OPPA European Social Fund
Prague & EU: We invest in your future.

Multi-agent Constraint Programming

Boi Faltings

Laboratoire d'Intelligence Artificielle
boi.faltings@epfl.ch
http://moodle.epfl.ch/

May 10, 2012

Multi-agent Constraint Satisfaction Problems (CSP)

Given $< X, D, C, A >$ where:

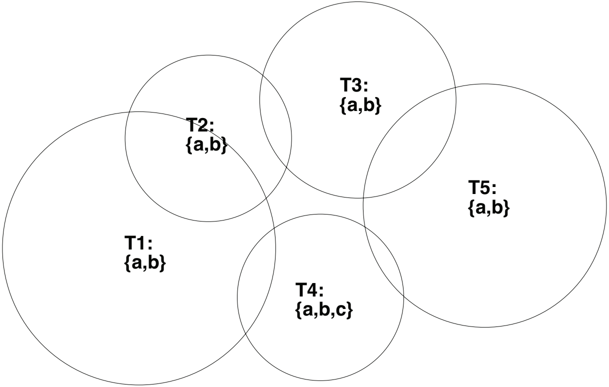
- $X = \{x_1, .., x_n\}$ is a set of n variables.
- $D = \{d_1, .., d_n\}$ is a set of n domains.
- $C = \{c_1, .., c_m\}$ is a set of m constraints.
- $A = \{a_1, .., a_n\}$ is a set of n agents, not necessarily all different.

Find solution $= (x_1 = v_1 \in d_1, ..., x_n = v_n \in d_n)$ such that for all constraints, value combinations are allowed by relations.

Example of a CSP: Radio Spectrum Allocation

Goal: select transmission channels that do not interfere with others:

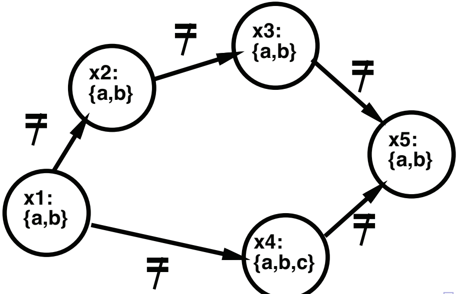
.



Resource Allocation (2)

CSP model:

- Variables = choice of frequency
- Domains = frequency bands
- Constraints = inequalities between overlapping ranges
- Agents control transmitters



Constraint Optimization

- Some solutions are better than others.
- Express using soft constraints: every tuple has a cost.
- Optimal solution =
  - solution that minimizes sum of costs (utiliarian).
  - solution that minimizes maximal cost (egalitarian).
  - mixture (semiring).
- Most real problems are optimization problems.

Overview

Distributed algorithms for solving CSP and COP.

- synchronous backtracking
- asynchronous backtracking/ADOPT
- dynamic programming/DPOP
- distributed local search
- random sampling

Follows survey article:

Faltings, B. *Distributed Constraint Programming*. In Rossi, F., van Beek, P. and Walsh, T. (editors), Handbook of Constraint Programming, pages 699-729. Elsevier, 2006 (also at http://liawww.epfl.ch/)

Solving a CSP

Importance of CSP: large theory and tools for computing solutions

2 common methods:

- backtrack search: assign one variable at a time, backtrack when no assignment without satisfying constraints.
- local search: start with random assignment, make local changes to reduce number of constraint violations.

Distributed CSP (DCSP)

- Problem is distributed in a network of *agents*.
- Each variable *belongs* to one agent who is responsible for setting its value (typically these are connected to complex local subproblems).
- Constraints are known to all agents with variables in it.
- Distributed $\neq$ parallel: distribution of variables to agents cannot be chosen to optimize performance.

Reasons for a distributed solution

Real world problems are often distributed:

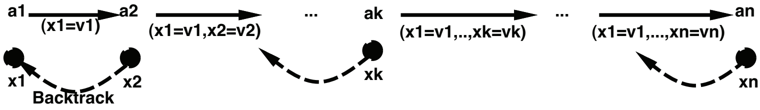
- no agreement on a common model.
- costly to formalize constraints and preferences for all possible cases.
- no trusted third party.
- privacy concerns.

but generally not efficiency!

Synchronous Backtracking

Agents agree on an variable order and repeat:

1. send partial solution up to x_{k-1} to k-th agent.
2. k-th agent generates the next extension to this partial solution.
3. if solution cannot be extended consistently, $k \leftarrow k-1$.
4. if solution can be extended consistently, $k \leftarrow k+1$.
5. if $k < 1$, stop: unsolvable.
6. if $k > n$, assigment $=$ solution.



Optimization: SyncBB

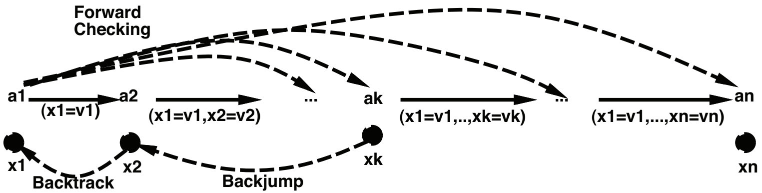
Extend synchronous backtracking to optimization:

- every constraint contributes a cost.
- upper bound = lowest cost of full assignment found so far.
- partial assignment extended while cost $<$ upper bound.
- result = solution with lowest cost.

Improvements

Synchronous backtracking allows common CSP heuristics:

- forward checking: send partial solution to all higher agents.
- dynamic variable ordering: select next variable according to domain size.
- backjumping: reduce *k* to last variable involved in conflict.



Implementing CSP heuristics

Distributed forward checking:

- $A(x_k)$ sends $(x_1 = v_1, .., x_k = v_k)$ to all $A(x_j)$, $j > k$
- $A(x_j)$ removes inconsistent values and initiates backtrack at x_k whenever domain becomes empty

Can be done aynchronously (asynchronous forward checking)
Dynamic variable ordering:

- $A(x_j)$ sends back size of remaining domain for x_j
- $A(x_k)$ chooses smallest one to be x_{k+1}

Backjumping:
reduce k to last variable involved in current conflict.

Performance metrics

- non-concurrent constraint checks (NCCC): longest chain of constraint checks with serial dependency (ignores message delivery time).
- concurrent time: (simulated) time taken in parallel execution.
- wall clock time (time taken by the simulator).
- number of messages (ignores computation time and size of messages).
- amount of information exchanged (ignores computation time).

Asynchronous Backtracking

- Agents work in parallel without synchronization.
- Global priority ordering among variables (ex.: unique processor id); assume x_i has higher priority than x_j whenever $i < j$.
- Asynchronous message delivery, but all messages arrive in order in which they were sent.
- constraints are binary.
- every agent a_i is responsible for one variable x_i.

ABT data structures

Each agent maintains

- a current value for its own variable.
- all constraints with higher priority variables.
- a list of all lower priority variables.
- an `agent view` that records the values of all known higher priority variables.
- for each value of its own variable, a set of `nogood` that indicate lower bounds on the cost that choosing this value has for lower priority variables.

Adjusting own variable value

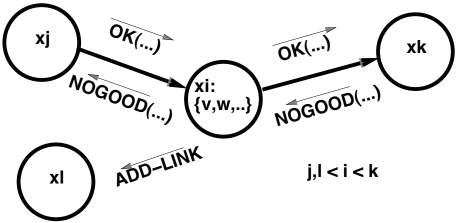
Own variable should be adjusted to the value with the lowest possible cost:

- cost(v) $\geq \sum$ constraints(agent view) $+ \sum$ nogoods(v)
- if all nogoods are exact, cost(v) is also exact.
- set variable $x \leftarrow$ v with lowest cost bound.
- if cost(v) > 0 send nogood to higher priority variable.
- similarly if cost is exact, indicate to higher priority variable.

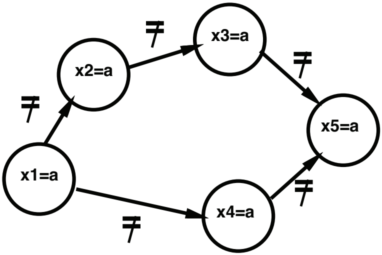
ABT messages

Agent informs:

- lower priority agents of value choice using `OK?` messages.
- closest higher priority agent of cost bounds using `nogood` messages.
- newly discovered agents using add-link messages.

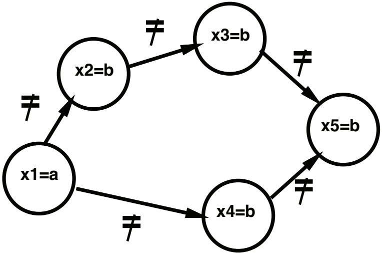


Example (1)



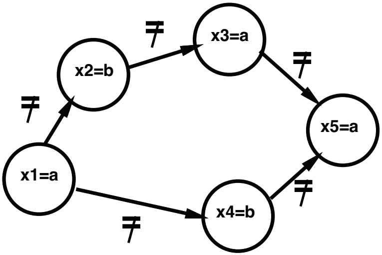
	message(s)	action
a_2	OK(x_1=a)	$x_2 \leftarrow$ b
a_3	OK(x_2=a)	$x_3 \leftarrow$ b
a_4	OK(x_1=a)	$x_4 \leftarrow$ b
a_5	OK(x_3=a)	$x_5 \leftarrow$ b
	OK(x_4=a)	

Example (2)



	message(s)	action
a_3	OK(x_2=b)	$x_3 \leftarrow$ a
a_5	OK(x_3=b)	$x_5 \leftarrow$ a
	OK(x_4=b)	

Example (3)



	message(s)	action
a_5	OK(x_3=a)	inconsistent!

$$x_3 = \mathrm{a} \Rightarrow x_5 \neq \mathrm{a}$$
$$x_4 = \mathrm{b} \Rightarrow x_5 \neq \mathrm{b}$$

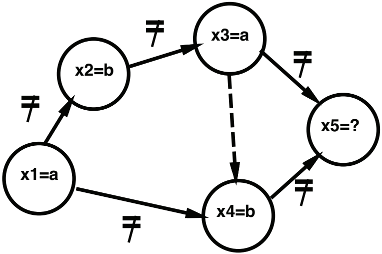
a_5 sends a nogood to a_4:

v = b, cond $= (x_3 = \mathrm{a})$, tag $= x_5$ cost = 1

Example(4)

- Nogoods give lower bounds on costs incurred by the lower priority variables mentioned in the `tag`:
  nogood.cond $\subseteq$ *self.agentview* $\wedge$ *nogood.v* = *self.x.v* $\Rightarrow$ *cost-sum(nogood.tag)* $\geq$ *nogood.cost*
- a_4 adds the nogood for value b, with tag x_5.
- However, this requires checking whether it is applicable, i.e. that nogood.cond corresponds to its agent view.
- a_4 does not know about x_3, so it requests a new link using an `add-link` message to a_3.
- Now it can be verified that the agentview satisfies the condition.

Example (5)



- a_4 now finds that value a is inconsistent because of x_1, and b is inconsistent because of the nogood.
- chooses a third value, c, and informs a_5.
- a_5 can now choose $x_5 = b$ and obtain a consistent solution.

Termination Detection

- x_5 has a value with cost=0 and no lower priority agents.

⇒ cost of x_5 is exact, a_5 sends an exact nogood with cost 0 to a_4 and a_3.

- a_4 now has an exact nogood for its only lower-priority agent, and itself sends an exact nogood with cost 0 to a_3.
- ...
- a_1 has no higher-priority agent: it generates an exact nogood but decides termination.
- when there is no solution, a_1 generates an exact nogood with cost $\neq 0$.

Extension to Optimization

- Nogoods give lower bounds on costs.
- Compute total cost of all lower priority agents by summing nogoods.
- Nogood tags must exactly cover all lower-priority variables, otherwise some variables are not counted or counted multiple times.
- If we can prevent this from happening, then ABT works fine for optimization as well.

Pseudotrees

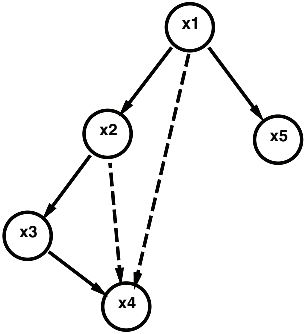
- Split constraint graph into spanning tree + back edges.
- Identify root: every node has one parent (path to the root).
- Pseudotree: all backedges go to ancestors of the node.
- A pseudotree exists for all graphs and choices of root node.
- Example: DFS tree.

Constructing a DFS ordering

Depth-first search traversal:

- move to neighbour not yet visited
- connect neighbours already in graph by *back edges*
- backtrack when no new neighbour

Note: all back edges connect to ancestors!



Properties of DFS trees

- nogoods are always sent to lowest-priority agent.

⇒ nogoods are never sent along back edges.

⇒ no variable can appear in nogoods from different branches.

⇒ exact nogoods always add up to an exact bound!

Asynchronous optimization: ADOPT

- using pseudotree ordering $\Rightarrow$ ABT algorithm with valued nogoods gives exact optimization.
- additional optimization: remember cost of nogoods that are erased after change in agent view; when context is revisited, install as bound using *backtrack thresholds*.
- result = ADOPT, a widely cited algorithm for distributed constraint optimization.

ADOPT-NG

- different optimization of ABT: send valued nogoods to all ancestors, not just the lowest one.

⇒ ancestors higher in the tree can form bounds on the relative quality of different valuations.

- greatly improves efficiency, even without backtrack threshold mechanism.

Properties of asynchronous backtracking

- Algorithm is complete: if there is a solution, it will be found (due to direct correspondence with backtracking algorithm).
- CSP heuristics costly to implement.
- Termination needs to be detected with termination detection algorithm (= consensus problems).
- Asynchronous behavior can create wasted search effort ⇒
  - more messages than synchronous backtracking, but
  - sometimes shorter execution time (parallelism)

Dynamic Programming Optimization Protocol (DPOP)

- Principle: replace variables by constraints.
- Consider variable x having constraint with y.
- For each value of x, there may be a consistent value of y.

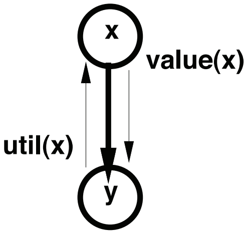
⇒ replace y by a constraint on x:

x=v is allowed if there is a consistent value of y.

- Optimization version:

utility(x=v) = utility(x=v,y=w); w = best possible value of y given x=v.

Example



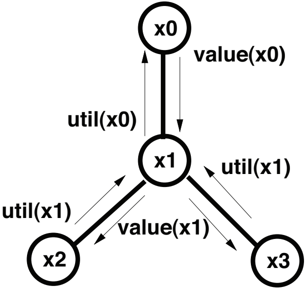
- y sends constraint in util(x) message.

⇒ x can decide (best) value locally.

- x informs y of value using value(x) message.

Dynamic programming in trees

- Rooted tree: every node has at most one parent
- Nodes send UTIL messages to their parents
- Best values of x2, x3 ⇒ unary constraint on x1
- x1 sums up UTIL messages + own constraint ⇒ unary constraint on x0
- x0 picks best value v(x0); sends value(x0=v(x0)) → x1
- x1 picks best value given x0 and informs x2,x3

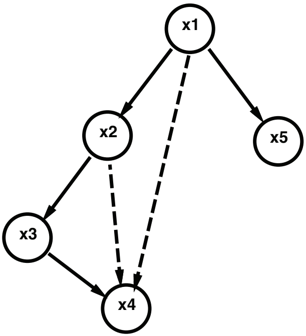


Dynamic programming in graphs

Use pseudotree/DFS ordering:

- send UTIL messages along the tree edges.
- add extra dimensions for variables involved in back edges.
- message size grows exponentially in number of dimensions.

Complexity exponential in treewidth of ordering!



Example Problem

$c(x_0, x_3)$

x_0 \ x_3	w	b
w	3	0
b	3	3

$c(x_0, x_1)$

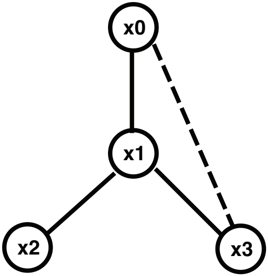
x_0 \ x_1	w	b
w	1	0
b	2	2

$c(x_1, x_2)$

x_1 \ x_2	w	b
w	1	0
b	0	1

$c(x_1, x_3)$

x_1 \ x_3	w	b
w	2	0
b	0	2

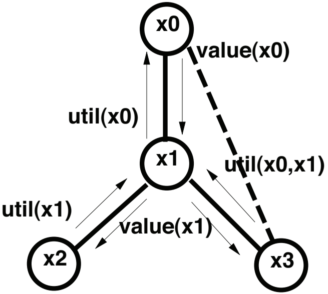


Distributed dynamic programming

$$UTIL(x_1) = \begin{array}{cc} \multicolumn{2}{c}{x_1} \\ w & b \\ \hline 0 & 0 \end{array}$$

$$UTIL(x_0, x_1) = x_0 \begin{array}{c|cc} & \multicolumn{2}{c}{x_1} \\ & w & b \\ \hline w & 0 & 2 \\ b & 3 & 3 \end{array}$$

$$UTIL(x_0) = \begin{array}{cc} \multicolumn{2}{c}{x_0} \\ w & b \\ \hline 1 & 3 \end{array}$$



x_0: w; send value$(x_0 = \mathtt{w}) \to x_1$
x_1: w; send value$(x_0 = \mathtt{w}, x_1 = \mathtt{w}) \to x_2, x_3$
x_2 and x_3: b

Complexity

- Two messages per variable (UTIL and VALUE).

$\Rightarrow$ *number* of messages grows linearly with the size of the problem.

- However, the maximum message *size* grows exponentially with the tree-width of the induced graph.
- In many distributed problems, the tree-width is relatively small.

DPOP variants

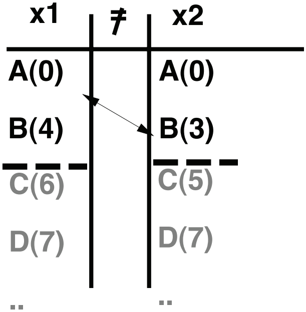
- S-DPOP (AAAI 2005): self-stabilizing.
- A-DPOP (CP 2005): approximation through dropping constraints.
- O-DPOP (AAAI 2006): Open DPOP: incremental elicitation.
- PC-DPOP (IJCAI 2007): DPOP with partial centralization.

MB-DPOP (IJCAI 2007)

- tradeoff between number and size of messages: combine search with dynamic programming.
- MB-DPOP limits message size and switches to search whenever message exceeds dimension limit.
- allows continuous scaling from pure search to pure dynamic programming.

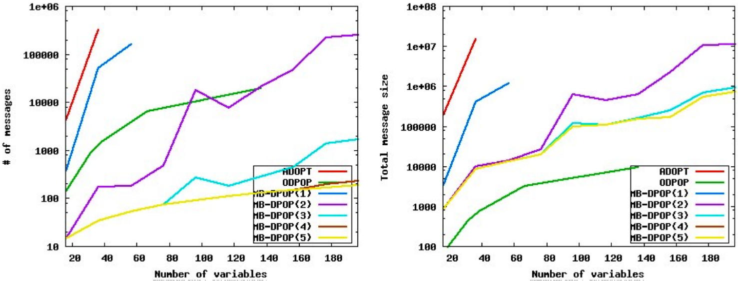
Open Constraint Optimization

- Observation: can solve CSP without knowing domains completely.
- Extends to optimization:



- If $x_1 = a, x_2 = b$ is consistent, no other solution can be better!
- Implemented in ODPOP.

DPOP performance



Distributed local search

Drawbacks of systematic search:

- need variable ordering (impossibility result by Dechter)
- no anytime behavior: have to wait for termination.
- often (too) costly.

Sacrifice completeness $\Rightarrow$ local search

Min-conflicts

- Assign random value to each variable in parallel (this will conflict with some constraints).
- At each step, find the change in variable assignment which most reduces the number of conflicts .
- Corresponds to search by "hill-climbing".

Distributed min-conflicts

- *Neighbourhood* of $N(x_i)$ = variables connected to x_i through constraints.
- Change to x_i can happen asynchronously with others as long as there is no other change in the neighbourhood.

⇒ two neighbouring agents are not allowed to change simultaneously:
  - highest improvement wins
  - ties broken by fixed ordering

⇒ parallel, distributed execution.

- also called `MGM`

Breakout Algorithm

- Similar to min-conflict, but assign dynamic priority to every conflict (constraint), initially =1
- Modify variable which reduces the most the sum of the priority values of all conflicts.
- When local minimum:

  increase weight of every existing conflict
- Eventually, new conflicts will have lower weight than existing ones ⇒ breakout

Local minima

If all improvements = 0:

1. increase weight of all constraint violations
2. restart asynchronous changes

Random Sampling

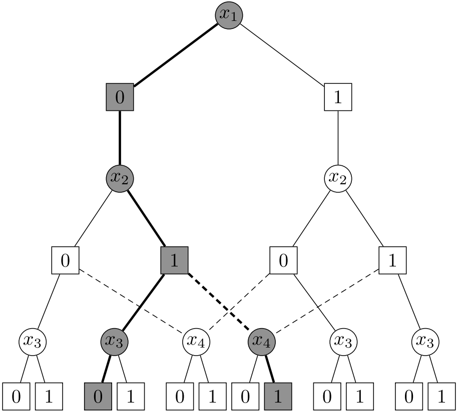
Carry out optimization as in synchronous branch-and-bound, but:

- instead of systematic enumeration, sample variable domains randomly
- for each sampled assignment, feed backwards sum of costs: each agent knows the cost to its children.
- keep a record of the best cost $\mu^t_{a,d}$ for each context a and sample value d, and also the best value d_a found at time t.

Termination: sequentially select best value from first to last agent.

Extension to Pseudotrees

Does not require linear order, but samples can be generated simultaneously for different branches in a pseudotree:



Distributed Upper Confidence Bounds on Trees (DUCT)

- for value d and context a, compute confidence interval $L^t_{a,d}$ using Hoeffding bound.

$\Rightarrow$ estimate distance from optimum of worst sample.

$\Rightarrow$ estimate bound $B^t_{a,d}$ on optimal cost for value d in context a.

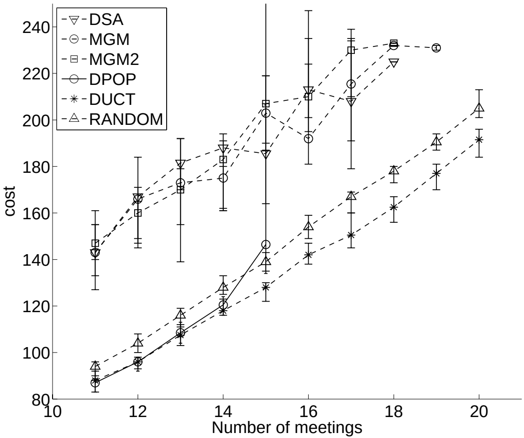
$\Rightarrow$ sample values with lowest estimate.

- bound probability that μ_{a,d_a} is further than δ from the optimum to be ϵ
  $\Rightarrow$ termination condition.

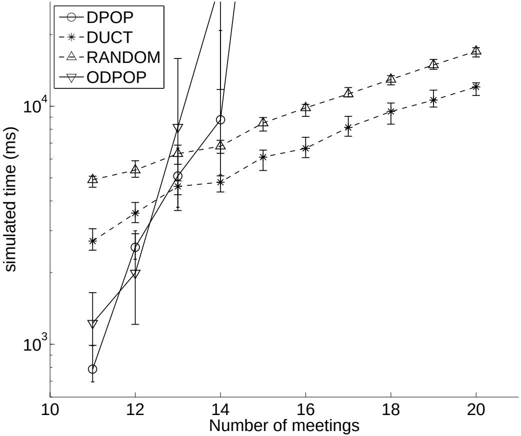
Note that all tests are local, no communication is required.
See:

Ottens, B., Dimitrakakis, C. and Faltings, B. *DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems.* In Proceedings of the 26th conference of the AAAI, 2012.

Performance: Cost



Performance: Time



Privacy Protection

- Distributed computation alone does not protect privacy.
- Homomorphic encryption can ensure complete privacy of preferences and final choices.
- With codenames, distributed computation can protect identities of agents and structure of constraints.

See:

Faltings, B., Léauté, T. and Petcu, A. Privacy Guarantees through Distributed Constraint Satisfaction. In Proceedings of the 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'08), pages 350-358, 2008
Léaut]'e, T. and Faltings, B. Privacy-Preserving Multi-agent Constraint Satisfaction. In 2009 IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT-09), pages 17-25, 2009
Léaut]'e, T. and Faltings, B. Coordinating Logistics Operations with Privacy Guarantees. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11), 2011

Self-Interest

- In many cases, agents just want to maximize their own benefit.
- Most solutions are better for some and worse for others.

$\Rightarrow$ need to compensate those who lose.

- Not a problem when utilities are publicly known: gain of the winners always exceeds losses of the losers.
- However, agents could manipulate the propagation.

Private Utilities

- When utilities are private, agents would exaggerate their own preferences.
- Counter by making each agent pay a VCG (Vickrey-Clarke-Groves) tax.
- VCG tax(a_i) $=$ cost increase on other agents due to agent a_i.

$\Rightarrow$ changes agent incentive from optimizing own cost to optimizing combined cost of all agents.

$\Rightarrow$ agent has no incentive to manipulate solving process (faithful execution)!

M-DPOP

- Computing VCG tax requires computing costs when agent a_i is not present (marginal economy).
- For much of the problem, this is the same as the full optimization: reuse this work.
- M-DPOP combines all propagations in parallel and makes this process efficient.

Software

Several open-source frameworks exist:

- FRODO (`http://frodo2.sourceforge.net/`): from EPFL-LIA, implements most algorithms using search, dynamic programming, local search and (soon) DUCT. Integration with open-source JaCoP solver for complex local problems.
- DisChoco (`http://www2.lirmm.fr/coconut/dischoco/`): from CNRS Montpellier, distributed framework for connecting Choco constraint solvers.
- various algorithms available individually.

Summary

- Multi-agent constraint satisfaction: interest of distributed algorithms.
- Synchronous and asynchronous backtracking.
- From satisfaction to optimization.
- DPOP: dynamic programming.
- Distributed local search.

OPPA European Social Fund
Prague & EU: We invest in your future.