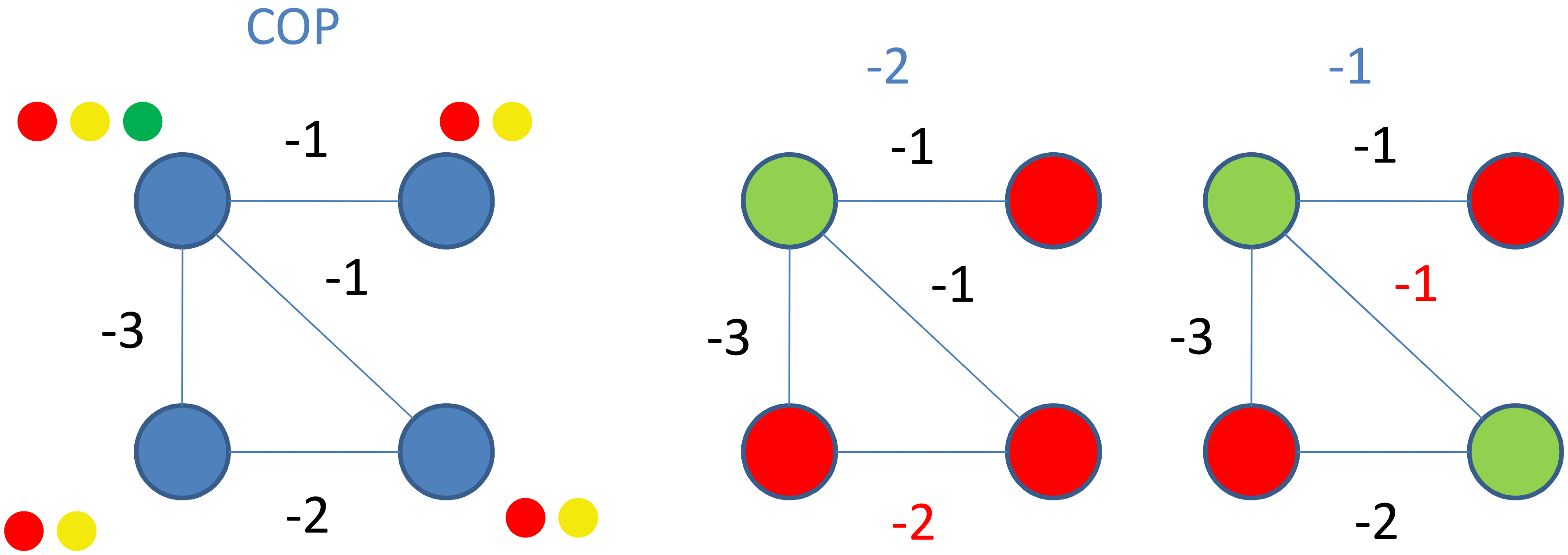






Weighted Graph Coloring - COP

- Optimization Problem
- Conflicts have a weight
- Maximise the sum of weights of violated constraints



Distributed Constraint Optimization: Exact Algorithms

Pedro Meseguer

Artificial Intelligence Research Institute, IIIA

Spanish Council for Scientific Research, CSIC

Distributed Constraint Optimization: Exact Algorithms

- Satisfaction: DisCSP
  - ABT (Yokoo et al 98)
- Optimization: DCOP
  - ADOPT (Modi et al 05)
  - BnB-ADOPT (Yeoh et al 10)
  - DPOP (Petcu & Faltings 05)

Distributed Algorithms

- *Synchronous*: agents take steps following some fixed order (or computing steps are done simultaneously, following some external clock).

- *Asynchronous*: agents take steps in arbitrary order, at arbitrary relative speeds.

- *Partially synchronous*: there are some restrictions in the relative timing of events

Synchonous vs Asynchronous

- *Synchronous*:
  - A few agents are active, most are waiting
  - Active agents take decisions with updated information
  - Low degree of concurrency / poor robustness
  - Algorithms: direct extensions of centralized ones
- *Asynchronous*:
  - All agents are active simultaneously
  - Information is less updated, obsolescence appears
  - High degree of concurrency / robust approaches
  - Algorithms: new approaches

From CSP to DisCSP

- CSP: (X, D, C)

  $X = \{x_1, x_2,\dots, x_n\}$ variables

  $D = \{d_1, d_2,\dots,d_n\}$ domains (finite)

  $C = \{c_1,c_2,\dots,c_r\}$ constraints

  c ⍰ C $var(c) = \{x_i, x_j,\dots, x_k\}$ *scope*

  $rel(c)$ ⍰ $d_i \times d_j \times .. \times d_k$ *permitted tuples*

- Solution: total assignment satisfying all constraints

- DisCSP: (X, D, C, A, ϕ)

  $A = \{a_1, a_2,\dots, a_k\}$ agents

  ϕ: X -> A maps variables to agents

  c is known by agents owning

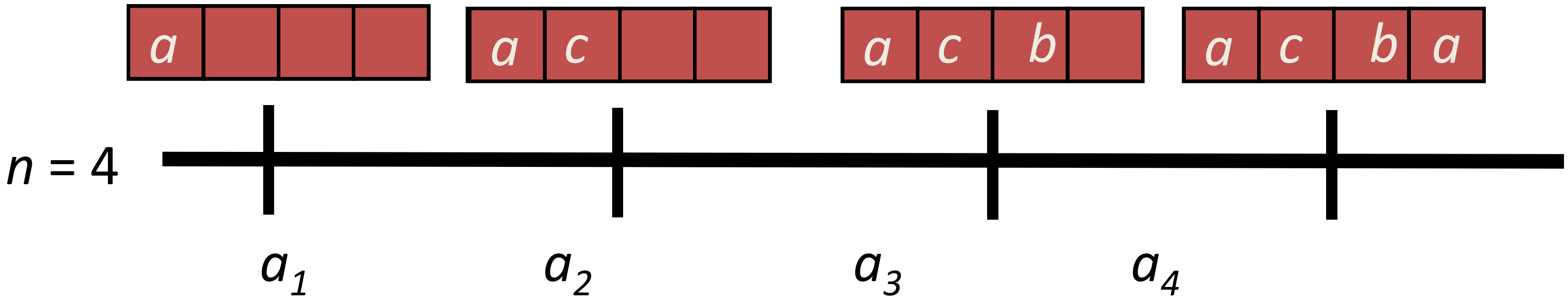
  $var(c)$

Assumptions

1. Agents communicate by *sending messages*
2. An agent can send messages to others, *iff it knowns their identifiers*
3. The *delay* transmitting a message is finite but random
4. For any pair of agents, *messages are delivered in the order they were sent*
5. Agents *know the constraints in which they are involved*, but not the other constraints
6. Each agent owns a *single variable (agents = variables)*
7. Constraints are *binary* (2 variables involved)

Synchronous Backtracking

- Total order of *n* agents: *like a railroad with n stops*
- Current partial solution: *train with n seats, forth/back*
- One place for each agent: when train reaches a stop
  - selects a new consistent value for the agent of this stop (wrt previous agents), it moves forward
  - otherwise, it moves backwards



SOLUTION

ABT: Asynchonous Backtracking

- The Distributed Constraint Satisfaction Problem: Formalization and Algorithms *Makoto Yokoo, Ed Durfee, Toru Ishida, Kazohiro Kuwabara*, IEEE Transactions Know. & Data Engineering, 1998

- First complete asynchronous algorithm for DisCSP solving

- Influential paper award in AAMAS 2010

Makoto Yokoo and colleagues

ABT: Description

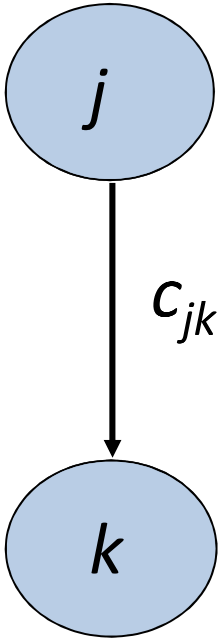
- Asynchronous:
  - All agents active, take a value and inform.
  - No agent has to wait for other agents
- Total order among agents: to avoid cycles

– *i* < *j* < *k* means that: *i* more priority than *j*,
j more priority than *k*

- Constraints are directed, following total order
- ABT plays in asyncronous distributed context the same role as backtracking in centralized

ABT: Directed Constraints

- Directed: from higher to lower priority agents



- Higher priority agent (*j*) informs the lower one (*k*) of its assignment
- Lower priority agent (*k*) evaluates the constraint with its own assignment
  - If permitted, no action
  - else it looks for a value consistent with *j*
    - If it exists, *k* takes that value
    - else, the agent view of *k* is a nogood, backtrack

generates nogoods: eliminate values of k

ABT: Nogoods

- Nogood: conjunction of (variable, value) pairs of higher priority agents, that removes a value of the current one
- Example: $x \neq y$, $d_x = d_y = \{a, b\}$, x higher than y

  when [x <- a] arrives to y, this agent generates the nogood

$$x=a \Rightarrow y \neq a$$

  that removes value a of d_y. If x changes value, when [x <- b] arrives to y, the nogood $x=a \Rightarrow y \neq a$ is eliminated, value a is again available and a new nogood removing b is generated.

ABT: Nogood Resolution

- When all values of variable *y* are removed, the conjunction of the left-hand sides of its nogoods is also a nogood.

- Resolution: the process of generating the new nogood.

- Example: : $x \neq y$, $z \neq y$, $d_x=d_y=d_z=\{a, b\}$, *x, z* higher than *y*

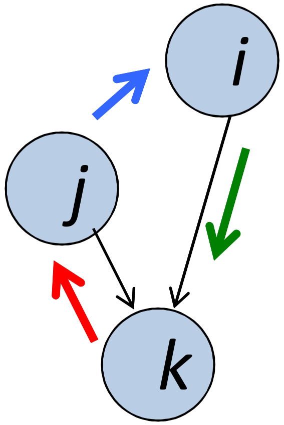
$x=a \Rightarrow y \neq a$; $x=a \wedge z=b$ is a nogood

$z=b \Rightarrow y \neq b$; $x=a \Rightarrow z \neq b$ (assuming *x* higher than *z*)

How ABT works

- ABT agents: asynchronous action; spontaneous assignment
- *Assignment*: *j* takes value *a*, *j* informs lower priority agents
- *Backtrack*: *k* has no consistent values with high priority agents, *k* resolves nogoods and sends a backtrack message
- *New links*: *j* receives a nogood mentioning *i*, unconnected with *j;* *j* asks *i* to set up a link
- *Stop*: "no solution" detected by an agent, stop
- Solution: when agents are silent for a while (quiescence),
  every constraint is satisfied-> solution;
  detected by specialized algorithms

ABT: Messages



- **Ok?**(*i->k,a*): *i* informs *k* that it takes value *a*

- **Ngd**(*k->j,i=a* ⇒ *j≠b*):
  all *k* values are forbidden,
  k requests *j* to backtrack,
  k forgets *j* value, *k* takes some value,
  j may detect obsolescence

- **AddI**(*j->i*): set a link from *i* to *j*, to know *i* value

- **Stop**: there is no solution

ABT: Data Structures

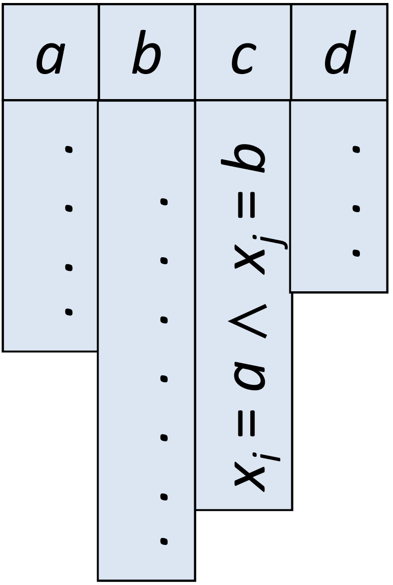
- *Current context* or *agent view*:
  values of higher priority constrained agents

x_i	x_j ...		
a	b		

- *Nogood store*:
  each removed value has a justifying nogood

$x_i = a \wedge x_j = b \Rightarrow x_k \neq c$ *justification*

higher priority



Stored nogoods must be active:

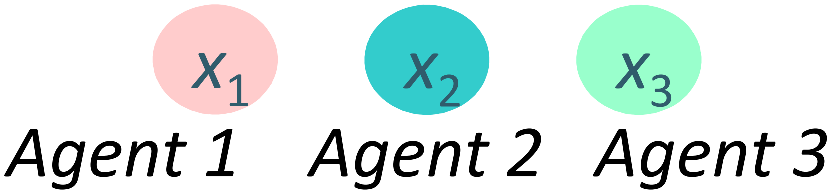
lhs $\subseteq$ current context

If a nogood becomes no active, it is removed (and the value is available again)

ABT: Graph Coloring Example

Variables x_1, x_2, x_3; $D_1 = \{b, a\}$, $D_2 = \{a\}$, $D_3 = \{a, b\}$

3 agents, lex ordered:

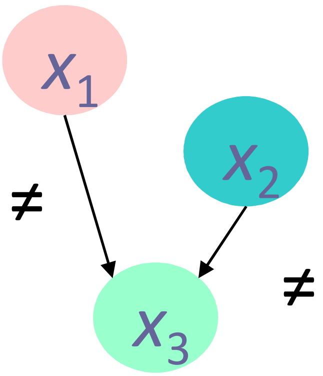


2 difference constraints: c_{13} and c_{23}

Constraint graph:

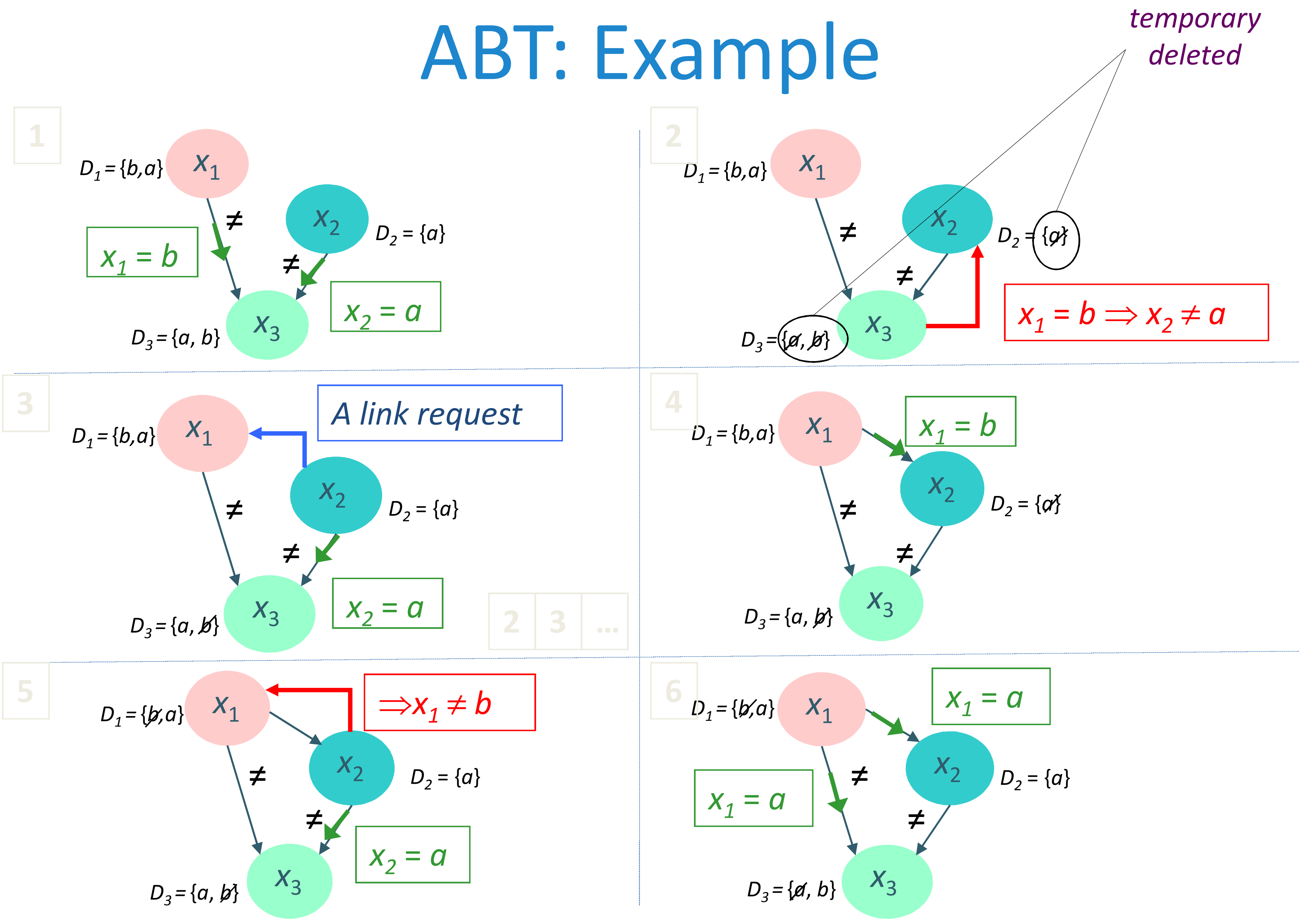
Value-sending agents: x_1 and x_2

Constraint-evaluating agent: x_3



Each agent *checks* constraints of incoming links: *Agent*$_1$ and *Agent*$_2$ check nothing, *Agent*$_3$ checks c_{13} and c_{23}

ABT: Example



temporary deleted

1

$D_1 = \{b,a\}$ x_1

$x_1 = b$

≠

x_2 $D_2 = \{a\}$

≠

$x_2 = a$

$D_3 = \{a, b\}$ x_3

2

$D_1 = \{b,a\}$ x_1

≠

x_2 $D_2 = \{\cancel{a}\}$

≠

$D_3 = \{\cancel{a}, \cancel{b}\}$ x_3

$x_1 = b \Rightarrow x_2 \neq a$

3

$D_1 = \{b,a\}$ x_1

A link request

x_2 $D_2 = \{a\}$

≠

≠

$x_2 = a$

$D_3 = \{a, \cancel{b}\}$ x_3

2 3 ...

4

$D_1 = \{b,a\}$ x_1

$x_1 = b$

x_2 $D_2 = \{\cancel{a}\}$

≠

≠

$D_3 = \{a, \cancel{b}\}$ x_3

5

$D_1 = \{\cancel{b},a\}$ x_1

$\Rightarrow x_1 \neq b$

x_2 $D_2 = \{a\}$

≠

≠

$x_2 = a$

$D_3 = \{a, \cancel{b}\}$ x_3

6

$D_1 = \{\cancel{b},a\}$ x_1

$x_1 = a$

x_2 $D_2 = \{a\}$

$x_1 = a$

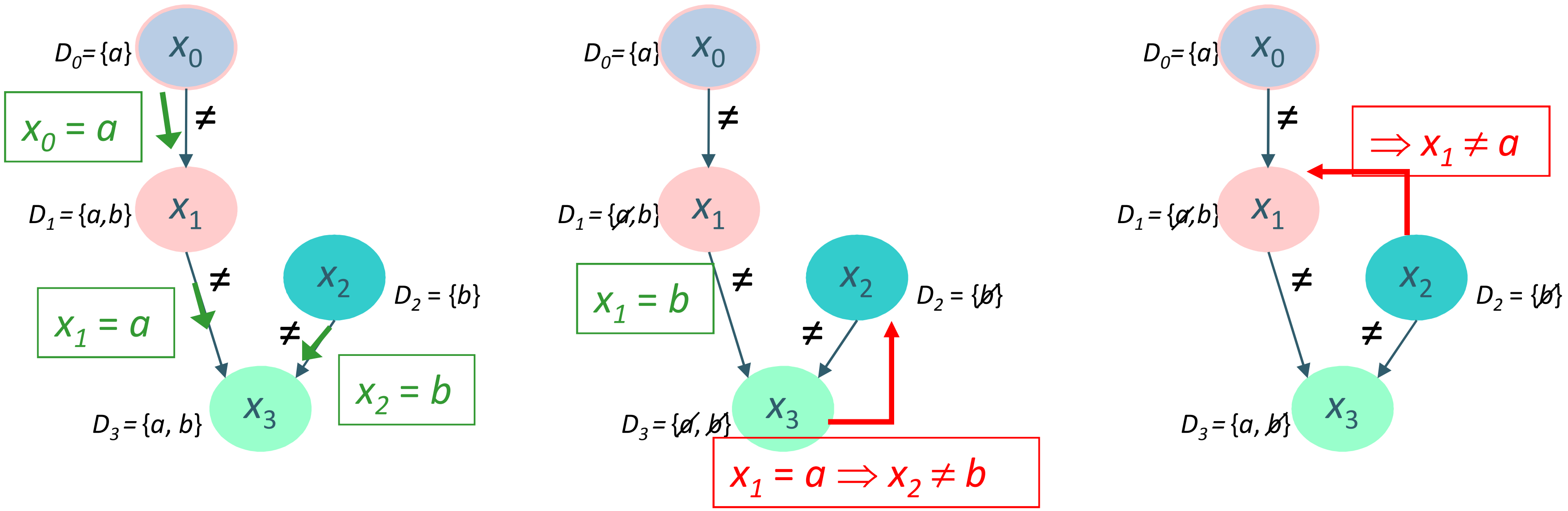
≠

≠

$D_3 = \{\cancel{a}, b\}$ x_3

ABT: Why AddLink?

- Imagine ABT without AddLink message



- *x*1 rejects Nogood message as obsolete, *x*2 keeps on sending it -> infinite loop!!
- AddLink avoids it: obsolete info is removed in finite time

ABT: Correctness / Completeness

- Correctness:
  - silent network <=> all constraints are satisfied

- Completeness:
  - ABT performs an exhaustive traversal of the search space
  - Parts not searched: those eliminated by nogoods
  - Nogoods are legal: logical consequences of constraints
  - Therefore, either there is no solution => ABT generates the empty nogood, or it finds a solution if exists

ABT: Termination

- There is no infinite loop
- Induction in depth of the agent:
  - Base case: the top agent x_1 receives Nogood messages only, with empty left-hand side; either it discard all values, generating the empty nogood, or remains in a value; either case it cannot be in an infinite loop
  - Induction case: assume x_1 … x_{k-1} are in stable state, x_k is looping
    x_k receives Nogood messages only containing x_1 … x_k
    either it discards all values, generating a Nogood for x_1 … x_{k-1}
    (breaking the assumption that x_1 … x_{k-1} are in stable state)
    or it finds a consistent value (breaking the assumption that x_k is in an infinite loop)

Evaluation Measures

- Synchronous cycles:
  - Cycle: Each agent wakes up, reads all incoming messages from previous cycle, and sends messages (ready to be read in the next cycle)
  - Suggested by Yokoo et al, in the original ABT paper
- Non-concurrent constraint checks:
  - Each agent keeps a counter of constraint checks
  - The counter is included in the messages sent by the agent
  - If an agent receives a message with a counter higher than its own counter, it copies the message counter
  - At the end, the highest counter is #NCCC
  - Suggested by Meisels et al, in 2002 (following logical clocks)

From DisCSPs to DCOPs

- DisCSP: (X, D, C, A, ϕ):

  C are boolean functions $c_i : \prod_{j \in var(c_j)} d_j$ -> {true, false}

- DCOP: (X, D, C, A, ϕ):

  C are cost functions $c_i : \prod_{j \in var(c_j)} d_j \rightarrow N$

  overall cost(t) = $\sum_i c_i(t)$

- Solution: total assignment with *acceptable* cost

- Optimal solution: total assignment with *minimum* cost

  arg min $\sum_{ij} c_{ij}(t)$ (binary cost functions)

Synchronous BnB

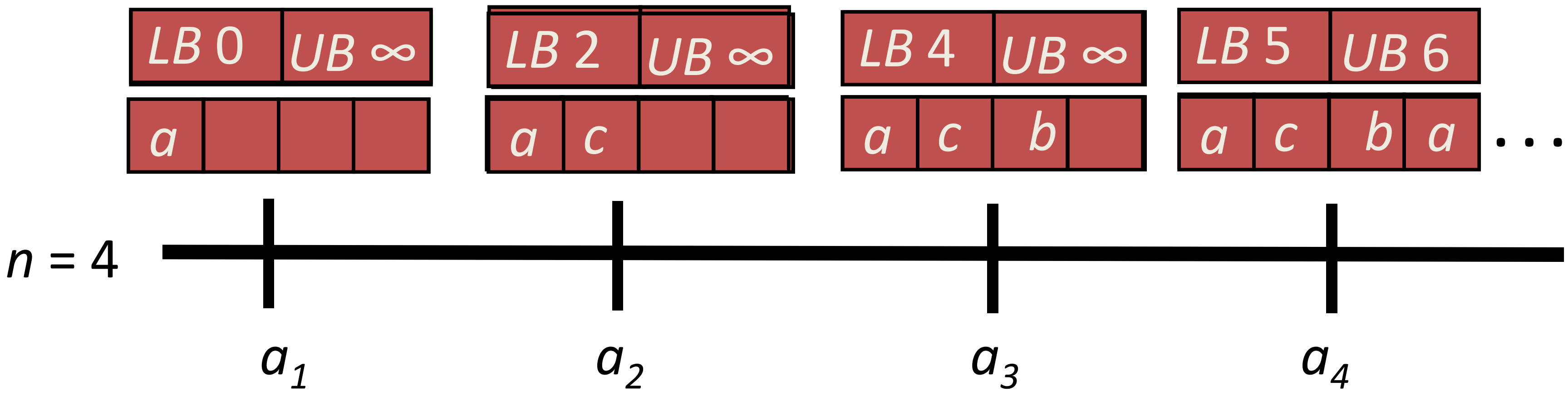
total order of agents: $a_1, a_2, \ldots, a_n$; *like a railroad with n stops*

CPS: current partial solution; *like a train that goes forth / back*

one place for each agent, plus LB, UB

When the train reaches a stop:

- *select a new value*
- *if LB < UB, move forward*
- *otherwise, select another value*
- *if no more values, move backward*



Inneficient Asynchronous DCOP

- DCOP: sequence of DisCSP, with decreasing thresholds

  DisCSP cost = *k*, DisCSP cost = *k*-1, DisCSP cost = *k*-2, ...

- ABT asynchronously solves each instance, until finding the first unsolvable instance.

- Synchrony on solving sequence instances

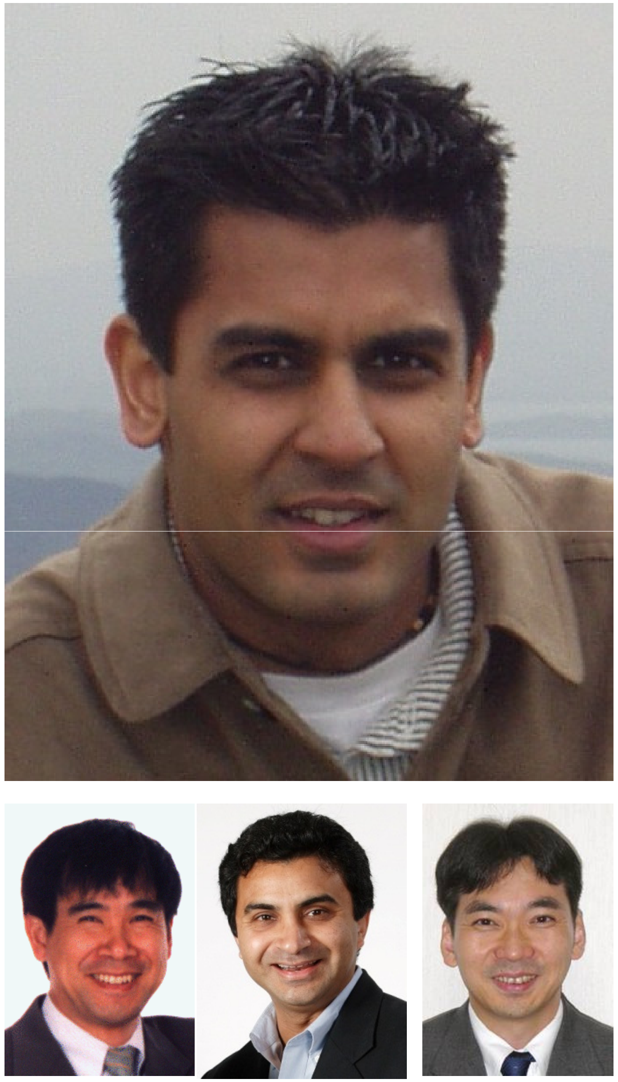
  cost *k* instance is solved before cost *k*-1 instance

- Very inefficient

ADOPT: Asynchronous Distributed Optimization

- ADOPT: asynchronous distributed constraint optimization with quality guarantees; *P. Jay Mody, W. M. Shen, M. Tambe, M. Yokoo* Artificial Intelligence, 2005

- First asynchonous complete algorithm for optimally solving DCOP

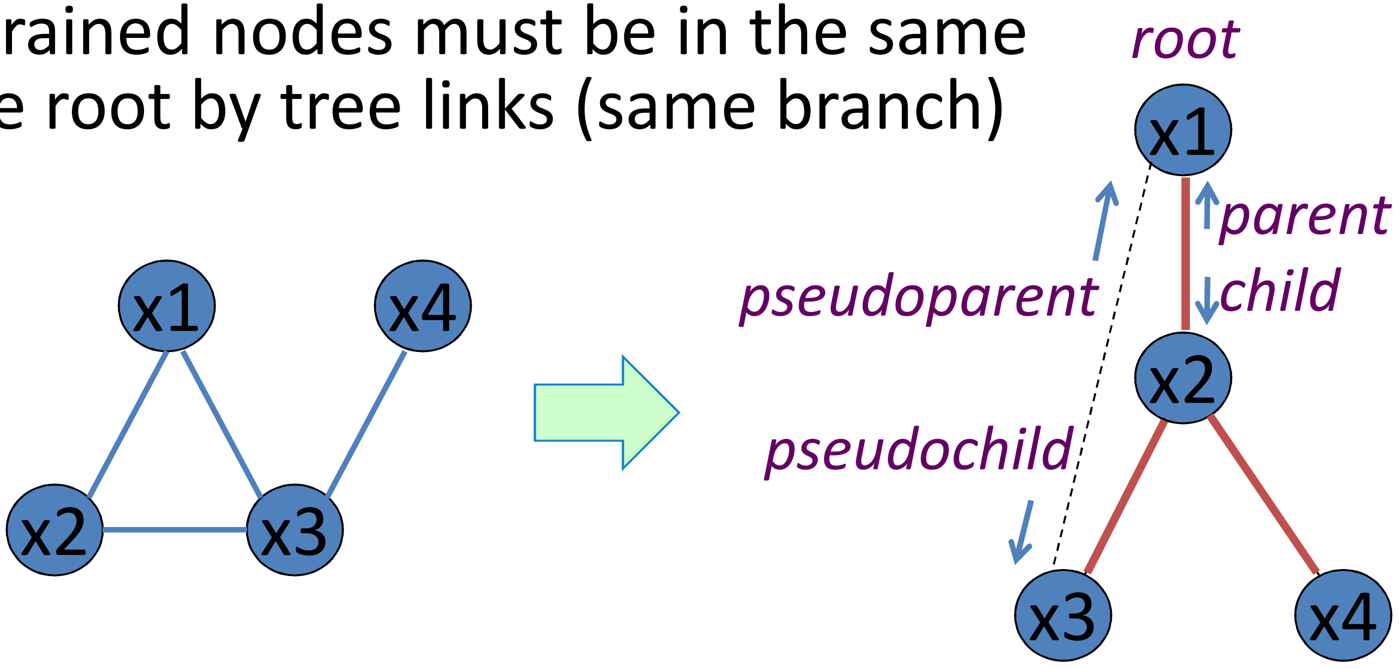


Pragnesh Jay Modi and colleagues

ADOPT: DFS tree (pseudotree)

ADOPT assumes that agents are arranged in a DFS tree:

- constraint graph -> rooted graph (select a node as root)
- some links form a tree / others are backedges
- two constrained nodes must be in the same path to the root by tree links (same branch)



Every graph admits a DFS tree: DFS graph traversal

ADOPT Description

- Asynchronous algorithm
- Each time an agent receives a message:
  - Processes it (the agent may take a new value)
  - Sends VALUE messages to its children and pseudochildren
  - Sends a COST message to its parent
- Context: set of (variable value) pairs (as ABT agent view) of ancestor agents (in the same branch)
- Current context:
  - Updated by each VALUE message
  - If current context is not compatible with some child context, the later is initialized (also the child bounds)

ADOPT Messages

- **value** (*parent -> children* U *pseudochildren*, *a*):

  parent informs descendants that it has taken value *a*

- **cost**(*child->parent, lower bound, upper bound, context*):

  child informs *parent* of the best cost of its assignement; attached context to detect obsolescence;

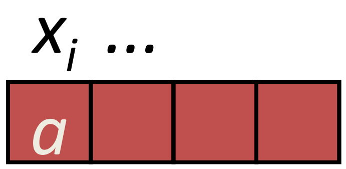
- **threshold** (*parent-> child, t*): minimum cost of solution in child is at least *t*

- **termination** (*parent-> children*): *LB = UB*

ADOPT Data Structures

1.Current context (agent view):

values of higher priority constrained agents



x_i	...		
a			

2. Bounds:

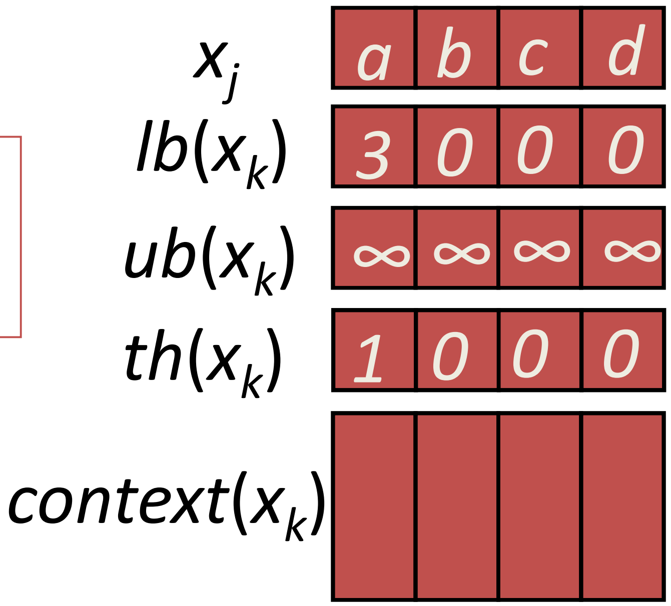
lower bounds

upper bounds

thresholds

contextes

for each value, child



x_j	*a*	*b*	*c*	*d*
$lb(x_k)$	*3*	*0*	*0*	*0*
$ub(x_k)$	∞	∞	∞	∞
$th(x_k)$	*1*	*0*	*0*	*0*
$context(x_k)$				

Stored contextes must be active:

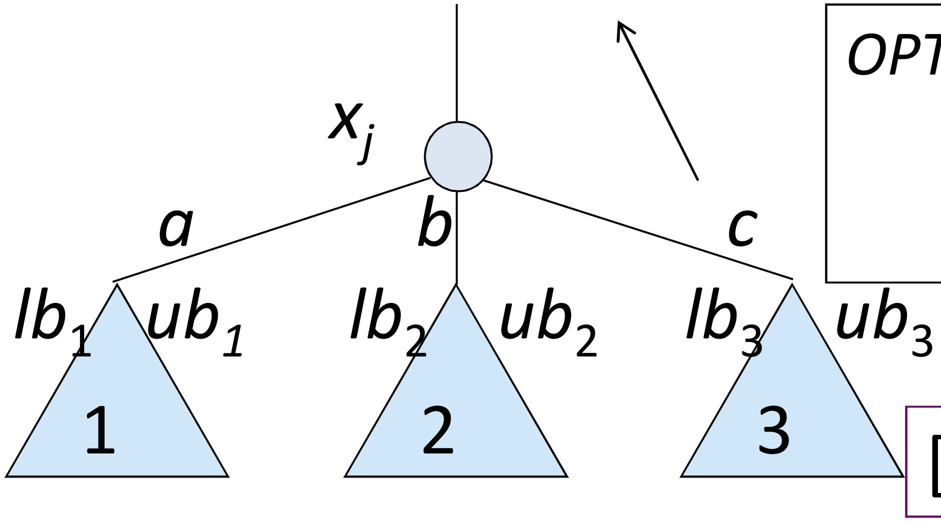
context $\subseteq$ *current context*

If a context becomes no active, it is removed *lb,th*<-0, *ub*<-∞

ADOPT Bounds

δ(*value*)= *cost with higher agents*

$$\delta(b) = \sum_{i \in \text{ current-context}} c_{ij}(a,b)$$



$$OPT(x_j, CurCont) = \min_{d \in dj} \delta(d) + \sum_{xk \in Children} OPT(x_k, CurCont \cup (x_j, d))$$

$[lb_k, ub_k]$ = *cost of lower agents*

$$LB(b)=\delta(b)+\sum_{xk \in children} lb(b,x_k)$$

$$LB=\min_{b \in dj} LB(b)$$

$$UB(b) = \delta(b) + \sum_{xk \in children} ub(b,x_k)$$

$$UB = \min_{b \in dj} UB(b)$$

ADOPT: Value Assignment

- An ADOPT agent takes the value with minimum LB
- Eager behavior:
  - Agents may constantly change value
  - Generates many context changes
- Threshold:
  - lower bound of the cost that children have from previous search
  - parent distributes threshold among children
  - incorrect distribution does not cause problems: the child with minor allocation would send a COST to the parent later, and the parent will rebalance the threshold distribution

ADOPT: Properties

- For any x_i , $LB \leq OPT(x_l,CurCont) \leq UB$
- For any x_i , its threshold reaches *UB*
- For any x_i , its final threshold is equal to $OPT(x_l,CurCont)$
  [ADOPT terminates with the optimal solution]

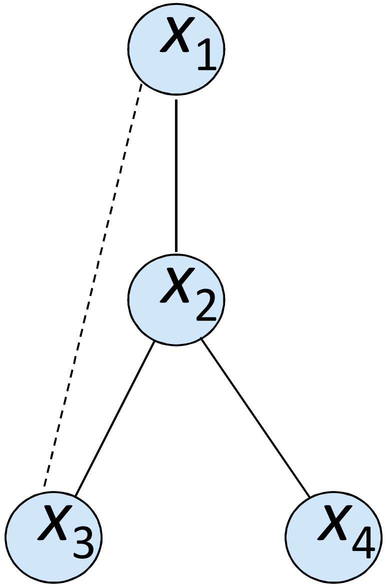
ADOPT: Example

4 Variables (4 agents) x_1, x_2, x_3 x_4 with $D = \{a, b\}$

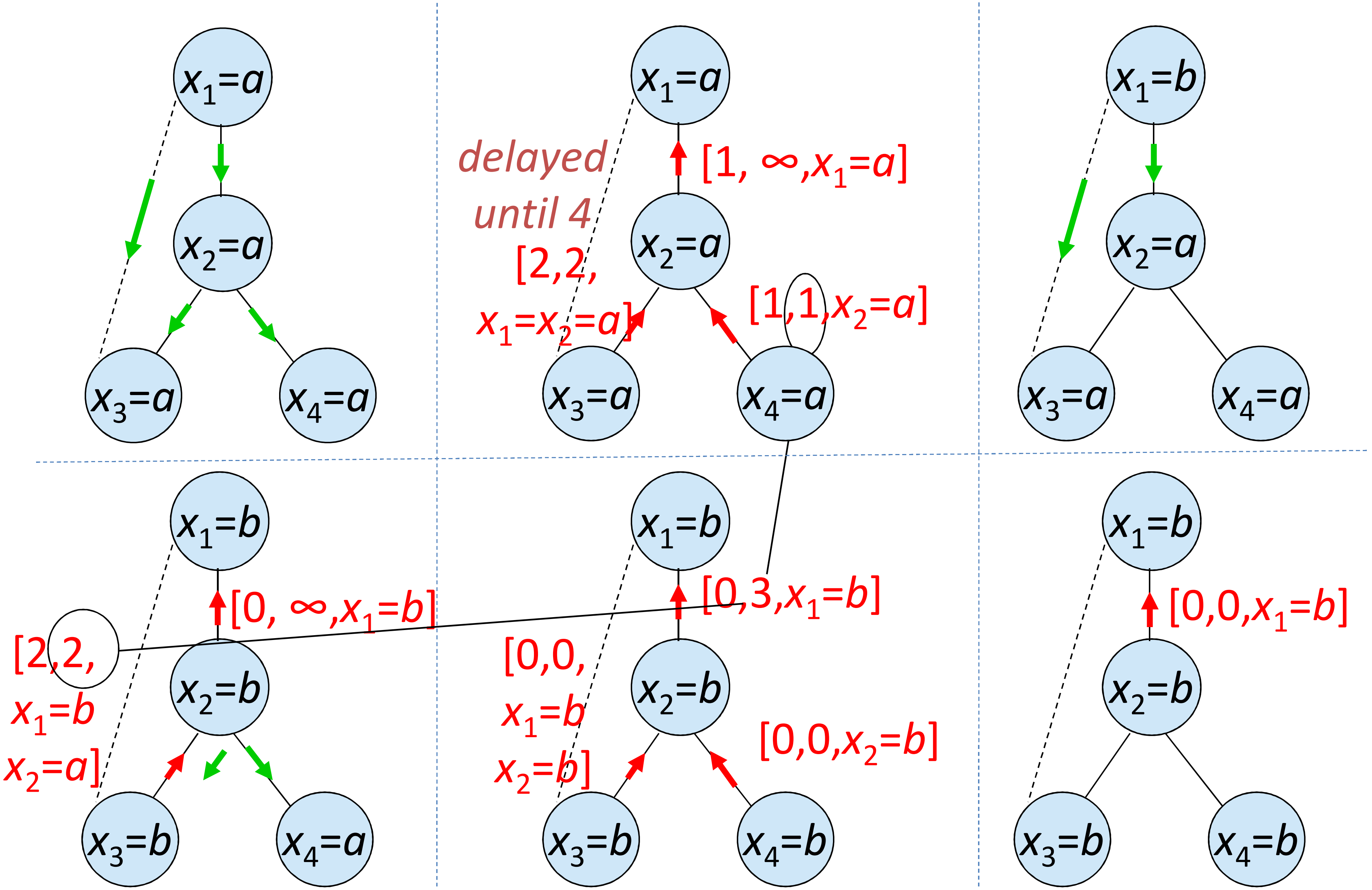
4 binary identical cost functions

x_i	x_j	
a	*a*	1
a	*b*	2
b	*a*	2
b	*b*	0

Constraint graph:



ADOPT: Example

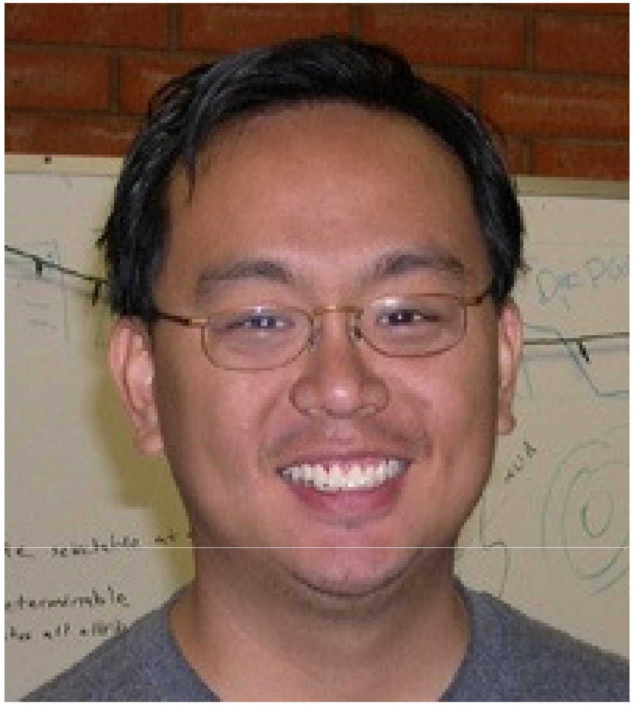


BnB-ADOPT

- BnB-ADOPT: an asynchronous branch-and-bound DCOP algorithm; *W. Yeoh, A. Felner, S. Koenig,* JAIR 2010

- ADOPT branch-and-bound version

- Changes best-first by depth-first branch-and-bound strategy



William Yeoh and colleagues

BnB-ADOPT: Description

- Basically same messages / data structures as ADOPT
- Changes:
  - All incoming messages are processed before taking value
  - Timestamp for each value
  - The agent context may be updated with VALUE and with COST
  - THRESHOLD is included in VALUE (so BnB-ADOPT messages are VALUE, COST and TERMINATE)
- **Main change:** *value selection*
  - Agent changes value when the current value is definitely worse than another value ($LB(current\text{-}value) \geq UB$)
  - Thresholds are upper bounds (not lower bound like in ADOPT)

BnB-ADOPT: Messages

- **value** (*parent* -> *children* U *pseudochildren*, *a*, *t*):

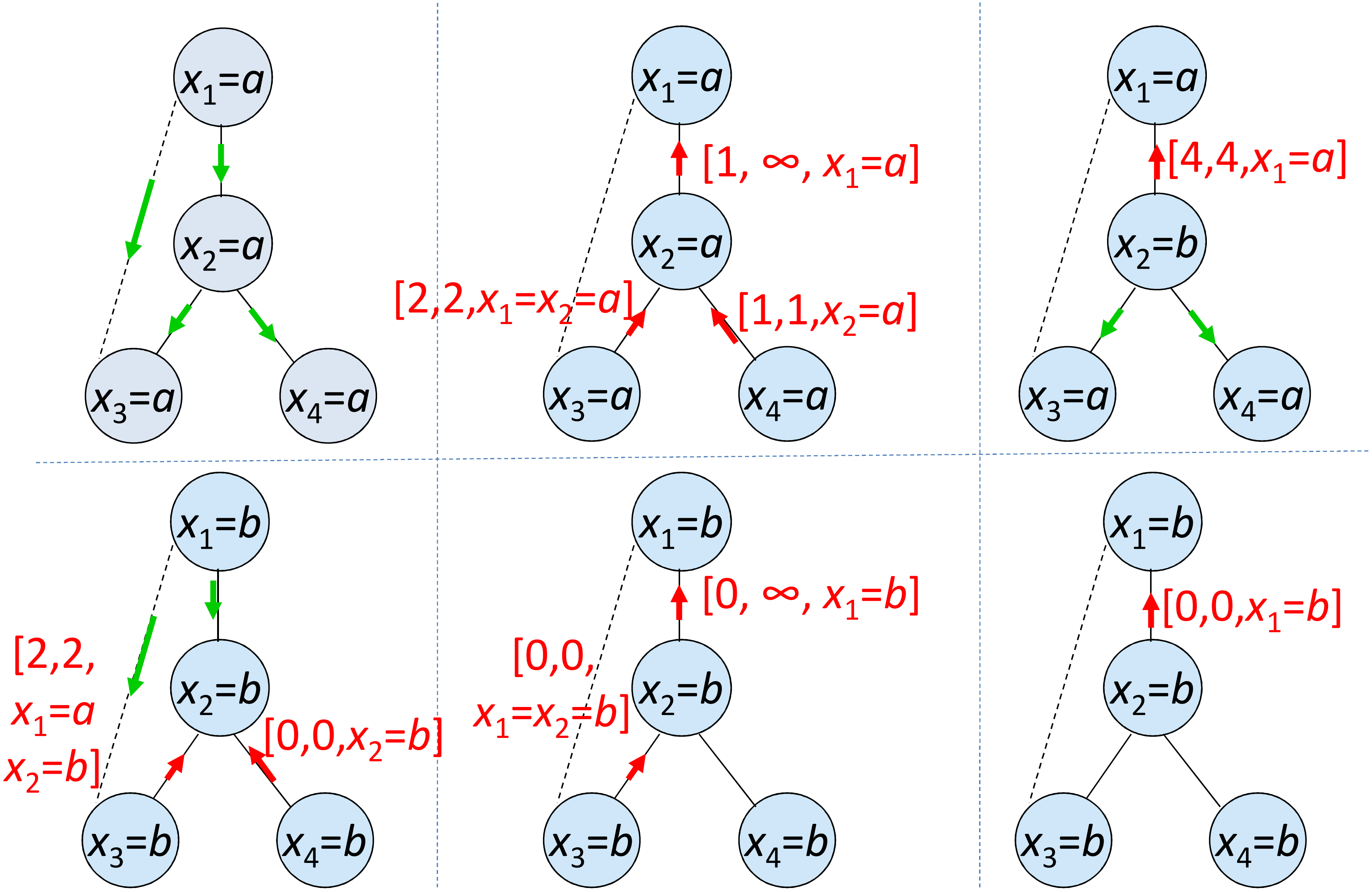
  parent informs descendants that it has taken value *a,*
  children threshold is *t* (pseudochildren threshold is ∞)

- **cost**(*child->parent*, *lower bound, upper bound, context*):

  child informs *parent* of the best cost of its assignement,
  attached context to detect obsolescence

- **termination** (*parent-> children*): *LB* = *UB*

BnB-ADOPT: Example



BnB-ADOPT: Performance

- BnB-ADOPT agents change value less frequently -> less context changes
- BnB-ADOPT uses less messages / less cycles than ADOPT
- Best-first strategy does not pay-off in terms of messages/cycles

BnB-ADOPT: Redundant Messages

- Many VALUE / COST messages are redundant

- Detection of redundant messages:
  [Gutierrez, Meseguer AAAI 2010]
  - VALUE to be sent: if it is equal to the last VALUE sent, it is redundant
  - COST to be sent: if it is equal to the last COST sent and there is no context change, it is redundant
  - For efficiency: take thresholds into account

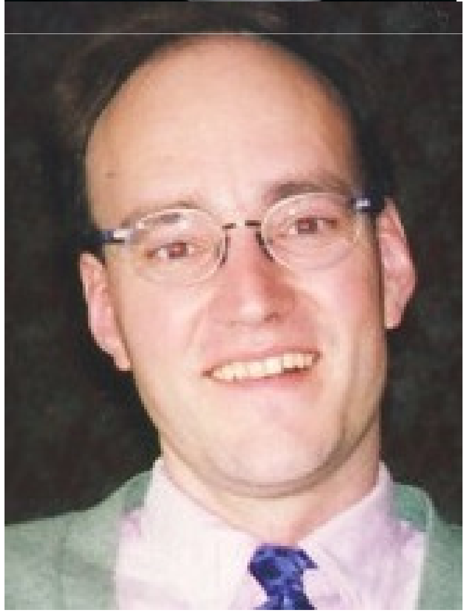
- Significant decrement in #messages, keeping optimality

A New Strategy

- So far, algorithms (ABT, ADOPT,…) exchanged individual assignments: *distributed search*
  - Small messages, but …
  - Exponentially many

- A different approach, exchanging cost functions: *distributed inference (dynamic programming)*
  - A few messages, but …
  - Exponentially large

DPOP: Dynamic Programming Optimization Protocol



Adrian Petcu and Boi Faltings

- DPOP: A scalable method for distributed constraint optimization *A. Petcu, B. Faltings*; IJCAI 2005
- In distributed, it plays the same role as ADC in centralized

DPOP phases/messages

PHASES	*MESSAGES*
1. DFS tree construction	• token passing
2. Utility phase: from leaves to root	• **util** (*child -> parent, constraint table* [-*child*])
3. Value phase: from root to leaves	• **value** (*parent -> children, pseudochildren, parent value*)

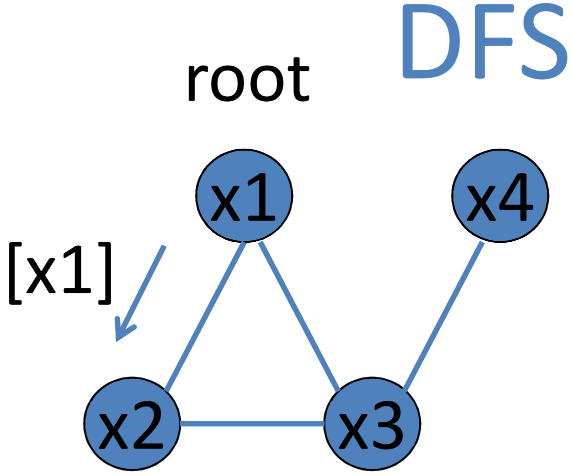
DPOP: DFS tree phase

Distributed DFS graph traversal: token, ID, *neighbors*(*X*)

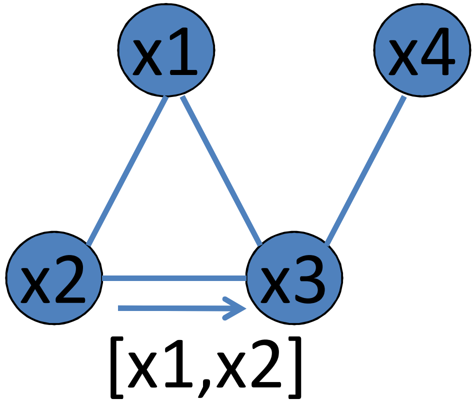
1. *X* owns the token: adds its own ID and sends it in turn to each of its neighbors, which become *children*.
2. *Y* receives the token from *X*: it marks *X* as visited. First time *Y* receives the token then *parent*(*Y*)=*X*. Other IDs in token which are also *neighbors*(*Y*) are *pseudoparent*. If *Y* receives token from neighbor *W* to which it was never sent, *W* is *pseudochild*.
3. When all *neighbors*(*X*) visited, *X* removes its ID from token and sends it to *parent*(*X*).

A node is selected as root, which starts. When all neighbors of root are visited, the DFS traversal ends.

DFS phase: Example

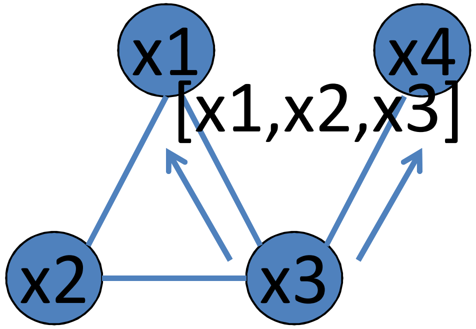


x1 parent of x2



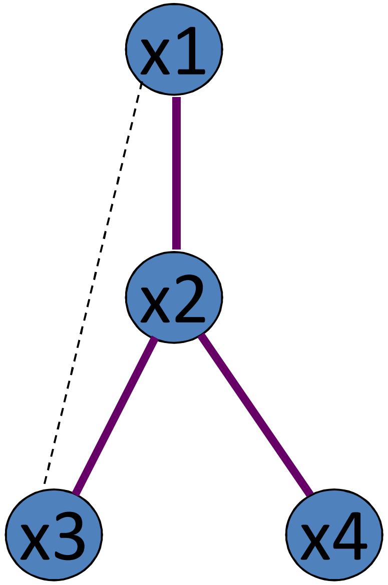
x2 parent of x3

x1 pseudoparent of x3



x3 parent of x4

x3 pseudochild of x1



DPOP: Util phase

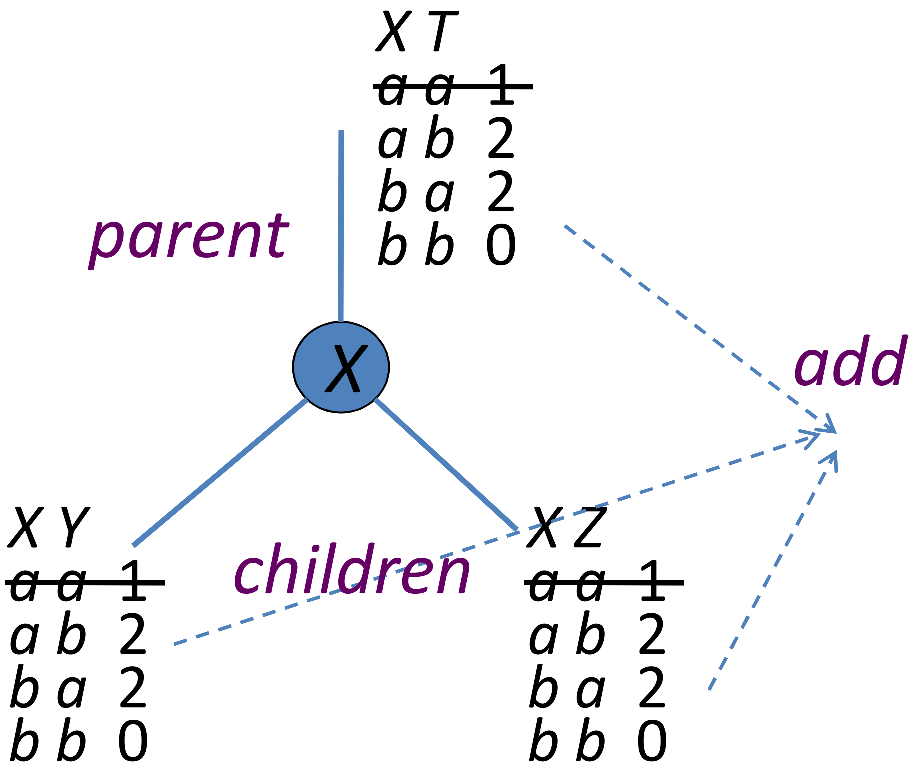
Agent *X*:

1. receives from each child Y_i a cost function: $C(Y_i)$
2. combines (adds, joins) all these cost functions with the cost functions with *parent*(*X*) and *pseudoparents*(*X*)
3. projects *X* out of the resulting cost function, and sends it to *parent*(*X*)

From the leaves to the root.

DPOP: util phase example

cost function to parent(X)



X	T	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

X	Y	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

X	Z	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

add

X	Y	Z	T	
~~a~~	~~a~~	~~a~~	~~a~~	~~3~~
a	a	a	b	4
a	a	b	a	4
a	a	b	b	5
a	b	a	a	4
a	b	a	b	5
a	b	b	a	5
a	b	b	b	6
b	a	a	a	6
b	a	a	b	4
b	a	b	a	4
b	a	b	b	2
b	b	a	a	4
b	b	a	b	2
b	b	b	a	2
b	b	b	b	0

All value combinations
Costs are the sum of applicable costs

Project out X

~~X~~	Y	Z	T	
~~a~~	~~a~~	~~a~~	~~a~~	~~3~~
~~a~~	a	a	b	4
~~a~~	a	b	a	4
~~a~~	a	b	b	5
~~a~~	b	a	a	4
~~a~~	b	a	b	5
~~a~~	b	b	a	5
~~a~~	b	b	b	6
~~b~~	a	a	a	6
~~b~~	a	a	b	4
~~b~~	a	b	a	4
~~b~~	a	b	b	2
~~b~~	b	a	a	4
~~b~~	b	a	b	2
~~b~~	b	b	a	2
~~b~~	b	b	b	0

Remove X
Remove duplicates keep the min cost

DPOP: Value phase

1. The root finds the value that minimizes the received cost function in the util phase, and informs its descendants (children U pseudochildren)
2. Each agent waits to receive the value of its parent / pseudoparents
3. Keeping fixed the value of parent/pseudoparents, finds the value that minimizes the received cost function in the util phase
4. Informs of this value to its children/pseudochildren

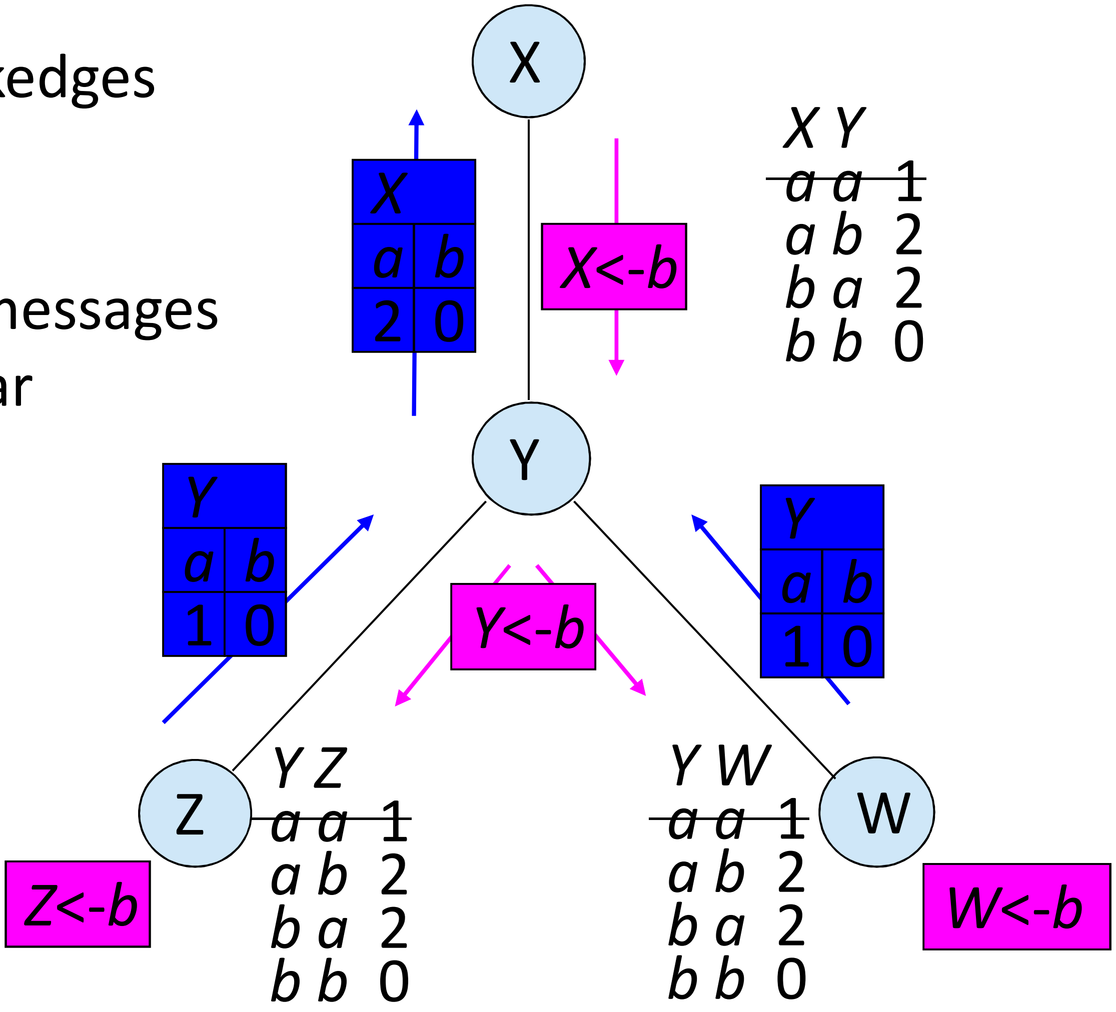
This process starts at the root and ends at the leaves

DTREE

For DCOPs without backedges

Optimal solution:

- linear number of messages
- message size: linear



X	
a	b
2	0

Y	
a	b
1	0

Y	
a	b
1	0

X	Y	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

Y	Z	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

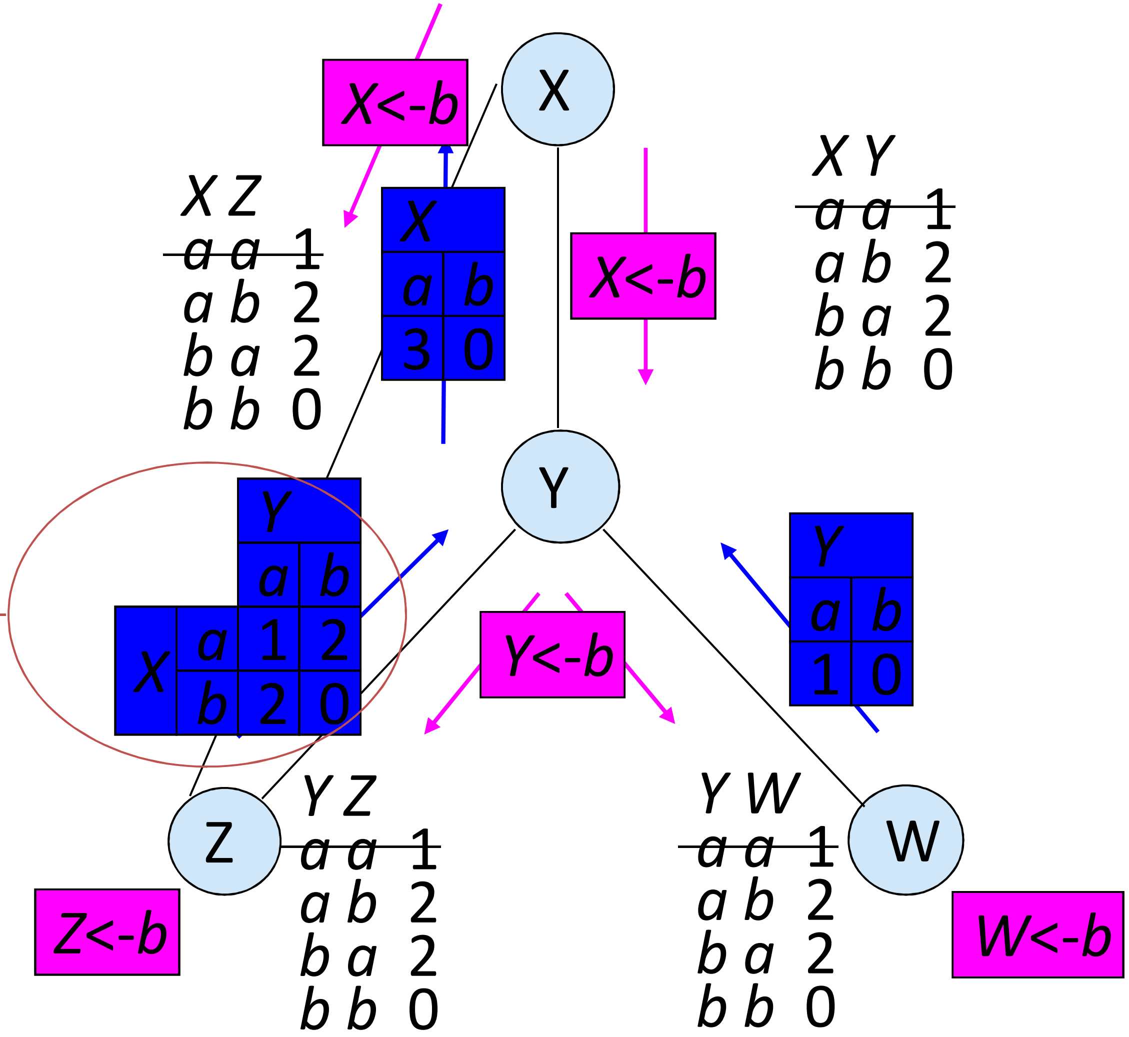
Y	W	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

DPOP: Example

For any DCOP

Optimal solution:

- linear number of messages
- message size: exponential



X	Z	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

X	Y	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

Y	Z	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

Y	W	
~~a~~	~~a~~	~~1~~
a	b	2
b	a	2
b	b	0

X	
a	b
3	0

	Y	
X	a	b
a	1	2
b	2	0

Y	
a	b
1	0

DPOP: Performance

- Synchronous algorithm, linear number of messages
- **util** messages can be exponentially large: main drawback
- Function filtering can alleviate this problem
- DPOP completeness: direct, from Adaptive Consistency results in centralized

Distributed Constraint Optimization: Approximate Algorithms

Alessandro Farinelli

Computer Science Department

University of Verona, Italy

Approximate Algorithms: outline

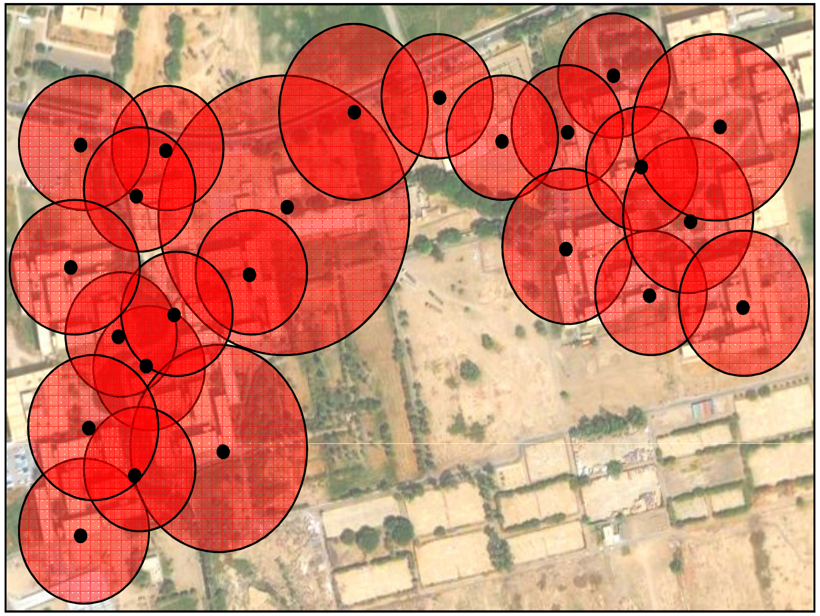
- No guarantees
  - DSA-1, MGM-1 (exchange individual assignments)
  - Max-Sum (exchange functions)
- Off-Line guarantees
  - K-optimality and extensions
- On-Line Guarantees
  - Bounded max-sum

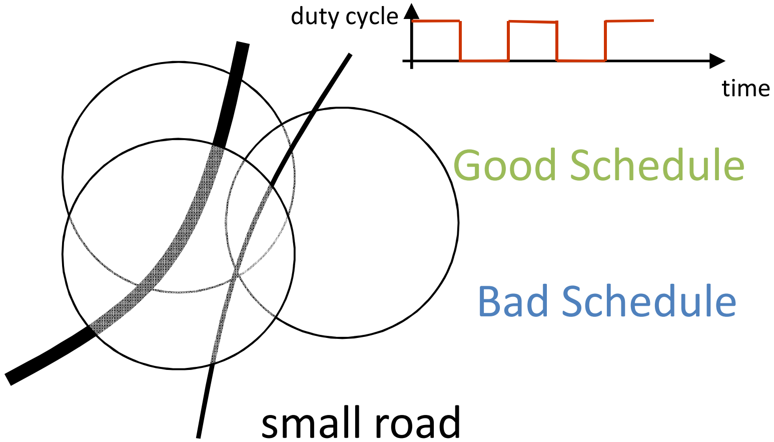
Why Approximate Algorithms

- Motivations
  - Often optimality in practical applications is not achievable
  - Fast good enough solutions are all we can have
- Example – Graph coloring
  - Medium size problem (about 20 nodes, three colors per node)
  - Number of states to visit for optimal solution in the worst case 3^20 = 3 billions of states
- Key problem
  - Provides guarantees on solution quality

Exemplar Application: Surveillance

- Event Detection
  - Vehicles passing on a road
- Energy Constraints
  - Sense/Sleep modes
  - Recharge when sleeping
- Coordination
  - Activity can be detected by single sensor
  - Roads have different traffic loads
- Aim
  - Focus on road with more traffic load





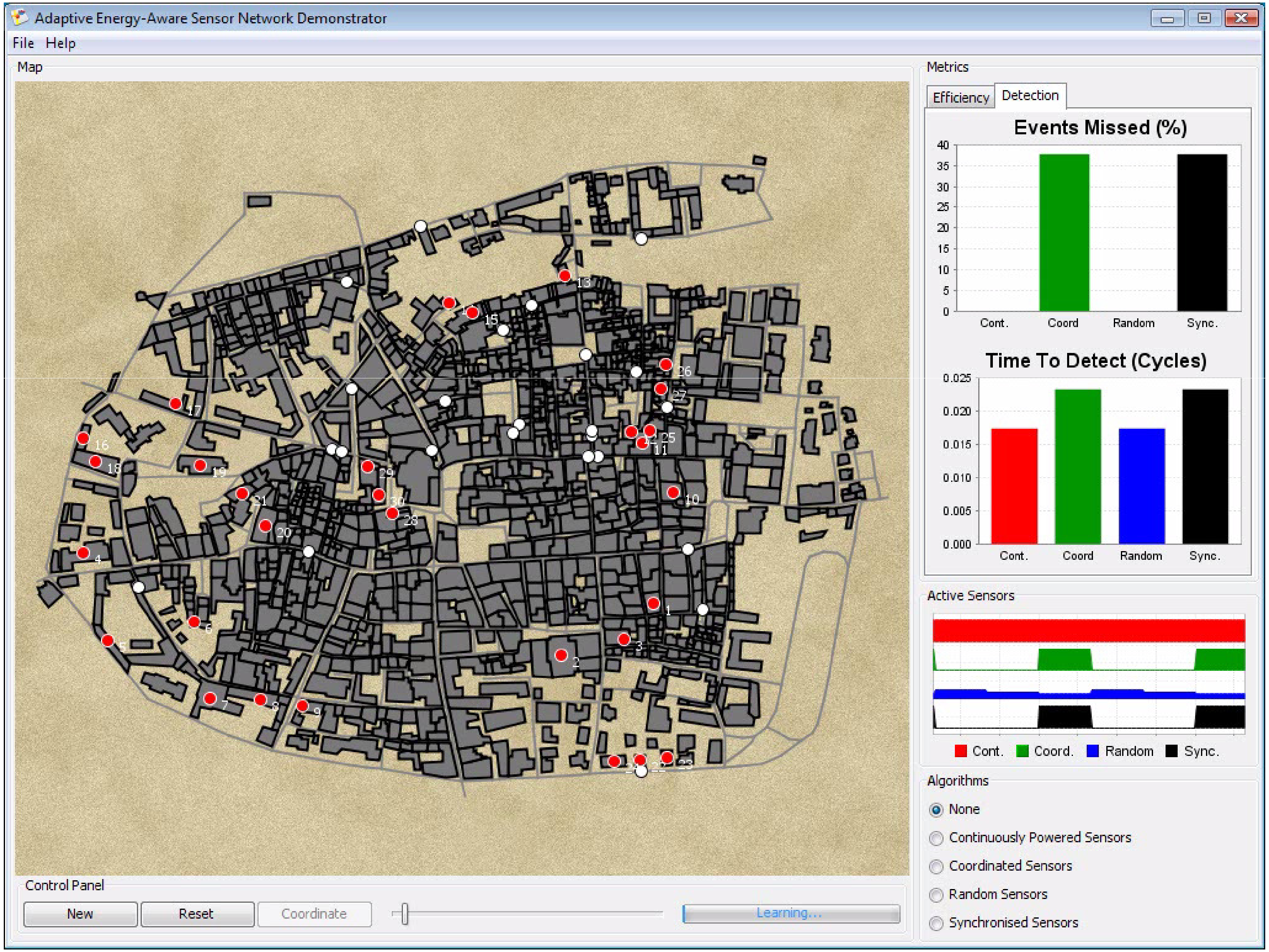
Good Schedule

Bad Schedule

Heavy traffic road

small road

Surveillance demo



Algorithms

● None

○ Continuously Powered Sensors

○ Coordinated Sensors

○ Random Sensors

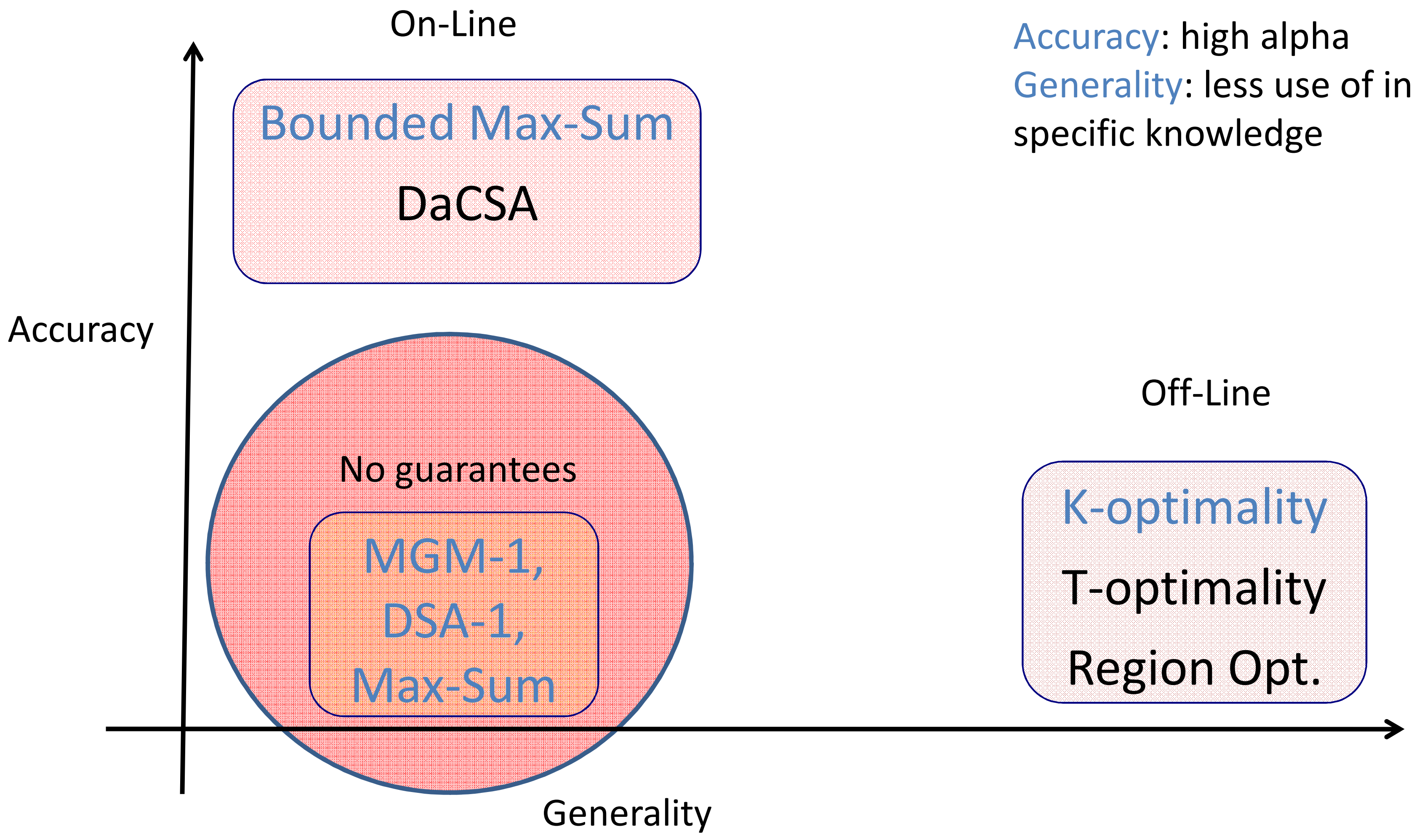
○ Synchronised Sensors

Joint work with:
A. Rogers,
N. R. Jennings

Guarantees on solution quality

- Key Concept: bound the optimal solution
  - Assume a maximization problem
  - X^* optimal solution, $\tilde{X}$ a solution
  - $F(\tilde{X}) \geq \alpha F(X^*)$
  - α percentage of optimality
    - [0,1]
    - The higher the better
  - $\rho = \frac{1}{\alpha}$ approximation ratio
    - >= 1
    - The lower the better
  - $\rho F(\tilde{X})$ is the bound

Types of Guarantees



On-Line

Bounded Max-Sum

DaCSA

Accuracy

No guarantees

MGM-1, DSA-1, Max-Sum

Off-Line

K-optimality

T-optimality

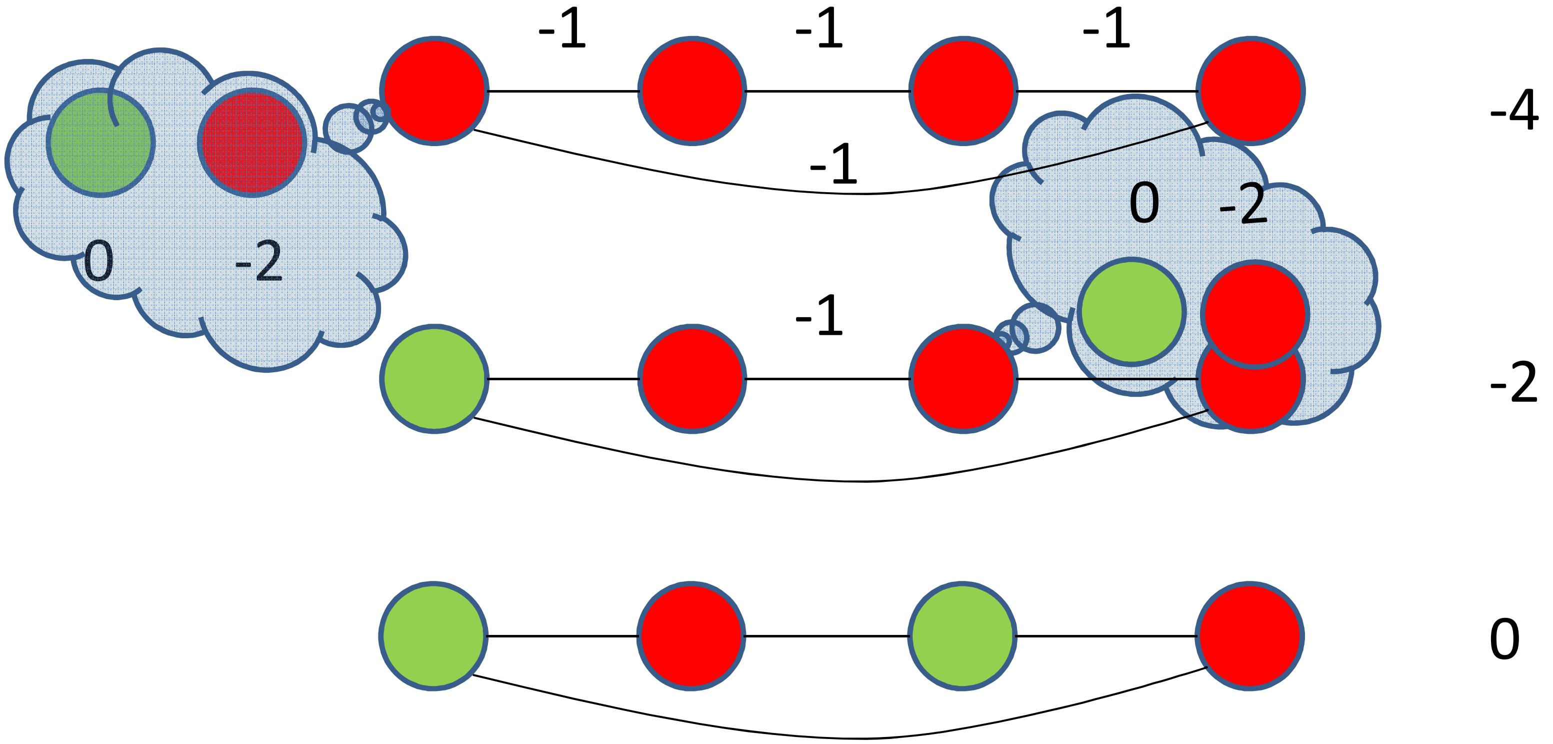
Region Opt.

Generality

Accuracy: high alpha
Generality: less use of instance specific knowledge

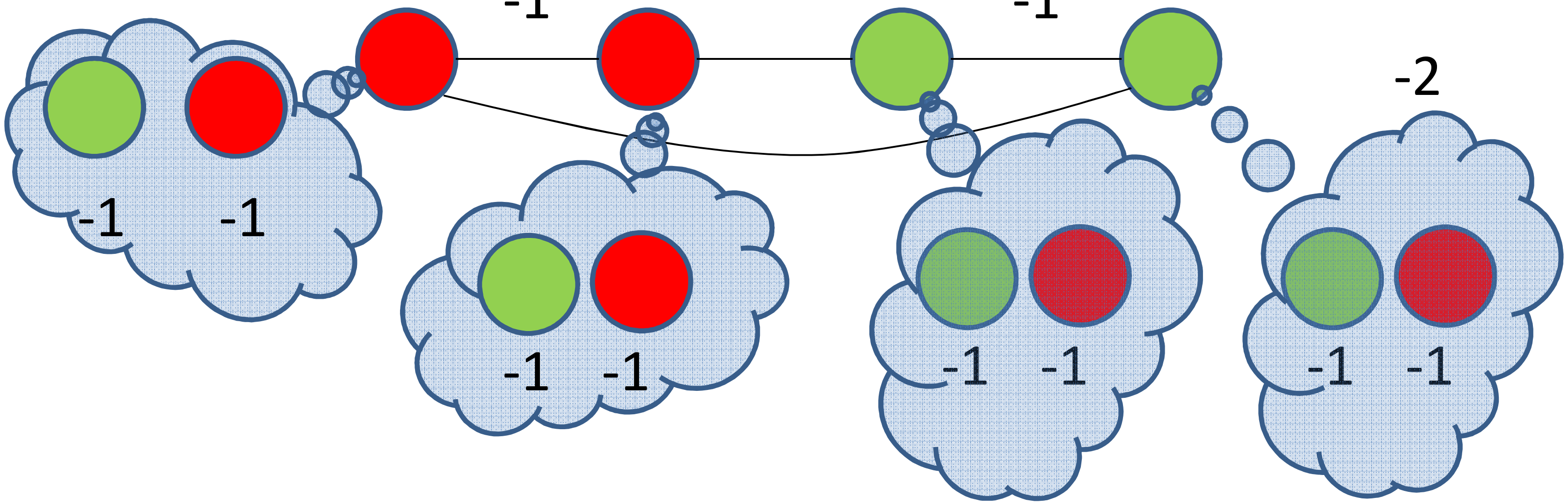
Centralized Local Greedy approaches

- Greedy local search
  - Start from random solution
  - Do local changes if global solution improves
  - Local: change the value of a subset of variables, usually one



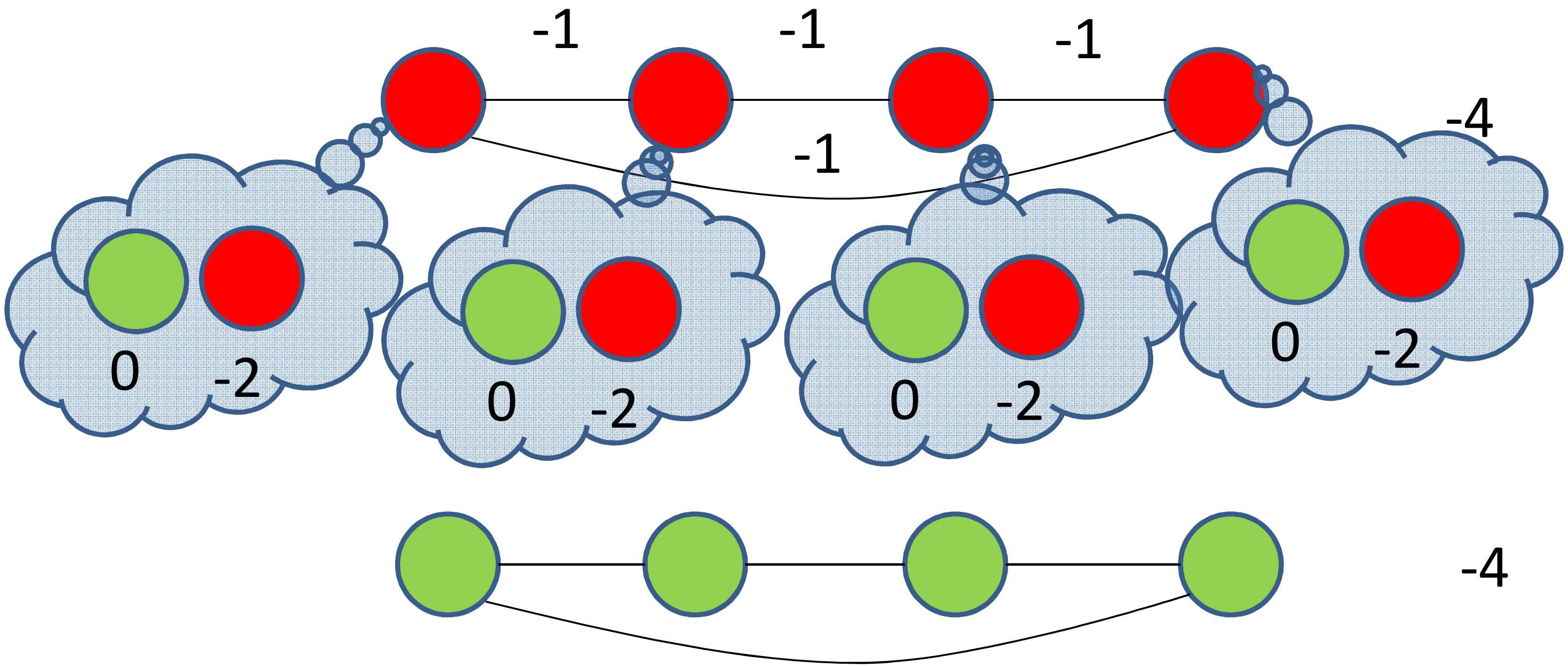
Centralized Local Greedy approaches

- Problems
  - Local minima
  - Standard solutions: RandomWalk, Simulated Annealing



Distributed Local Greedy approaches

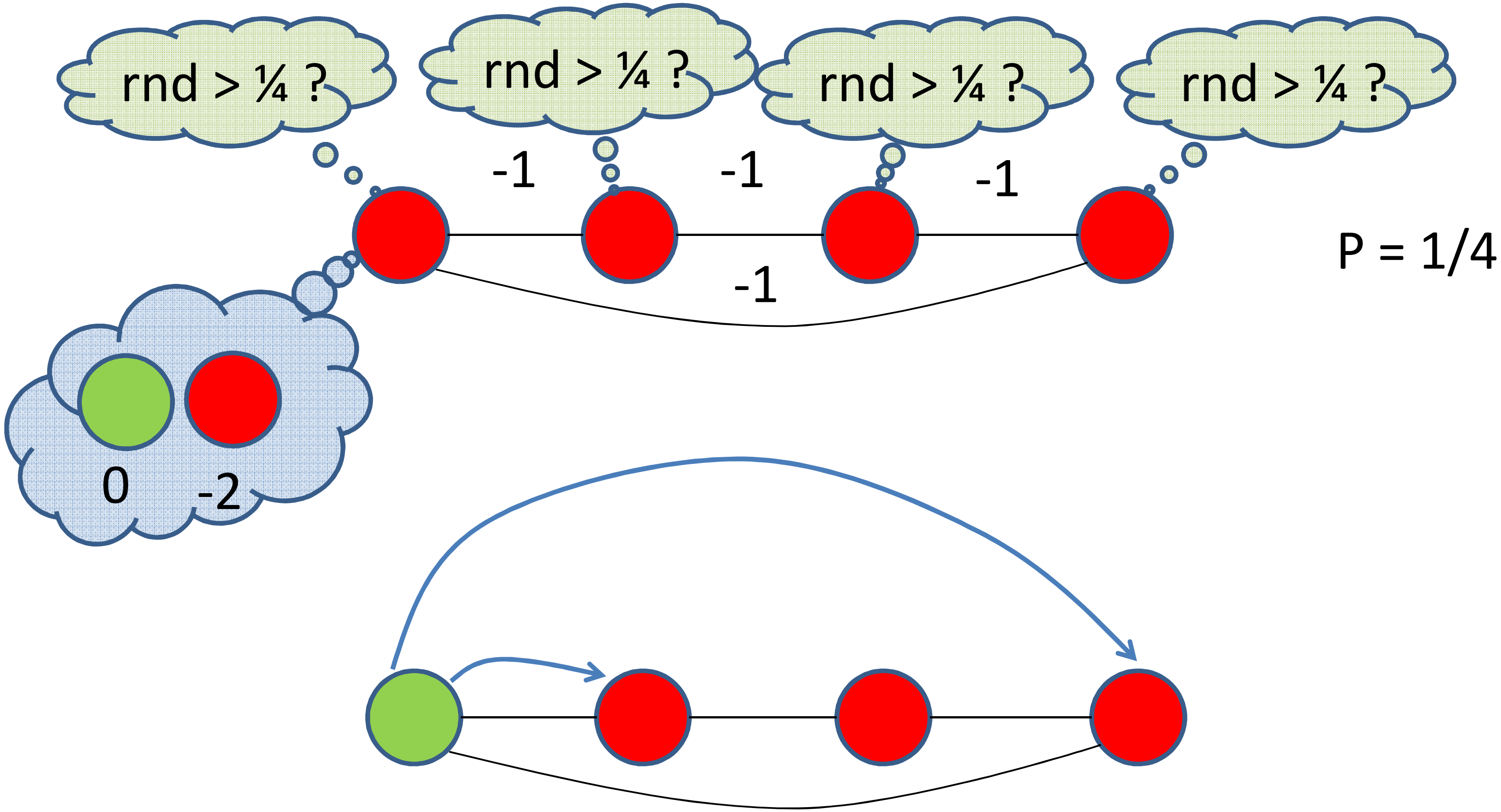
- Local knowledge
- Parallel execution:
  - A greedy local move might be harmful/useless
  - Need coordination



Distributed Stochastic Algorithm

- Greedy local search with activation probability to mitigate issues with parallel executions
- DSA-1: change value of one variable at time
- Initialize agents with a random assignment and communicate values to neighbors
- Each agent:
  - Generates a random number and execute only if rnd less than activation probability
  - When executing changes value maximizing local gain
  - Communicate possible variable change to neighbors

DSA-1: Execution Example



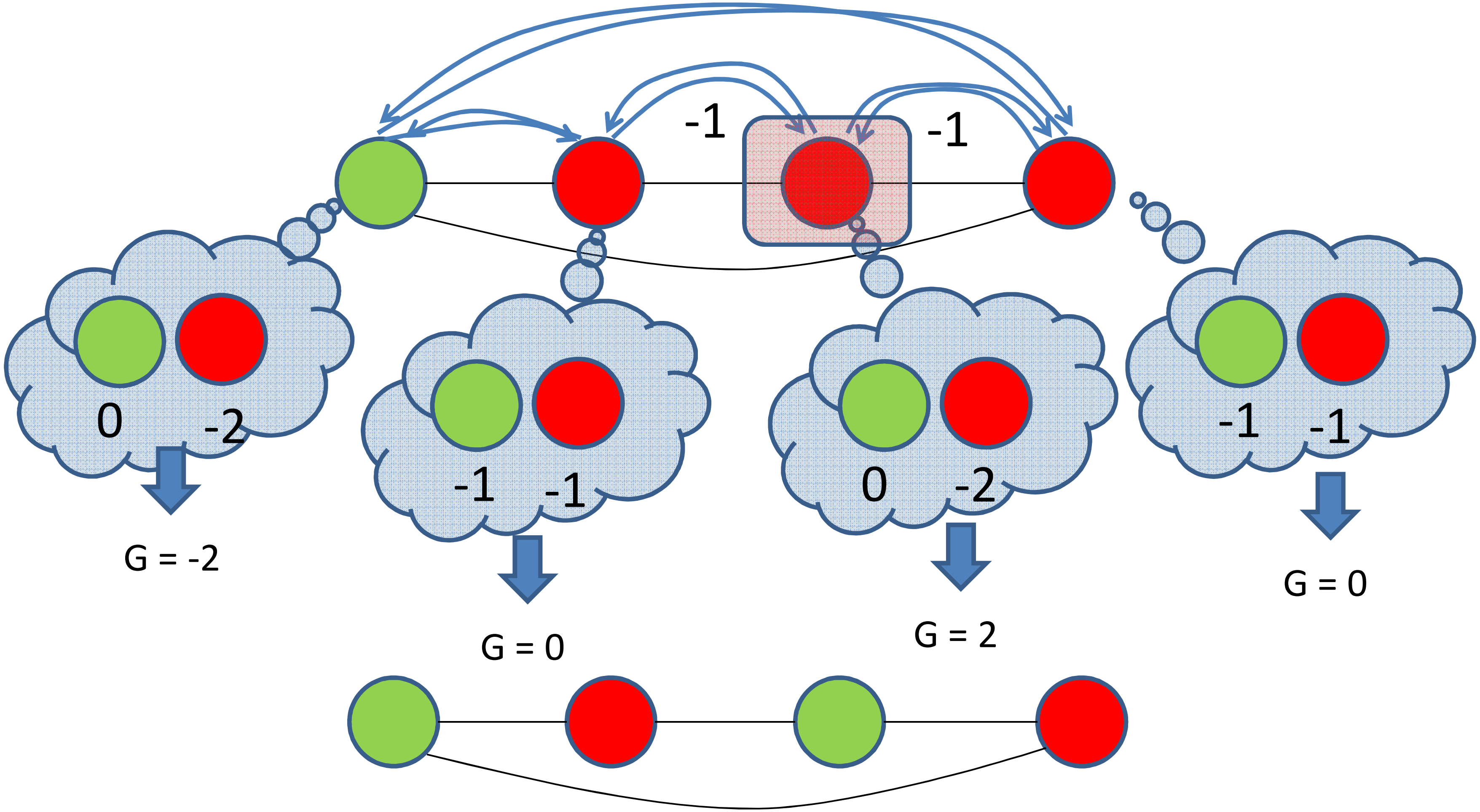
DSA-1: discussion

- Extremely “cheap” (computation/communication)
- Good performance in various domains
  - e.g. target tracking [Fitzpatrick Meertens 03, Zhang et al. 03],
  - Shows an anytime property (not guaranteed)
  - Benchmarking technique for coordination
- Problems
  - Activation probablity must be tuned [Zhang et al. 03]
  - No general rule, hard to characterise results across domains

Maximum Gain Message (MGM-1)

- Coordinate to decide who is going to move
  - Compute and exchange possible gains
  - Agent with maximum (positive) gain executes
- Analysis [Maheswaran et al. 04]
  - Empirically, similar to DSA
  - More communication (but still linear)
  - No Threshold to set
  - Guaranteed to be monotonic (Anytime behavior)

MGM-1: Example

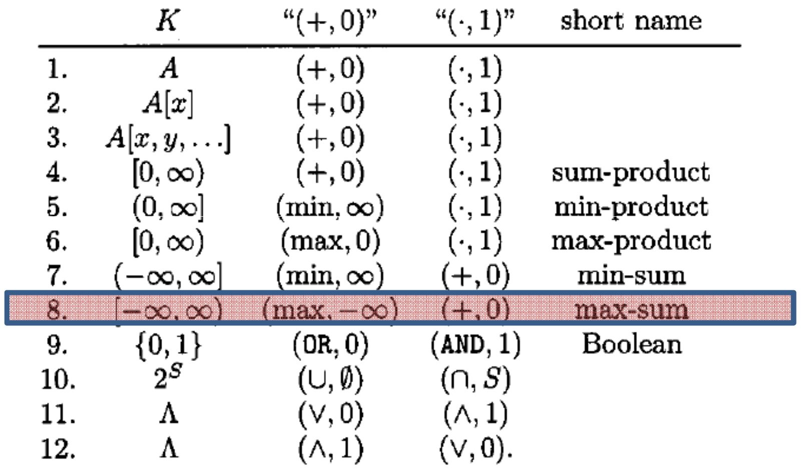


Local greedy approaches

- Exchange local values for variables
  - Similar to search based methods (e.g. ADOPT)
- Consider only local information when maximizing
  - Values of neighbors
- Anytime behaviors
- Could result in very bad solutions

GDL based approaches

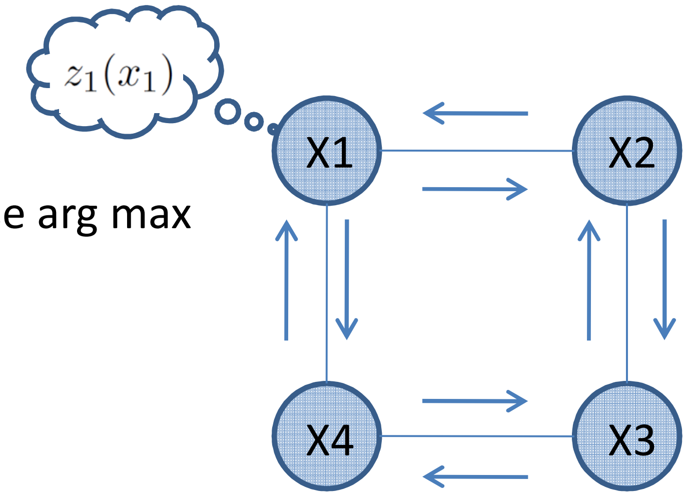
- Generalized Distributive Law
  - Unifying framework for inference in Graphical models
  - Builds on basic mathematical properties of semi-rings
  - Widely used in Info theory, Statistical physics, Probabilistic models
- Max-sum
  - DCOP settings: maximise social welfare



	K	“$(+,0)$”	“$(\cdot,1)$”	short name
1.	A	$(+,0)$	$(\cdot,1)$	
2.	$A[x]$	$(+,0)$	$(\cdot,1)$	
3.	$A[x,y,\ldots]$	$(+,0)$	$(\cdot,1)$	
4.	$[0,\infty)$	$(+,0)$	$(\cdot,1)$	sum-product
5.	$(0,\infty]$	$(\min,\infty)$	$(\cdot,1)$	min-product
6.	$[0,\infty)$	$(\max,0)$	$(\cdot,1)$	max-product
7.	$(-\infty,\infty]$	$(\min,\infty)$	$(+,0)$	min-sum
8.	$[-\infty,\infty)$	$(\max,-\infty)$	$(+,0)$	max-sum
9.	$\{0,1\}$	$(\mathtt{OR},0)$	$(\mathtt{AND},1)$	Boolean
10.	2^S	$(\cup,\emptyset)$	$(\cap,S)$	
11.	Λ	$(\vee,0)$	$(\wedge,1)$	
12.	Λ	$(\wedge,1)$	$(\vee,0)$.	

Max-sum

Agents iteratively computes local functions that depend only on the variable they control



Choose arg max

Shared constraint

$$m_{1\to 2}(x_2) = max_{x_1}(\boxed{F_{12}(x_1, x_2)} + \boxed{m_{4\to 1}(x_1))}$$

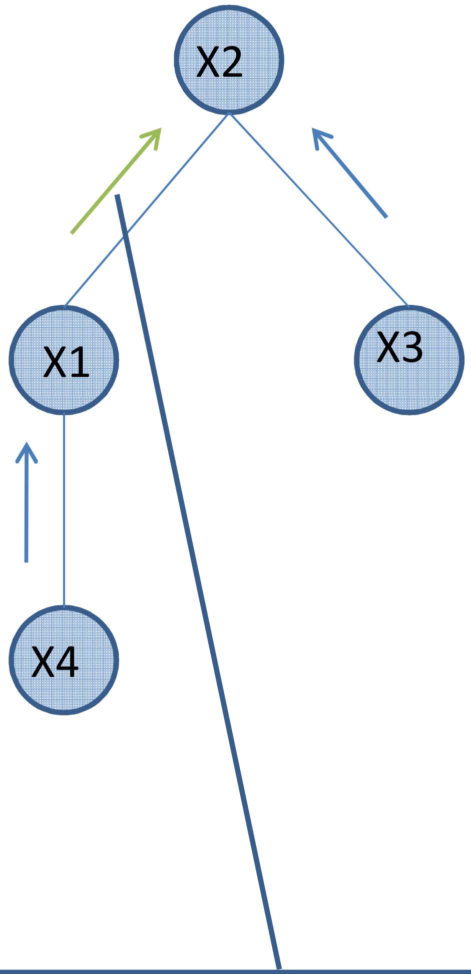
All incoming messages except x2

$$z_1(x_1) = \boxed{m_{4\to 1}(x_1) + m_{2\to 1}(x_1)}$$

All incoming messages

Max-Sum on acyclic graphs

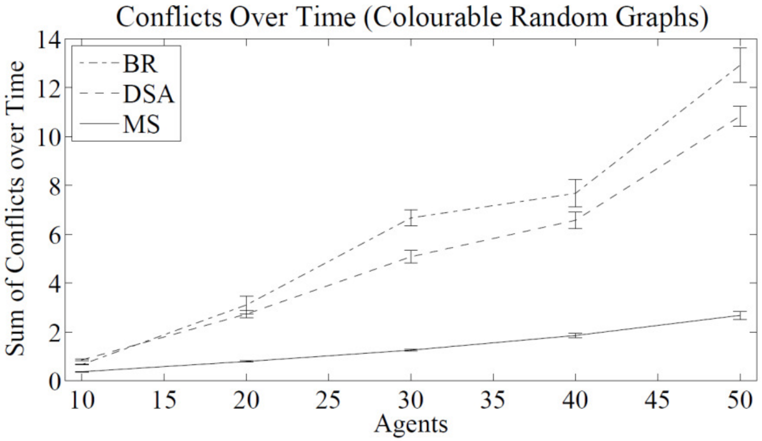
- Max-sum Optimal on acyclic graphs
  - Different branches are independent
  - Each agent can build a correct estimation of its contribution to the global problem (z functions)
- Message equations very similar to Util messages in DPOP
  - Sum messages from children and shared constraint
  - Maximize out agent variable
  - GDL generalizes DPOP [Vinyals et al. 2010a]

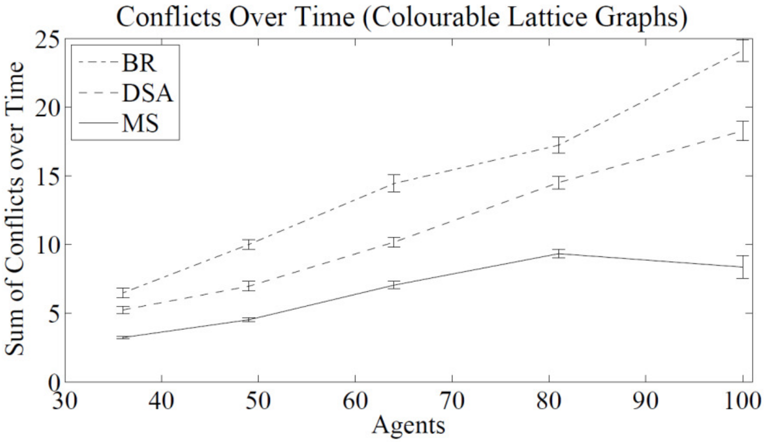


$$m_{1\rightarrow 2}(x_2) = max_{x_1}(F_{12}(x_1, x_2) + m_{4\rightarrow 1}(x_1))$$

Max-sum Performance

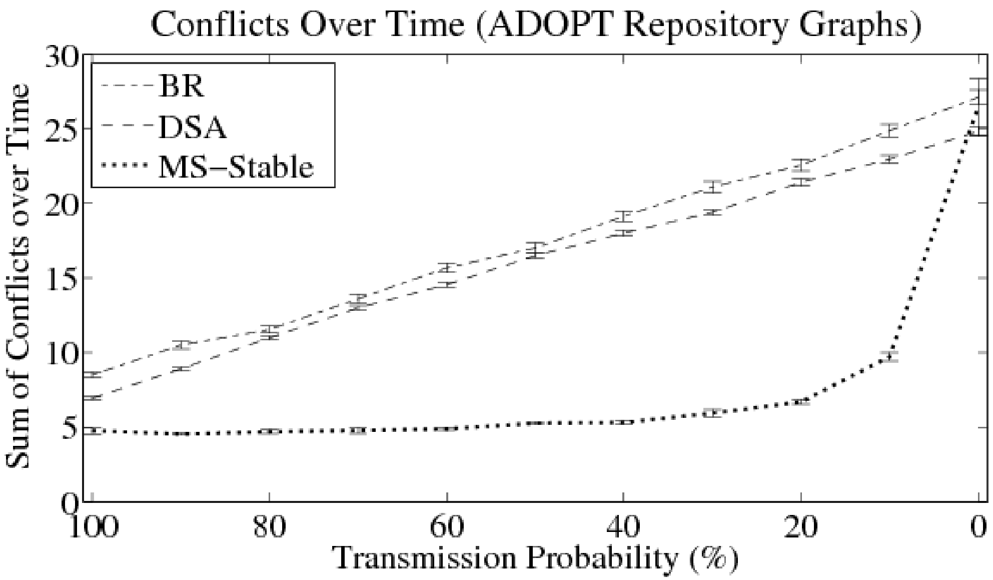
- Good performance on loopy networks [Farinelli et al. 08]
  - When it converges very good results
    - Interesting results when only one cycle [Weiss 00]
  - We could remove cycle but pay an exponential price (see DPOP)





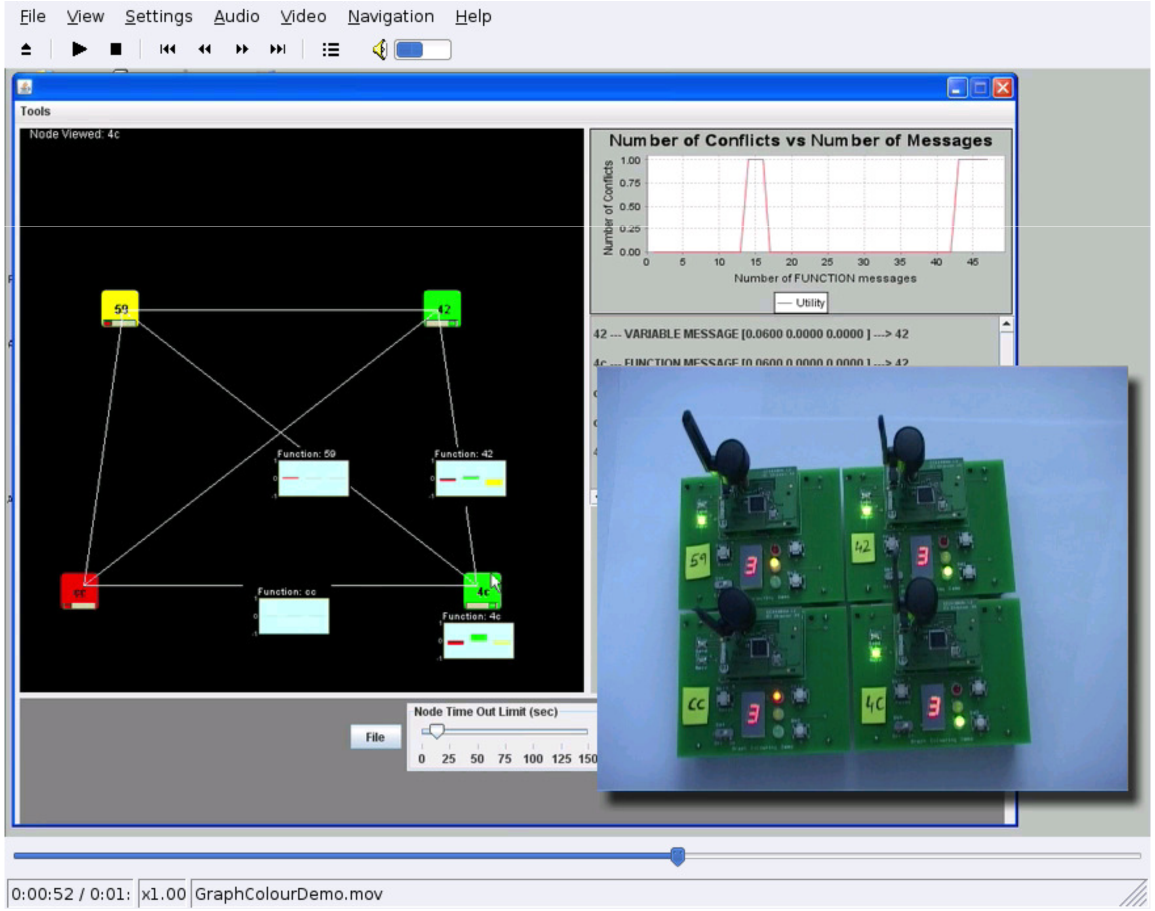
Max-Sum for low power devices

- Low overhead
  - Msgs number/size
- Asynchronous computation
  - Agents take decisions whenever new messages arrive
- Robust to message loss



Max-sum on hardware

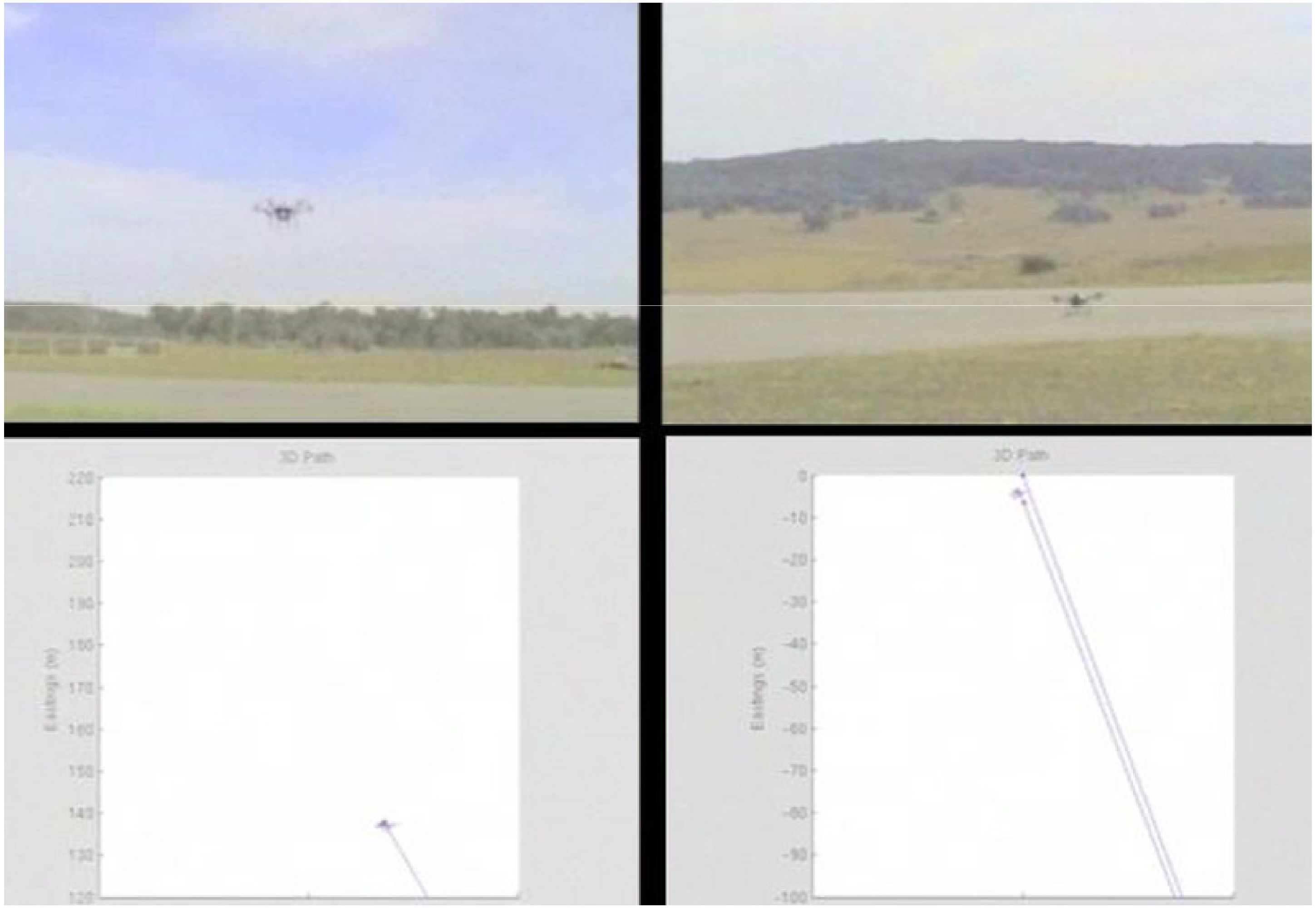
- Chipcon CC2431
  - Low-power devices, 8 KByte RAM



```
42 --- VARIABLE MESSAGE [0.0600 0.0000 0.0000 ] ---> 42
4c --- FUNCTION MESSAGE [0.0600 0.0000 0.0000 ] ---> 42
```

Joint work with:
L. Teacy, N. J. Grabham,
P. Padhy, A. Rogers,
N. R. Jennings

Max-Sum for UAVs

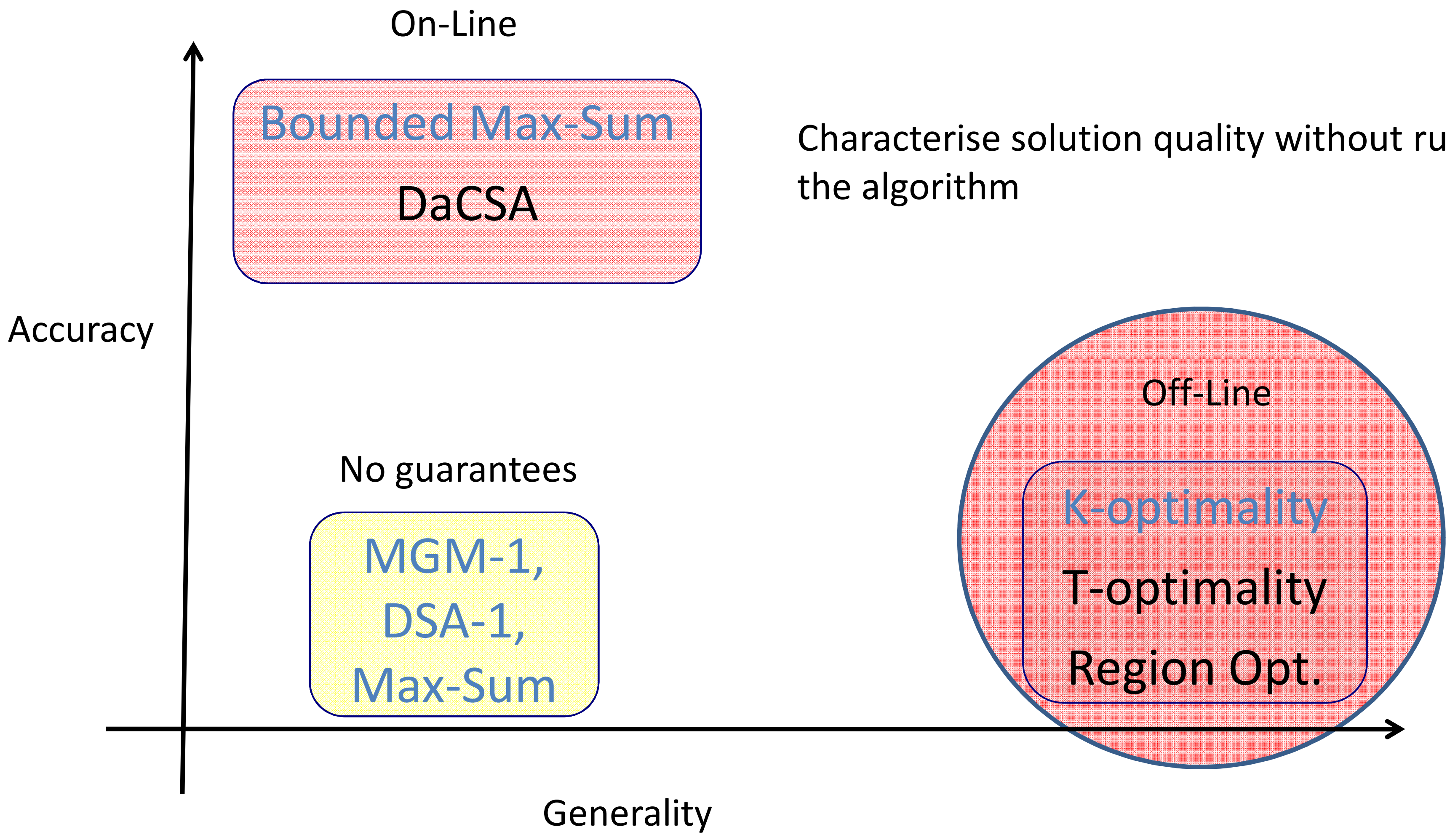


Ack: F. Delle Fave, A. Rogers, N.R. Jennings and ACFR

Quality guarantees for approx. techniques

- Key area of research
- Address trade-off between guarantees and computational effort
- Particularly important for many real world applications
  - Critical (e.g. Search and rescue)
  - Constrained resource (e.g. Embedded devices)
  - Dynamic settings

Off-Line guarantees



On-Line

Bounded Max-Sum

DaCSA

Characterise solution quality without running the algorithm

Accuracy

Off-Line

No guarantees

MGM-1,
DSA-1,
Max-Sum

K-optimality

T-optimality

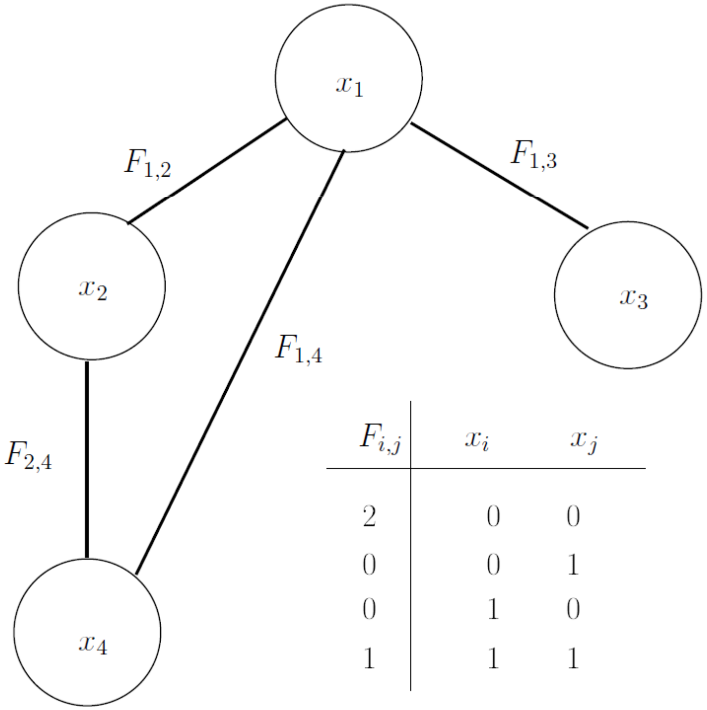
Region Opt.

Generality

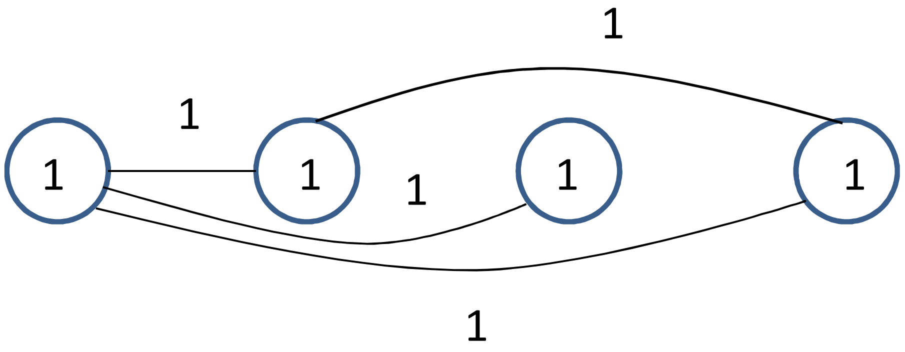
K-Optimality framework

- Given a characterization of solution gives bound on solution quality [Pearce and Tambe 07]
- Characterization of solution: k-optimal
- K-optimal solution:
  - Corresponding value of the objective function can not be improved by changing the assignment of k or less variables.

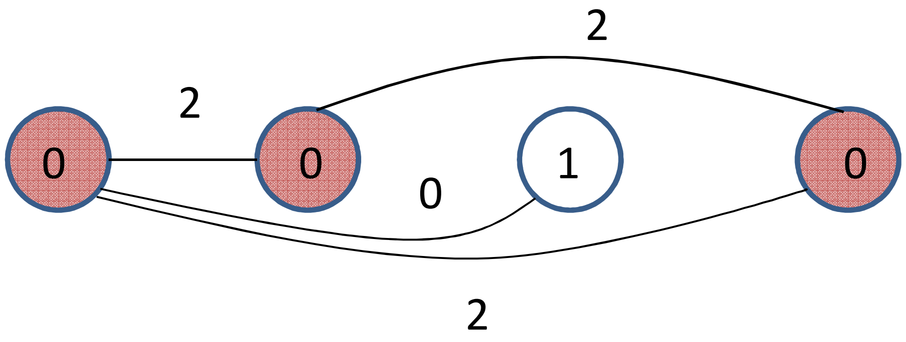
K-Optimal solutions



$F_{i,j}$	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1



2-optimal ? Yes 3-optimal ? No



Bounds for K-Optimality

For any DCOP with non-negative rewards [Pearce and Tambe 07]

Number of agents

Maximum arity of constraints

$$F(\tilde{X}) \geq \underbrace{\frac{\binom{n-m}{k-m}}{\binom{n}{k}-\binom{n-m}{k}}}_{\alpha} F(X^*)$$

K-optimal solution

Binary Network (m=2):

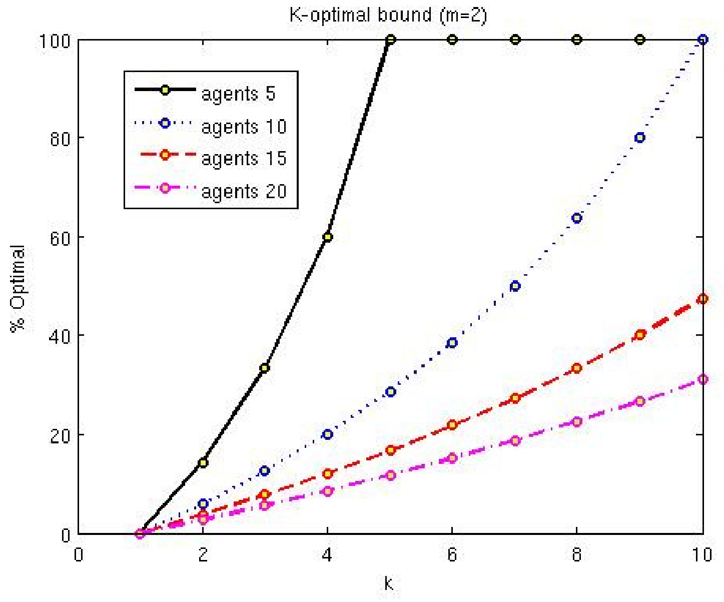
$$F(\tilde{X}) \geq \frac{k-1}{2n-k-1} F(X^*)$$

K-Optimality Discussion

- Need algorithms for computing k-optimal solutions
  - DSA-1, MGM-1 k=1; DSA-2, MGM-2 k=2 [Maheswaran et al. 04]
  - DALO for generic k (and t-optimality) [Kiekintveld et al. 10]
- The higher k the more complex the computation (exponential)

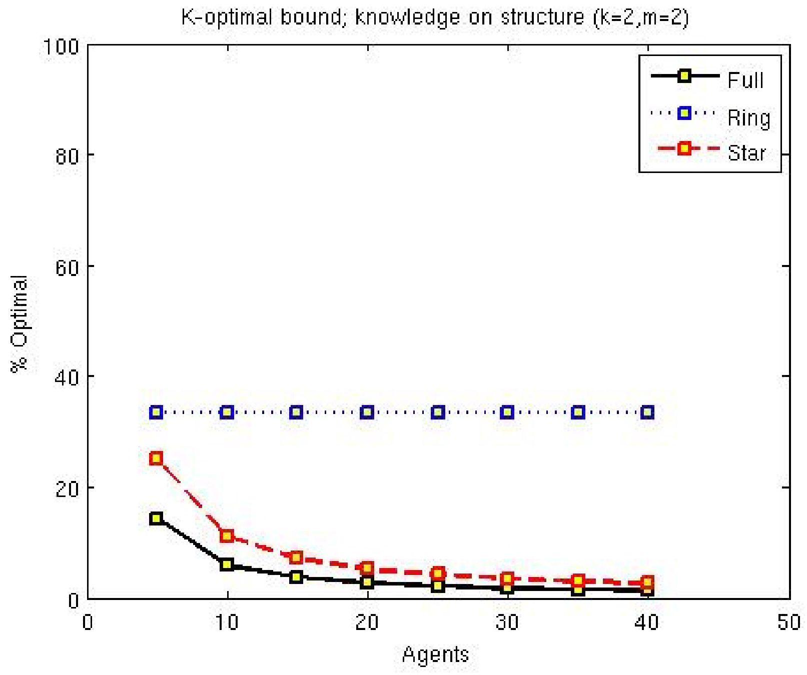
Percentage of Optimal:

- The higher k the better
- The higher the number of agents the worst



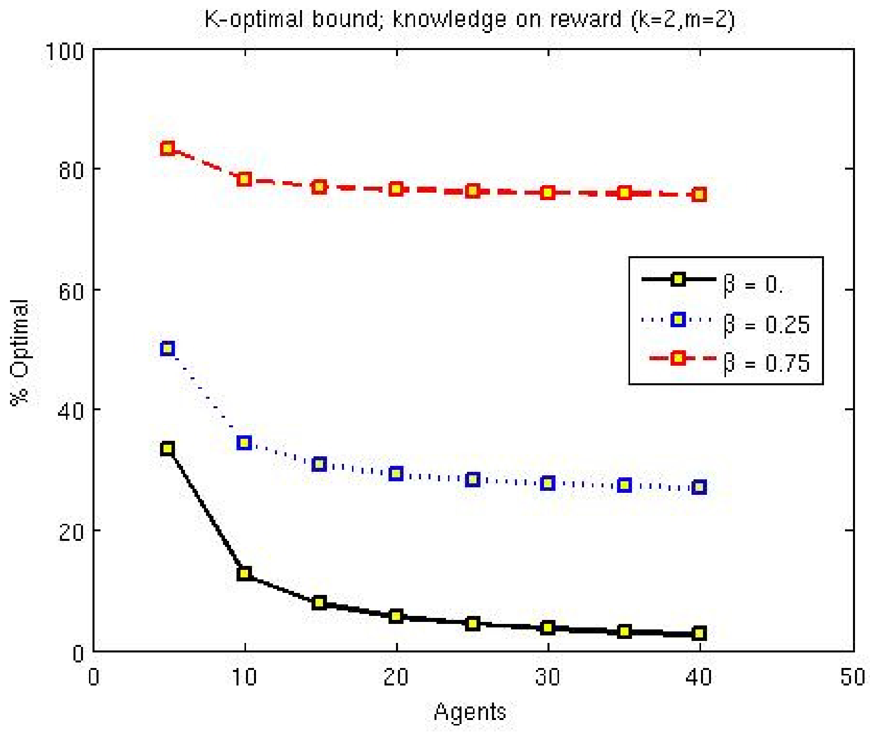
Trade-off between generality and solution quality

- K-optimality based on worst case analysis
- assuming more knowledge gives much better bounds
- Knowledge on structure [Pearce and Tambe 07]



Trade-off between generality and solution quality

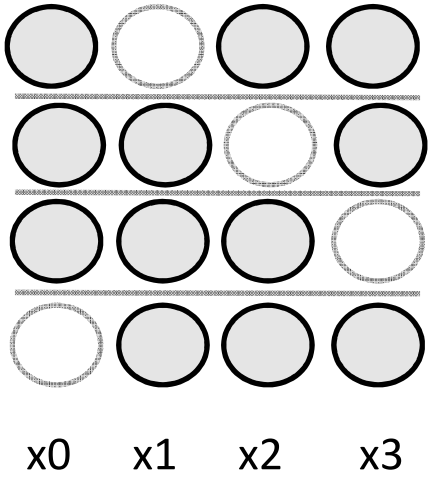
- Knowledge on reward [Bowring et al. 08]
- Beta: ratio of least minimum reward to the maximum



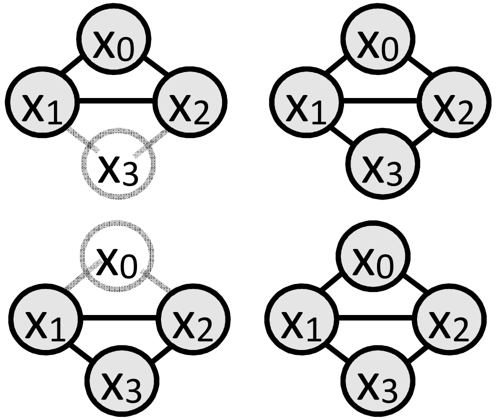
Off-Line Guarantees: Region Optimality

- k-optimality: use size as a criterion for optimality
- t-optimality: use distance to a central agent in the constraint graph
- Region Optimality: define regions based on general criteria (e.g. S-size bounded distance) [Vinyals et al 11]
- Ack: Meritxell Vinyals

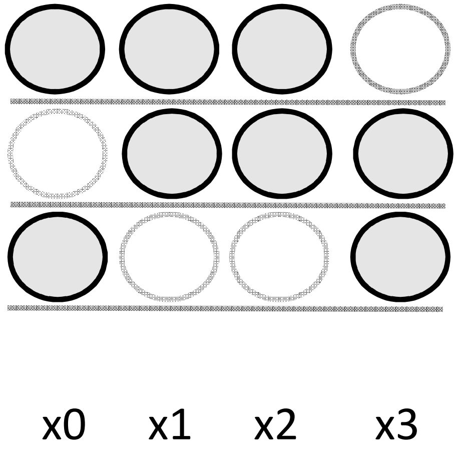
3-size regions



1-distance regions



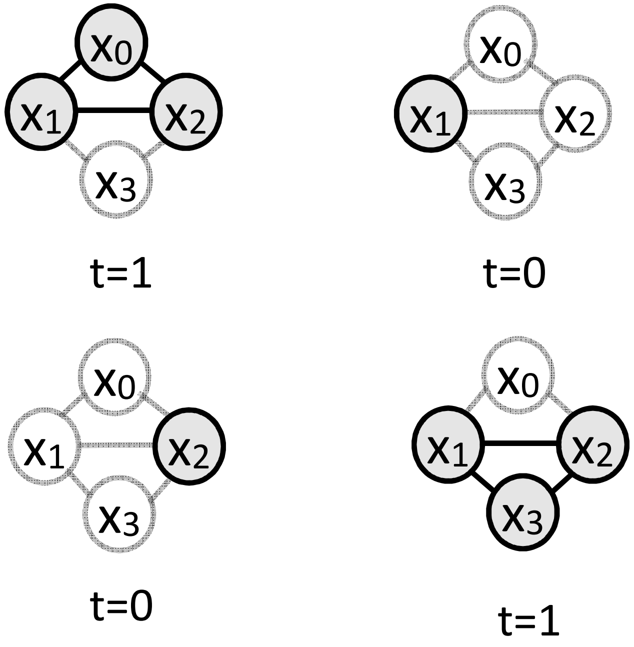
C regions



Size-Bounded Distance

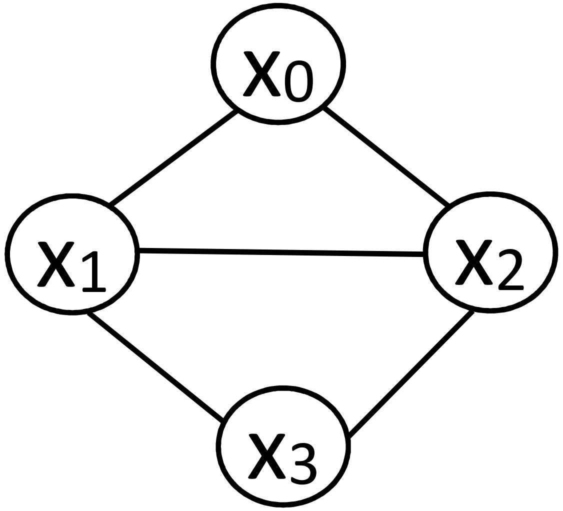
- Region optimality can explore new regions: s-size bounded distance
- One region per agent, largest t-distance group whose size is less than s
- S-Size-bounded distance
  - C-DALO extension of DALO for general regions
  - Can provides better bounds and keep under control size and number of regions

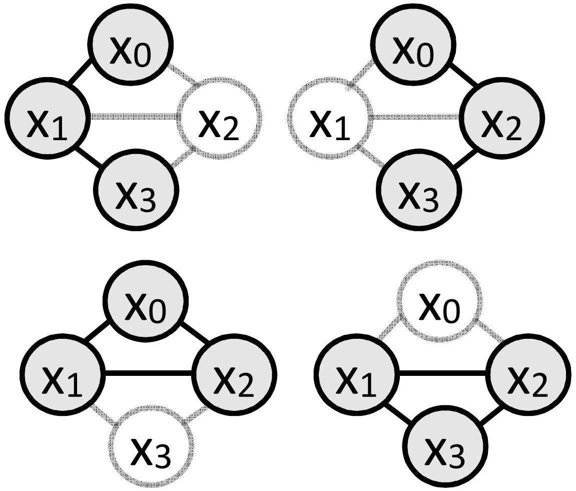
3-size bounded distance



Max-Sum and Region Optimality

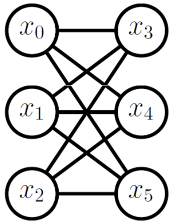
- Can use region optimality to provide bounds for Max-sum [Vinyals et al 10b]
- Upon convergence Max-Sum is optimal on SLT regions of the graph [Weiss 00]
- Single Loops and Trees (SLT): all groups of agents whose vertex induced subgraph contains at most one cycle



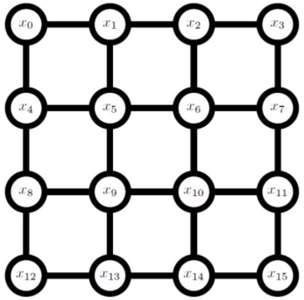


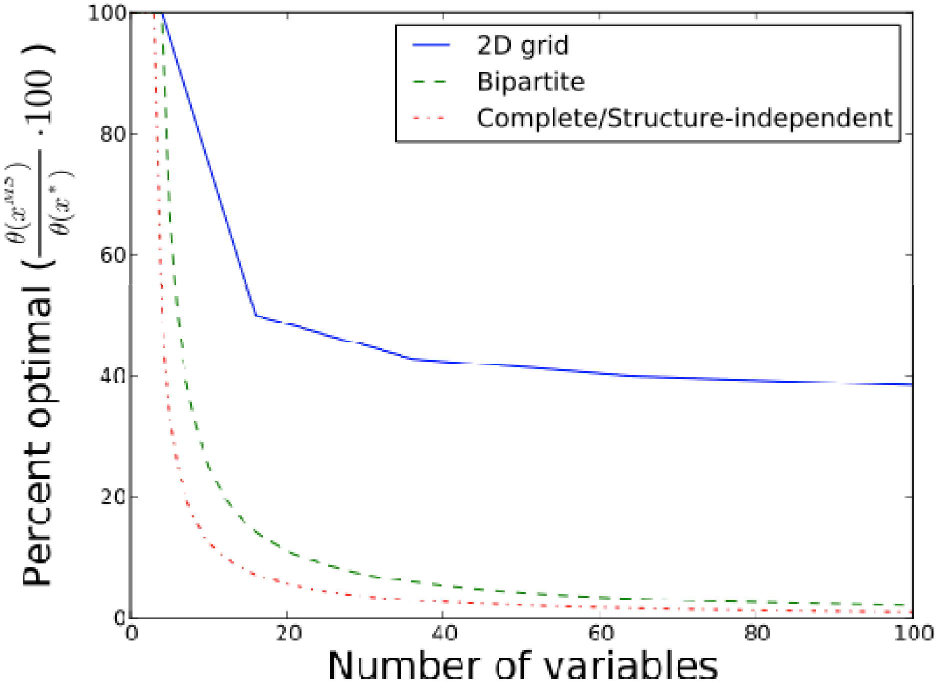
Bounds for Max-Sum

- Complete: same as 3-size optimality
- bipartite



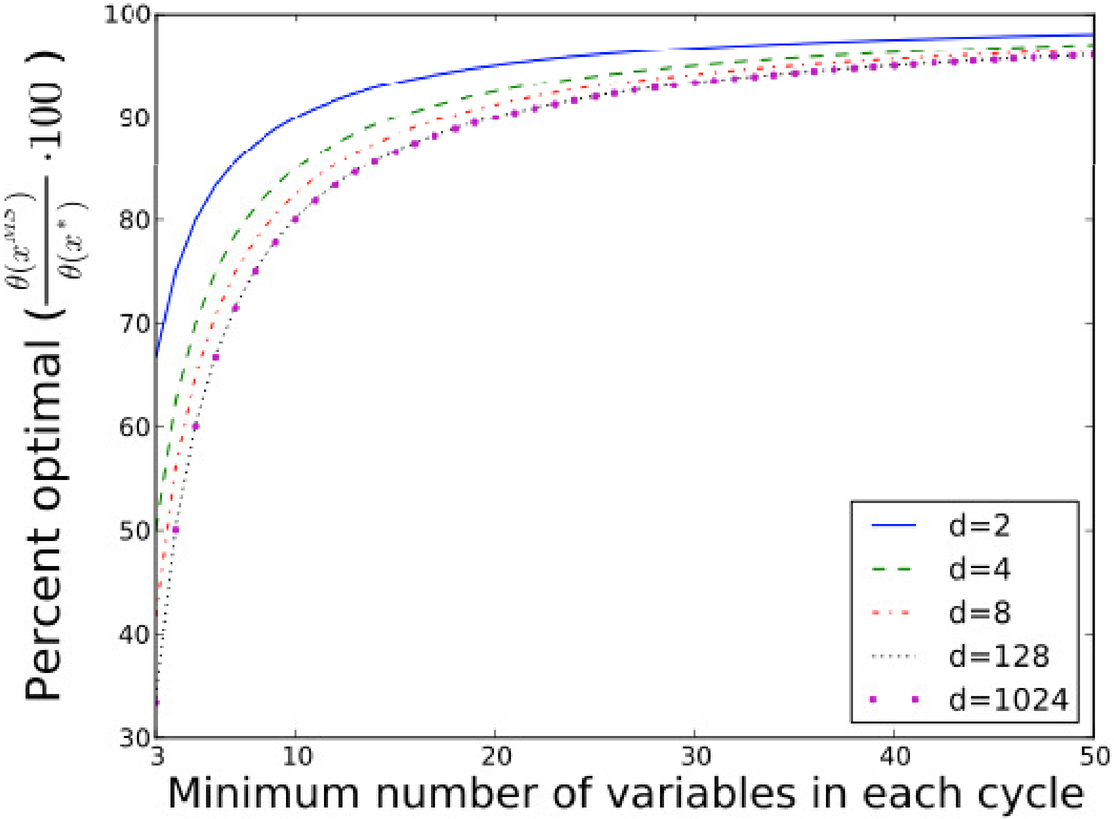
- 2D grids

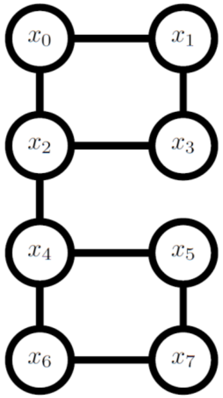




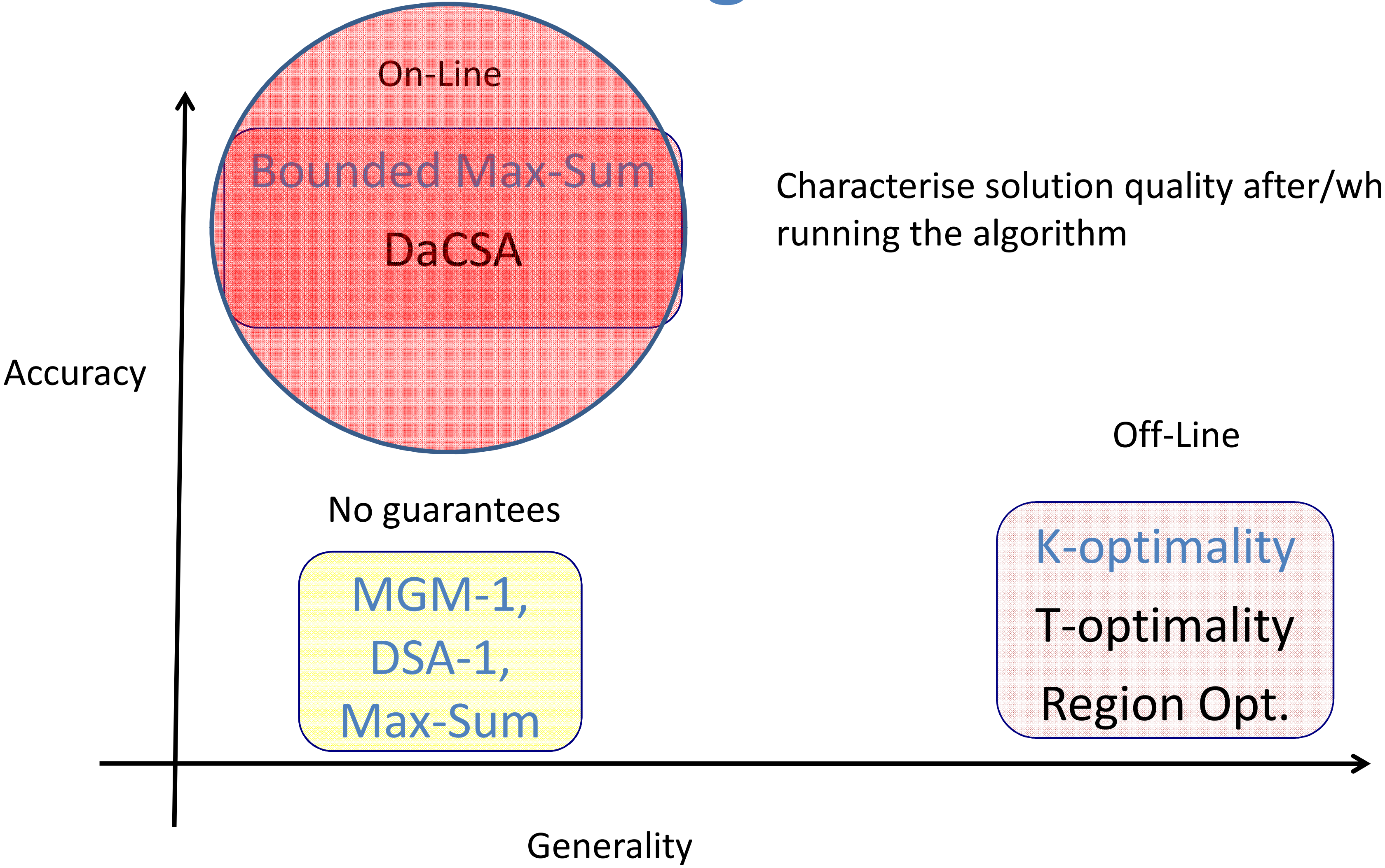
Variable Disjoint Cycles

Very high quality guarantees if smallest cycle is large





On-Line guarantees

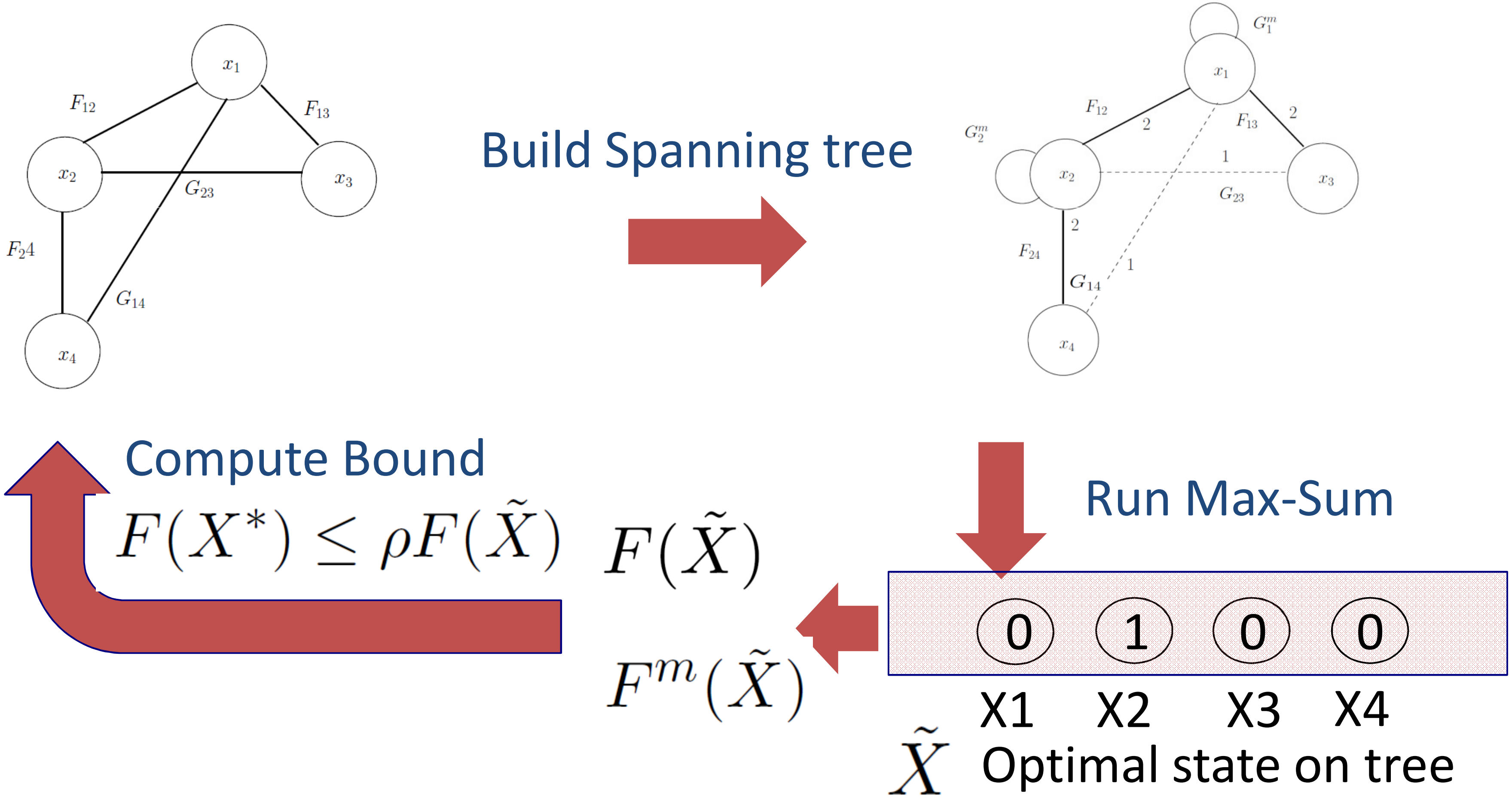


Characterise solution quality after/while running the algorithm

Bounded max-sum

Aim: Remove cycles from constraint graph avoiding exponential computation/communication [Rogers et al. 11]

Avoid pseudo trees/junction trees



Build Spanning tree

Run Max-Sum

0 1 0 0

X1 X2 X3 X4

$\tilde{X}$ Optimal state on tree

$F(\tilde{X})$

$F^m(\tilde{X})$

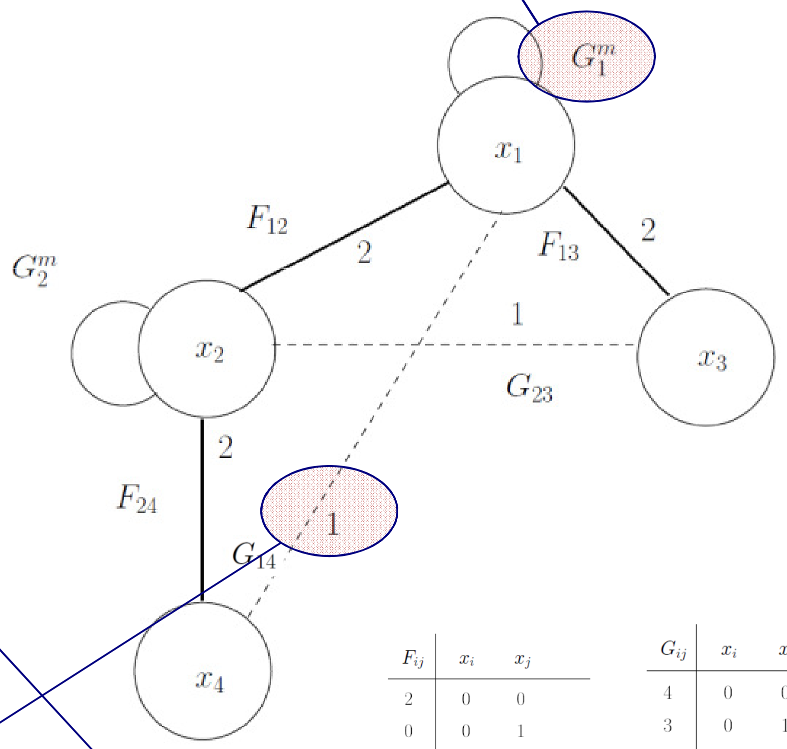
Compute Bound

$F(X^*) \leq \rho F(\tilde{X})$

Computing the weights

- Compute a weight for each edge
  - maximum possible impact of the constraint on the solution
  - Maximum Spanning Tree
  - Remove edges and modify constraints
  - $\rho F(\tilde{X}) = F^m(\tilde{X}) + W$

$$G_1^m = \min_{x_4} G_{14}$$



F_{ij}	x_i	x_j
2	0	0
0	0	1
0	1	0
1	1	1

G_{ij}	x_i	x_j
4	0	0
3	0	1
1	1	0
2	1	1

$$w'_{14} = max_{x_4}\,[max_{x_1} G_{14} - min_{x_1} G_{14}] = 3$$

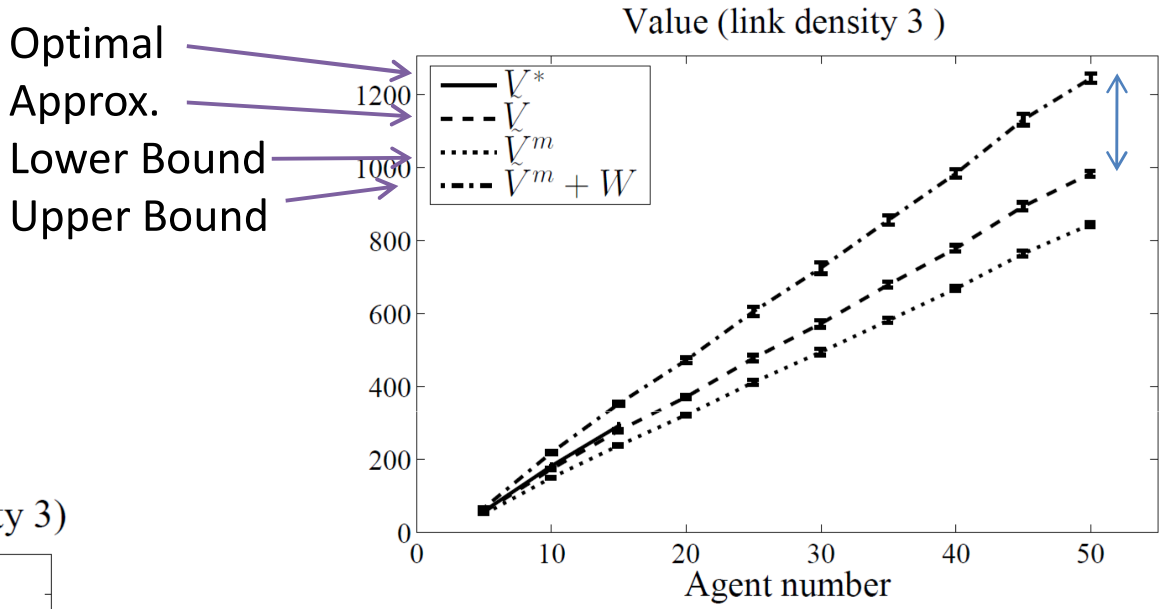
$$w''_{14} = max_{x_1}\,[max_{x_4} G_{14} - min_{x_4} G_{14}] = 1$$

$$w_{ij} = \min\{w'_{ij}, w''_{ij}\}$$

W = sum of weights of removed edges

Results (Random Binary Network)

- Bound is significant
  - Approx. ratio is typically 1.23 (81 %)

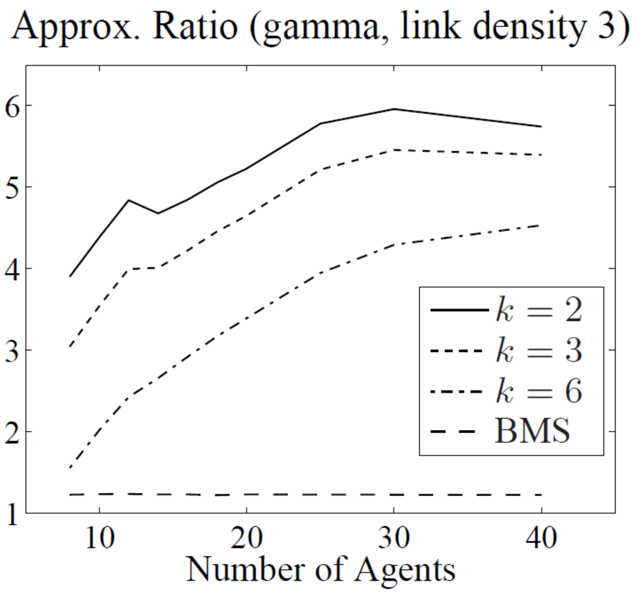


Optimal

Approx.

Lower Bound

Upper Bound



Comparison with k-optimal with knowledge on reward structure

Much more accurate less general

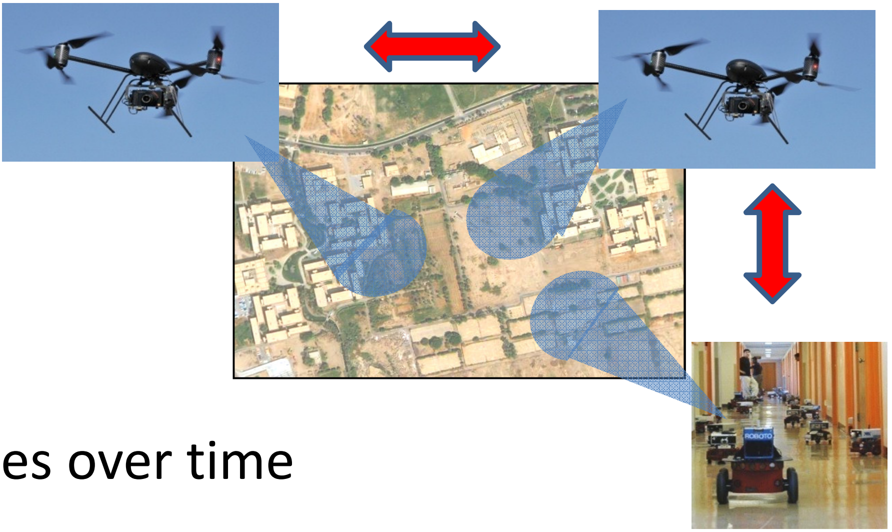
Discussion

- Discussion with other data-dependent techniques
  - BnB-ADOPT [Yeoh et al 09]
    - Fix an error bound and execute until the error bound is met
    - Worst case computation remains exponential
  - ADPOP [Petcu and Faltings 05b]
    - Can fix message size (and thus computation) or error bound and leave the other parameter free
- Divide and coordinate [Vinyals et al 10]
  - Divide problems among agents and negotiate agreement by exchanging utility
  - Provides anytime quality guarantees

Future Challenges for DCOP

- Handle Uncertainty
  - E[DPOP] [Leaute et al. 09]
  - Distributed Coordination of Exploration and Exploitation [Taylor et al. 11]
- Handle Dynamism
  - S-DPOP [Petcu and Faltings 05]
  - Robustness in DCR [Lass et al. 09]
  - Fast-Max-Sum [MacArhur et al. 09]

Challenging domains: Robotics



- Cooperative exploration, Surveillance, Patrolling, etc.
- Main Challenges
  - Reward is unknown/uncertain
  - Structure of the problem changes over time
  - High probability of failures
    - Messages, robots, etc.
- Work in this direction
  - [Stranders et al. 09]
  - [Taylor et al. 11]

Challenging domains: Energy

- Intelligent Building control, power distribution configuration, electric vehicle management, etc.
- Main challenges
  – Large scale, dynamic system
  – Decomposing agents’ interactions
  – Interaction with end-users
- Work in this direction
  – [Kumar et al. 09]
  – [Kambooj et al. 11]

References I

- [Kumar et al. 09] Distributed Constraint Optimization with Structured Resource Constraints, AAMAS 09
- [Kamboj at al. 09] Deploying Power Grid-Integrated Electric Vehicles as a Multi-Agent System, AAMAS 11
- [Leaute et al. 09] E[DPOP]: Distributed Constraint Optimization under Stochastic Uncertainty using Collaborative Sampling, DCR 09
- [Taylor et al. 11] Distributed On-line Multi-Agent Optimization Under Uncertainty: Balancing Exploration and Exploitation, Advances in Complex Systems
- [Stranders at al 09] Decentralised Coordination of Mobile Sensors Using the Max-Sum Algorithm, AAAI 09
- [Petcu and Faltings 05] S-DPOP: Superstabilizing, Fault-containing Multiagent Combinatorial Optimization, AAAI 05
- [Lass et al. 09] Robust Distributed Constraint Reasoning, DCR 09
- [Vinyals et al. 10] Divide and Coordinate: solving DCOPs by agreement. AAMAS 10
- [Petcu and Faltings 05b] A-DPOP: Approximations in Distributed Optimization, CP 2005
- [Yeoh et al. 09] Trading off solution quality for faster computation in DCOP search algorithms, IJCAI 09
- [Vinyals et al 10a] Constructing a unifying theory of dynamic programming DCOP algorithms via the Generalized Distributive Law, JAAMAS 2010.
- [Rogers et al. 11] Bounded approximate decentralised coordination via the max-sum algorithm, Artificial Intelligence 2011.

References II

- [Yeoh et al. 09] Trading off solution quality for faster computation in DCOP search algorithms, IJCAI 09
- [Vinyals et al 10b] Worst-case bounds on the quality of max-product fixed-points, NIPS 10
- [Vinyals et al 11] Quality guarantees for region optimal algorithms, AAMAS 11
- [Maheswaran et al. 04] Distributed Algorithms for DCOP: A Graphical Game-Based Approach, PDCS-2004
- [Kiekintveld et al. 10] Asynchronous Algorithms for Approximate Distributed Constraint Optimization with Quality Bounds, AAMAS 10
- [Pearce and Tambe 07] Quality Guarantees on k-Optimal Solutions for Distributed Constraint Optimization Problems, IJCAI 07
- [Bowring et al. 08] On K-Optimal Distributed Constraint Optimization Algorithms: New Bounds and Algorithms, AAMAS 08
- [Farinelli et al. 08] Decentralised coordination of low-power embedded devices using the max-sum algorithm, AAMAS 08
- [Weiss 00] Correctness of local probability propagation in graphical models with loops, Neural Computation
- [Fitzpatrick and Meertens 03] Distributed Coordination through Anarchic Optimization, Distributed Sensor Networks: a multiagent perspective.
- [Zhang et al. 03] A Comparative Study of Distributed Constraint algorithms, Distributed Sensor Networks: a multiagent perspective.
- [MacArthur et al 09] Efficient, Superstabilizing Decentralised Optimisation for Dynamic Task Allocation Environments, OptMAS workshop 2009.

Market-based Resource Allocation

Jesús Cerquides
Juan A. Rodríguez-Aguilar
Artificial Intelligence Research Institute, IIIA
Spanish Council for Scientific Research, CSIC

At the end of this block you will be able to

1. Classify auctions along several dimensions.
2. Determine the winner in complex auctions:
   - And how complex this can get to be.
3. Apply auctions to solve supply chain formation problems
4. Apply auctions to robustly solve task allocation problems

Introduction

Historical note

- Auctions used in Babylon as early as 500 B.C.
- 193 A.D. After having killed Emperor Pertinax, Prætorian Guard sold the Roman Empire by means of an auction

What is an auction?

An auction is a protocol that allows agents to indicate their interests in one or more resources and that uses these indications of interest to determine both an allocation of resources and a set of payments by the agents.

[Shoham & Leyton-Brown 2009]

Where are auctions used nowadays?

- Resource allocation
  - Treasury auctions
  - Right to drill oil, off-shore oil lease
  - Use the EM spectrum
  - Private and public goods and services acquisition
  - Internet auctions
- Market-based computing

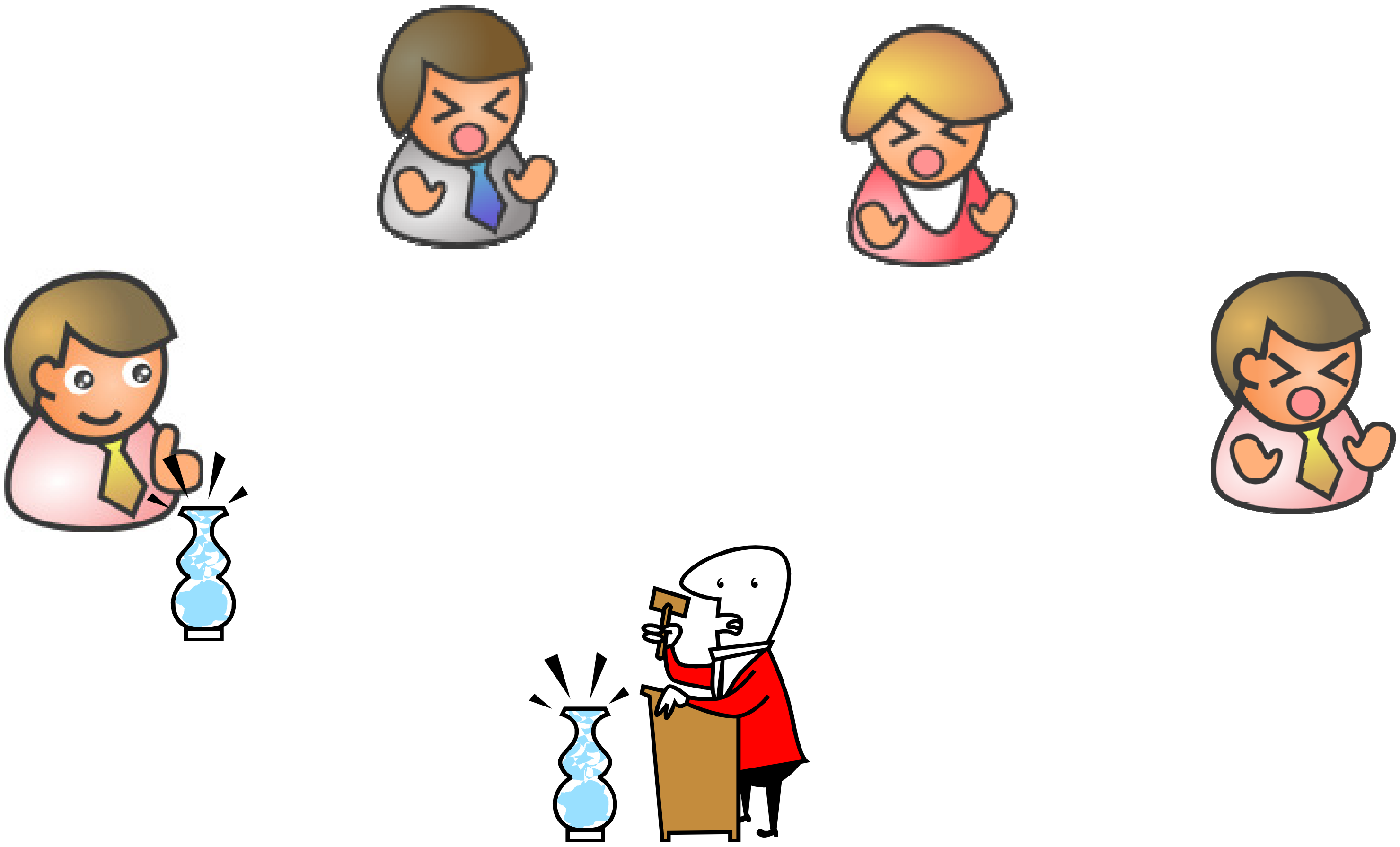
  The use of a market-based method, such as an auction to compute the outcome of a distributed problem.

  - Air-conditioning control
  - Production control
  - Robot navigation
  - Sensor networks

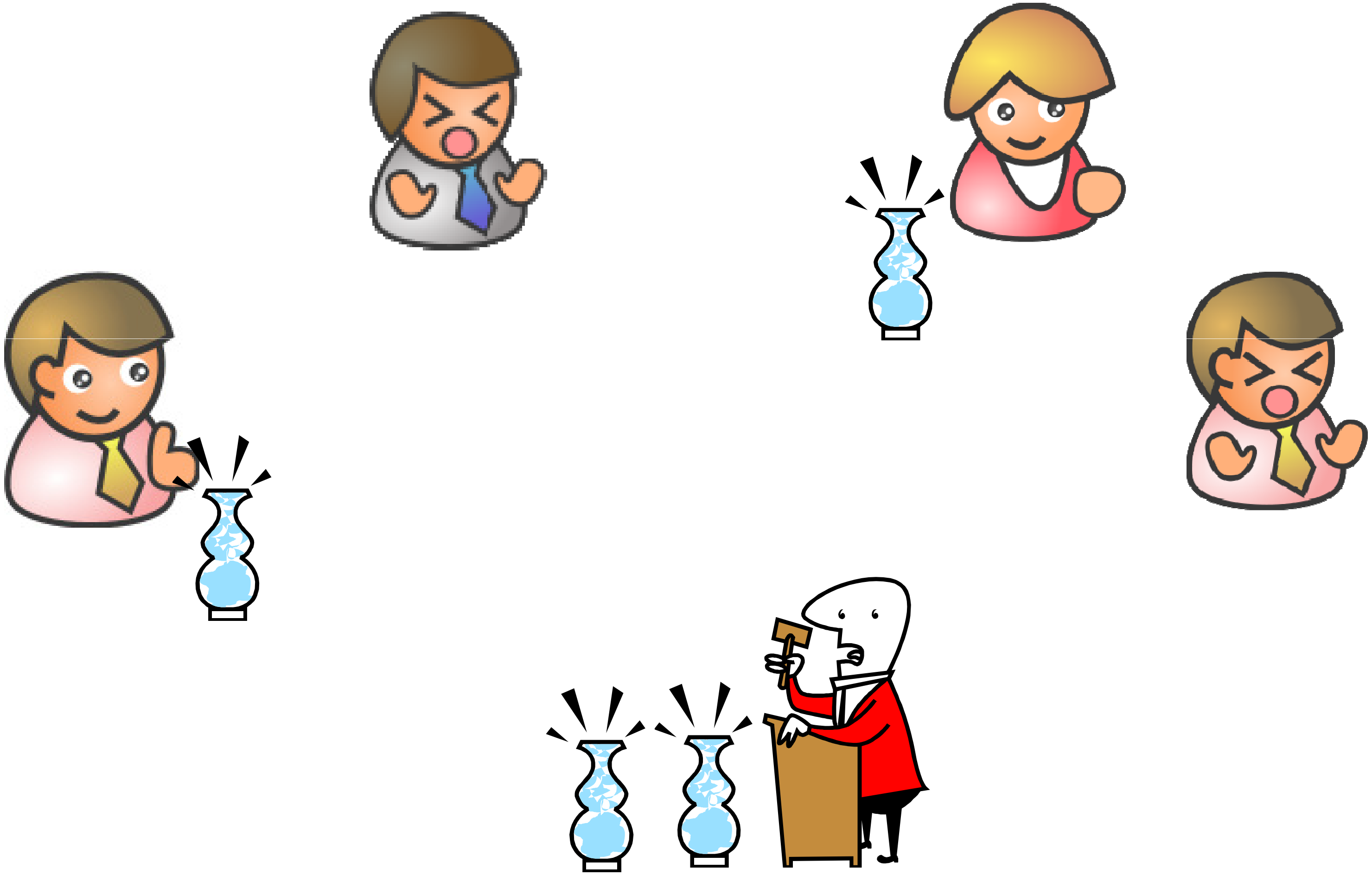
Why auctions?

- Market-based price setting → For objects of unknown value, the value is dynamically assessed by the market!
- Flexible → Any object type
- Dynamic → Interactive
- Automated
  - use of simple rules reduces complexity of negotiations
  - ideal for computer implementation
- *Revenue*-maximising and *efficient* allocations are achievable

Single good auctions



Multi-unit auctions

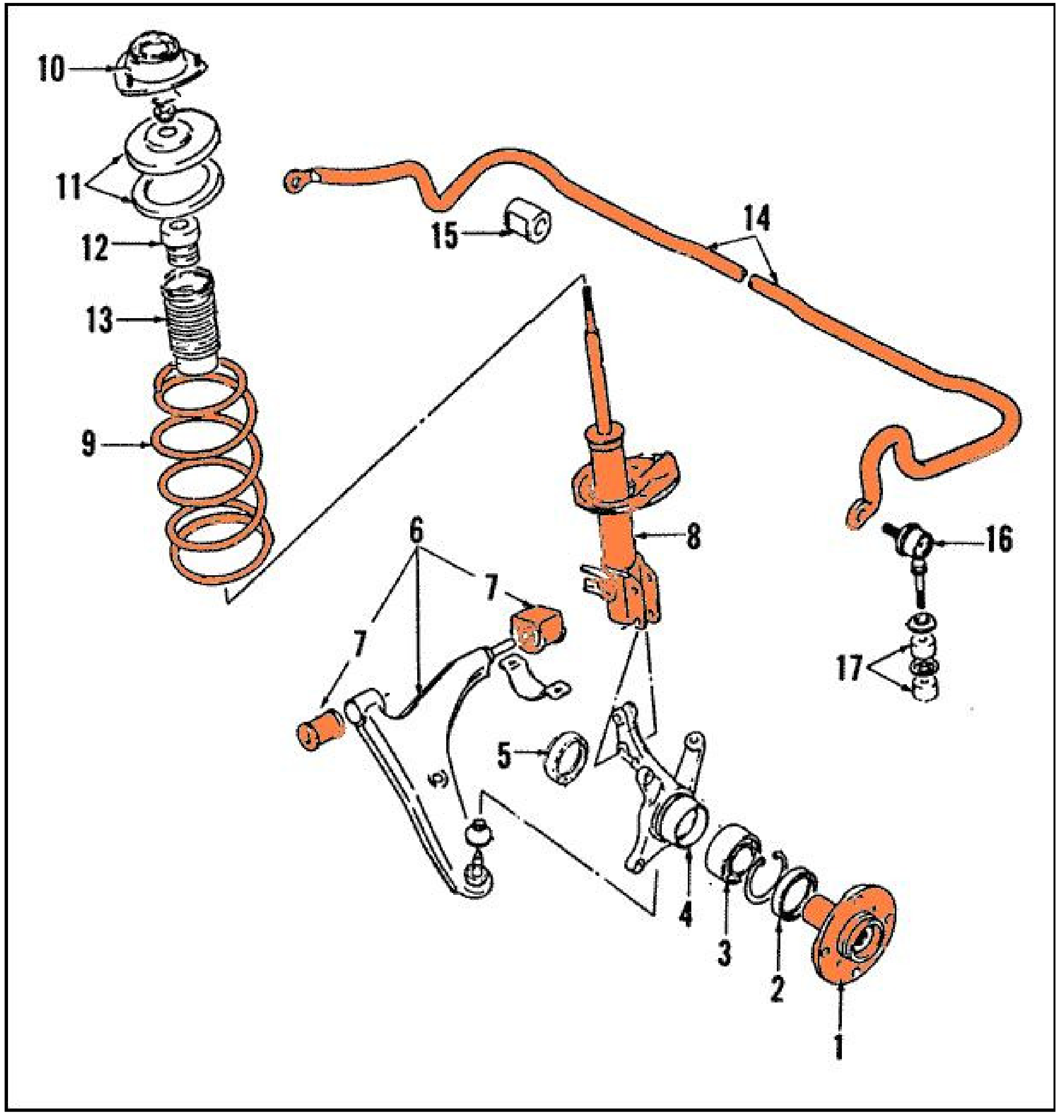


Multi-item auctions

Multi-item auctions (II)

Reverse auctions

FRONT SUSPENSION, FRONT WHEEL BEARING ACQUISITION



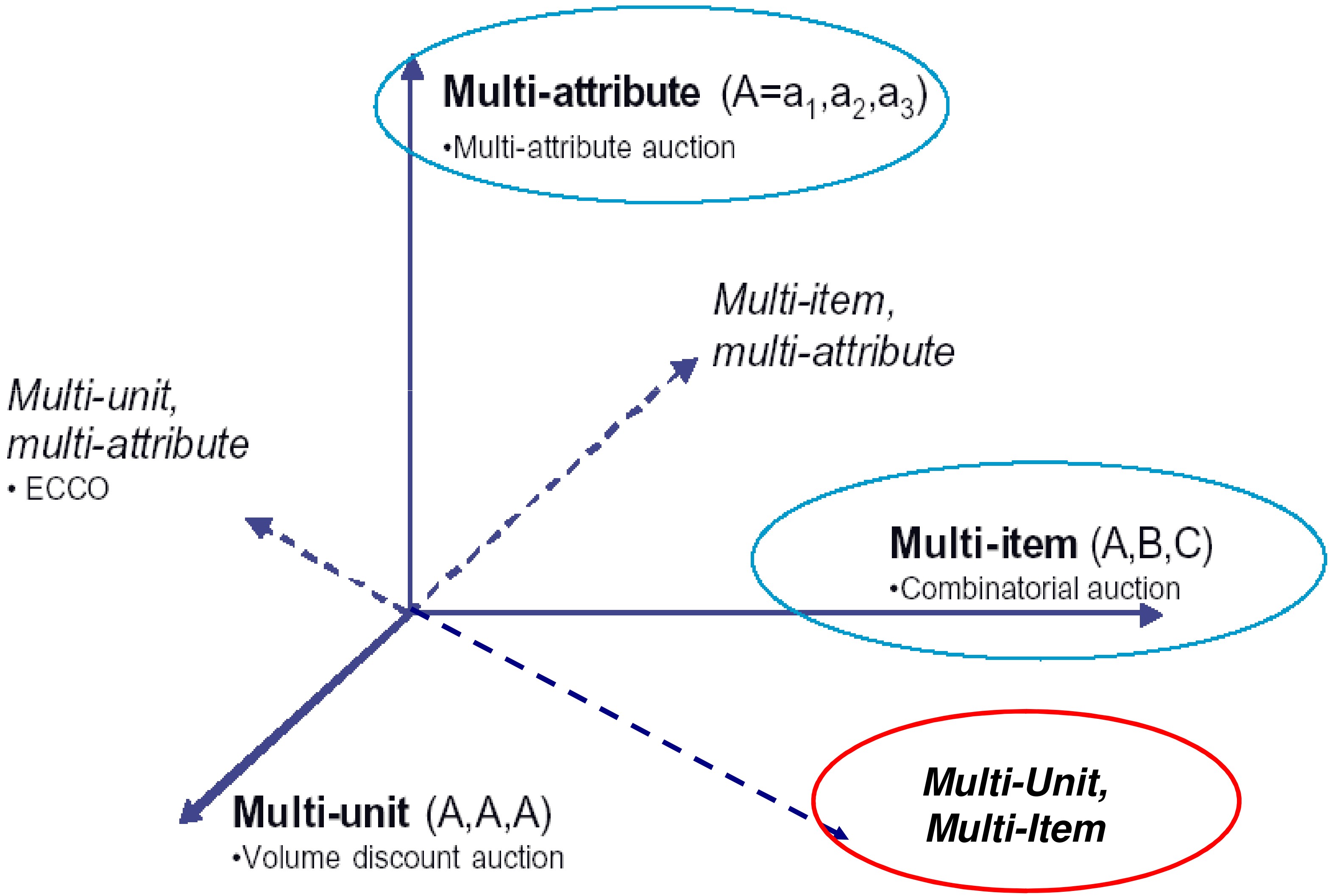
- Goal: Buy parts to produce a front suspension.
- The buyer issues a request for bids to his providers.

PART #	DESCRIPTION
1	FRONT HUB
7	LOWER CONTROL ARM BUSHINGS
8	STRUT
9	COIL SPRING
14	STABILIZER BAR

Multi-attribute auctions

- They allow negotiation over further attributes beyond price (e.g. color, weight, or delivery time).
- For instance:
  - Provider John Doe offers to deliver a **stainless-steel stabilizer bar** that **weighs 500 g** at the cost of **200 EUR** by **July 18th 2011**.
- *They promise higher market efficiency through a more effective information exchange of buyer's preferences and supplier's offerings.*

Multidimensional Auctions



Multi-attribute (A=a_1,a_2,a_3)

- Multi-attribute auction

Multi-item, multi-attribute

Multi-unit, multi-attribute

- ECCO

Multi-item (A,B,C)

- Combinatorial auction

Multi-unit (A,A,A)

- Volume discount auction

Multi-Unit, Multi-Item

Outline

- Single-item auctions
- Multi-item auctions
- Supply chain formation
- Robust auctions
- Research opportunities

Outline

- Single-item auctions
- Multi-item auctions
- Supply chain formation
- Robust auctions
- Research opportunities

Single-unit auction protocols

- English
- Japanese
- Dutch
- Sealed-bid
  - First-price
  - Second-price (Vickrey)

Auction rules

- *Bidding rules.*
  - How offers are made:
    - *By whom*
    - *When*
    - *What their content can be*
- *Clearing rules.*
  - Who gets which goods (allocation) and what money changes hands (payment).
- *Information rules.*
  - Who knows what about the state of the negotiation and when.

Auction rules

- *Bidding rules*.
  - How offers are made:
    - *By whom*
    - *When*
    - *What their content can be*
- *Clearing rules*.
  - Who gets which goods (allocation) and what money changes hands (payment).
- *Information rules.*
  - Who knows what about the state of the negotiation and when.

Winner Determination Problem (WDP)

- Given a set of bids, allocate the good to the bidder whose bid maximizes the auctioneer's revenue.
- WDPs for single-unit auctions are easy:
  - English: Last bid wins
  - Japanese: Last remaining bidder wins
  - Dutch: First bid wins
  - Sealed-bid: Highest bid wins

Multi-unit auctions

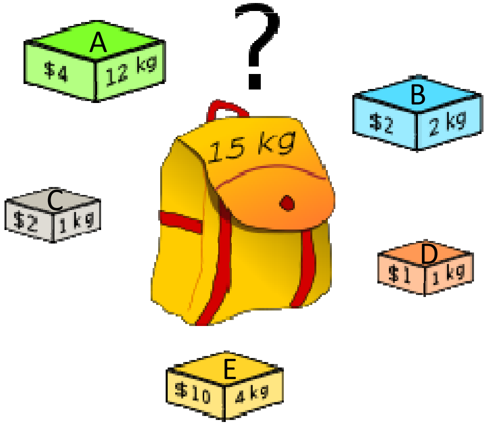
- An auctioneer wants to sell 15 apples maximizing the revenue.
- Sequential auction / Parallel auction
  - Bidding extremely difficult
  - Bidders can end up with less units than needed
- As a whole.
  - There are multi-unit formats of the single-unit protocols:
    - English, Japanese, Dutch, Sealed-bid
  - Bidders bid on quantity and price

Multi-unit auctions

- An auctioneer wants to sell 15 apples maximizing the revenue.
- The auctioneer receives the following offers:
  - A: Buy 12 apples for 4€
  - B: Buy 2 apples for 2€
  - C: Buy 1 apple for 2€
  - D: Buy 1 apple for 1€
  - E: Buy 4 apples for 10€
- What is the optimal allocation?

Multi-unit auctions

- An auctioneer wants to sell 15 apples maximizing the revenue.
- The auctioneer receives the following offers:
  - A: Buy 12 apples for 4€
  - B: Buy 2 apples for 2€
  - C: Buy 1 apple for 2€
  - D: Buy 1 apple for 1€
  - E: Buy 4 apples for 10€



Weighted knapsack problem (example)

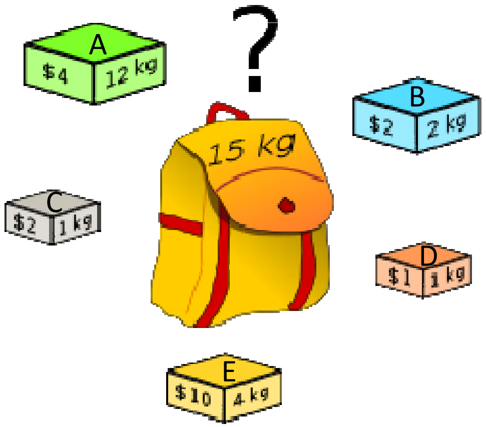
- A binary variable for each bid: x_A, x_B, x_C, x_D, x_E
- Maximize the revenue obtained by filling the knapsack

$$\text{maximize } 4x_A + 2x_B + 2x_C + x_D + 10x_E$$

$$\text{subject to}$$

$$12x_A + 2x_B + x_C + x_D + 4x_E \le 15$$

$$x_A, x_B, x_C, x_D, x_E \in \{0,1\}$$



Multi-unit auctions allow more complex bids

- An auctioneer wants to sell 15 apples maximizing the revenue.
- The auctioneer receives the following offers:
  - A: Buy either 12 apples for 4€ OR 6 apples for 3€
  - B: Buy 2 apples for 2€
  - C: Buy 1 apple for 2€
  - D: Buy 1 apple for 1€
  - E: Buy either 4 apples for 10€ OR 2 apples for 6€
- What is the optimal allocation?

Bidding languages

- In general, in a multi-unit auction, bidders need to specify their valuation for every number of units.
- If we are selling three apples, each bidder should tell us something like

🍎	4€
🍎 🍎	6€
🍎🍎🍎	7€

- Imagine we are selling 100 apples…

$$v_i : \{1,\ldots,n\} \to \mathbb{R}^+$$

- Bidding languages allow bidders to convey this information more compactly

Bidding language examples

- Send a single value *c*
  - Additive valuation:

$$v_i(n) = n \cdot c$$

  - Single item valuation:

$$v_i(n) = \begin{cases} 0 & if\ n = 0 \\ c & otherwise \end{cases}$$

- Sending two values *c,B*:
  - Fixed budget valuation:

$$v_i(n) = \min(c \cdot n, B)$$

Weighted knapsack problem

- Binary variable $x_{k,i}$ indicates that bidder i is allocated k units
- Each bidder has sent a valuation function v_i such that $v_i(k)$ is the amount bidder i will pay if allocated k units

$$\text{maximize} \quad \sum_{i=1}^{N} \sum_{k=1}^{m} v_i(k) \cdot x_{k,i}$$

$$\text{subject to} \quad \sum_{i=1}^{N} \sum_{k=1}^{m} k \cdot x_{k,i} \le m$$

$$\sum_{k=1}^{m} x_{k,i} \le 1 \qquad \forall i, 1 \le i \le N$$

$$x_{k,i} \in \{0,1\} \qquad \forall k, 1 \le k \le m, \forall i, 1 \le i \le N$$

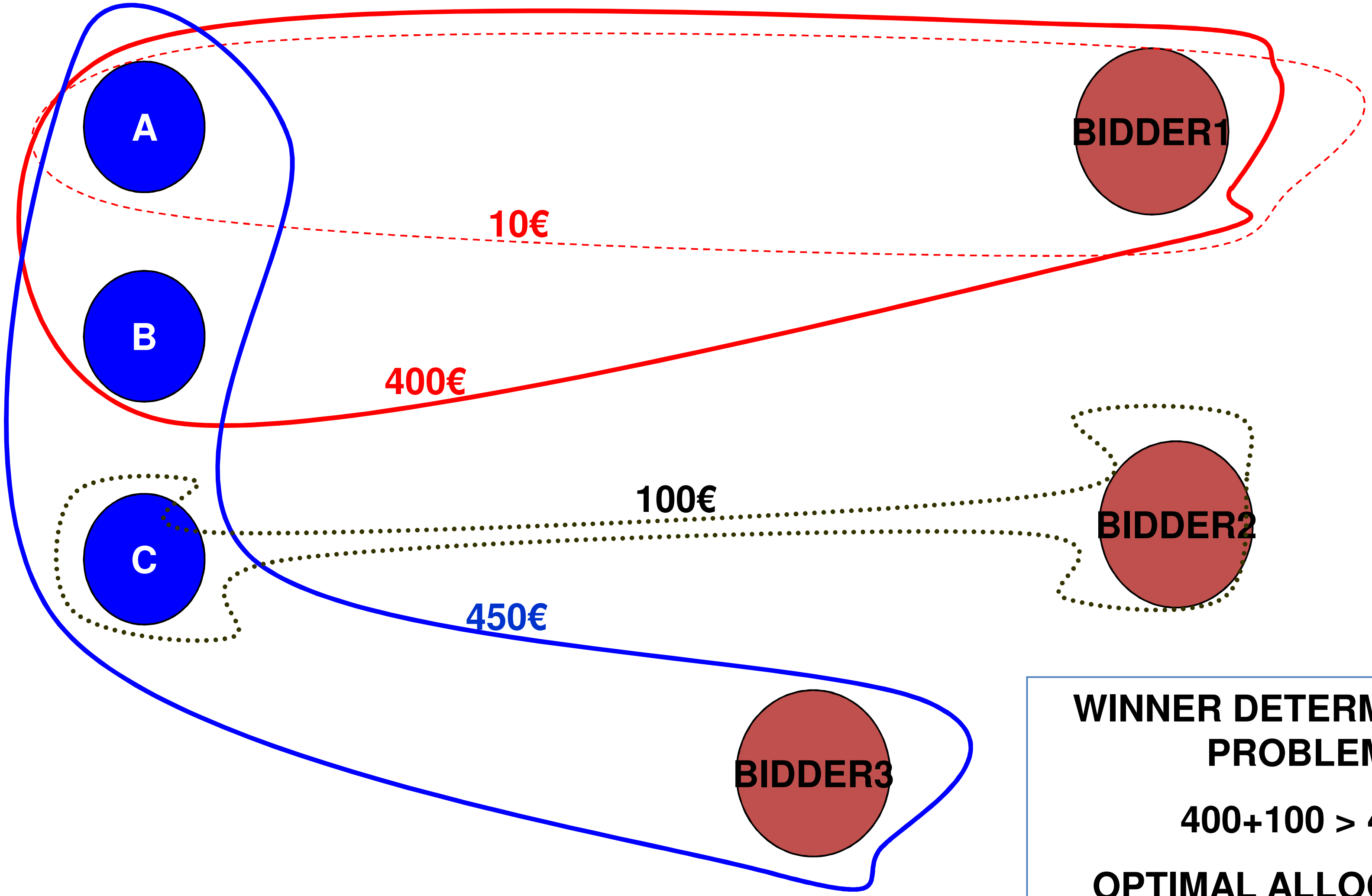
Outline

- Single-item auctions
- Multi-item auctions
- Supply chain formation
- Robust auctions
- Research opportunities

Combinatorial auctions

- In multi-unit auctions goods are interchangeable.
- In combinatorial auctions different goods are traded simultaneously.
- It is important when bidders' valuations depend strongly on which set of goods they receive.
- Examples:
  - Energy auctions
  - Corporate procurement auctions
  - Auctions for paths in a network (i.e. comm. connections)

Combinatorial Auctions



WINNER DETERMINATION PROBLEM:

400+100 > 450

OPTIMAL ALLOCATION: Bidders 1 and 2 win

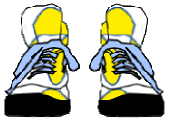
Valuation functions

- A set of bidders $N = \{1,\ldots,n\}$
- A set of goods $G = \{1,\ldots,m\}$
- For each bidder $i \in N$, we have a valuation funtion v_i

$$v_i : \mathrm{P}(G) \to \mathbb{R}$$

- Interesting when bidders have nonadditive valuation functions:
  - Substitutability

  - Complementarity

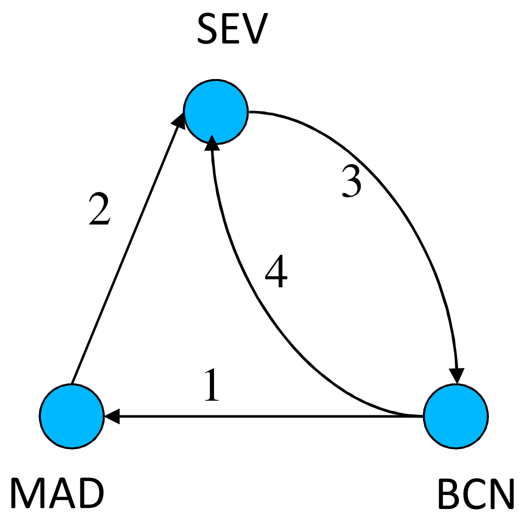


- Sequential / Parallel auctions: The exposure problem.

CA in Transportation [Sheffi 2004]

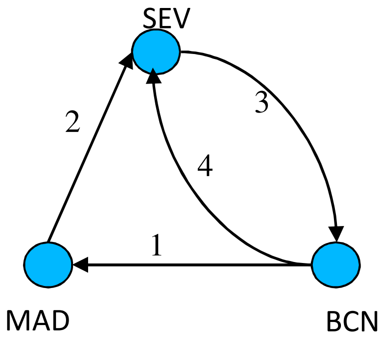
Given three cities

- Traffic Lanes (1,2,3,4)
- Similar Frequency & Distance
- Potential Operators (A;B)



Which operator gets which lanes?

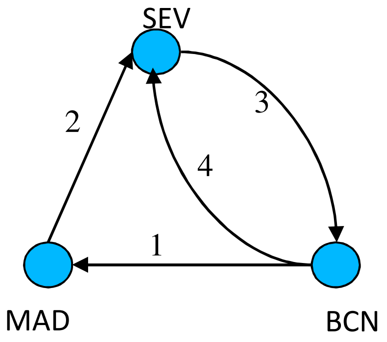
CA in Transportation



- Single round, sealed-bids.
- Information Exchange:
  - Auctioneer gives aggregated volume estimate
  - Operators submit lane offers
- Assignment Mechanism:
  - Lane-by-lane analysis
  - Highest bid wins

	Operator	
Lane	A	B
BCN-MAD	500	450
MAD-SEV	600	700
SEV-BCN	400	500
BCN-SEV	600	500

CA in Transportation

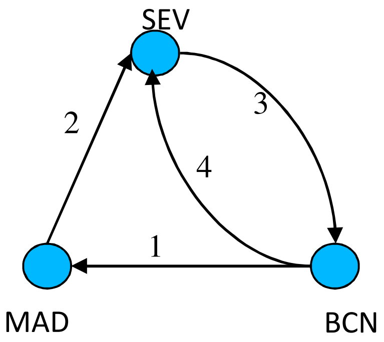


- Single round, sealed-bids.
- Information Exchange:
  - Auctioneer gives aggregated volume estimate
  - Operators submit lane offers
- Assignment Mechanism:
  - Lane-by-lane analysis
  - Highest bid wins

	Operator	
Lane	A	B
BCN-MAD	500	450
MAD-SEV	600	700
SEV-BCN	400	500
BCN-SEV	600	500

Income: 2300

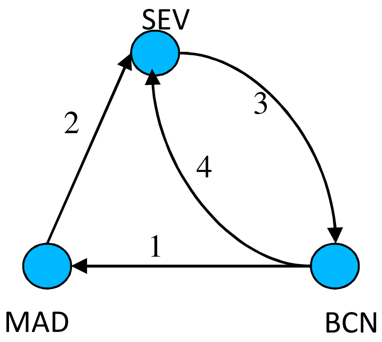
Transportation. Combinatorial bidding



	Operator A’s Bids						Operator B´s Bids					
Lanes	#1	#2	#3	#4	#5	#6	#1	#2	#3	#4	#5	#6
BCN-MAD	1				1		1				1	
MAD-SEV		1			1			1			1	
SEV-BCN			1		1	1			1		1	1
BCN-SEV				1		1				1		1
Bid	500	600	400	600	1.600	1.100	450	700	500	500	1.750	1.100

Which operator gets which lanes?

Transportation. Combinatorial bidding



	Operator A's Bids						Operator B´s Bids					
Lanes	#1	#2	#3	#4	#5	#6	#1	#2	#3	#4	#5	#6
BCN-MAD	1				1		1				1	
MAD-SEV		1			1			1			1	
SEV-BCN			1		1	1			1		1	1
BCN-SEV				1		1				1		1
Bid	500	600	400	600	1.600	1.100	450	700	500	500	1.750	1.100

Income: 2350

Valuation functions

- A set of bidders $N = \{1, \ldots, n\}$
- A set of goods $G = \{1, \ldots, m\}$
- For each bidder $i \in N$, we have a valuation funtion v_i

$$v_i : \mathrm{P}(G) \rightarrow \mathbb{R}$$

- Note that the size of $\mathrm{P}(G)$ is 2^m

Winner determination problem

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{S \subseteq G} v_i(S) \cdot x_{S,i}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \sum_{S \subseteq G,\ \{j\}} x_{S \cup \{j\},i} \leq 1 \qquad \forall j \in G$$

$$\sum_{S \subseteq G} x_{S,i} \leq 1 \qquad \forall i \in N$$

$$x_{S,i} \in \{0,1\} \qquad \forall S \subseteq G, \forall i \in N$$

Bidding languages

- We want to:
  - Communicate less information.
  - Have fewer decision variables when solving.
- A bidding language allows for a more efficient way of communicating a value function.
- Atomic bids: “I will buy A,B, and C for 10€”
- OR language: Each bidder sends a set of atomic bids. The auctioneer can accept as many as he wants.
- XOR language: Each bidder sends a set of atomic bids. The auctioneer can only accept a single bid from each bidder.

OR language Winner Determination Problem

- A set of atomic bids $B = \{\langle G_1, p_1 \rangle, \ldots, \langle G_n, p_n \rangle\}$
- A set of goods $G = \{1, \ldots, m\}$

$$
\begin{aligned}
&\text{maximize} \quad \sum_{i=1}^{n} p_i x_i \\
&\text{subject to} \quad \sum_{i \text{ s. t. } j \in G_i} x_i \leq 1 \qquad \forall j \in G \\
&\qquad\qquad x_i \in \{0,1\} \qquad \forall i \in B
\end{aligned}
$$

CATS – Testing WDP algorithms [Leyton-Brown 2006]

- Generator of CA instances to test CA algorithms.
- Distributions of instances from realistic domains:
  - Paths in space (e.g. truck routes)
  - Proximity in space (e.g. adjacent pieces of real state)
  - Arbitrary relationships (e.g. Pollution rights auction)
  - Temporal matching (e.g. airport take-off & landing rights)
  - Temportal scheduling (e.g. Job-shop scheduling)
- …as well as a collection of artificial distributions used in the literature.

Available @ http://www.cs.ubc.ca/~kevinlb/CATS/

Task allocation

- Consider a scenario where agents want to allocate video editing tasks to task performers

Rendering task

Bret

Jemaine

Centre

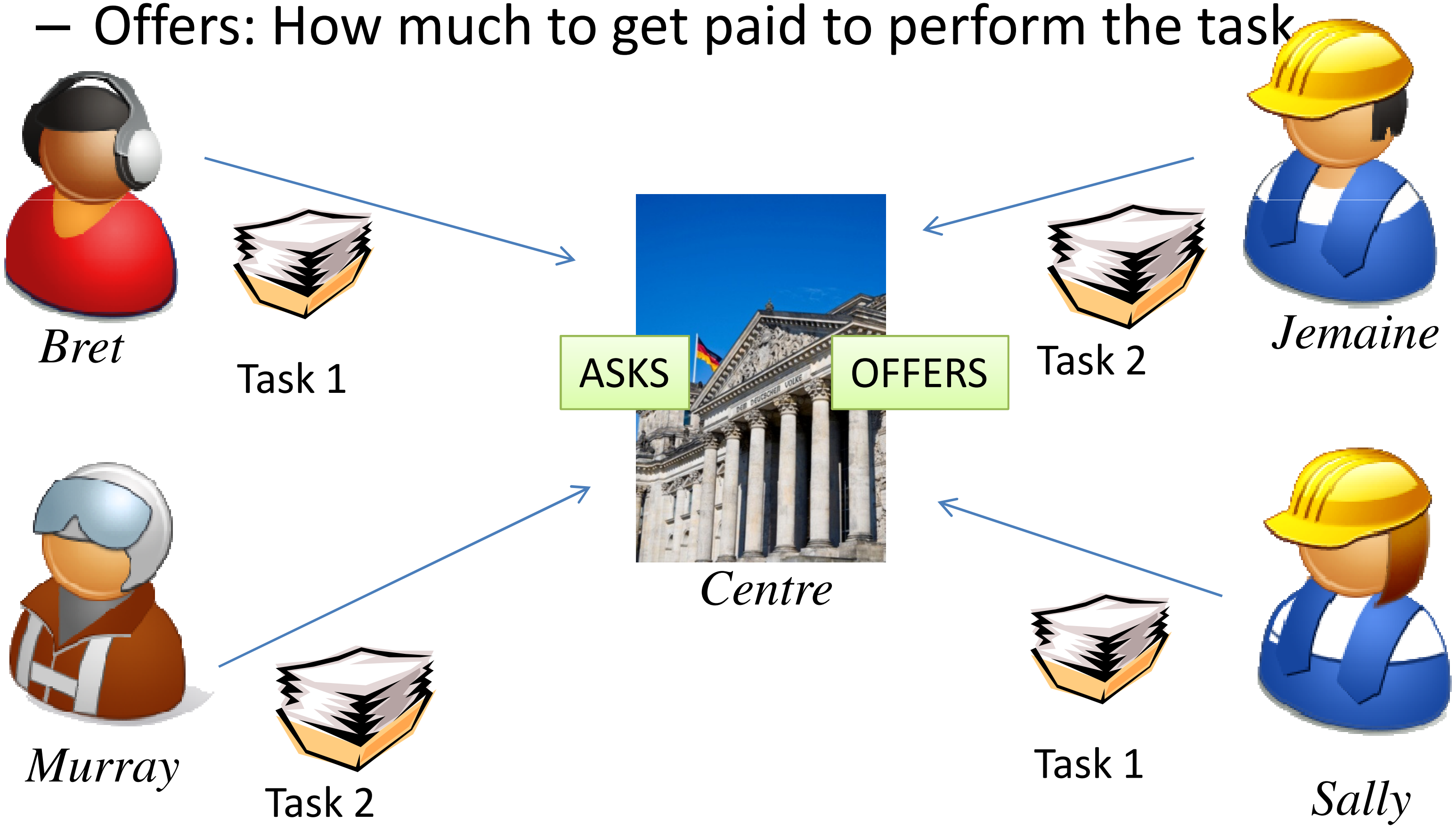
Murray

Rendering task

Sally

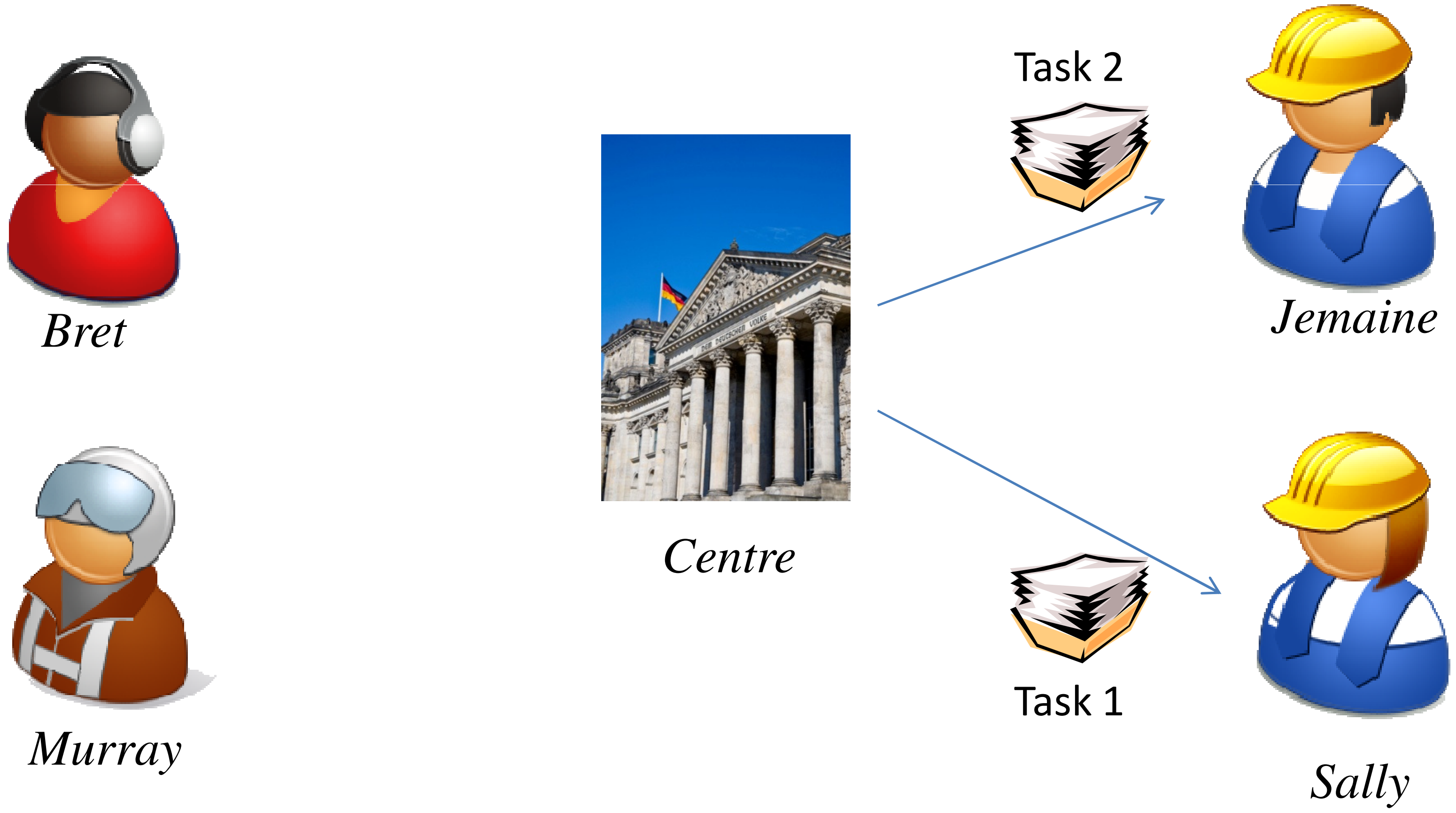
Task allocation

- Agents send to the centre:
  - Asks: How much to pay to have the task performed
  - Offers: How much to get paid to perform the task



Task allocation

- The centre allocates the tasks and pays the agents

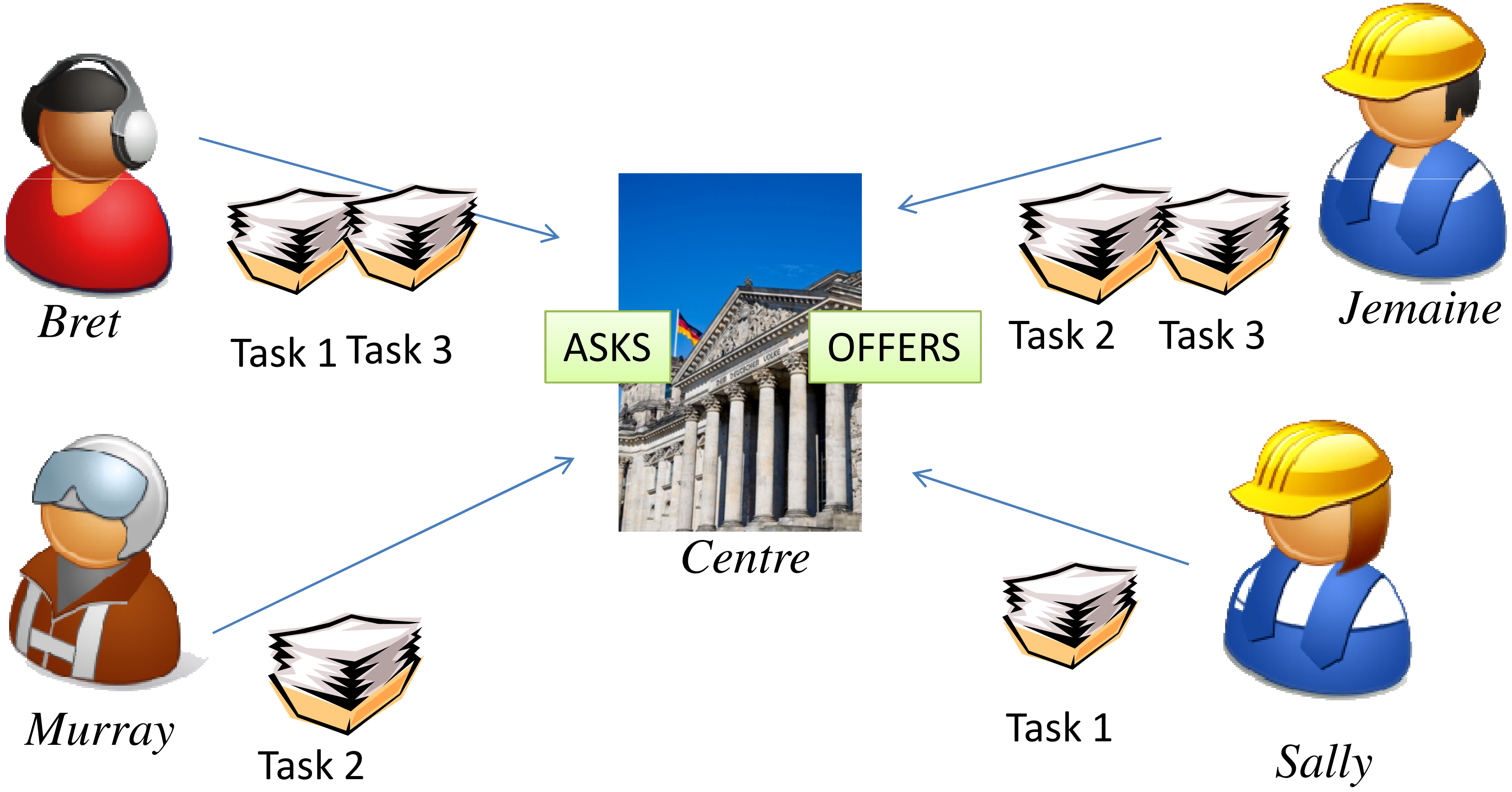


Combinatorial exchanges

- In markets with many buyers and many sellers, exchanges are a natural choice for a market mechanism.
- In a *combinatorial exchange, the trades* can involve multiple buyers and multiple sellers each.
- Participants in a combinatorial exchange are allowed to both buy and sell (*bundles* of) items, or just buy or just sell.
- Single-unit exchanges, multi-unit exchanges are particular cases of combinatorial exchanges.

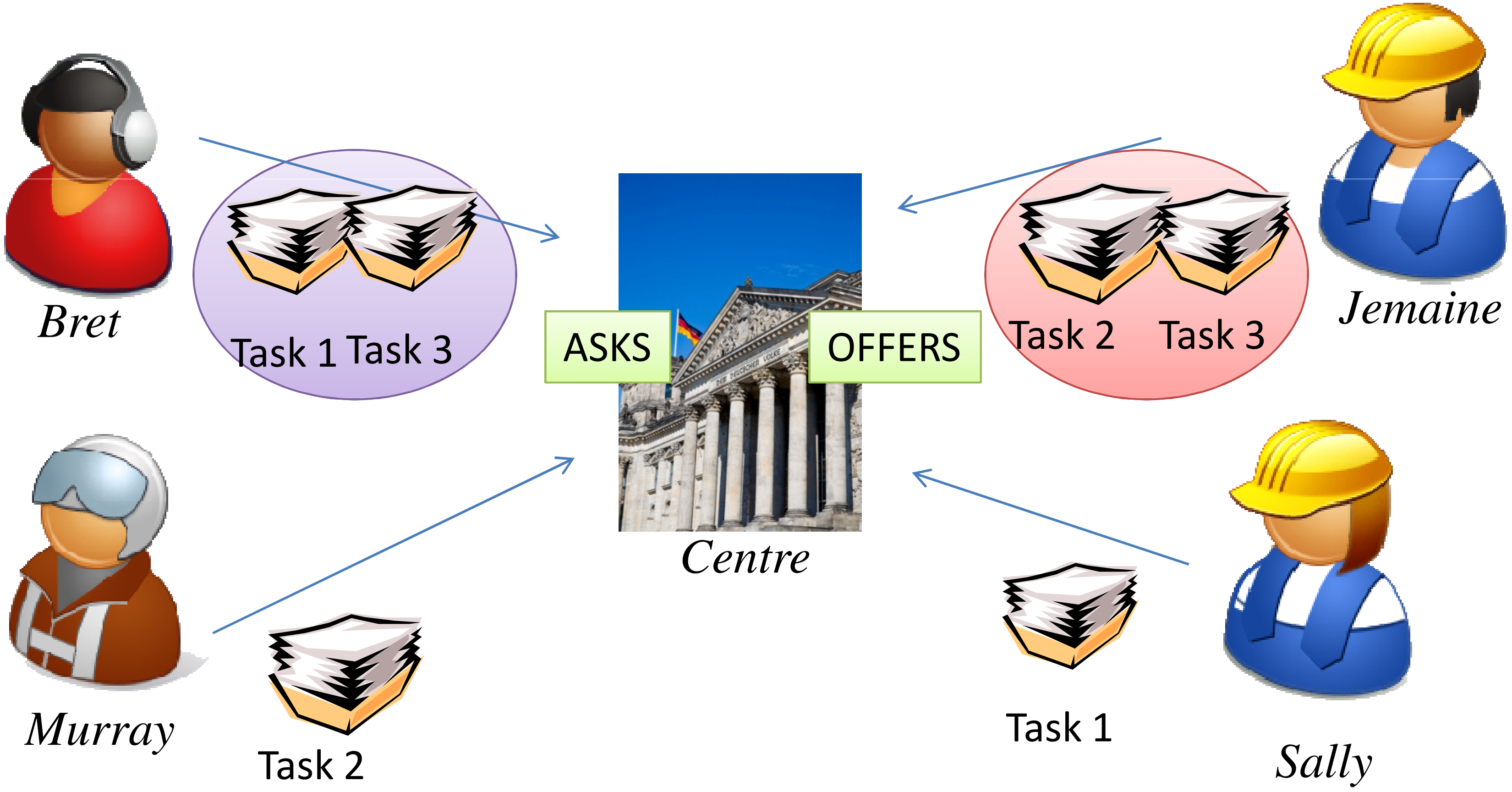
Task allocation
Combinatorial exchange

- Agents send to the centre combinatorial asks and combinatorial offers on bundles of tasks.



Task allocation
Combinatorial exchange

- Agents send to the centre combinatorial asks and combinatorial offers on bundles of tasks.



Combinatorial exchanges

- A set of bidders $N = \{1, \ldots, n\}$
- A set of items $G = \{1, \ldots, m\}$
- For each bidder $i \in N$, a bid B_i in this setting is

$$B_i = \left\langle \left(q_i^1, \ldots, q_i^m \right), p_i \right\rangle$$

- q_i^k is *the asked/offered number of units of item k and*
- p_i is the price
- $q_i^k > 0$ *means buying;* $q_i^k < 0$ *means selling*
- $p_i > 0$ *means offering;* $p_i < 0$ *means asking*

Winner determination problem

To label the bids as winning or losing so as to maximize surplus under the constraint that demand does not exceed supply:

$$\text{maximize} \quad \sum_{i=1}^{n} p_i x_i$$

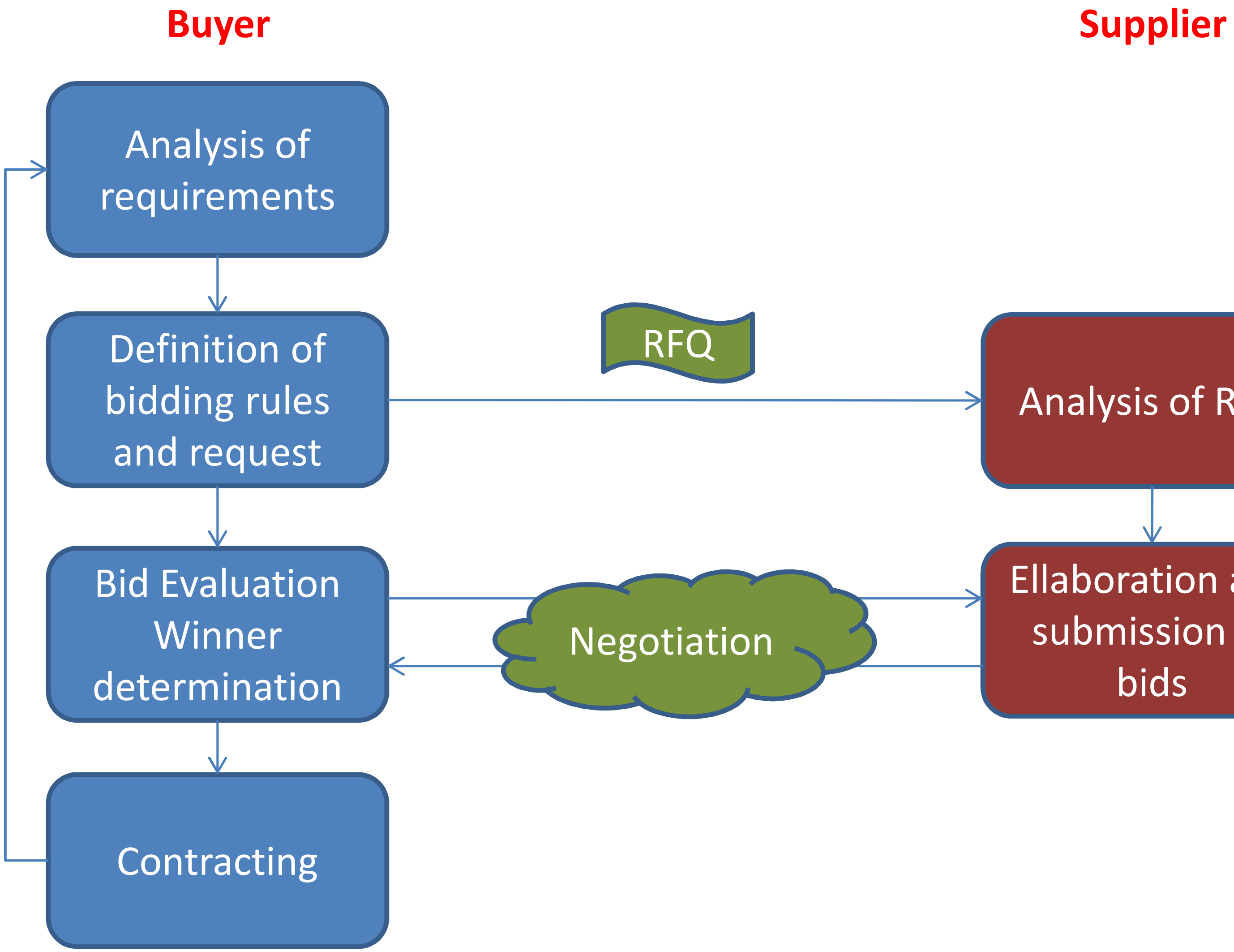
$$\text{subject to} \quad \sum_{i=1}^{n} q_i^j x_i \leq 0 \qquad \forall j \in G$$

$$x_i \in \{0,1\} \qquad \forall i \in N$$

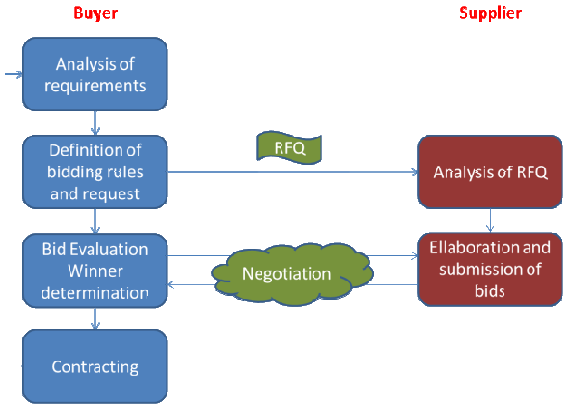
Outline

- Single-item auctions
- Multi-item auctions
- Supply chain formation
- Robust auctions
- Research opportunities

Purchasing process


Buyer
Supplier
Analysis of requirements
Definition of bidding rules and request
RFQ
Analysis of RFQ
Bid Evaluation Winner determination
Negotiation
Ellaboration and submission of bids
Contracting

Reverse auctions for industrial procurement



Buyer
Supplier
Analysis of requirements
Definition of bidding rules and request
RFQ
Analysis of RFQ
Bid Evaluation Winner determination
Negotiation
Ellaboration and submission of bids
Contracting

- The first reverse auction took place in the 1990s, when the Internet was introduced as an auction tool.
- The buyer advertises a need for an item or service.
- Sellers then place bids for the amount they expect to be paid in order to perform such a service or provide such an item.
- Generally, the seller who places the lowest bid will win the job or sell the item.
- As of 2007 more than 50% of the large companies were using reverse auctions for their purchases with 176M$ mean expenditure.

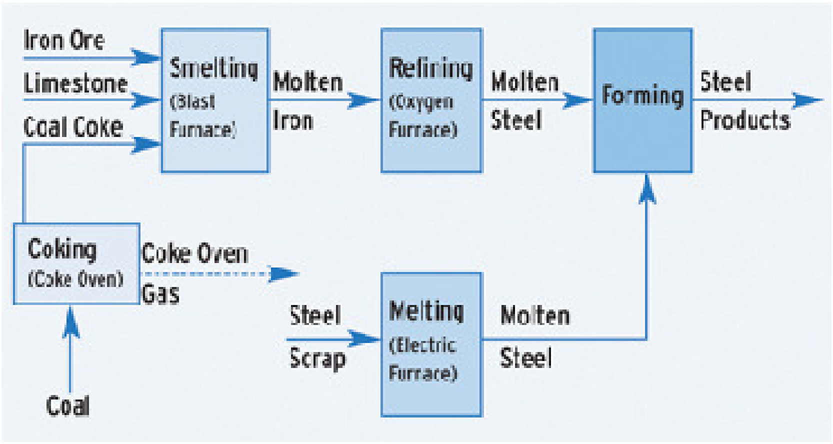
Reverse auctions for industrial procurement [Bichler et al. 2006]

- RFQs range from ten to 90,000 goods with an average of 250 goods:
  - Reverse combinatorial auctions are the right tool
- Auction design objectives:
  - Cost minimization
  - Supplier perceived fairness
- Auction protocol:
  - Public procurement: First-price sealed bid
  - Private procurement: Multiple round

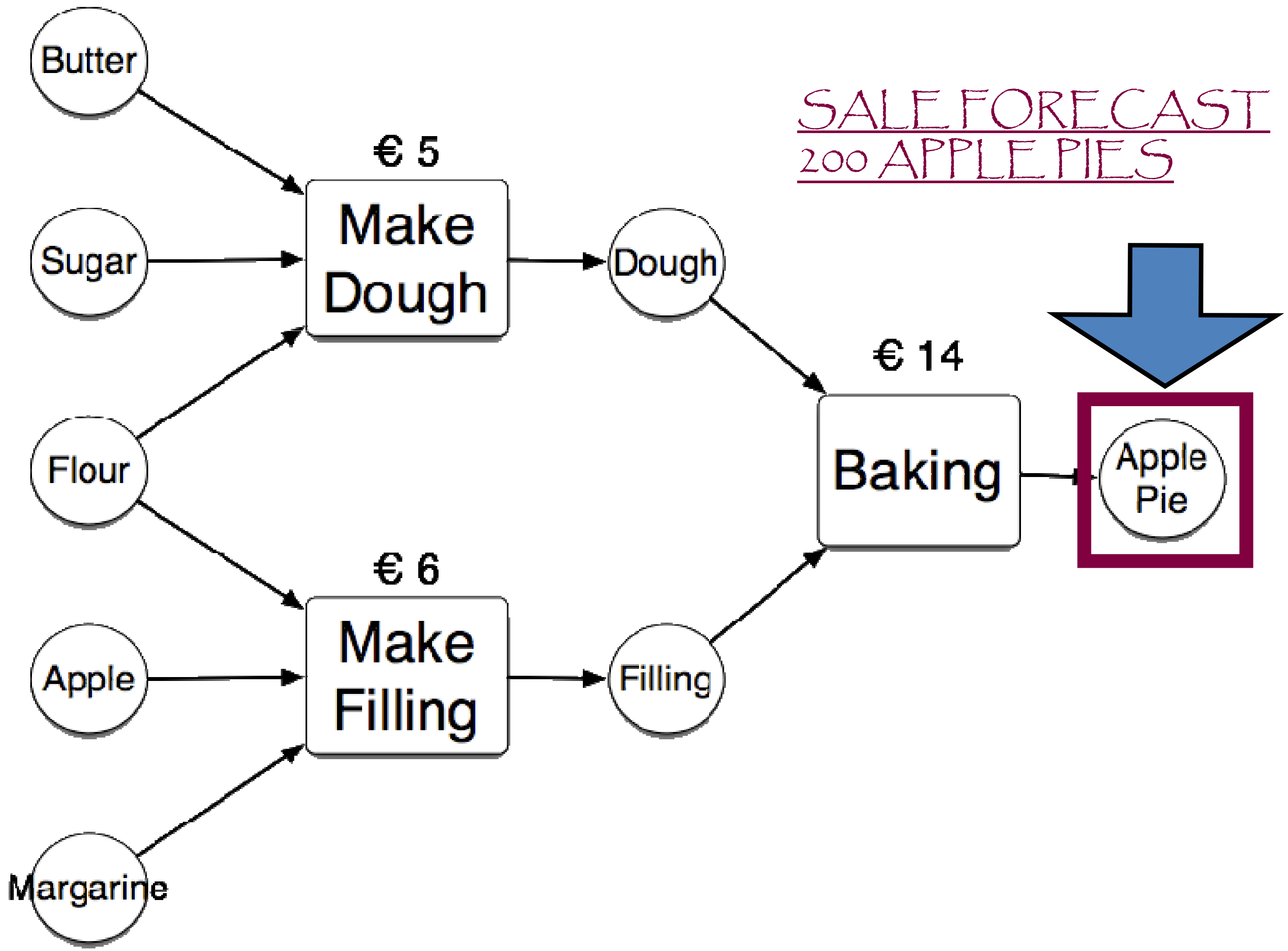
Supply chain formation

- A supply chain is a set of organizations directly linked by the flow of products, services, finances, and information from a source to a customer.

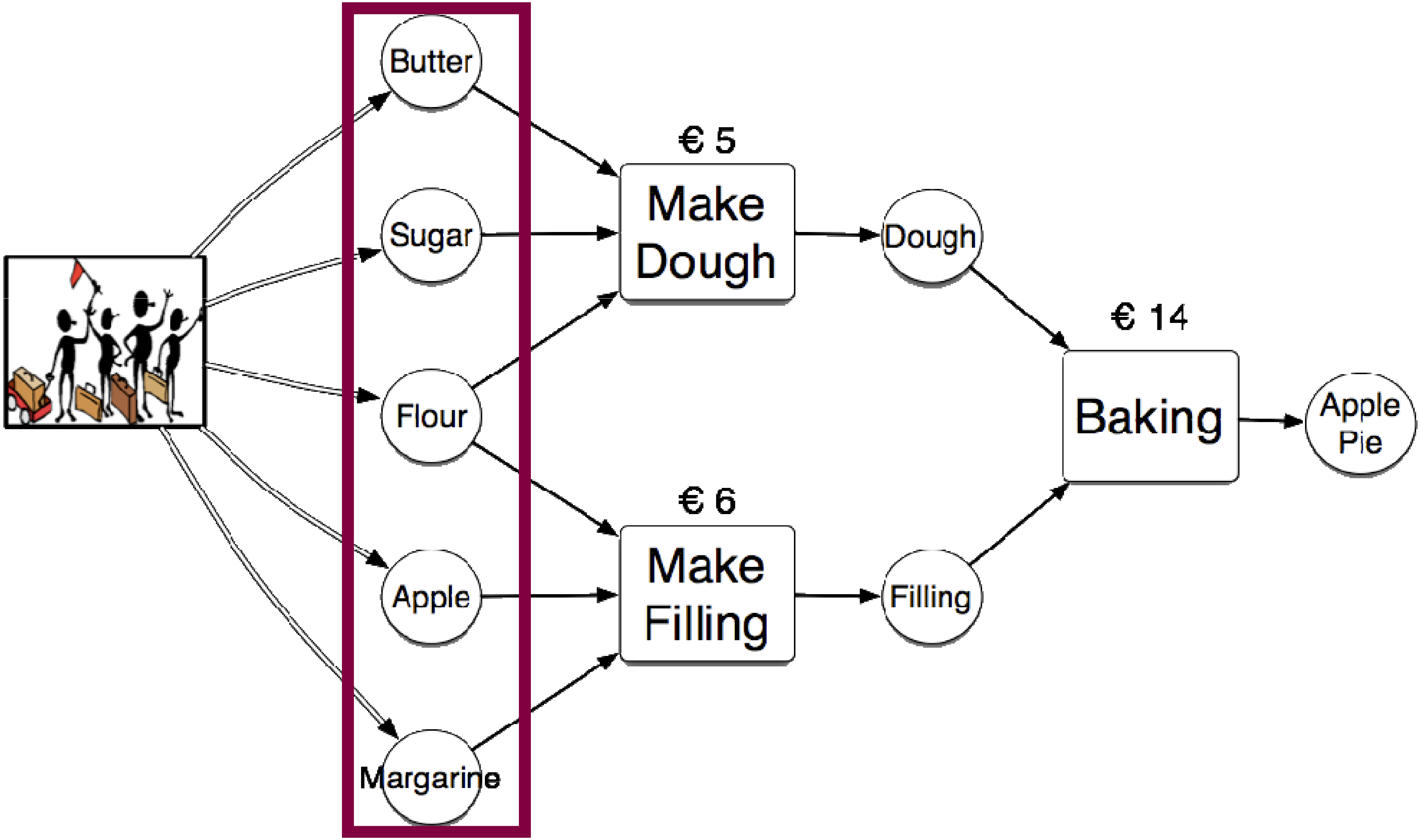
The iron and steel manufacturing process



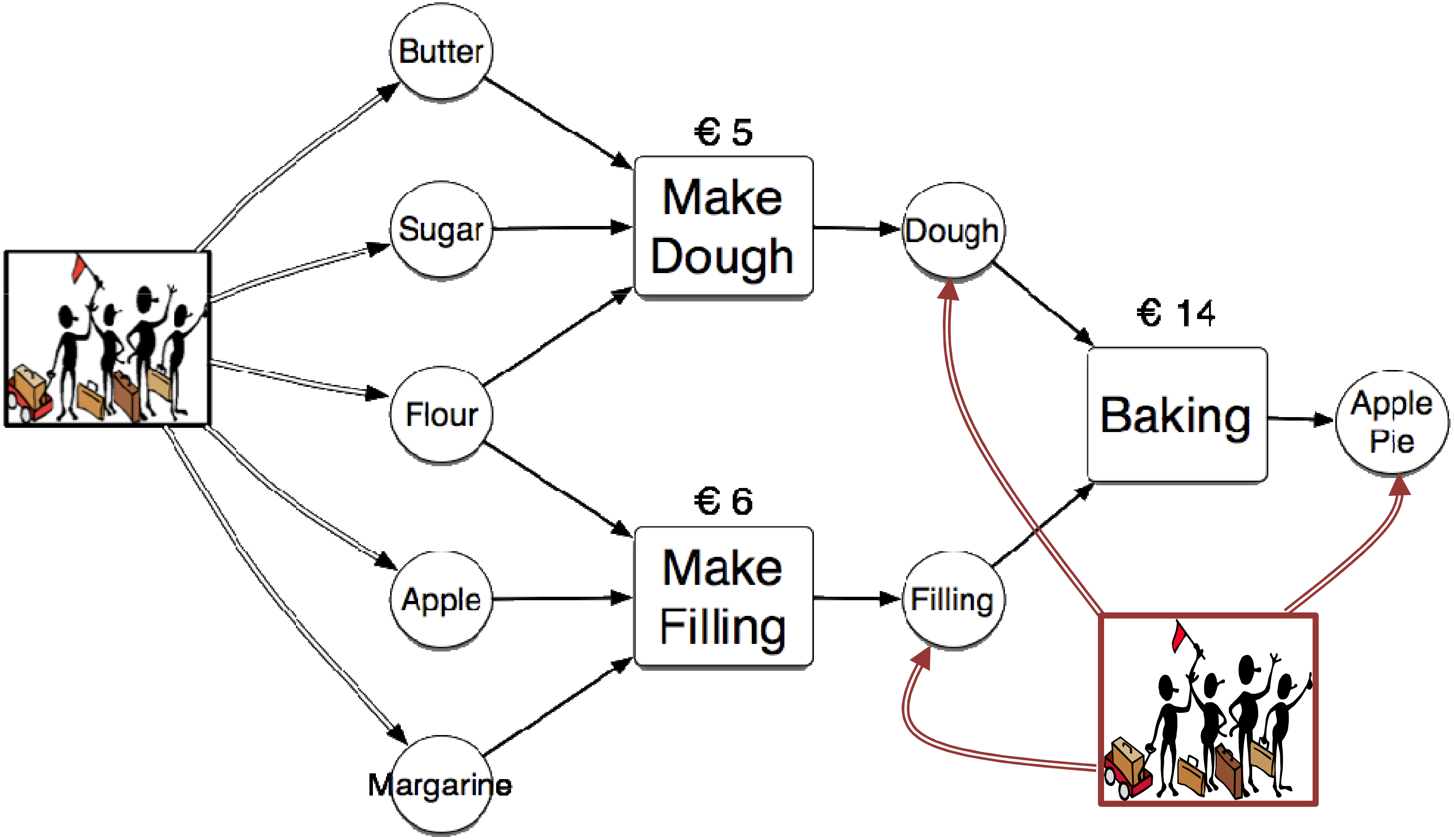
Production process example



Purchasing (Buy)

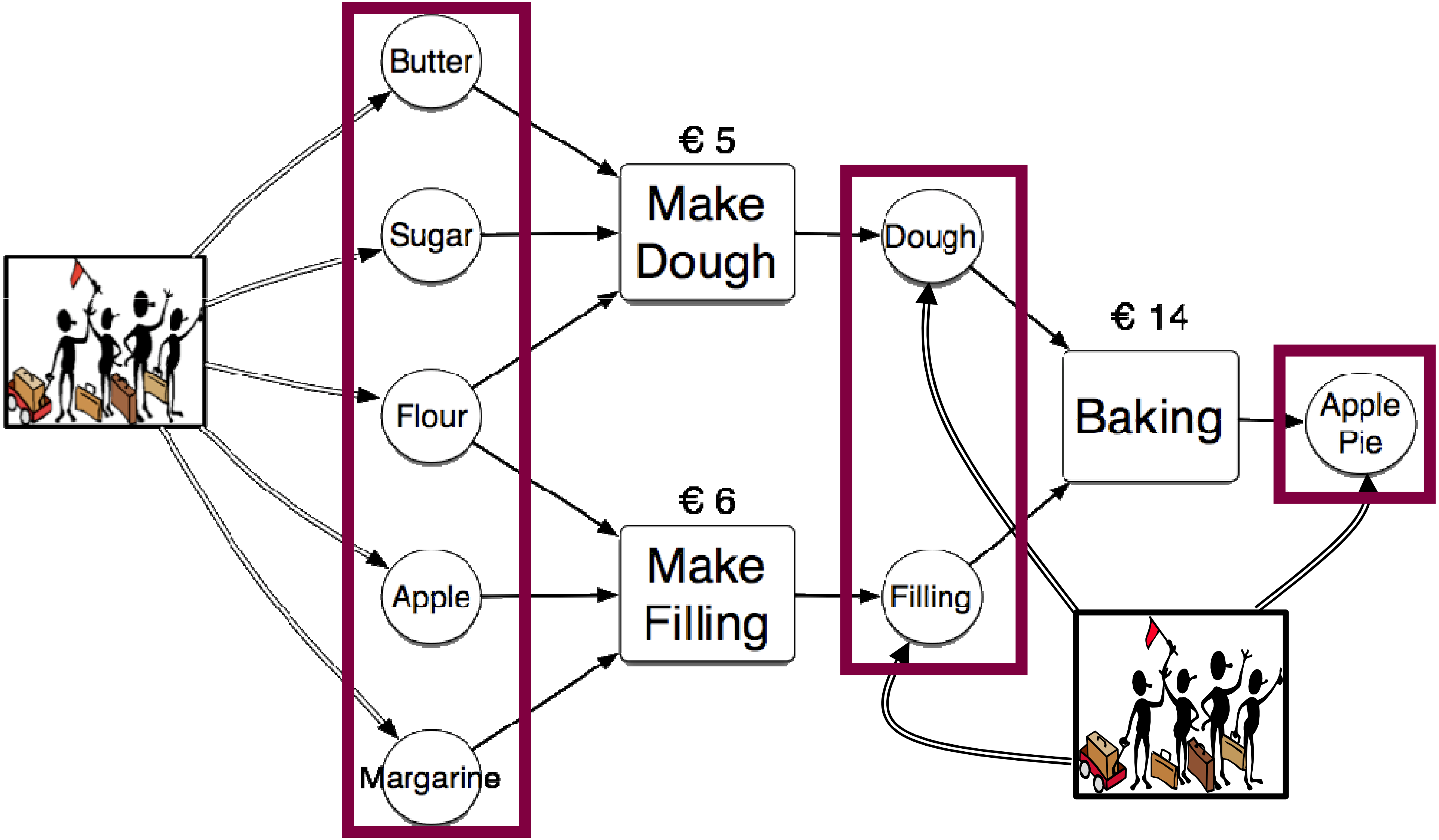


Make-or-Buy

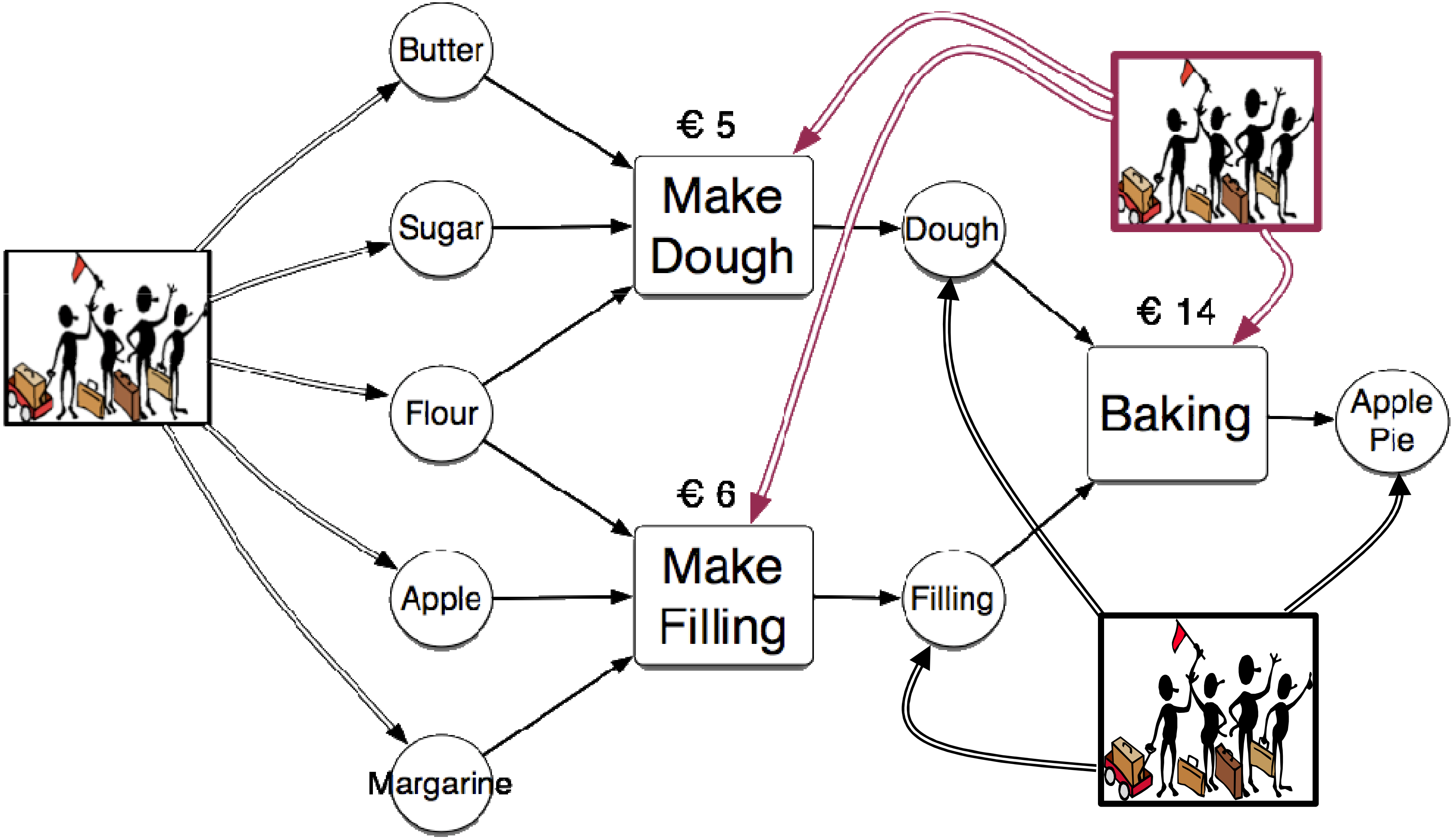


Butter
Sugar
Flour
Apple
Margarine
€ 5
Make Dough
€ 6
Make Filling
Dough
Filling
€ 14
Baking
Apple Pie

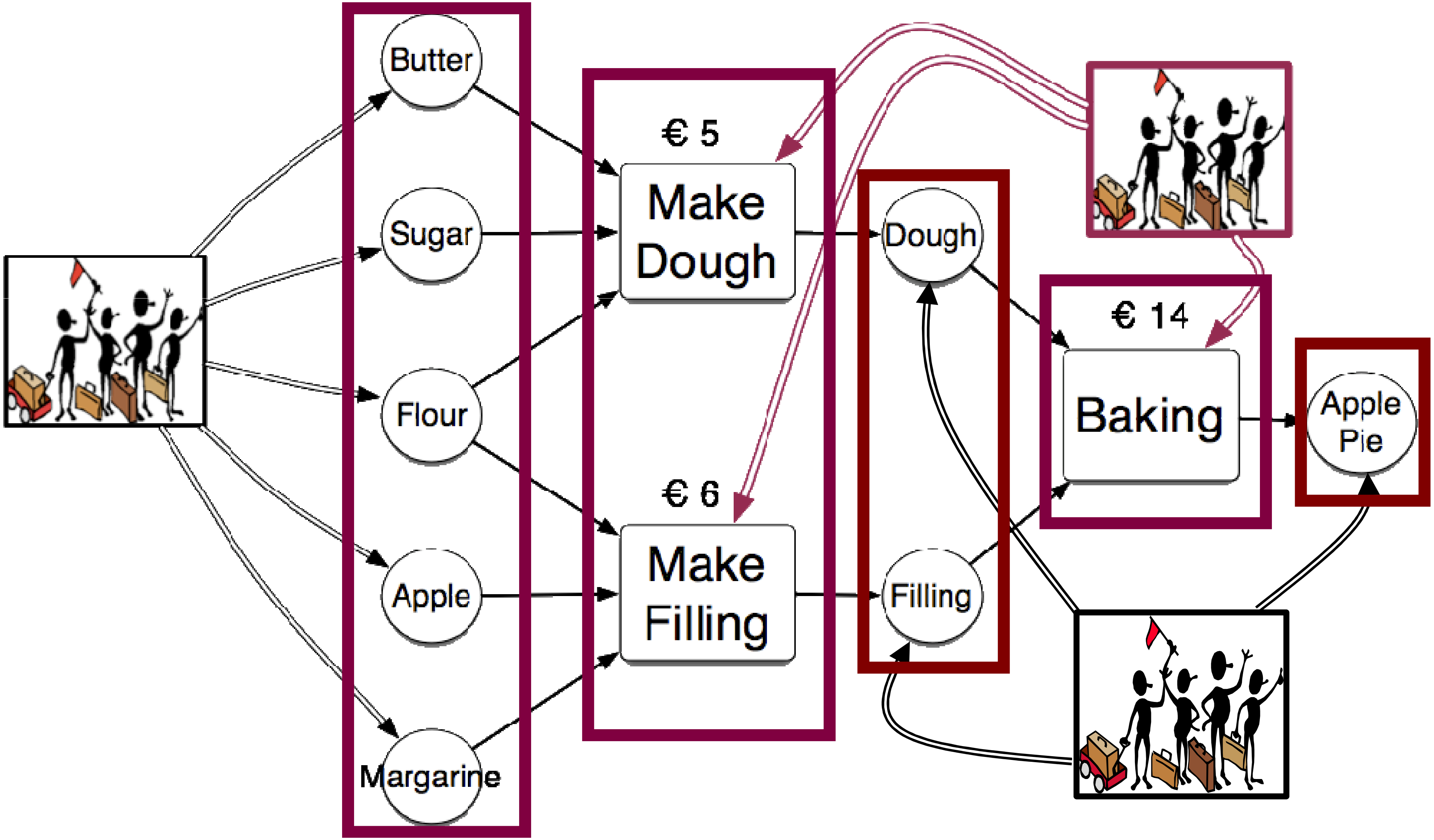
Make-or-Buy


Butter
Sugar
Flour
Apple
Margarine
€ 5
Make Dough
€ 6
Make Filling
Dough
Filling
€ 14
Baking
Apple Pie

Make-or-Buy-or-Collaborate



Make-or-Buy-or-Collaborate



Combinatorial auctions for supply chain formation [Walsh et al., 2000]

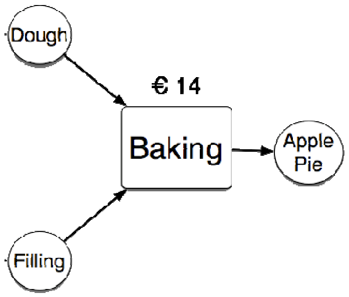
- Provided that:
  - The graph is acyclic.
  - Each transformation produces only one unit of a single output good
- The optimal allocation for a supply chain formation problem can be obtained by means of a combinatorial auction.

Combinatorial auctions for supply chain formation [Walsh et al., 2000]

- Each agent i participating in the supply chain sends a bid of the form:

$$\left\langle p_i, \left[g_1, q_i^1\right], \ldots, \left[g_n, q_i^n\right] \right\rangle$$

  - I charge you p_i € (to give you/ if you give me) q_i^1 units of g_1 and (to give you/ if you give me) q_i^2 units of g_2 and …



I charge you 14 € to give you 1 unit of apple pie if you give me 1 unit of dough and 1 unit of filling.

Combinatorial auctions for supply chain formation [Walsh et al., 2000]

- The WDP defines a variable x_i for each bid

$$\left\langle p_i, \left[g_1, q_i^1\right], \ldots, \left[g_n, q_i^n\right] \right\rangle$$

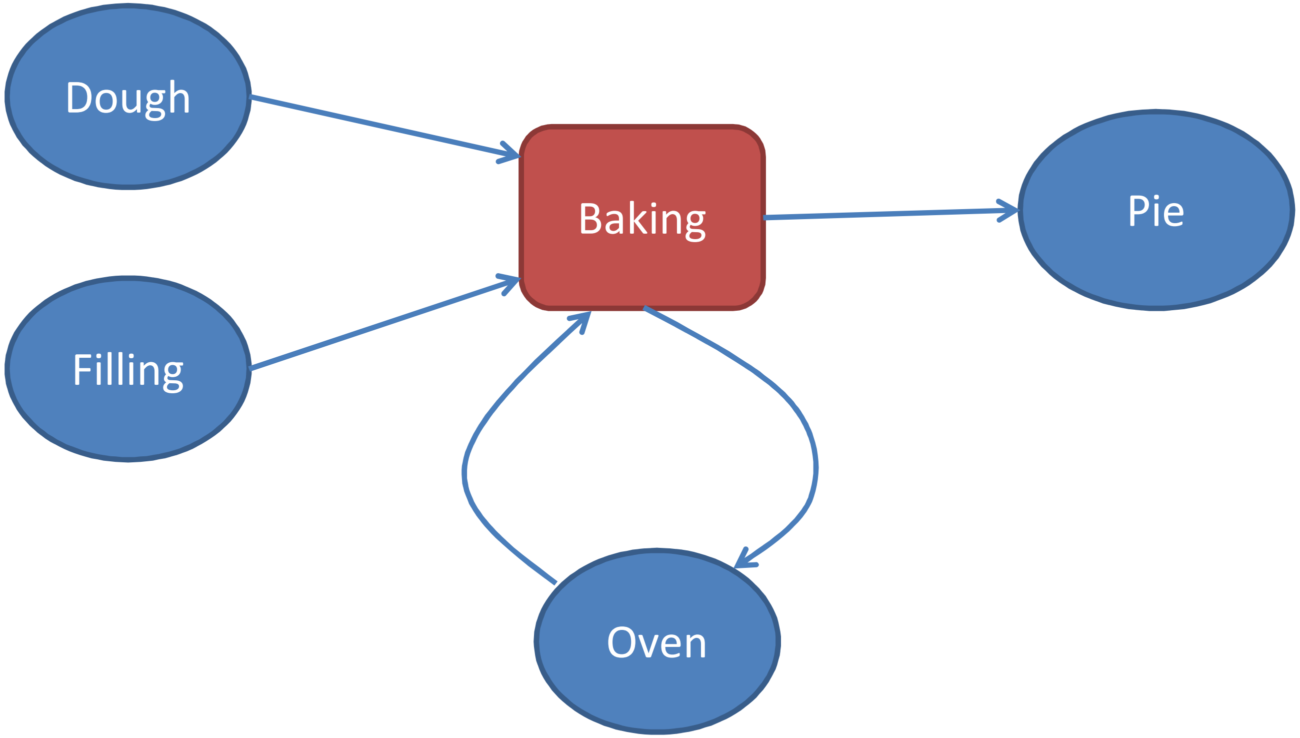
- Then it solves the following integer program:

$$\text{maximize} \sum_{i=1}^{B} p_i x_i$$

$$\text{subject to} \sum_{i=1}^{B} q_i^j x_i = 0 \qquad \forall 1 \le j \le G$$

Mixed Multi Unit Combinatorial Auction (MMUCA)

- Sometimes reasonable business collaborations can only be represented by means of loops and need more than one output



MMUCA

- An extension of Combinatorial Auctions that provides:
  - A bidding language to express preferences over operations across the supply chain
  - An optimisation problem that selects:
    - The best business partners
    - A feasible sequence of operations

Automatically selects the best Make-or-Buy-or-Collaborate decisions

MMUCA Bidding language

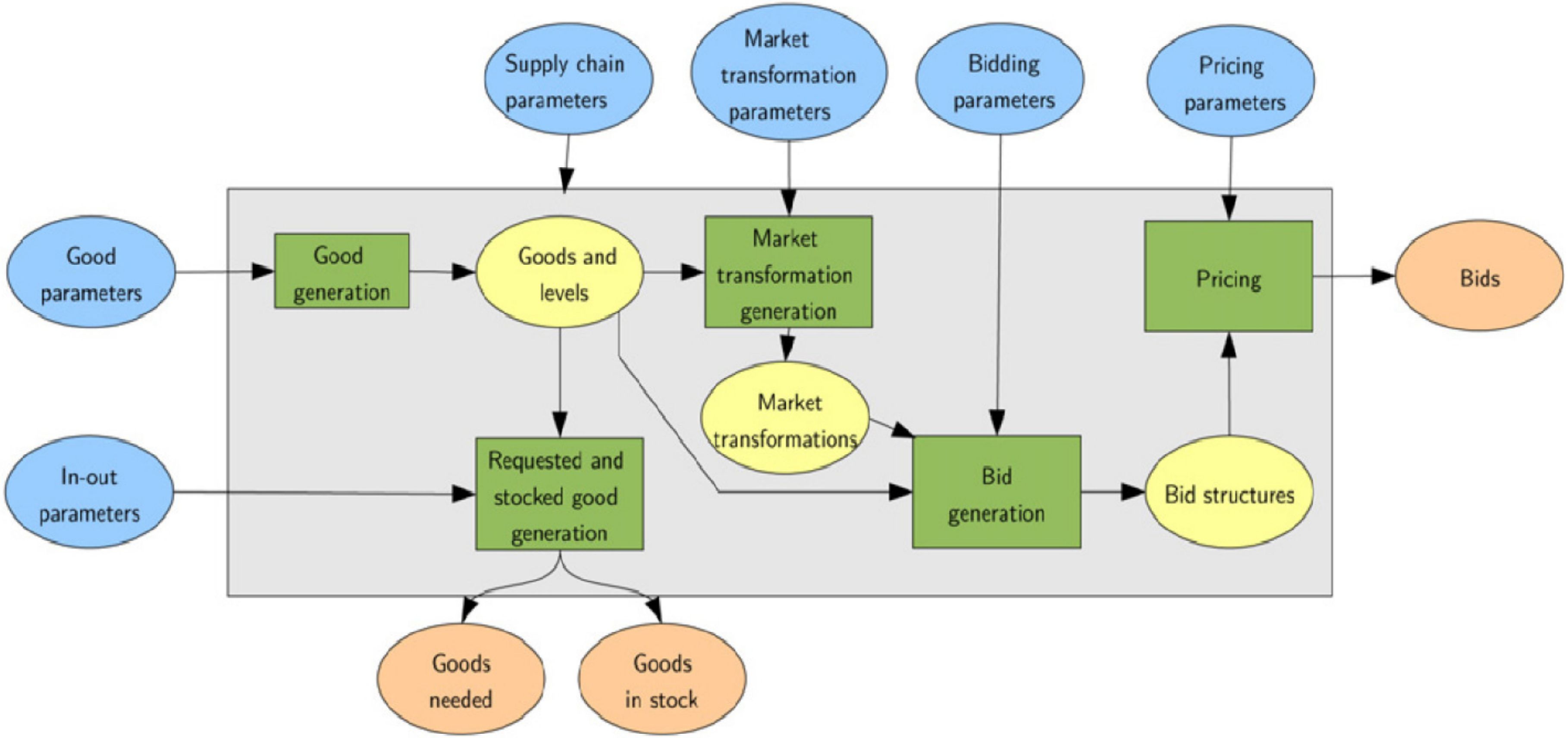
- A bidder can express preferences over bundles of transformations (Atomic Bid)
- A bidder can submit combinations of Atomic Bids (e.g. XOR, OR)
- Theorem [Cerquides et al. 2007]: XOR is expressive enough to represent any valuation
- MMUCA generalizes:
  - reverse auctions
  - direct auctions
  - exchanges
  - CAs for SCF

MMUCA Solvers [Giovannucci 2008]

SOLVER	TOPOLOGY	#Decision Variables
Petri-Nets Based Integer Program	ACYCLIC	O(N)
Direct Integer Program	ANY	$O(N^2)$
Connected Components IP	ANY	O(N) ≤ ??<< $O(N^2)$

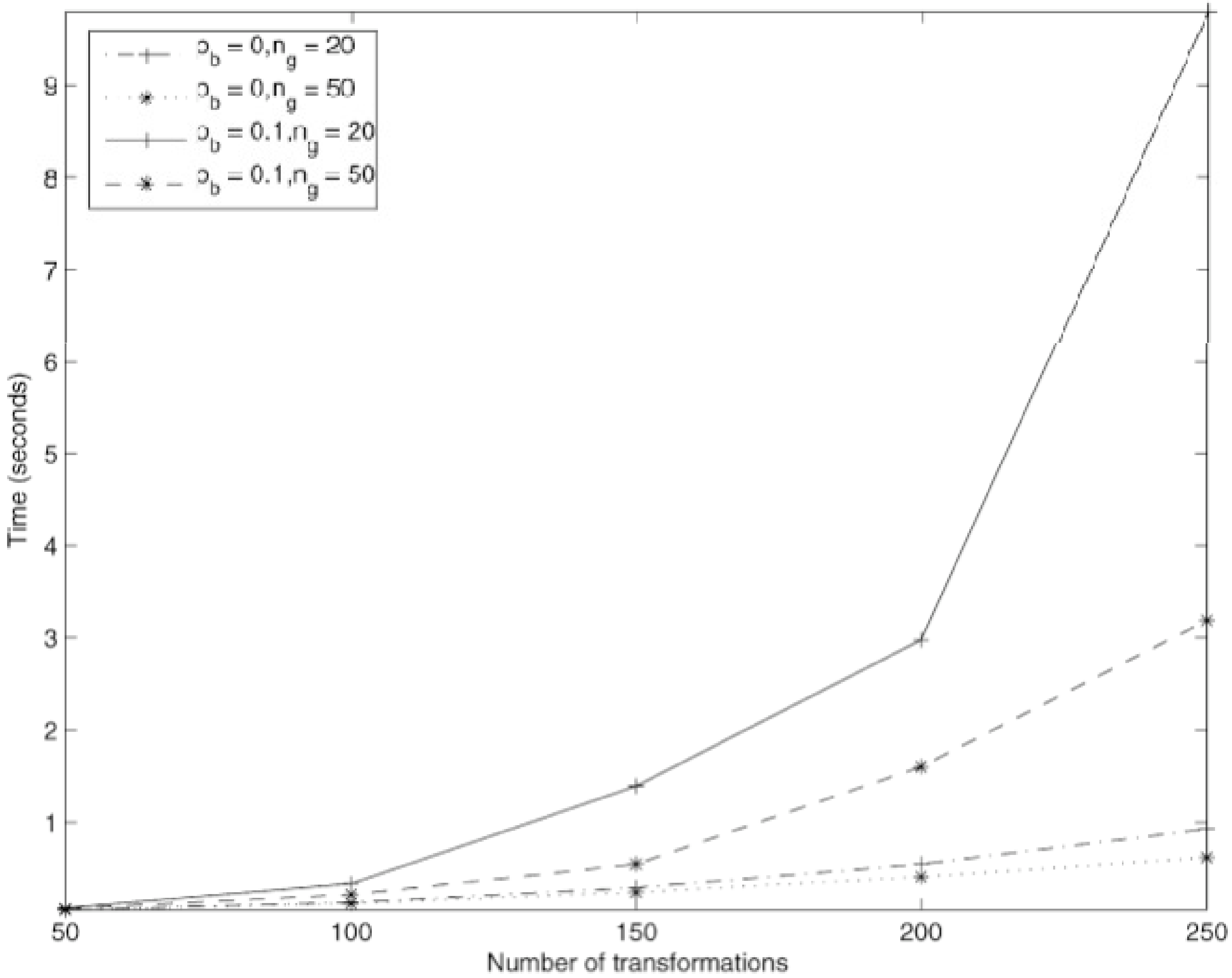
N: overall number of transformations bid

MMUCA generator [Vinyals et al. 2008]



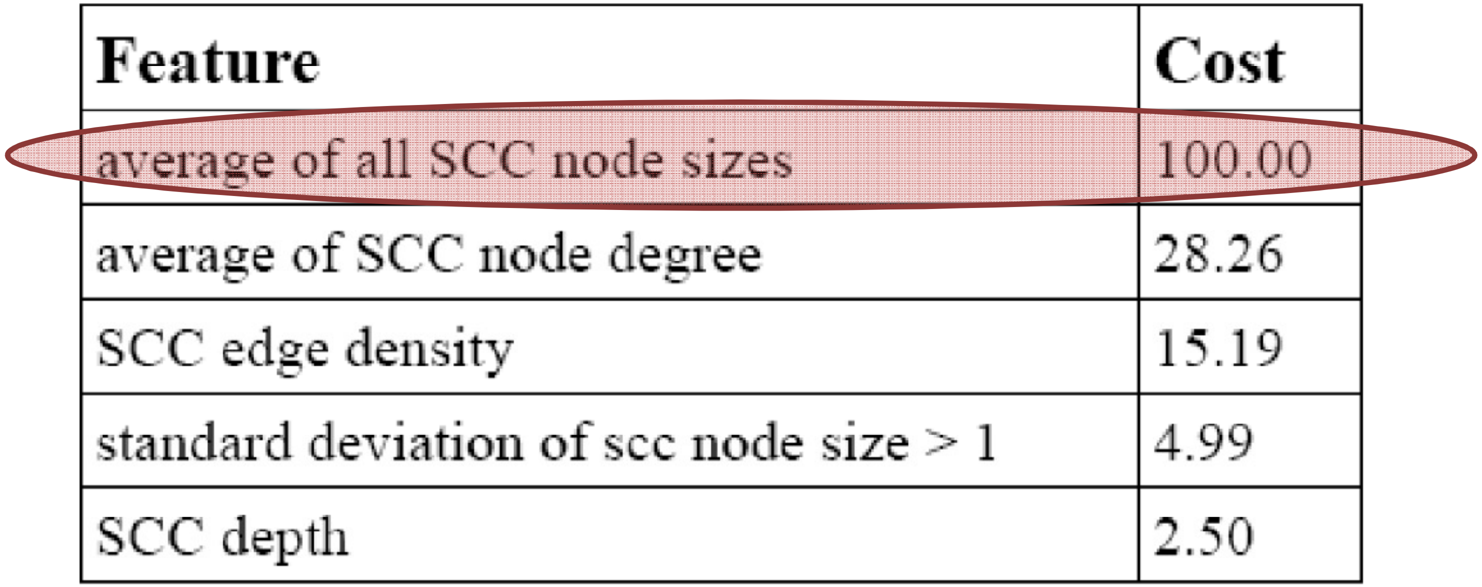
Available @t http://zeus.maia.ub.es/~cerquide/mmucaGenerator
http://www.iiia.csic.es/~rosell/MMUCAP/

MMUCA resolution time



Analysing MMUCA hardness: Empirical findings [Almajano et. al, 2010]

- We generated problems with different characteristics
- Used machine learning to predict MMUCA solving time
- Anlyzed the relevance of each feature in this model.

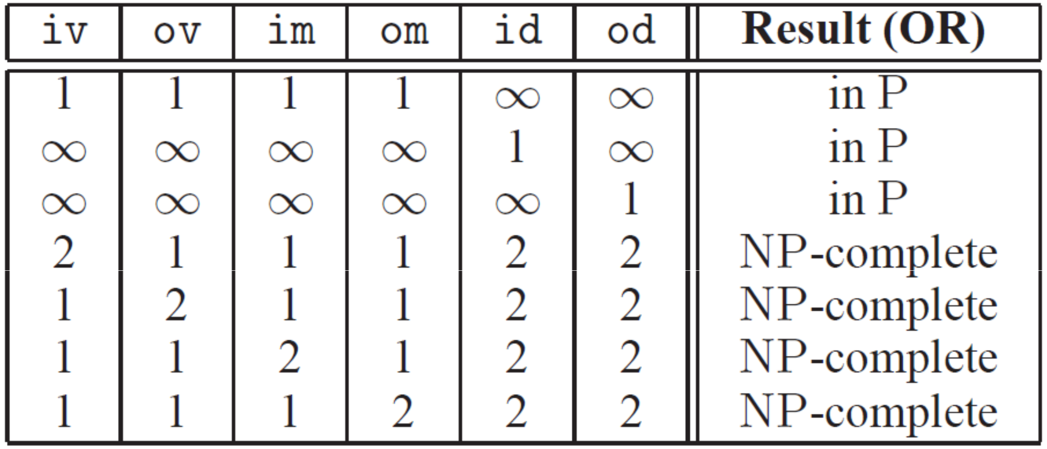


Feature	Cost
average of all SCC node sizes	100.00
average of SCC node degree	28.26
SCC edge density	15.19
standard deviation of scc node size > 1	4.99
SCC depth	2.50

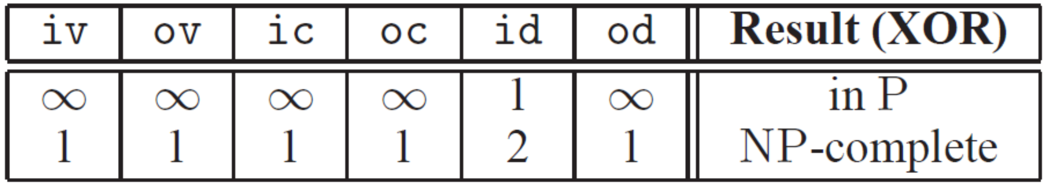
- The larger a feature’s cost, the higher its importance.

Analysing MMUCA hardness: Theoretical results [Fionda & Greco, 2009]

- Determining FEASIBILITY is NPC for most classes



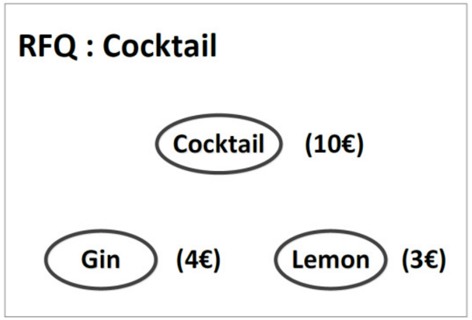
iv	ov	im	om	id	od	**Result (OR)**
1	1	1	1	∞	∞	in P
∞	∞	∞	∞	1	∞	in P
∞	∞	∞	∞	∞	1	in P
2	1	1	1	2	2	NP-complete
1	2	1	1	2	2	NP-complete
1	1	2	1	2	2	NP-complete
1	1	1	2	2	2	NP-complete

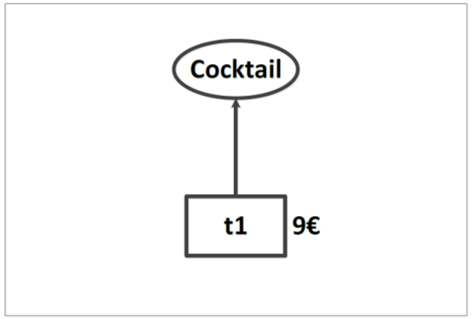


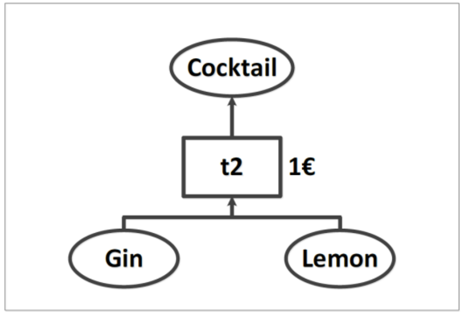
iv	ov	ic	oc	id	od	**Result (XOR)**
∞	∞	∞	∞	1	∞	in P
1	1	1	1	2	1	NP-complete

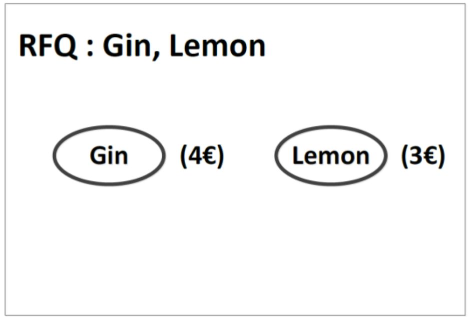
- If the auction has limited intricacy and can be represented as a hypergraph with bounded hypertree width then the WDP is in P.

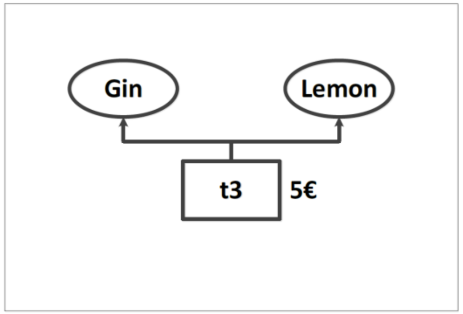
Sequential MMUCA [Mikhaylov 2011]

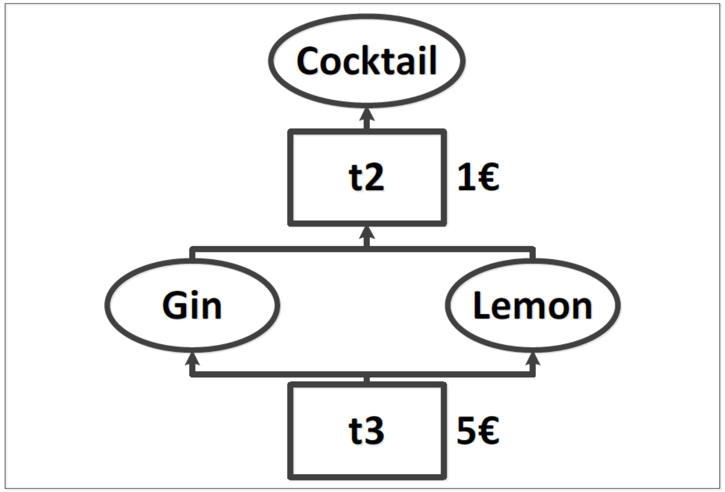












Decentralized SCF [Winsper & Chli, 2010]

- Map Walsh & Wellman Supply Chain Formation problem to a DCOP
- Solve distributedly using Loopy Belief Propagation (Max-Sum)
- Finally post-process the solution to turn the result into a feasible supply chain.

Outline

- Single-item auctions
- Multi-item auctions
- Supply chain formation
- Robust auctions
- Research opportunities

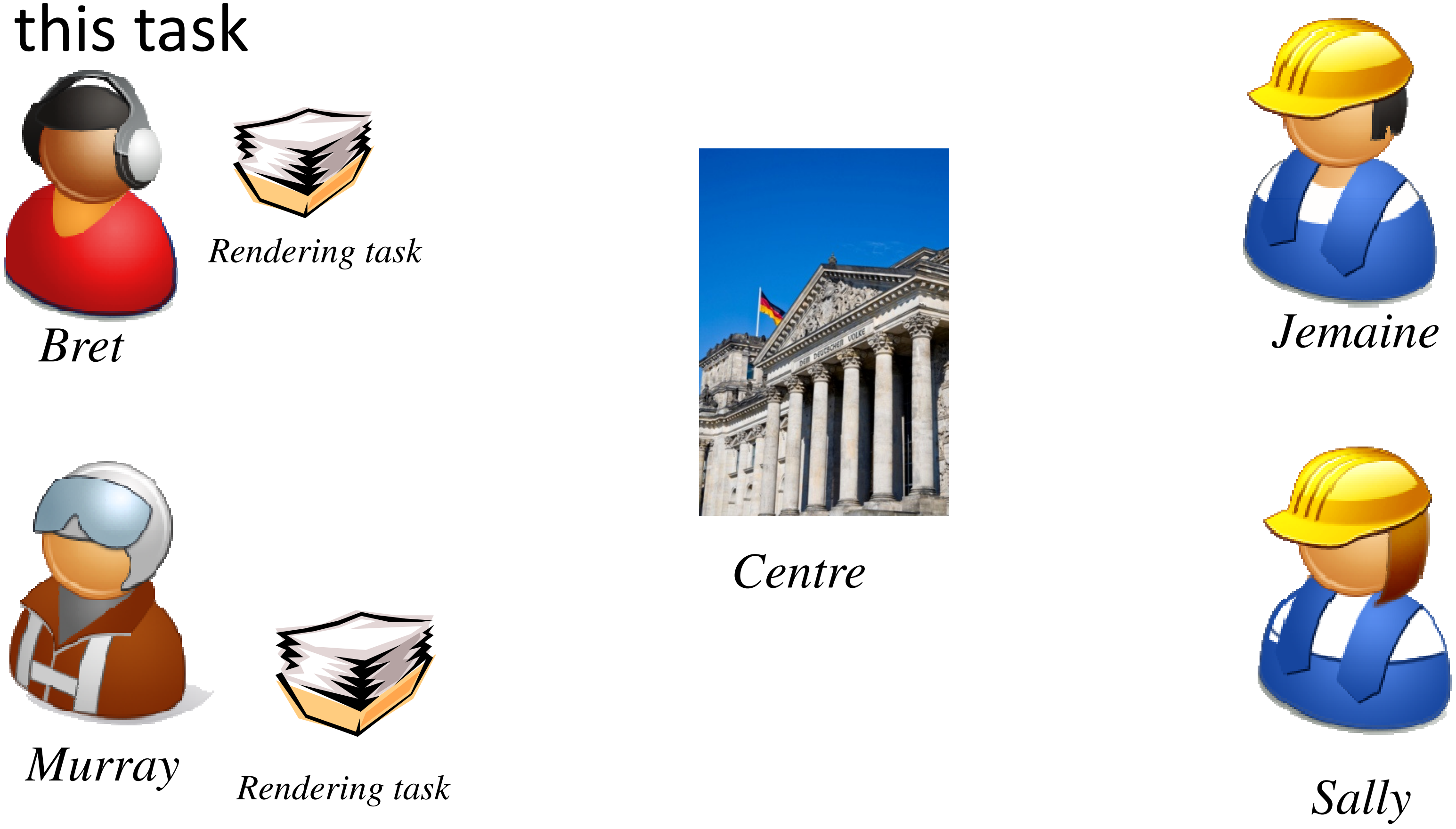
Robust auctions

- Common assumption: agents always successfully complete their allocated tasks.
- Unrealistic because agents can and often do fail.
- Agents have subjective perceptions about an agent's probability of succeeding at a given task (*trust*).

- **Challenge**: How to asess allocations that *prevent* failures exploiting trust information
  - Trust-based mechanisms (Ramchurn et al., JAIR 2009)

Task Allocation with Execution Uncertainty

- Consider a scenario where agents want to allocate a video editing task and the task performers may fail at this task



Bret

Rendering task

Jemaine

Centre

Murray

Rendering task

Sally

Trust-based task allocation

- In combinatorial exchanges, task requesters are indifferent to task performers.
- Now task requesters do care about task performers: valuations need to take into account the trust of the task requester in the task performer.
- The goal is to design a mechanism (allocation rule + payment function) that:
  - Has agents truthfully report on their expected QoS
  - Finds the task allocation that maximises the expected utility of the agents within the system (efficient allocation) instead of the revenue-maximising allocation.

Efficiency vs Revenue-maximisation

- Revenue-maximimising is a common goal in auctions.
- Efficiency (value-maximising allocations) is an alternative in resource allocation problems.

$$\arg\max_{a \in Allocations} \sum_{i \in I} v_i(a)$$

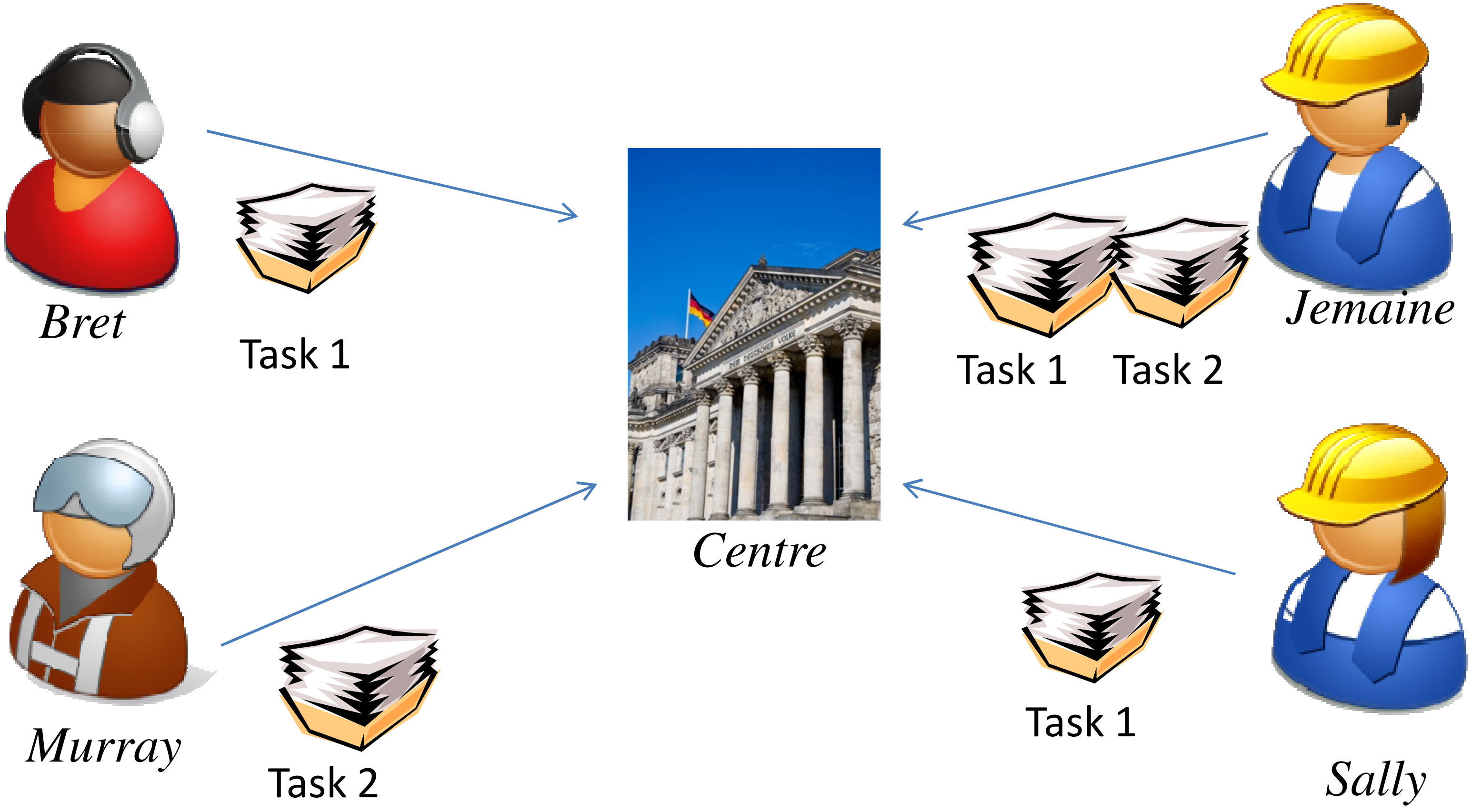
- Efficiency aims at allocating resources to those who value them most.
- From the perspective of a seller, a revenue-maximising allocation is a natural goal.
- From the perspective of the society as a whole, efficiency may be more important.

Efficiency vs Revenue-maximisation

- To make efficient choices agents must *truthfully* report their valuations.
- To provide *incentives* to report truthfully, it may be necessary to tax or subsidise various individuals.
- A *transfer (payment) function* sets utility adjustments so that individuals are taxed or subsidised depending on their reported valuations.
- Transfer functions are based on computing the marginal contribution of each agent to the mechanism (what if agent *i* wasn't there?).

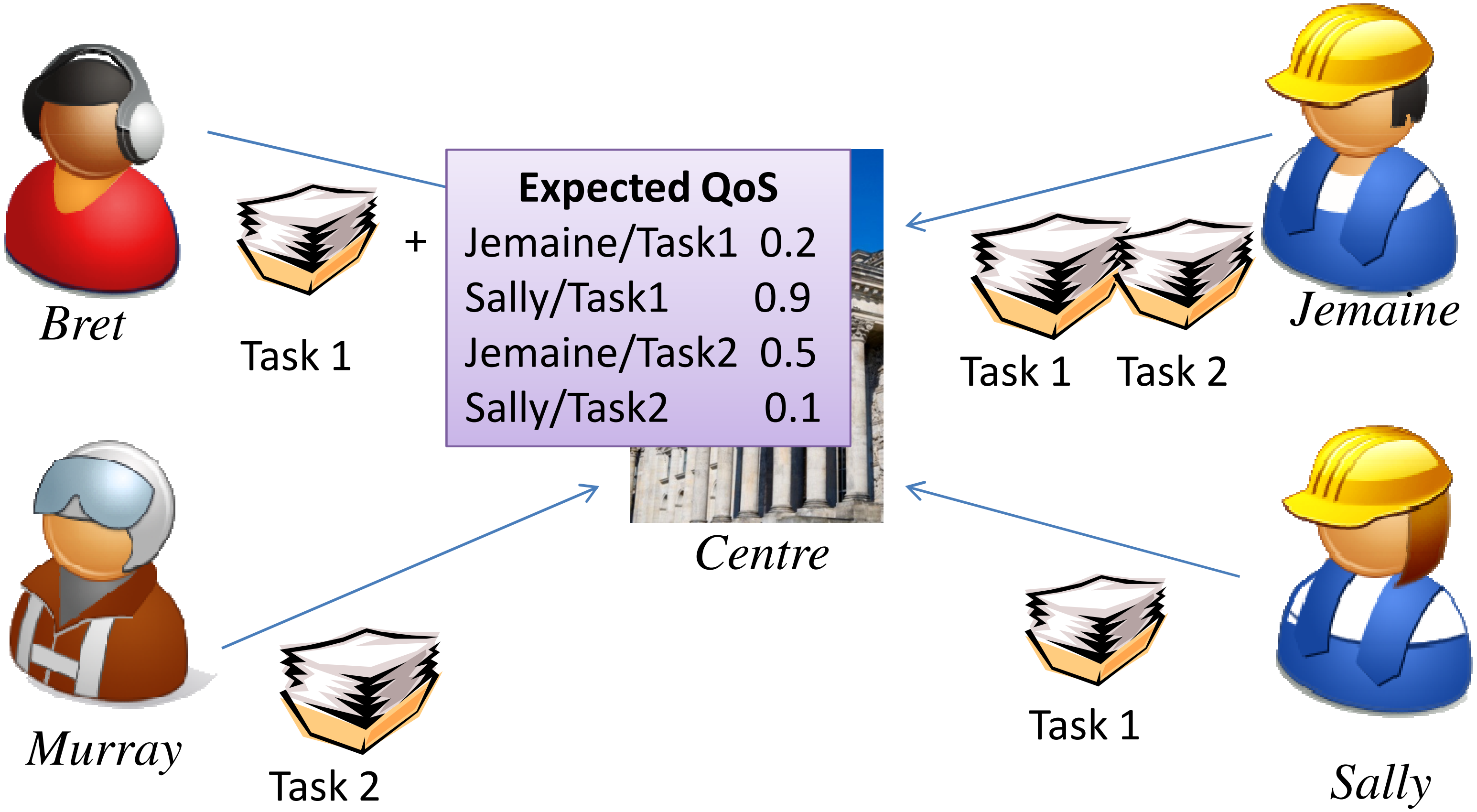
Trust-based task allocation

- Each ask: How much to pay to have some tasks performed **plus** how reliable the agent believes other agents are
- Each offer: How much to get paid to perform some tasks



Trust-based task allocation

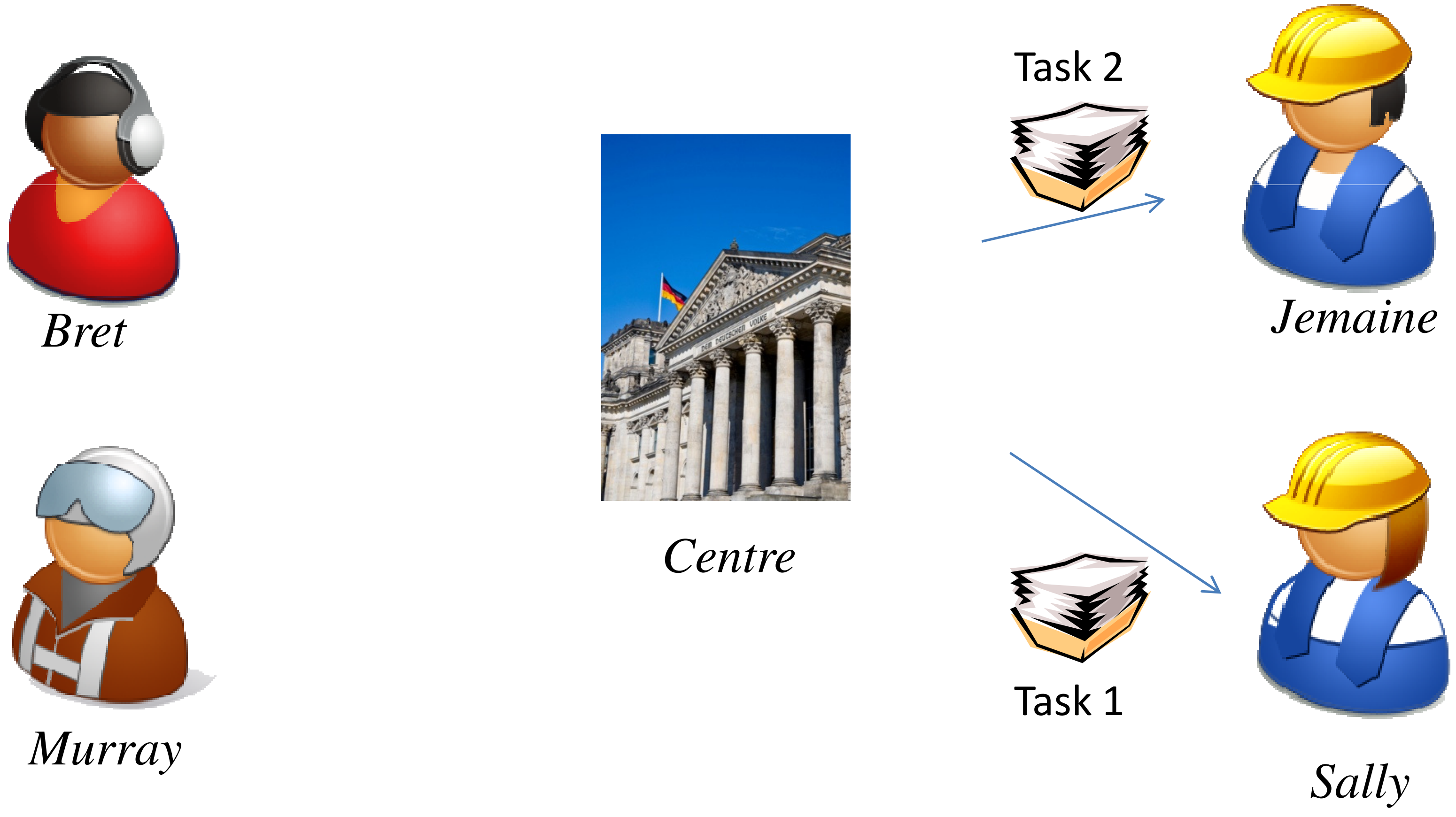
- Each ask: How much to pay to have some tasks performed plus how reliable the agent believes other agents are
- Each offer: How much to get paid to perform some tasks



Expected QoS	
Jemaine/Task1	0.2
Sally/Task1	0.9
Jemaine/Task2	0.5
Sally/Task2	0.1

Trust-based task allocation

- The Centre allocates the tasks and pays the agents depending on how they succeed at it.



Task 2

Bret

Jemaine

Centre

Murray

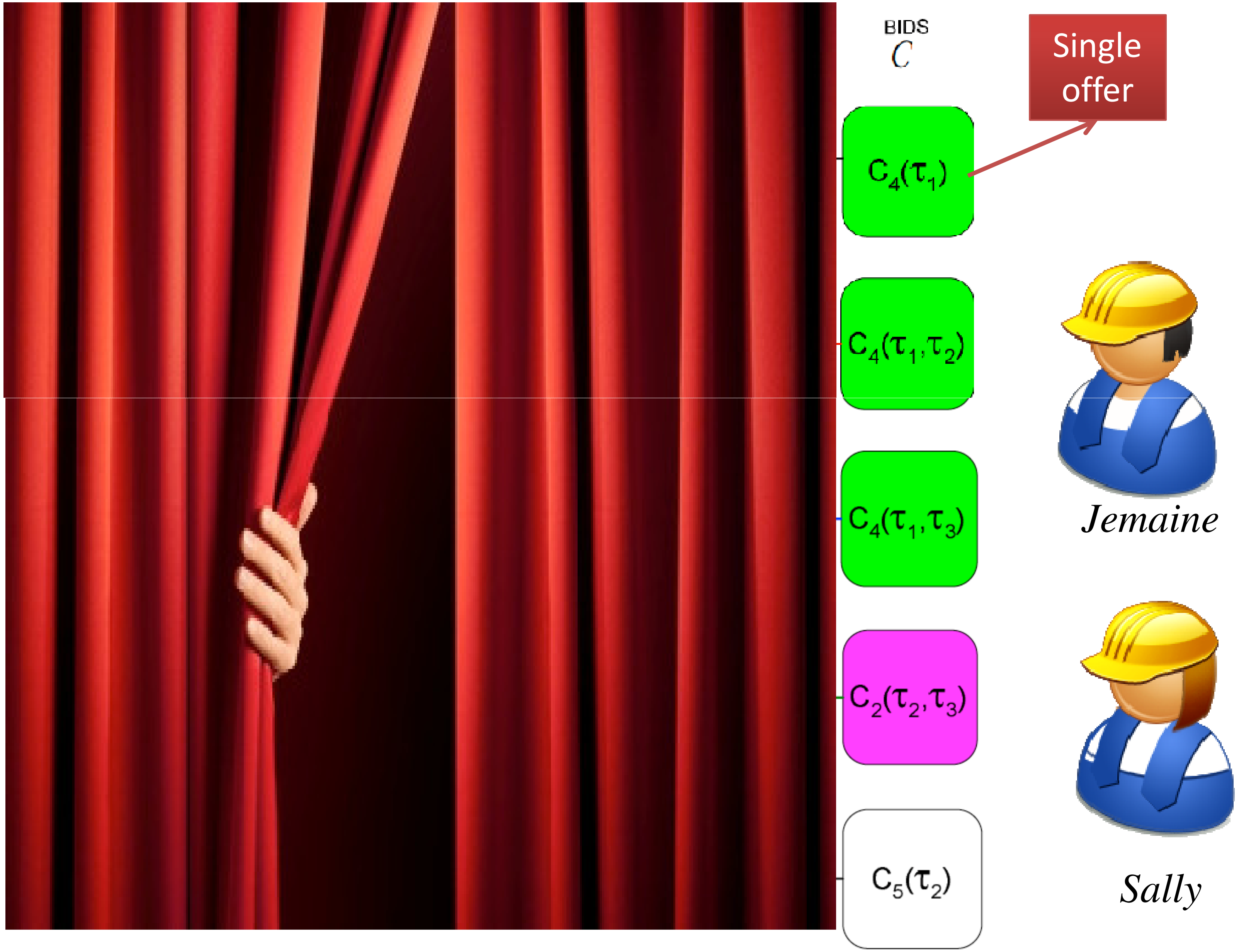
Task 1

Sally

Trust-based task allocation
Formal Model

- Tasks that can be requested or performed Υ
- Each agent $i \in I$ has:
  - A cost function $c_i(T)$ for the cost of performing tasks $T \subseteq \Upsilon$
  - A valuation function $v_i(T)$ over the tasks $T \subseteq \Upsilon$
  - A partial view about the probability of success of agent j when performing task $\tau \in \Upsilon$, noted as $\eta_i^j(\tau)$
- An agent type is $\theta_i = \{v_i, c_i, \eta_i\}$
- Trust of an agent i on another agent j as the aggregate expectation combining:
  - Previous personal interactions between i and j
  - Reputation in the society of j

Representing the space of allocations



BIDS

C

$C_4(\tau_1)$

$C_4(\tau_1,\tau_2)$

$C_4(\tau_1,\tau_3)$

$C_2(\tau_2,\tau_3)$

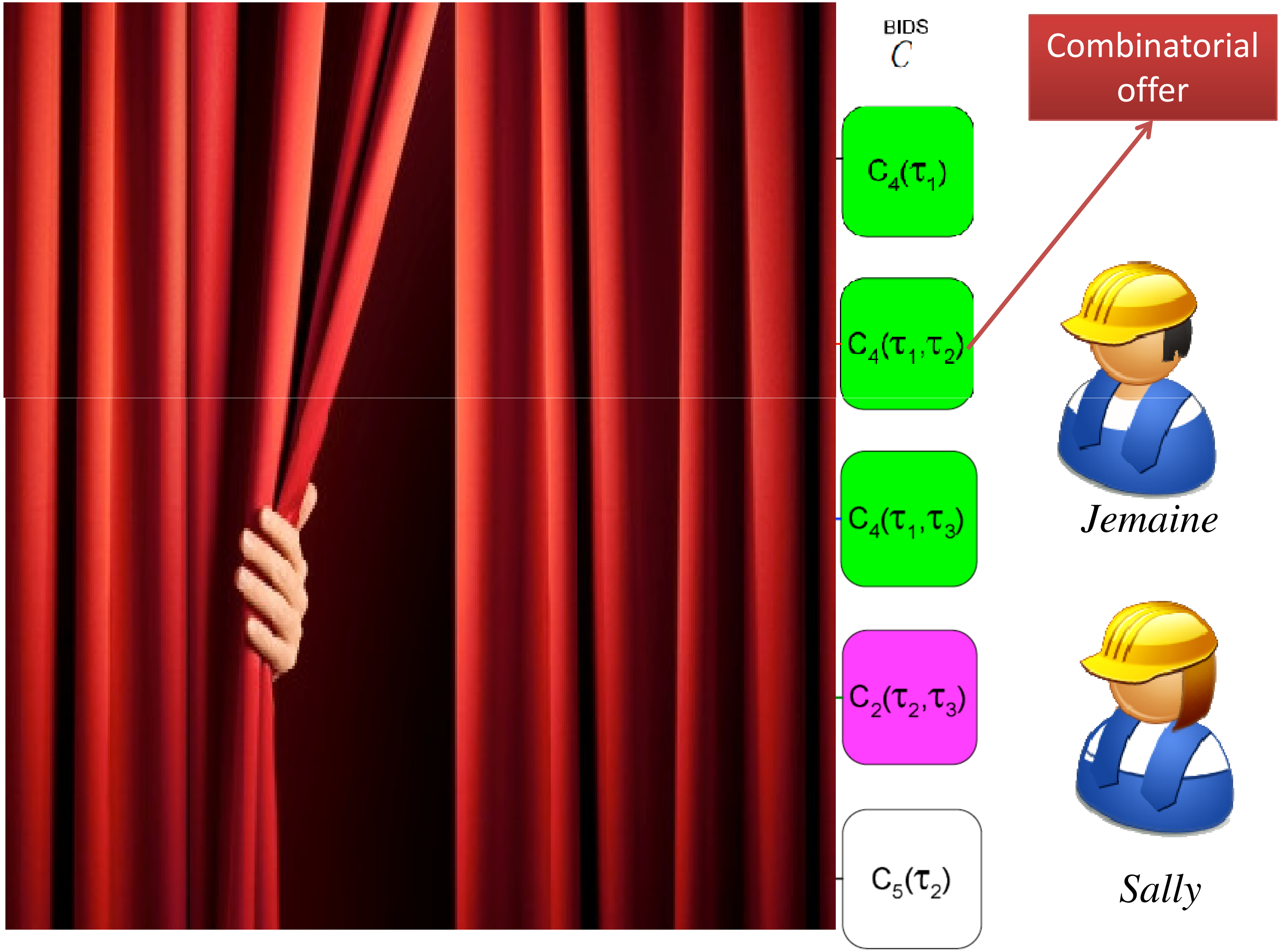
$C_5(\tau_2)$

Single offer

Jemaine

Sally

Representing the space of allocations



BIDS

C

$C_4(\tau_1)$

$C_4(\tau_1,\tau_2)$

$C_4(\tau_1,\tau_3)$

$C_2(\tau_2,\tau_3)$

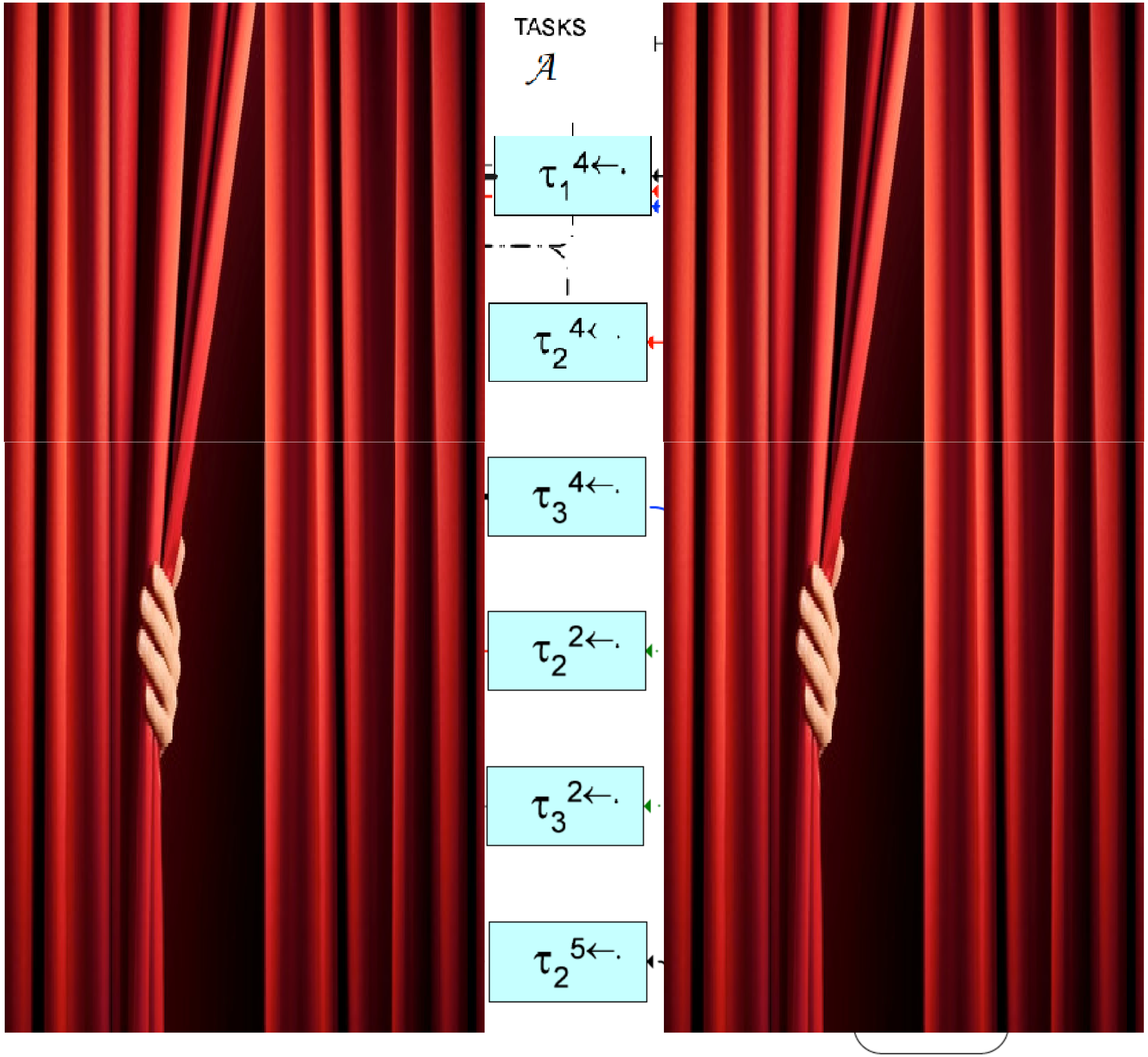
$C_5(\tau_2)$

Combinatorial offer

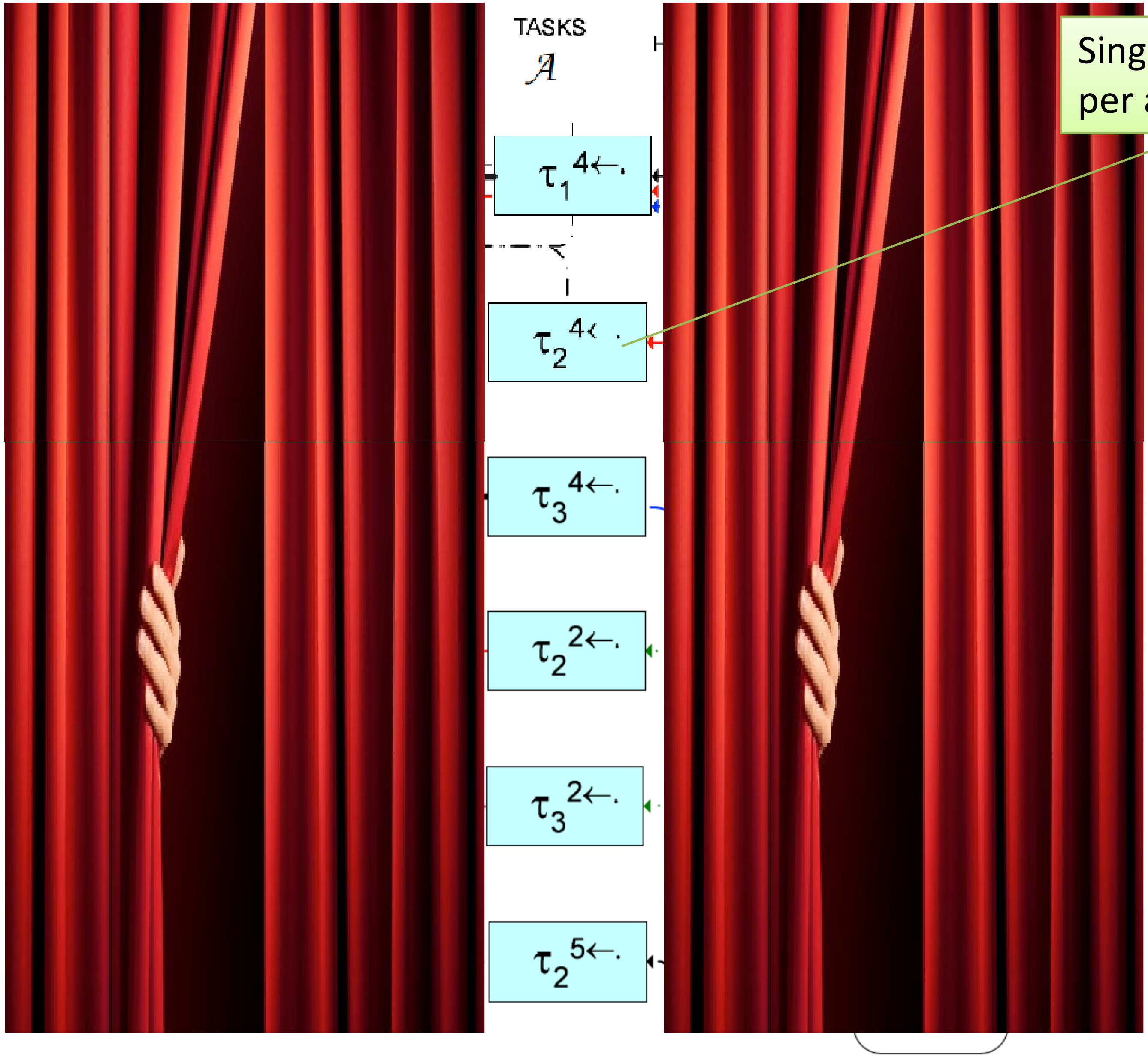
Jemaine

Sally

Representing the space of allocations

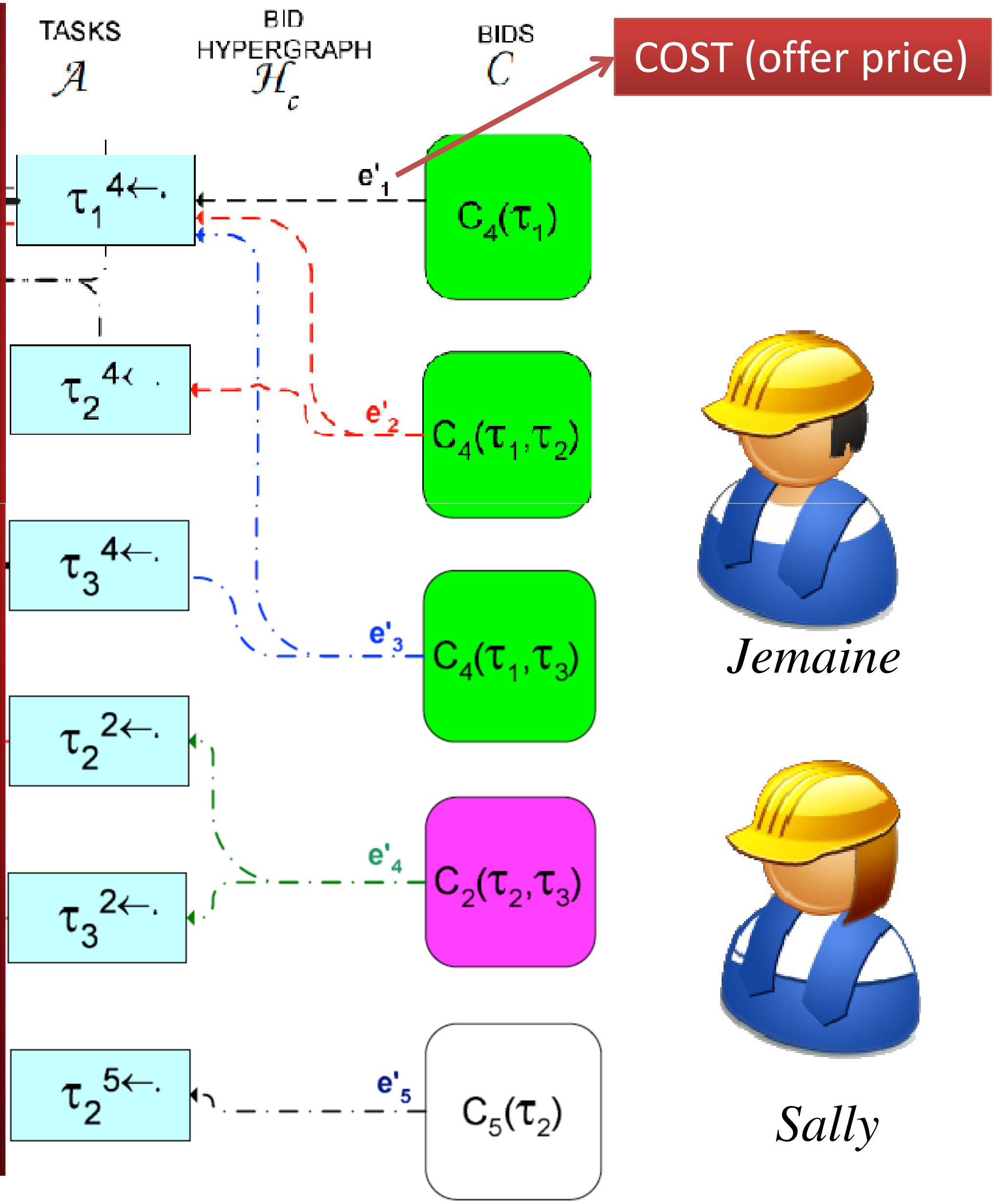


Representing the space of allocations



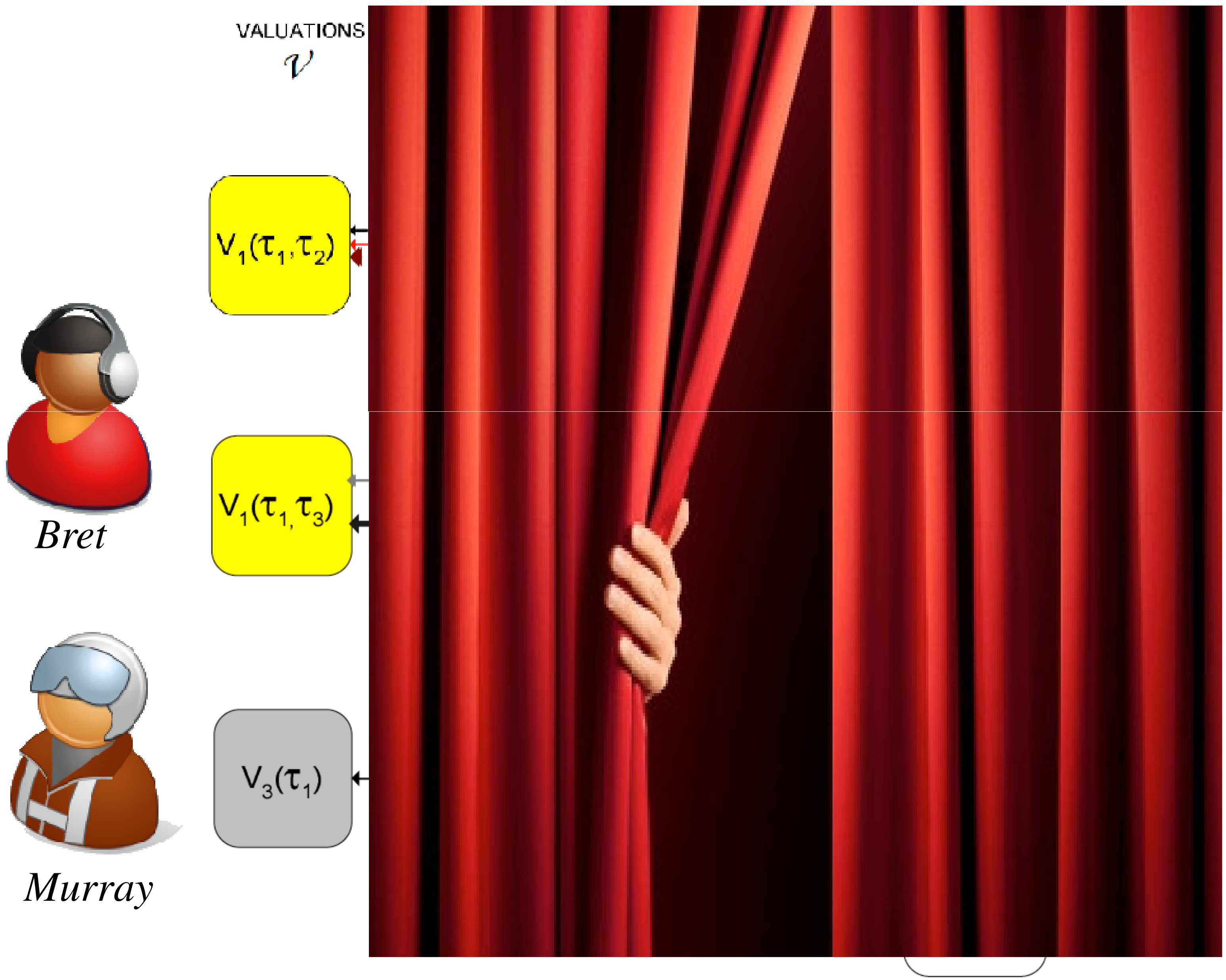
Single task per agent

Representing the space of allocations



COST (offer price)

Representing the space of allocations



VALUATIONS

$\mathcal{V}$

$V_1(\tau_1,\tau_2)$

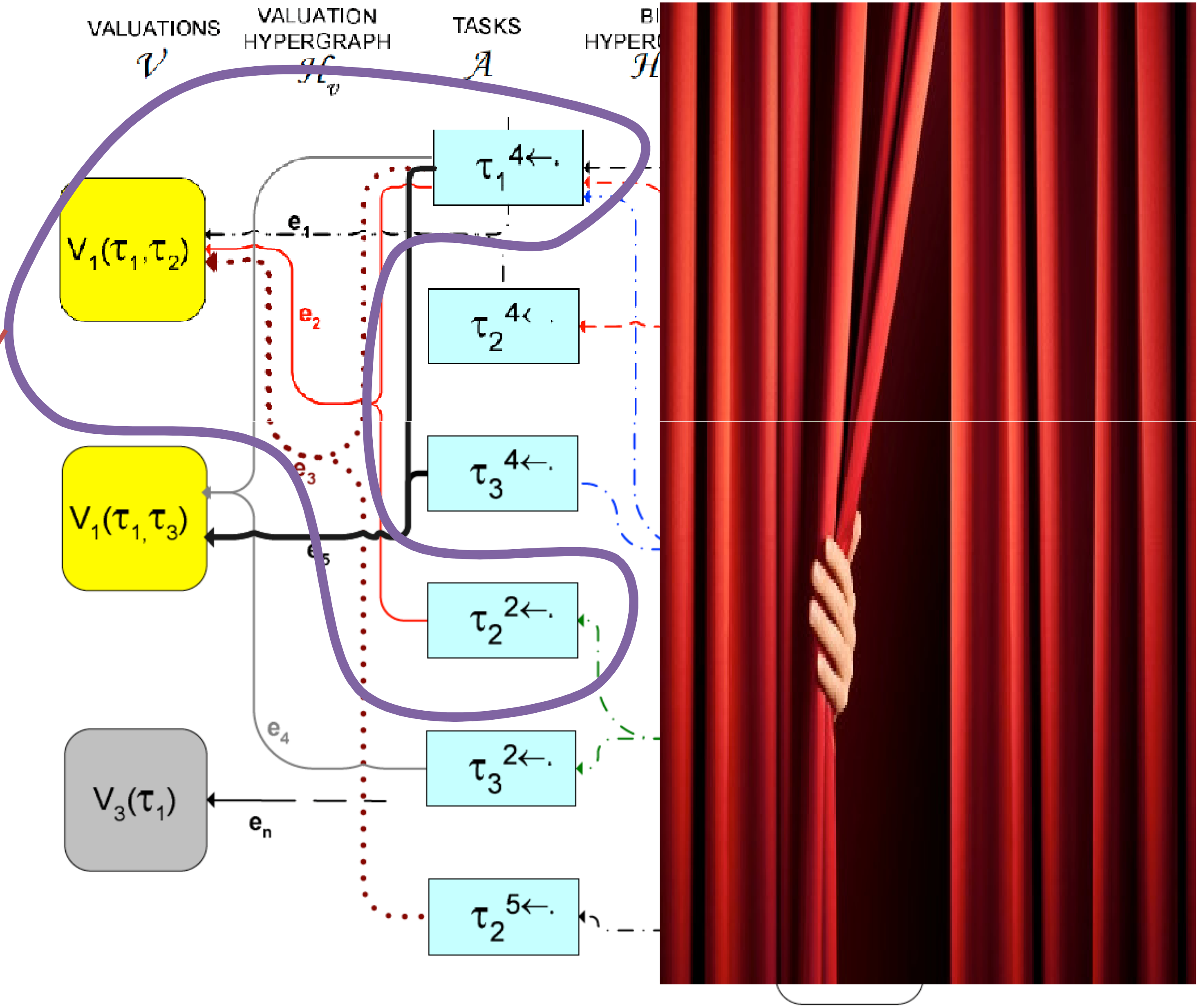
$V_1(\tau_1,\tau_3)$

$V_3(\tau_1)$

Bret

Murray

Representing the space of allocations



Expected valuation

Representing the space of allocations

- Expected valuation (example): The expected valuation associated to agent allocating task 1 to agent 4 and task 2 to agent 2 depends on the probability of success (POS) of agents 4 and 2 when performing task 1 and task 2 respectively.

Representing the space of allocations

- Expected valuation (example): The expected valuation associated to agent allocating task 1 to agent 4 and task 2 to agent 2 depends on the POS of agents 4 and 2 when performing task 1 ($p_4(\tau_1^{4\leftarrow 1})$) and task 2 ($p_2(\tau_2^{2\leftarrow 1})$) respectively.

$$\begin{aligned}\overline{v}_1(\tau_1^{4\leftarrow 1}, \tau_2^{2\leftarrow 1}) = {} & v_1(\tau_1, \tau_2) \cdot p_4(\tau_1^{4\leftarrow 1}) \cdot p_2(\tau_2^{2\leftarrow 1}) + \\ & v_1(\tau_1) \cdot p_4(\tau_1^{4\leftarrow 1}) \cdot (1 - p_2(\tau_2^{2\leftarrow 1})) + \\ & v_1(\tau_2) \cdot (1 - p_4(\tau_1^{4\leftarrow 1})) \cdot p_2(\tau_2^{2\leftarrow 1})\end{aligned}$$

Representing the space of allocations

- Expected valuation (example): The expected valuation associated to agent allocating task 1 to agent 4 and task 2 to agent 2 depends on the POS of agents 4 and 2 when performing task 1 ($p_4(\tau_1^{4\leftarrow 1})$) and task 2 ($p_2(\tau_2^{2\leftarrow 1})$) respectively.

$$\overline{v}_1(\tau_1^{4\leftarrow 1}, \tau_2^{2\leftarrow 1}) = v_1(\tau_1, \tau_2) \cdot p_4(\tau_1^{4\leftarrow 1}) \cdot p_2(\tau_2^{2\leftarrow 1}) +$$

Both agents succed

Representing the space of allocations

- Expected valuation (example): The expected valuation associated to agent allocating task 1 to agent 4 and task 2 to agent 2 depends on the POS of agents 4 and 2 when performing task 1 ($p_4(\tau_1^{4\leftarrow 1})$) and task 2 ($p_2(\tau_2^{2\leftarrow 1})$) respectively.

$$\overline{v}_1(\tau_1^{4\leftarrow 1}, \tau_2^{2\leftarrow 1}) =$$

Agent 4 succeeds

$$v_1(\tau_1) \cdot p_4(\tau_1^{4\leftarrow 1}) \cdot (1 - p_2(\tau_2^{2\leftarrow 1})) +$$

Agent 2 fails

Representing the space of allocations

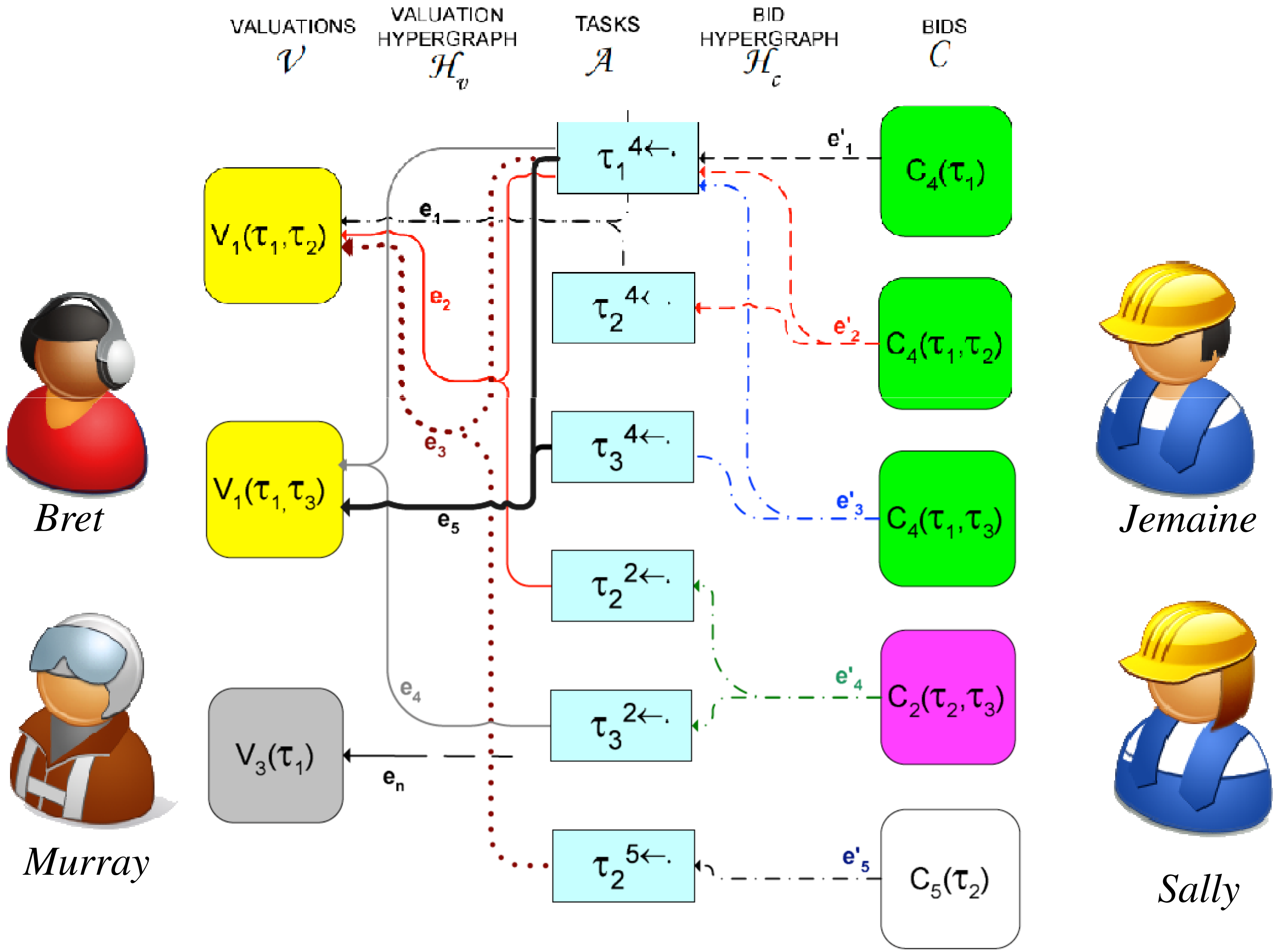
- Expected valuation (example): The expected valuation associated to agent allocating task 1 to agent 4 and task 2 to agent 2 depends on the POS of agents 4 and 2 when performing task 1 ($p_4(\tau_1^{4\leftarrow 1})$) and task 2 ($p_2(\tau_2^{2\leftarrow 1})$) respectively.

$$\overline{v}_1(\tau_1^{4\leftarrow 1}, \tau_2^{2\leftarrow 1}) =$$

Agent 2 succeeds
Agent 4 fails

$$v_1(\tau_2) \cdot (1 - p_4(\tau_1^{4\leftarrow 1})) \cdot p_2(\tau_2^{2\leftarrow 1})$$

Representing the space of allocations



Trust-based task allocation

- **Trust-based task allocation problem**: assessing the task allocation that maximises the expected utility of all agents within the system amounts to solving:

$$\arg\max_{a \in A} \; EU(a)$$

- where A stands for all feasible allocations and EU(a) stands for the expected utility of allocation a.
- It can be solved as an integer program with $\mathrm{O}(n)$ decision variables, where n is the number of expected valuations.

Integer Programming formulation

- Maximise $$\sum_{e\in\mathcal{E}^v} x_e \cdot w_v(e) - \sum_{e'\in\mathcal{E}^c} y_{e'} \cdot w_c(e')$$
  - where $x_e, y_{e'}$ are decision variables for valuations and costs
- subject to
  - Bids and valuations comply with the XOR bidding language

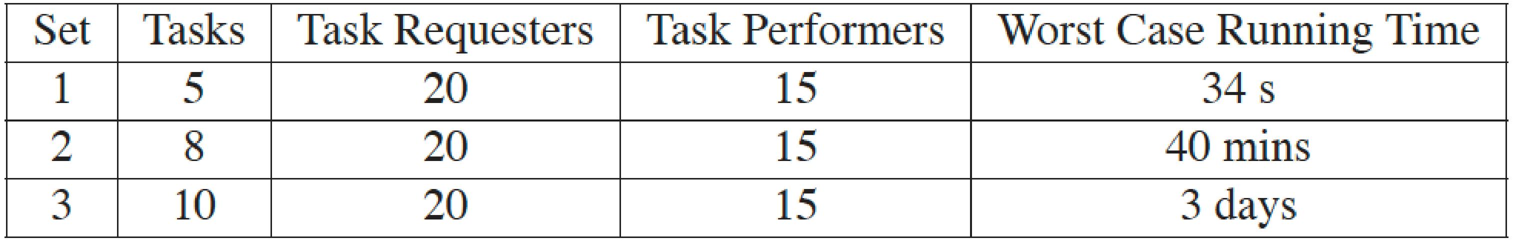
$$\sum_{e'\in\mathcal{E}_i^c} y_{e'} \leq 1 \quad \forall i \in \mathcal{I} \qquad \sum_{e\in\mathcal{E}_i^v} x_e \leq 1 \quad \forall i \in \mathcal{I}$$

  - Valuation on a task performed by a bidder is linked to the bidder performing the task (but demand dos not exceed supply)

$$\sum_{e\in\mathcal{E}^v,\tau_k^{j\leftarrow\cdot}\in e} x_e \leq \sum_{e'\in\mathcal{E}^c,\tau_k^{j\leftarrow\cdot}\in e'} y_{e'} \qquad \forall \tau_k^{j\leftarrow\cdot} \in \mathcal{A}$$

Worst-case empirical analysis

- *All* task performers bid over *all* tasks and *all* requesters submit a single valuation over *all* tasks.
- Small, medium-size scenarios (< $2x10^5$ variables) solvable in reasonable time (< 40 seconds).
- Time grows exponentially with the number of tasks.



Set	Tasks	Task Requesters	Task Performers	Worst Case Running Time
1	5	20	15	34 s
2	8	20	15	40 mins
3	10	20	15	3 days

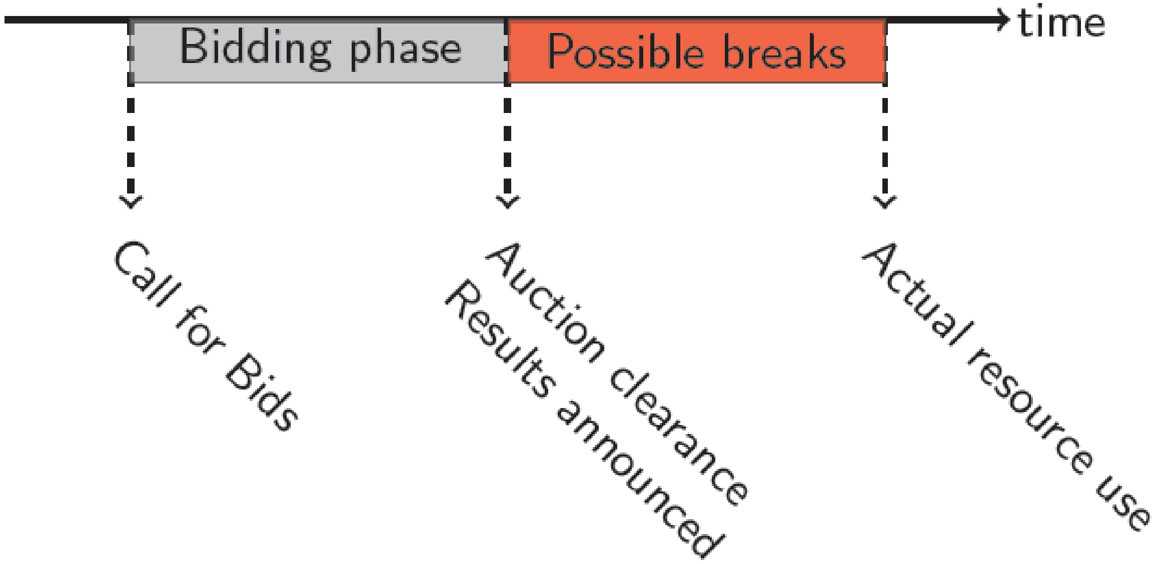
- Conclusion: Approximate algorithms required, but… withouth losing much efficiency!

Payment (utility transfer) function

- Payments are conditional on completion of tasks allocated.
- It is possible to define a payment function such that the resulting mechanism is:
  - Incentive compatible (agents truthfully report valuations, costs, expected QoS)
  - Efficient
  - Individually rational (agents prefer to participate in the mechanism)
- …but the computation of payments is costly! It requires to solve several optimisation problems.

Mending auctions [Muñoz 2011]

- Once an auction is cleared, there is a period of time where changes may occur



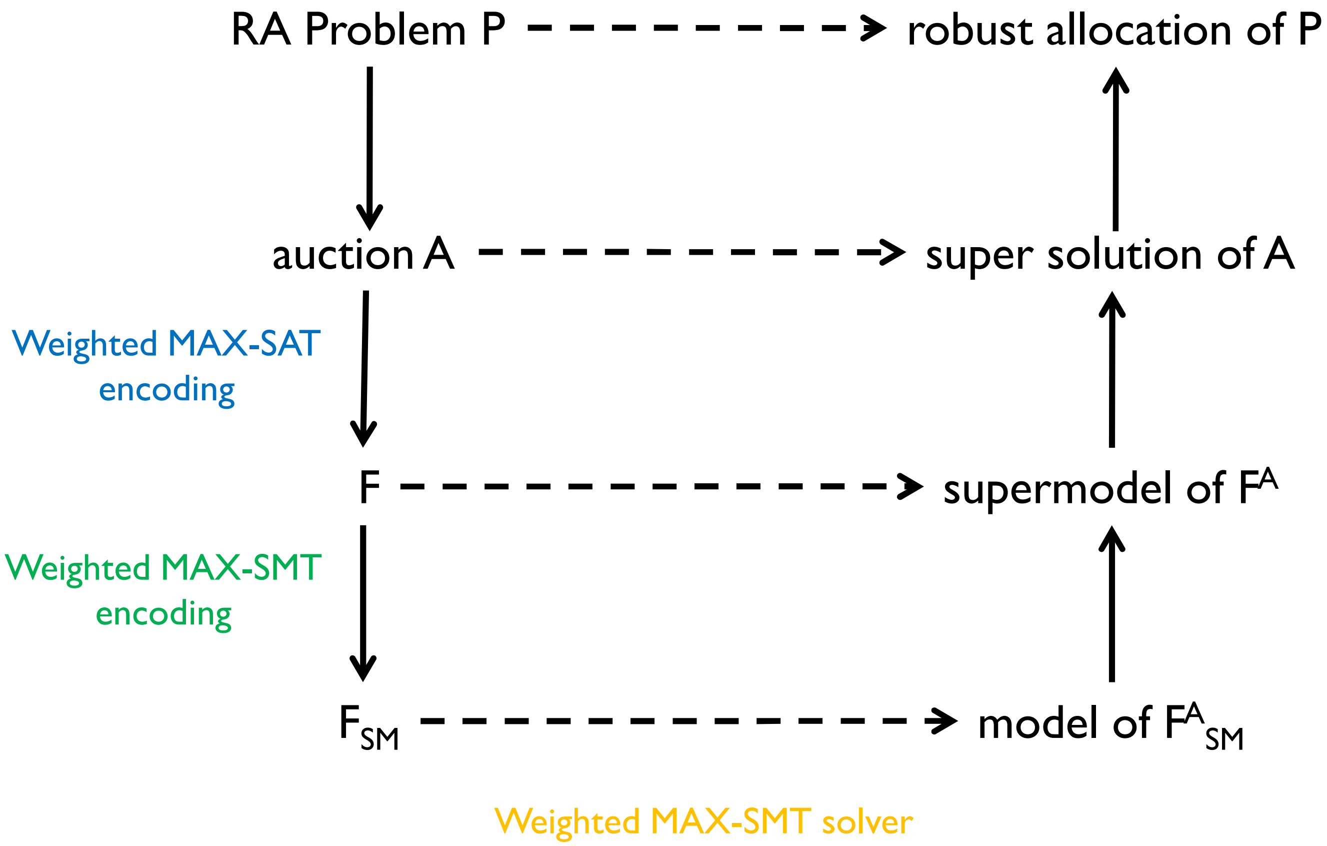
- Solving again the auction is not a good option
  - the bidders were informed about the winning allocation and a new auction could lead to a completely different allocation
- The intention is to minimize changes from the initial solution while losing as least revenue as possible

Robustness for CA

- **Definition of Robust Auction:**
  - **(a,b,β)-super solution**
  - *An (a,b,β)-super solution of an auction is a* ***maximal revenue*** *solution for the auction such that, if* ***at most a*** *goods become unavailable, then it can be repaired by changing* ***at most b*** *bids in the solution and, moreover, the* ***revenue of the solution and all possible repaired solutions is at least β****.*

Robustness for CA

- Schematic View



RA Problem P

robust allocation of P

auction A

super solution of A

Weighted MAX-SAT encoding

F

supermodel of F^A

Weighted MAX-SMT encoding

F_{SM}

model of $F^A{}_{SM}$

Weighted MAX-SMT solver

SMT = SAT Modulo Theories

Boolean satisfiability

- SAT
– Determine if a Boolean formula is satisfiable
- MAX-SAT
– The maximum number of clauses that can be satisfied by some assignment
- Weighted MAX-SAT
– The maximum weight which can be satisfied by any assignment
- Partial Weighted MAX-SAT
– Idem, with some non-weighted clauses

$F = (p \vee q) \wedge (\neg p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg q, 10)$

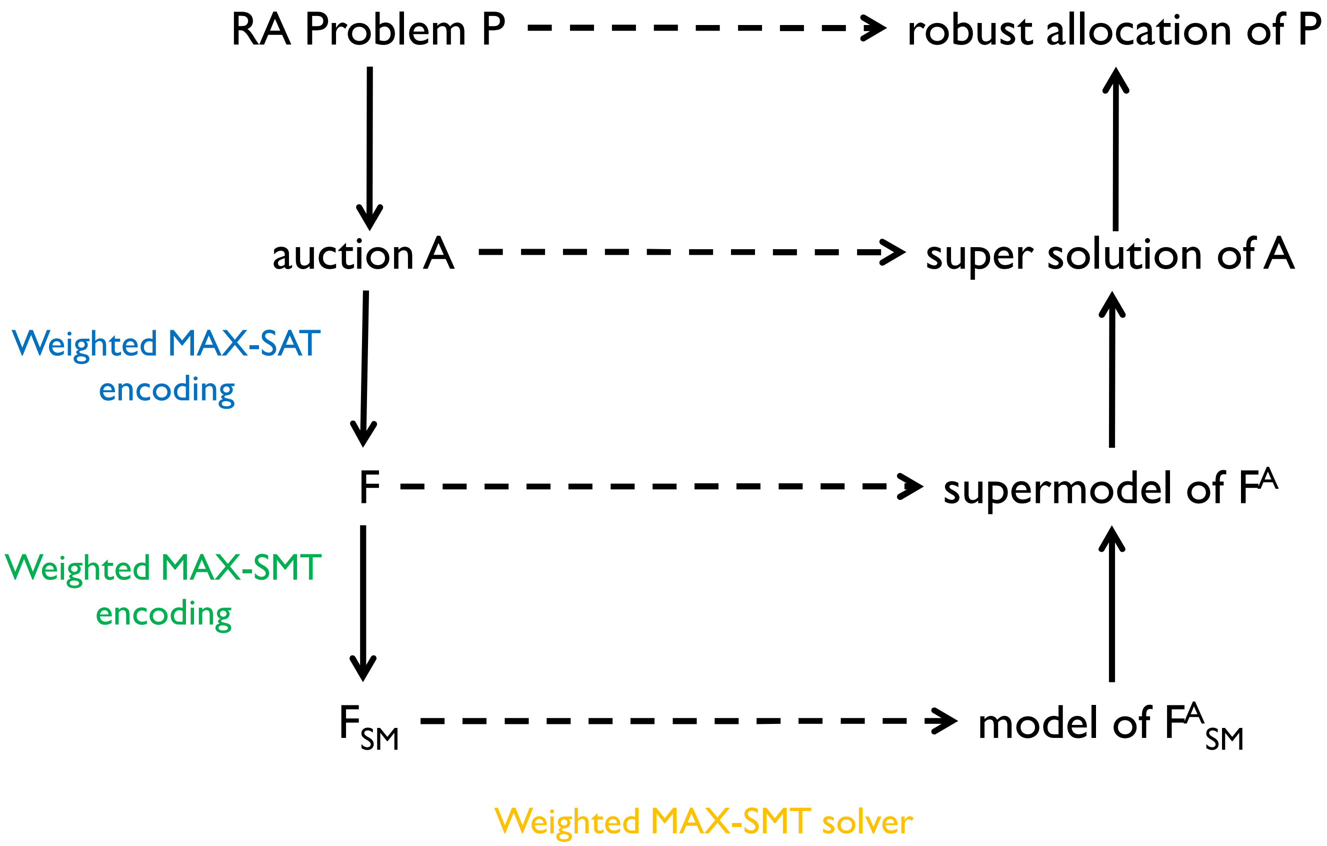
Result: 0 (p=true, q=true)

SAT Modulo theories

- SMT
  - Inclusion of arithmetic operations
  - $F = (p \vee q) \wedge (q \vee (y = 2)) \wedge (p \vee (x = 2)) \wedge (x + y < 4)$
- MAX-SMT
- Weighted MAX-SMT
- Partial Weighted MAX-SMT

Robustness for CA

- Schematic View



RA Problem P

robust allocation of P

auction A

super solution of A

Weighted MAX-SAT encoding

F

supermodel of F^A

Weighted MAX-SMT encoding

F_{SM}

model of $F^A{}_{SM}$

Weighted MAX-SMT solver

SMT = SAT Modulo Theories

Encoding

- Partial weighted MAX-SAT encodes CA WDP
  - F = M $\wedge$ W
  - Mandatory clauses (M)
    - Incompatibilities between bids for bids offering same goods (e.g. ($\neg B_2 \vee \neg B_4$) if B_2 offers ($g_{3,}g_5$) and B_4 offers (g_1,g_3)
    - Resource requirements (e.g $B_2 \rightarrow g_3 \wedge g_5$)
  - Weighted clauses (W)
    - Bids' prices (e.g. (B_2, 100))
- Partial weighted MAX-SMT encodes robust WDP
  - F_{SM} = F (CA WDP clauses) ^

    clause constraining solution value($\sum_{i=1}^{r} B_i \geq \beta$) ^

    repairing clauses (a,b)

Research opportunities

Research opportunities

- Coping with time
  - Scheduling tasks in a supply chain [Collins 2008]
  - Time-aware bidding language &WDP for mixed auctions [Witzel 2010]
- Coping with uncertainty
  - Preventing failures
  - Mending failed (task) allocations
- Coping with distribution
  - Scalable, distributed supply chain formation

References

[Almajano 2010] Pablo Almajano, J. Cerquides, J. Rodriguez-Aguilar (2010) Empirical hardness for mixed auctions, 161–170. In *Lecture Notes in Computer Science 5988*.

[Cerquides 2007] Jesús Cerquides, Ulle Endriss, Andrea Giovannucci, Juan A. Rodríguez-Aguilar: Bidding Languages and Winner Determination for Mixed Multi-unit Combinatorial Auctions. IJCAI 2007: 1221-1226

[Collins 2008] John Collins, Maria Gini. Scheduling tasks using combinatorial auctions: the MAGNET approach. In Gedas Adomavicius and Alok Gupta, editors, Handbooks in Information Systems Series: Business Computing, Elsevier, 2008.

[Cramton 2006] Peter Cramton, Yoav Shoham, and Richard Steinberg. *Combinatorial Auctions*. Cambridge, MA: MIT Press, 2006 Chapters 9,12,18,19,21,23.

[Fionda & Greco 2009] Valeria Fionda and Gianluigi Greco. Charting the tractability frontier of mixed multi-unit combinatorial auctions. In *Proceedings of the 21st international jont conference on Artifical intelligence* (IJCAI'09),

[Giovannucci 2008] Andrea Giovannucci. Computationally Manageable Combinatorial Auctions for Supply Chain Automation, PhD Thesis, Universitat Autònoma de Barcelona, 2008.

[Leyton-Brown 2006] A Test Suite for Combinatorial Auctions. K. Leyton-Brown, Y. Shoham. Chapter 18 in *Combinatorial Auctions*, P. Cramton, Y. Shoham, R. Steinberg (eds.), MIT Press, 2006.

[Mikhaylov 2011] Boris Mikhaylov, Jesús Cerquides, Juan A. Rodríguez-Aguilar. Sequential Mixed Auctions. To appear in IEEE International Conference on Electronic Commerce (ICEC) 2011.

[Muñoz 2011] Víctor Muñoz, 2011. Robustness on resource allocation problems. PhD thesis. University of Girona.

[Ramchurn 2009] Ramchurn, S. D., Mezzetti, C., Giovannucci, A., Rodriguez, J. A., Dash, R. K. and Jennings, N. R. Trust-Based. Mechanisms for Robust and Efficient Task Allocation in the Presence of Execution Uncertainty*. Journal of Artificial Intelligence Research*, 35 . pp. 119-159, 2009

[Sheffi 2004] Yossi Sheffi. Combinatorial Auctions in the Procurement of Transportation Services. *Interfaces* 34, 4 (August 2004), 245-252.

[Shoham & Leyton-Brown 2009] Y. Shoham, K. Leyton-Brown*. Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundation*s. Cambridge University Press, 2009.

[Vinyals 2008]) M.Vinyals, A.Giovannucci, J.Cerquides , P. Meseger, J.A. Rodriguez-Aguilar. A Test Suite for the Evaluation of Mixed Multi-Unit Combinatorial Auctions. *Journal of Algorithms*, Volume 63, Issue 1-3, pp. 130-150, 2008

[Walsh 2000] William E. Walsh, Michael P. Wellman, and Fredrik Ygge. Combinatorial auctions for supply chain formation. In *Proceedings of the 2nd ACM conference on Electronic commerce* (EC '00). ACM, New York, NY, USA, 260-269

[Winsper & Chli 2010] Michael Winsper and Maria Chli. 2010. Decentralised Supply Chain Formation: A Belief Propagation-based Approach. In *Proceeding of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence.*

[Witzel 2010] Andreas Witzel , Ulle Endriss. Time Constraints in Mixed Multi-unit Combinatorial Auctions. In *AMEC at AAMAS 2010.*

Backup

Linear programs

- A linear program is defined by:
  - A set of real-valued variables;
  - A linear objective function; and
  - A set of linear constraints.
- Let the set of variables be $\{x_1, \ldots, x_n\}$

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} w_i x_i \\ \text{subject to} & \sum_{i=1}^{n} a_{ij} x_i \le b_j \end{array} \quad \Longleftrightarrow \quad \begin{array}{ll} \text{maximize} & \mathbf{w}^T \mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \le \mathbf{b} \end{array}$$

- When there is no objective function it reduces to a constraint satisfaction or feasibility problem

Solving linear programs

- Linear programs can be solved in polynomial time through interior point methods
- The simplex algorithm
  - requires an exponential number of steps in the worst case…
  - …but is usually much more efficient that interior point methods in practice

Integer programs

- Integer programs are linear programs in which the variables are required to take integral (rather than real) values.
- In 0-1 integer programs each variable is constrained to take either the value 0 or the value 1.

$$\begin{array}{lll} \text{maximize} & \mathbf{w}^T\mathbf{x} & \\ \text{subject to} & \mathbf{A}\mathbf{x} \leq \mathbf{b} & \\ & x_i \in \{0,1\} & \forall i, 1 \leq i \leq n \end{array}$$

- Mixed integer programs involve a combination of integer and real-valued variables.

Solving integer programs

- Integer programs are undecidable in the worst case and NP-hard provided that the domains are finite.
- The most commonly used technique is branch and bound search.
  - At each search node the linear program relaxation of the integer program is solved
- Other techniques are branch-and-price or branch-and-cut.
- Provided that
  - A is totally unimodular (every square submatrix has determinant 0, 1 or -1); and
  - b is integral

integer programs can be solved in polynomial time.

Optimisation in Multi-Agent Systems

Coalition Formation

Sarvapali D. (Gopal) Ramchurn

Agents, Interaction, Complexity
School of Electronics and Computer Science,
University of Southampton

At the end of this talk, you will be able to:

1. Define what is coalition formation
   - Differences from and links to teamwork, decentralised coordination and supply-chain formation.
2. List the key steps of abstract coalition formation
   - Cooperative and Competitive settings
   - Computational challenges
   - Lack of realism
3. Define complex coalition formation applications
   - Dynamic CF: Scheduling and Routing
   - Uncertainty and Dynamism

Contents

- Part 1: An Overview of Coalition Formation
  - Differences and Similarities with other coordination technologies
  - Competitive and Cooperative settings
- Part 1: Abstract Coalition Formation
  - Steps of the CF process
  - Computational challenges
  - Overlapping Coalitions
- Part 2: Complex Coalition Formation
  - A scenario
  - Solutions
  - Other examples

Part 1

An overview of coalition formation

A definition of coalition formation

The coming together of a set of, possibly self-interested, agents that cooperate to perform joint actions as part of a group where the reward form performing such actions is attributed to the group.

A definition of coalition formation

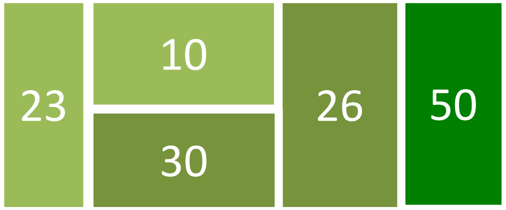
The coming together of a set of, possibly self-interested, agents that cooperate to perform joint actions as part of a group where the reward from performing such actions is attributed to the group.

Coalition Formation and other coordination paradigms

- Are Teams and Coalitions the same?
  - Coalition members may only care about their own profit.
  - Coalition Formation research focuses on who to group for best synergies and how to share rewards/resources
  - Teamwork: focus is on roles/team operation

Coalition Formation and other coordination paradigms

- Finding the best coalition = running a combinatorial auction?
  - Combinations of goods/services = combinations of agents?
  - CAs = set packing problem (find the sets of goods that minimise cost)
  - CF = can be a set partitioning problem (allocate **ALL** agents to a set)



There are diverse cooperative and competitive scenarios

- Cooperative
  - Team formation
  - Resource Allocation
  - Task Allocation
  - Dynamic Coalition Formation
- Competitive (i.e. selfish agents)
  - Cooperatives
  - Consortium
  - Buyers and Sellers

In cooperative settings we focus on optimisation

- Optimise the performance of a coalition
  - Resources or Tasks
  - Intra Group Synergies

- Find the best set of coalitions to form
  - Group Externalities
  - Coalition Values

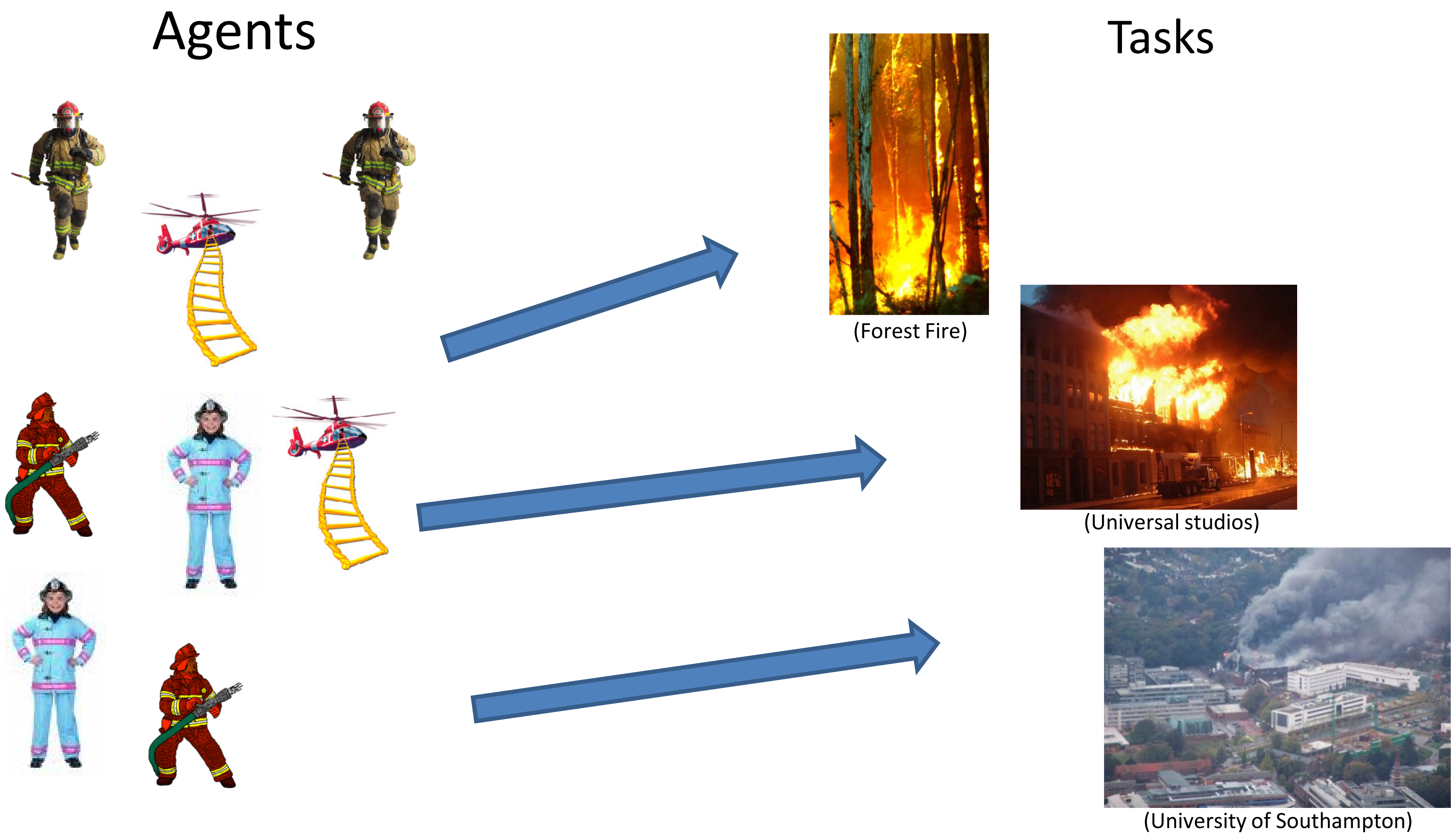
In competitive settings we focus on stability

- Maintaining the grand coalition
  - Assume it's the best coalition to form (super-additivity)
  - Game theory
- Avoiding coalitions from disbanding through payments
  - Payments within the coalition
  - Payments between coalitions
- BUT: do such payments exist? -> worst optimisation problem (GT typically ignores it…).

Part 2

Abstract Coalition Formation

Coalition Formation



Agents

Tasks

(Forest Fire)

(Universal studios)

(University of Southampton)

Coalition Formation

(Forest Fire)

v(Coalition 1) = 10

(Universal studios)

v(Coalition 2) = 14

v(Coalition 4) = 14

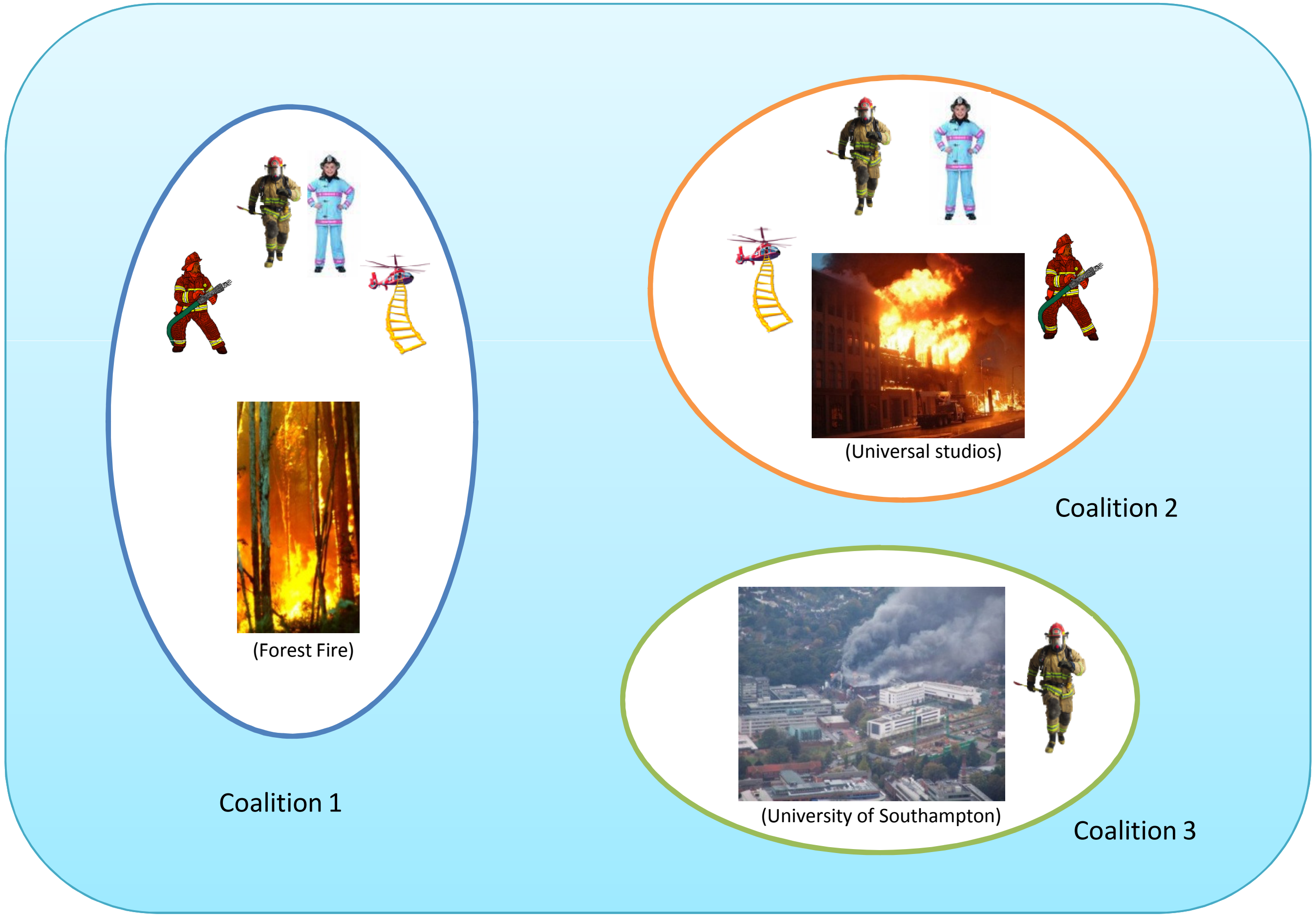
(University of Southampton)

v(Coalition 3) = 1

v(Coalition 3) = 5

Coalition Structure

$V(CS) = 10 + 14 + 1 = 25$



(Forest Fire)

Coalition 1

(Universal studios)

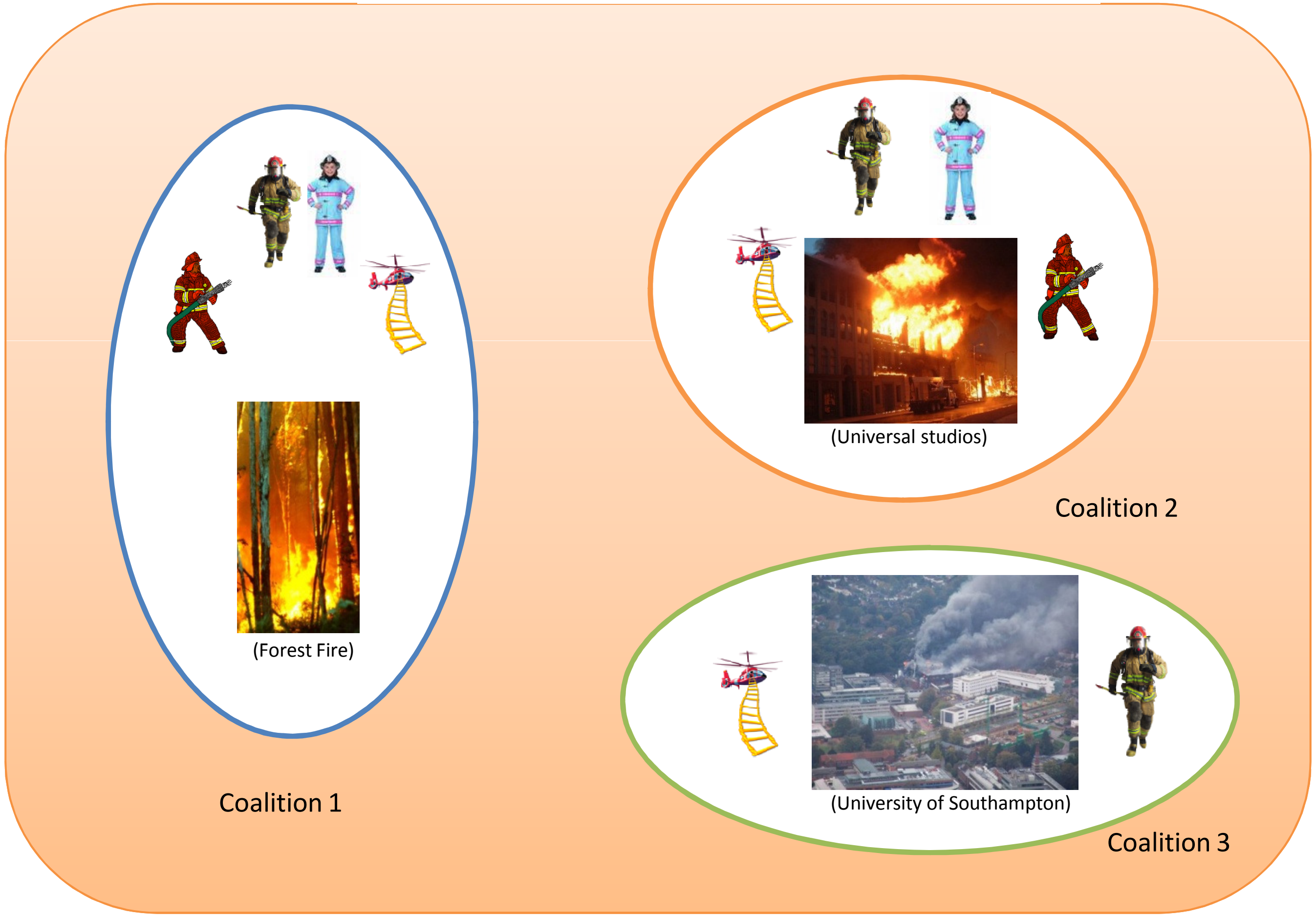
Coalition 2

(University of Southampton)

Coalition 3

Coalition Structure

V(CS) = 10 + 14 + 5 = 26



(Forest Fire)

Coalition 1

(Universal studios)

Coalition 2

(University of Southampton)

Coalition 3

Coalition Formation Stages

- Coalition Value Calculation and Optimisation
  - Intra-coalition task/resource allocation
  - Evaluating a coalition's effectiveness
  - In Competitive settings?
- Payoff Calculation
  - Stability of the coalition
  - Network games
- Coalition Structure Generation
  - The set of sets of agents i.e., the partition of the set of agents
  - Constraints
  - Winner Determination problem
  - Dynamic? (wait for PART II)

Coalition Formation Stages

- Coalition Value Calculation and Optimisation
  - Intra-coalition task/resource allocation
  - Evaluating a coalition’s effectiveness
  - In Competitive settings?
- Payoff Calculation
  - Stability of the coalition
  - Network games
- Coalition Structure Generation
  - The set of sets of agents i.e., the partition of the set of agents
  - Constraints
  - Winner Determination problem
  - Dynamic? (wait for PART II)

Coalition Value Calculation

- Computational Cost: 2^n where n = no. of agents.
- Standard Enumeration techniques:
  1. Build a forest
  2. Share coalition values computed (I'll compute the ones I can form and you do the same and we'll share) – i.e., Shehory et al.

  Problems: redundancy – compute coalitions twice, communication cost if to be distributed.

Coalition Value Calculation

- Wilf and Nijenhuis, 1989, DCVC (Rahwan&Jennings, 2008)
  - The last coalition in the list is: $\{1, \ldots, s\}$ (s = size of the coalition) e.g. last coalition of size 4 out of 6 agents is, 1,2,3,4.
  - Given any coalition C_i , the agent can calculate C_{i-1} by checking the values $c_{i,s}, c_{i,s-1}, c_{i,s-2}, \ldots$ until it finds a value $c_{i,x}$ such that
    - $c_{i,x} < c_{1,x}$, then:
      - $c_{i-1},k = c_i,$ k: 1 k <x
      - $c_{i-1},k = c_i,$ k +1: $k = x$
      - $c_{i-1},k = c_{i-1},$ k−1 +1: x <k <s
- Key point is: Generate the next coalition from the previous one with minimal changes.
- Distributed indices = distributed computation.

Coalition Formation Stages

- Coalition Value Calculation and Optimisation
  - Intra-coalition task/resource allocation
  - Evaluating a coalition's effectiveness
  - In Competitive settings?
- Payoff Calculation
  - Stability of the coalition
  - Network games
- Coalition Structure Generation
  - The set of sets of agents i.e., the partition of the set of agents
  - Constraints
  - Winner Determination problem
  - Dynamic? (wait for PART II)

Fairness and Coalitional Stability

- Let’s play a game
  - Each participant gets X in coalition Y
  - Participants have to choose which coalitions to join
  - How do we find the coalition we want to be in.
- Shapley Value
  - Fairness – agents obtain the reward they deserve
  - NP-complete
- Core
  - Stability – agents obtain the reward which avoids the coalition breaking up
  - NP-complete

Algorithms

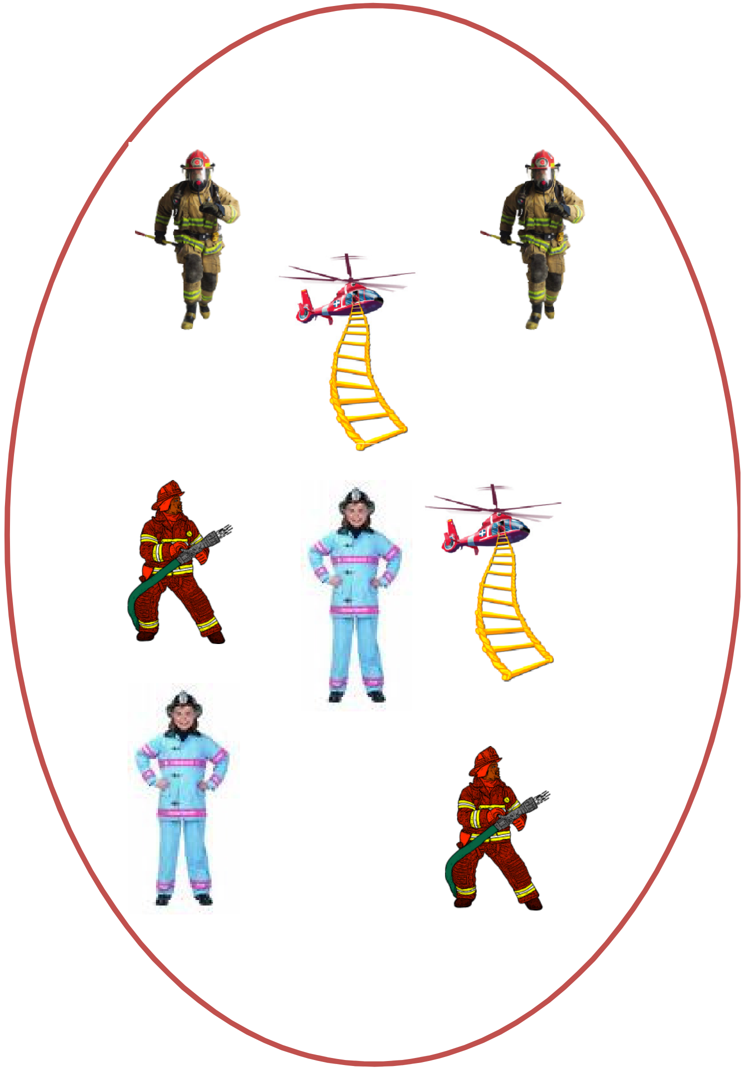
- Conitzer 2004
  - Core and Shapley value using decomposable characteristic functions
- Castro et al. 2008
  - Using sampling process on the ordering of agents' inserted into coalitions
- Dreschel and Kimms, 2010
  - Core computation using mathematical programming (150 agents).
- Issues: How are coalition values obtained? Manipulation issues? Who does the computation? Rationality assumptions? Approximate solutions?

Coalition Formation Stages

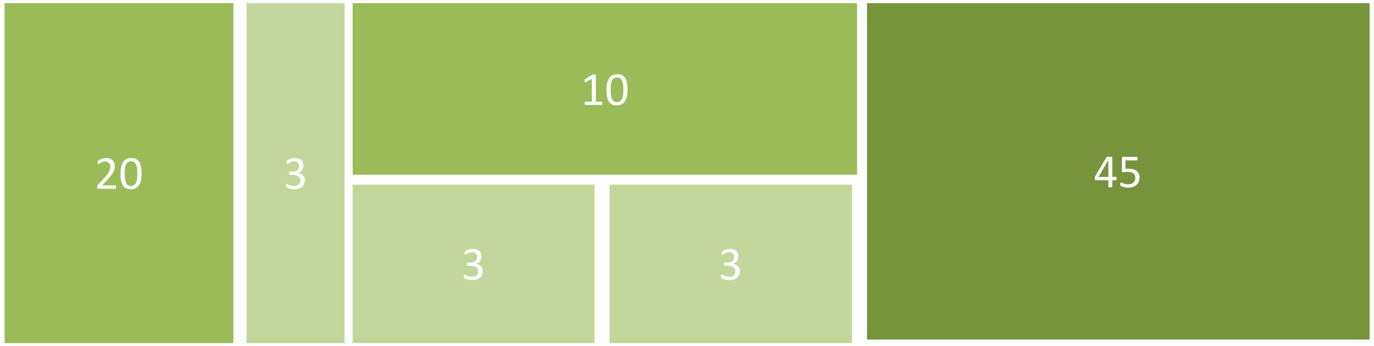
- Coalition Value Calculation and Optimisation
  - Intra-coalition task/resource allocation
  - Evaluating a coalition's effectiveness
  - In Competitive settings?
- Payoff Calculation
  - Stability of the coalition
  - Network games
- Coalition Structure Generation
  - The set of sets of agents i.e., the partition of the set of agents
  - Constraints
  - Winner Determination problem
  - Dynamic? (wait for PART II)

Coalition Structure Generation

- The grand coalition may not always be best
  - The characteristic function v(C) i.e., coalitions' values are independent of which other coalitions are formed.
- The set partitioning problem
  - Divide a set of agents into subsets to maximise the sum of the values of the subsets chosen
  - Crew Scheduling
  - Special case of the Winner Determination problem for combinatorial auctions (all items have to be allocated)



The Set Partitioning Problem



20

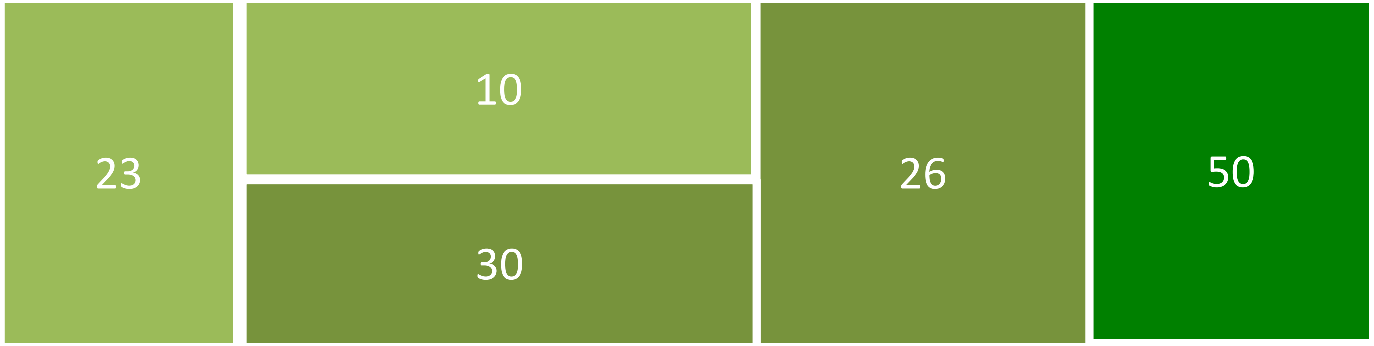
3

10

3

3

45



23

10

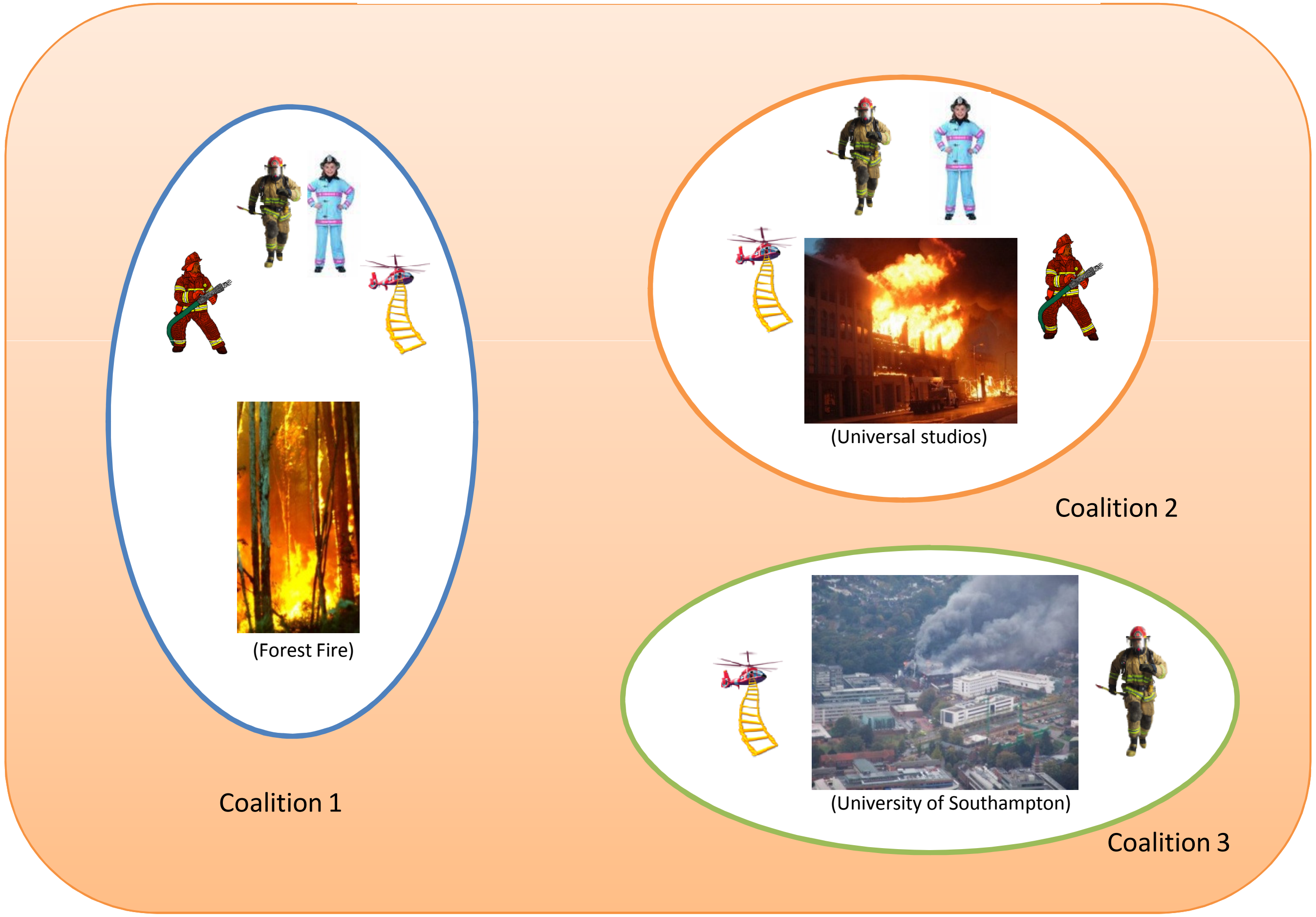
30

26

50

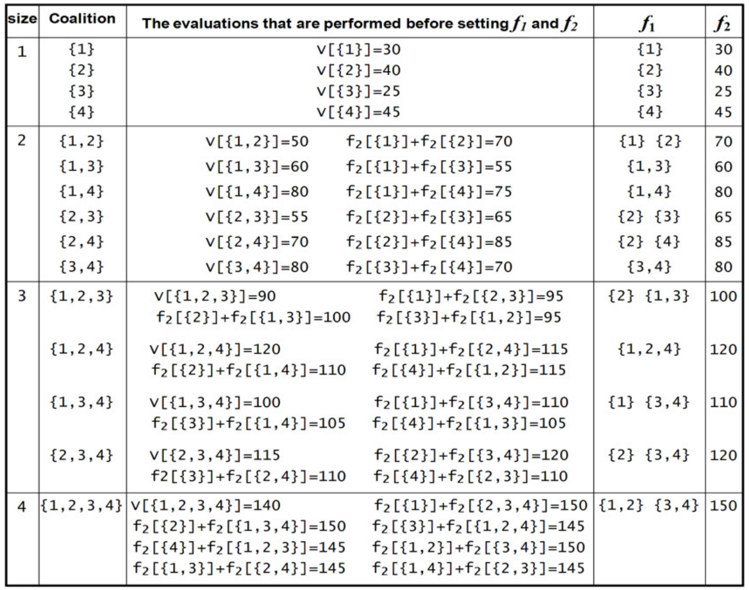
Coalition Structure

$$V(CS) = 10 + 14 + 5 = 26$$

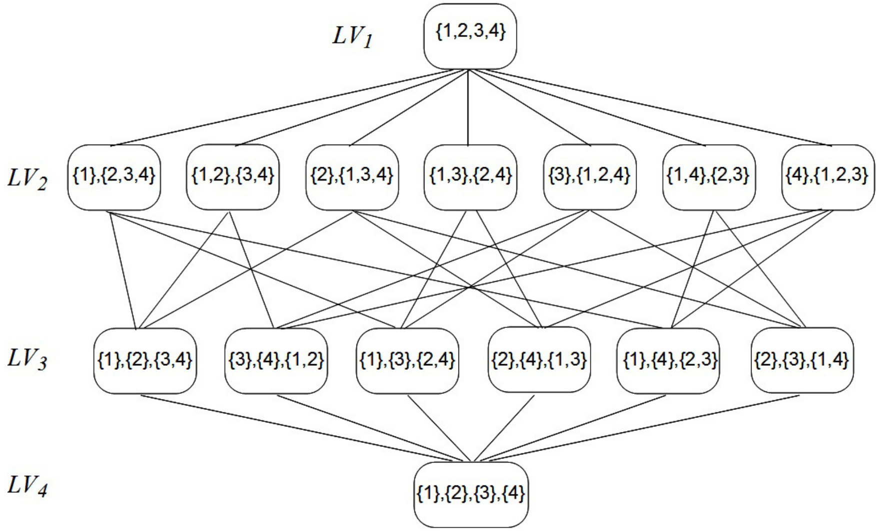


Coalition Structure Generation

- The Representation:
  - Rothkopf/Yun Yeh – Dynamic Programming
  - Sandholm et al.
  - Rahwan et al. (IP/IDP-IP etc…)

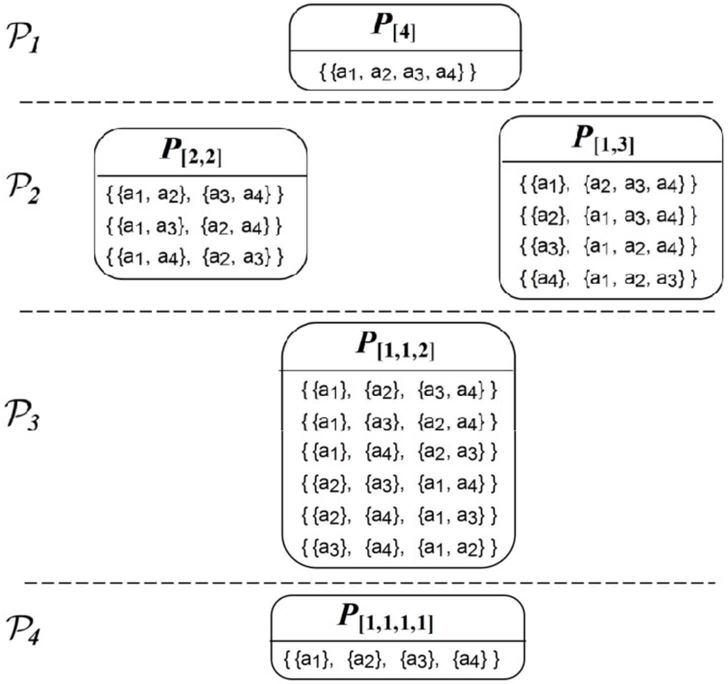


size	Coalition	The evaluations that are performed before setting f_1 and f_2	f_1	f_2
1	{1}	V[{1}]=30	{1}	30
	{2}	V[{2}]=40	{2}	40
	{3}	V[{3}]=25	{3}	25
	{4}	V[{4}]=45	{4}	45
2	{1,2}	V[{1,2}]=50 f_2[{1}]+f_2[{2}]=70	{1} {2}	70
	{1,3}	V[{1,3}]=60 f_2[{1}]+f_2[{3}]=55	{1,3}	60
	{1,4}	V[{1,4}]=80 f_2[{1}]+f_2[{4}]=75	{1,4}	80
	{2,3}	V[{2,3}]=55 f_2[{2}]+f_2[{3}]=65	{2} {3}	65
	{2,4}	V[{2,4}]=70 f_2[{2}]+f_2[{4}]=85	{2} {4}	85
	{3,4}	V[{3,4}]=80 f_2[{3}]+f_2[{4}]=70	{3,4}	80
3	{1,2,3}	V[{1,2,3}]=90 f_2[{1}]+f_2[{2,3}]=95 f_2[{2}]+f_2[{1,3}]=100 f_2[{3}]+f_2[{1,2}]=95	{2} {1,3}	100
	{1,2,4}	V[{1,2,4}]=120 f_2[{1}]+f_2[{2,4}]=115 f_2[{2}]+f_2[{1,4}]=110 f_2[{4}]+f_2[{1,2}]=115	{1,2,4}	120
	{1,3,4}	V[{1,3,4}]=100 f_2[{1}]+f_2[{3,4}]=110 f_2[{3}]+f_2[{1,4}]=105 f_2[{4}]+f_2[{1,3}]=105	{1} {3,4}	110
	{2,3,4}	V[{2,3,4}]=115 f_2[{2}]+f_2[{3,4}]=120 f2[{3}]+f_2[{2,4}]=110 f_2[{4}]+f_2[{2,3}]=110	{2} {3,4}	120
4	{1,2,3,4}	V[{1,2,3,4}]=140 f_2[{1}]+f_2[{2,3,4}]=150 f_2[{2}]+f_2[{1,3,4}]=150 f_2[{3}]+f_2[{1,2,4}]=145 f_2[{4}]+f_2[{1,2,3}]=145 f_2[{1,2}]+f_2[{3,4}]=150 f_2[{1,3}]+f_2[{2,4}]=145 f_2[{1,4}]+f_2[{2,3}]=145	{1,2} {3,4}	150



Coalition Structure Generation

- IP algorithm
  - Using Integer partitions of the number of agents
  - Create a set of subspaces for which we can compute upper and lower bounds
  - Apply BnB over these spaces
  - Apply BnB within these spaces
- CF with constraints
  - Restrict sizes
  - Restrict number of coalitions according to number of tasks
  - ….



$\mathcal{P}_1$

$P_{[4]}$

$\{\{a_1, a_2, a_3, a_4\}\}$

$\mathcal{P}_2$

$P_{[2,2]}$

$\{\{a_1, a_2\}, \{a_3, a_4\}\}$
$\{\{a_1, a_3\}, \{a_2, a_4\}\}$
$\{\{a_1, a_4\}, \{a_2, a_3\}\}$

$P_{[1,3]}$

$\{\{a_1\}, \{a_2, a_3, a_4\}\}$
$\{\{a_2\}, \{a_1, a_3, a_4\}\}$
$\{\{a_3\}, \{a_1, a_2, a_4\}\}$
$\{\{a_4\}, \{a_1, a_2, a_3\}\}$

$\mathcal{P}_3$

$P_{[1,1,2]}$

$\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$
$\{\{a_1\}, \{a_3\}, \{a_2, a_4\}\}$
$\{\{a_1\}, \{a_4\}, \{a_2, a_3\}\}$
$\{\{a_2\}, \{a_3\}, \{a_1, a_4\}\}$
$\{\{a_2\}, \{a_4\}, \{a_1, a_3\}\}$
$\{\{a_3\}, \{a_4\}, \{a_1, a_2\}\}$

$\mathcal{P}_4$

$P_{[1,1,1,1]}$

$\{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\}$

Part 3

Complex Coalition Formation Applications

Advanced Coalition Formation

- Task/Resource allocation
  - to coalitions
  - to agents within coalitions
  - To be shared by coalitions
- Scheduling
  - Dynamic
  - Temporal constraints
- Routing
  - Spatial constraints
  - Dynamism
- Distributed

Autonomous Straddle Carrier in Operation

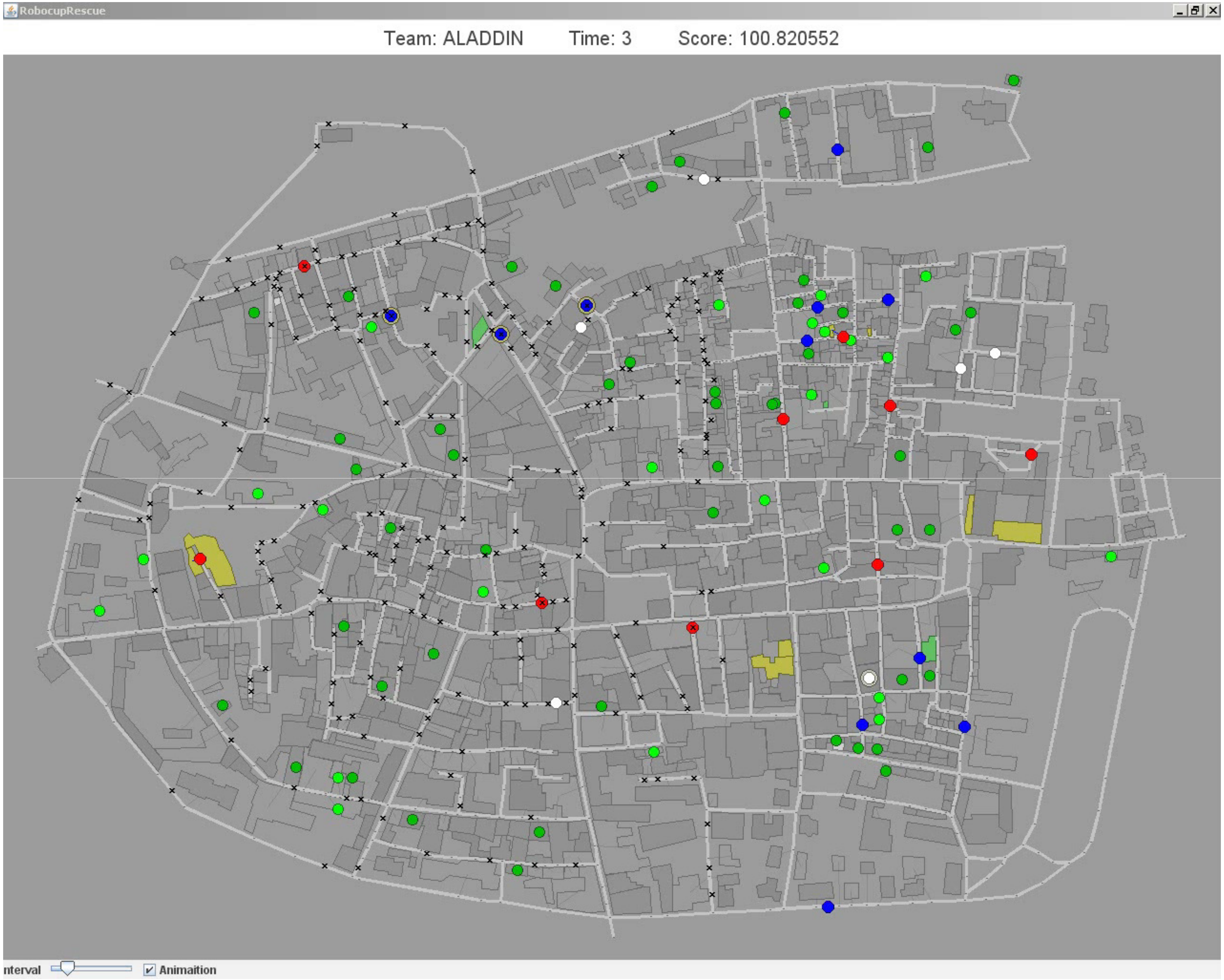
Task Allocation via Coalition Formation

- Shehory & Kraus, 1995
  - Agents communicate to decide which coalitions to form to perform tasks
  - Tasks with precedence order
  - Overlapping coalitions (agents can be in two coalitions)
- Sandholm and Lesser
  - Distributed Vehicle Routing context
  - Boundedly rational agents

Example application

ROBOCUP RESCUE SIMULATION

http://sourceforge.net/projects/roborescue/



Team: ALADDIN Time: 3 Score: 100.820552

Spatial and Temporal Constraints

- The RoboCupRescue Grand Challenge

  Design algorithms to allocate teams of ambulances and fire brigades in a disaster space in order to save civilians and extinguish fires.

  Ambulances located in different parts of the city have to work together to dig out victims

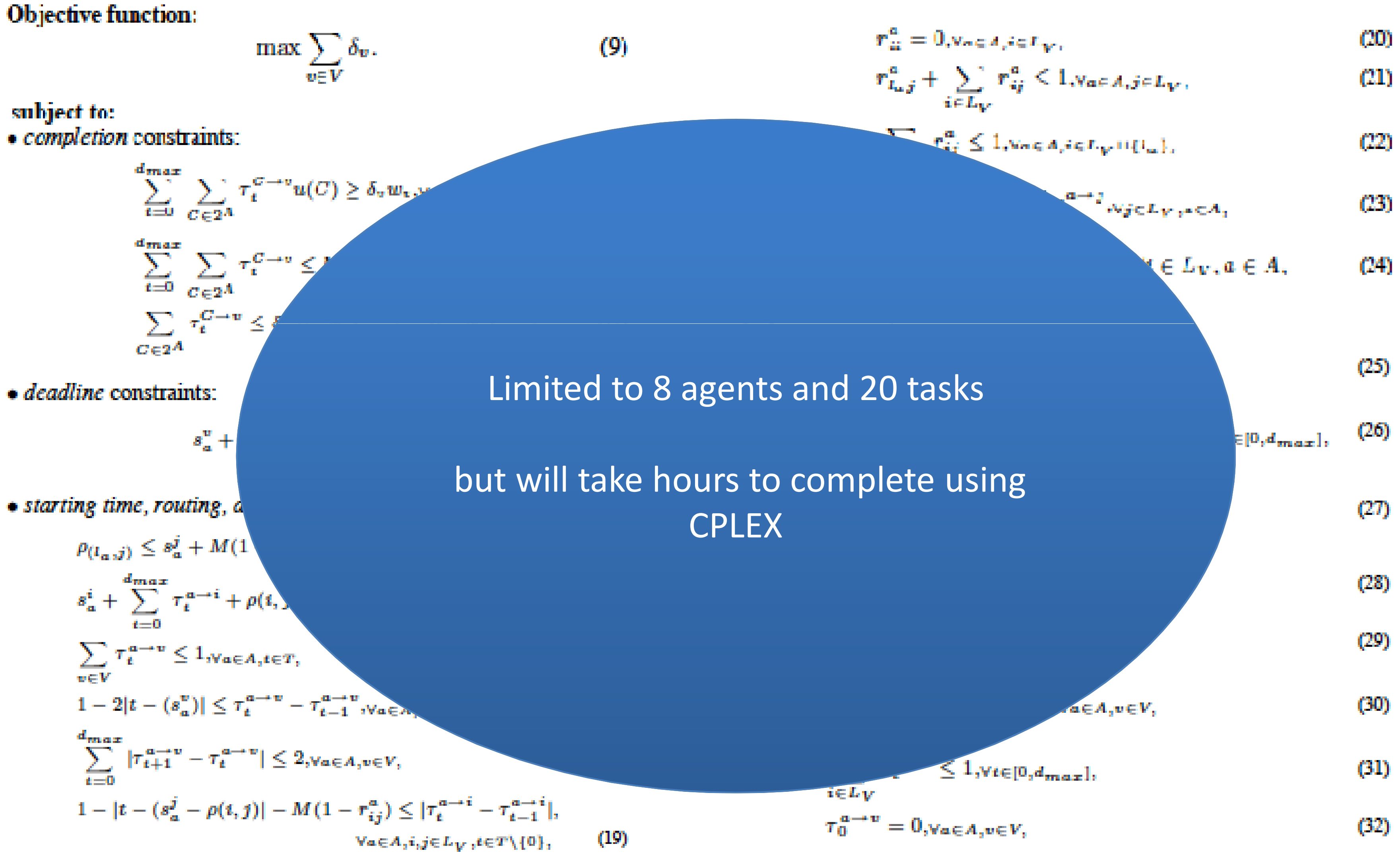
  Fire brigades located in different parts of the city have to work together to extinguish rapidly evolving fires – otherwise the city burns down

The example of the Ambulance Allocation problem – Spatial and Temporal issues

- Each victim has a:
  - Deadline
  - Workload (time to dig a victim out)
- Adding more ambulances linearly increases the speed at which the workload can be reduced

- Objective: Allocate coalitions of ambulances to maximise the number of victims saved, where each ambulance/victim is located at different points in the space.

Optimal Algorithm for Ambulance Allocation



Objective function:

$$\max \sum_{v \in V} \delta_v . \qquad (9)$$

subject to:

- *completion* constraints:

$$\sum_{t=0}^{d_{max}} \sum_{C \in 2^A} \tau_t^{C \to v} u(C) \geq \delta_v w_{t,v}$$

$$\sum_{t=0}^{d_{max}} \sum_{C \in 2^A} \tau_t^{C \to v} \leq$$

$$\sum_{C \in 2^A} \tau_t^{C \to v} \leq$$

- *deadline* constraints:

$$s_a^v +$$

- *starting time, routing,*

$$\rho_{(l_a, j)} \leq s_a^j + M(1$$

$$s_a^i + \sum_{t=0}^{d_{max}} \tau_t^{a \to i} + \rho(i,$$

$$\sum_{v \in V} \tau_t^{a \to v} \leq 1, _{\forall a \in A, t \in T},$$

$$1 - 2|t - (s_a^v)| \leq \tau_t^{a \to v} - \tau_{t-1}^{a \to v}, _{\forall a \in A}$$

$$\sum_{t=0}^{d_{max}} |\tau_{t+1}^{a \to v} - \tau_t^{a \to v}| \leq 2, _{\forall a \in A, v \in V},$$

$$1 - |t - (s_a^j - \rho(i,j))| - M(1 - r_{ij}^a) \leq |\tau_t^{a \to i} - \tau_{t-1}^{a \to i}|, \quad _{\forall a \in A, i,j \in L_V, t \in T \setminus \{0\}}, \qquad (19)$$

$$r_{ii}^a = 0, _{\forall a \in A, i \in L_V}, \qquad (20)$$

$$r_{l_a j}^a + \sum_{i \in L_V} r_{ij}^a \leq 1, _{\forall a \in A, j \in L_V}, \qquad (21)$$

$$\sum r_{ij}^a \leq 1, _{\forall a \in A, i \in L_V \cup \{l_a\}}, \qquad (22)$$

$$^{a \to j}, _{\forall j \in L_V, a \in A}, \qquad (23)$$

$$\in L_V, a \in A, \qquad (24)$$

(25)

$$\in [0, d_{max}], \qquad (26)$$

(27)

(28)

(29)

$$_{\forall a \in A, v \in V}, \qquad (30)$$

$$\sum_{i \in L_V} \leq 1, _{\forall t \in [0, d_{max}]}, \qquad (31)$$

$$\tau_0^{a \to v} = 0, _{\forall a \in A, v \in V}, \qquad (32)$$

Limited to 8 agents and 20 tasks

but will take hours to complete using CPLEX

CFST

- Admissible Heuristics
  - Maximise the number of tasks completed
  - Maximise working time of the agents
  - Minimise time to complete maximum number of tasks
- Iterative approach
  - Space is limited to only N tasks and, in the worst case 2^m coalitions (but this is unlikely due to spatial distribution!)
- DCOP version
  - Fast-Max-Sum (discussed earlier this morning)

Example application 2

VIRTUAL POWER PLANTS IN THE SMART GRID

Uncertainty and Dynamism

- Key computational issues
  - How to find the best agents to team up with?
  - How to share the benefits if the agents are not reliable in their prediction of output?
  - How to change coalitions over time?

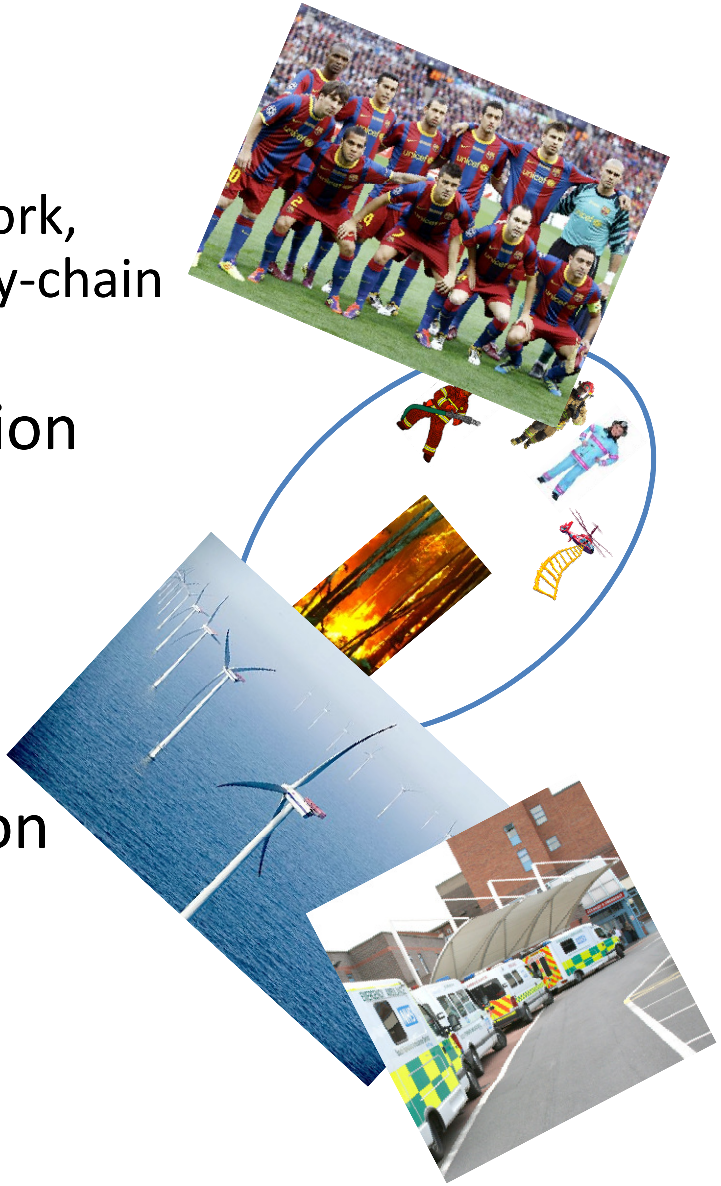
Initial Solutions

- Chalkiadakis et al., AAMAS 2010
  - Cooperative game theoretic issues.
- Vasirani et al., ATES 2011
  - Use a bidding process to allocate vehicles to the virtual power plant

- Possible PhD topic: Algorithms yet for creating virtual power plants on the fly given changing weather conditions/demand/etc…

Now, you should be able to:

1. Define what is coalition formation
   - Differences from and links to teamwork, decentralised coordination and supply-chain formation.
2. List the key steps of abstract coalition formation
   - Cooperative and Competitive settings
   - Computational challenges
   - Lack of realism
3. Identify complex coalition formation applications
   - Dynamic CF: Scheduling and Routing
   - Uncertainty and Dynamism



List of references I

- [1] G. Chalkiadakis and C. Boutilier. Bayesian reinforcement learning for coali- tion formation under uncertainty. 2004.
- [2] G. Chalkiadakis, V. Robu, R. Kota, A. Rogers, and N. Jennings. Cooper- atives of distributed energy resources for efficient virtual power plants. In The Tenth International Conference on Autonomous Agents and Multia- gent Systems (AAMAS-2011), pages 787–794, May 2011.
- [3] V. Dang, R. Dash, A. Rogers, and N. Jennings. Overlapping coalition formation for efficient data fusion in multi-sensor networks. In Proceedings of the National Conference on Artificial Intelligence, volume 21, page 635. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.
- [4] G. Demange and M. Wooders. Group formation in economics: networks, clubs and coalitions. Cambridge Univ Pr, 2005.
- [5] K. Macarthur, R. Stranders, S. Ramchurn, and N. Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In Twenty-Fifth Conference on Artificial Intelligence. AAAI Press, August 2011.
- [6] R. Myerson. Game theory: analysis of conflict. Harvard Univ Pr, 1997.
- [7] T. Rahwan and N. Jennings. Coalition structure generation: dynamic programming meets anytime optimisation. In Proceedings of the 23rd Confer- ence on Artificial Intelligence (AAAI), pages 156–161, 2008.

List of References II

- [8] T. Rahwan and N. Jennings. An improved dynamic programming algorithm for coalition structure generation. In Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3, pages 1417–1420. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
- [9] T. Rahwan, T. Michalak, N. Jennings, M. Wooldridge, and P. McBurney. Coalition structure generation in multi-agent systems with positive and negative externalities. In Proceedings of the 21st international jont confer- ence on Artifical intelligence, pages 257–263. Morgan Kaufmann Publishers Inc., 2009.
- [10] T. Rahwan, S. Ramchurn, N. Jennings, and A. Giovannucci. An anytime algorithm for optimal coalition structure generation. Journal of Artificial Intelligence Research, 34(1):521–567, 2009.
- 1[11] S. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. Jennings. Coalition formation with spatial and temporal constraints. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 3-Volume 3, pages 1181–1188. International Foundation for Autonomous Agents and Multiagent Systems, 2010.
- [12] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohḿ e. Any- time coalition structure generation with worst case guarantees. Arxiv preprint cs/9810005, 1998.
- [13] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating tasks in extreme teams. In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pages 727–734. ACM, 2005.
- [14] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. Artificial Intelligence, 101(1-2):165–200, 1998.
- [15] D. Yun Yeh. A dynamic programming approach to the complete set parti- tioning problem. BIT Numerical Mathematics, 26(4):467–474, 1986.