Distributed Constraint Optimization

Michal Jakob

Agent Technology Center, Dept. of Cybernetics, FEE Czech Technical University

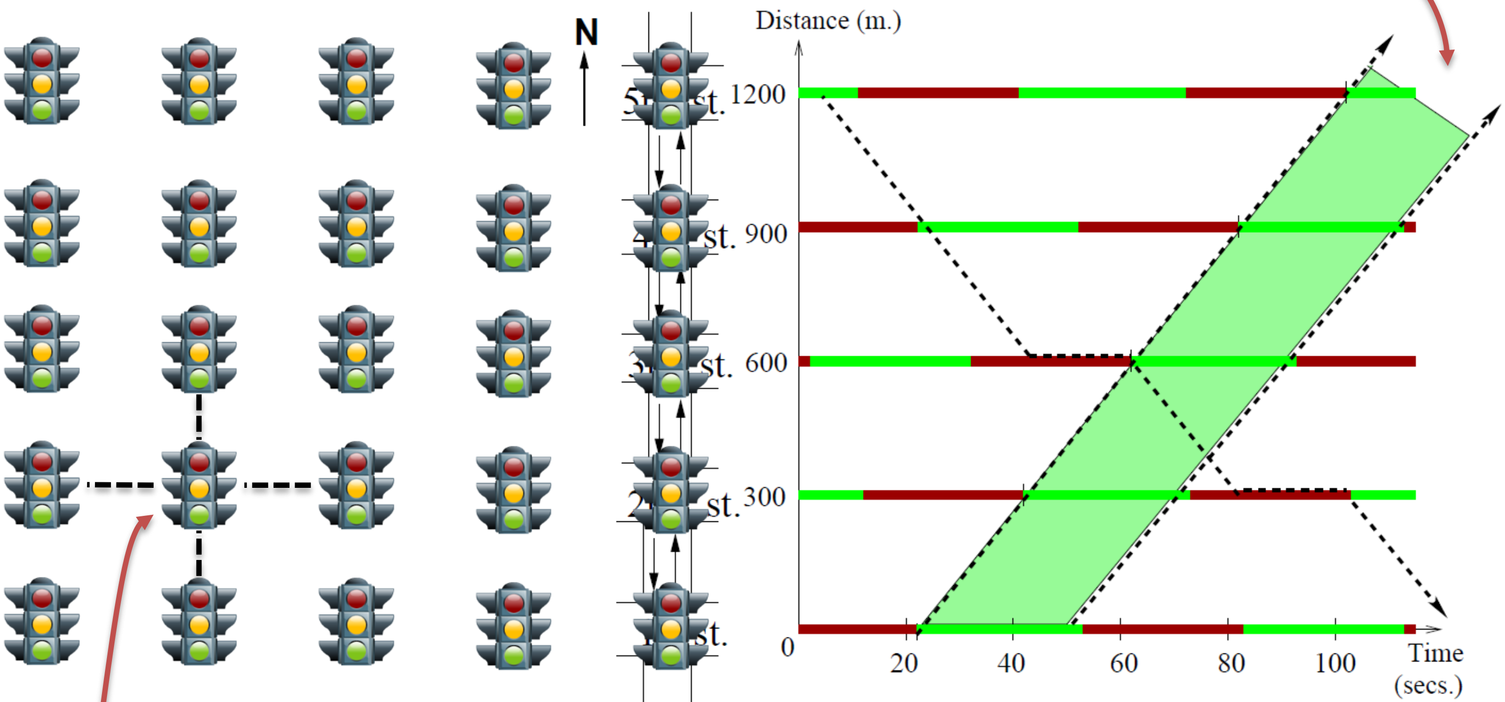
A4M33MAS Autumn 2010 - Lect. 12

(based partially on slides by Jay Modi)

Dynamic Traffic Light Scheduling

- Minimize traffic delay in a road grid by synchronizing lights

green wave



traffic light agent

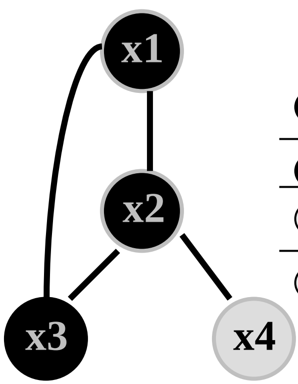
- can communicate locally
- can adopt four synchronization strategies (NS,SN,WE, EW)

Constraint Optimization Problem (COP)

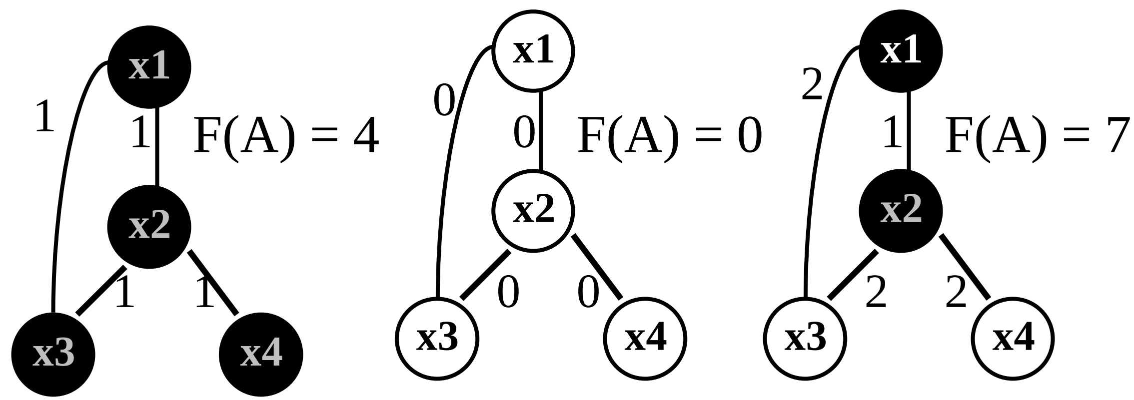
- Problem specification:
  - $X = \{x_i, \dots, x_m\}$ set of **variables**
  - $D = \{D_i, \dots, D_m\}$ set of **domains** for the variables, i.e. $x_i \in D_i$; if $D_i$is finite, let $D_i = \{v_{i,1}, \dots, v_{i,d(i)}\}$
  - $C = \{c_1, \dots, c_k\}$ set of **constraints** over X; the constraint c_i is represented by a real-valued function $P_i(y_1, \dots, y_j) \to \boldsymbol{R}$, $\{y_1, \dots, y_j\} \subseteq X$ that determines the **degree of constraint violation** (**constraint value**) for a given assignment
- Solution:
  - find an **assignment** of variables $\{x_i, \dots, x_m\}$ such that the **sum of all constraint values is minimized**

Example

Constraint Graph



di	dj	f(di,dj)
●	●	1
●	○	2
○	●	2
○	○	0



Distributed Constraint Optimization Problem (DCOP)

- $A = \{A_1, \dots, A_n\}$ set of agents
- Each agent A_i is responsible for one variable x_i
  - extension to multiple variables per agent possible
- Agent can communicate by sending messages
- Example

(D)COP with Binary Non-Negative Constraints

- Constraints involve **two variables** at maximum: $P(x_i, x_j)$
- Constraint values are **not negative:** $P(x_i, x_j) \geq 0$
- => addition of constraints **cannot decrease** the overall sum

Solution Algorithms

- Requirements on a good algorithm:
  - **terminates** in a finite number of steps
  - is **complete**: finds an optimum solution (always exists)
  - (is **sound**: the solution returned is indeed optimum)
- Existing algorithms
  - **Asynchronous Distributed Optimization (ADOPT)**
  - Optimal Asynchronous Partial Overlay (OptAPO)
  - Dynamic Parameter Optimization Problem (DPOP)
- Trade-offs between running time (number of cycles), number of messages and size of messages
- Recently, incomplete local search algorithms also proposed

Centralized Branch-and-Bound Algorithm

BRANCH-AND-BOUND-COP()

$c* \leftarrow \infty$ ▷ Minimum cost found. Global variable.
$g* \leftarrow \emptyset$ ▷ Best solution found. Global variable.
BRANCH-AND-BOUND-COP-HELPER$(1, \emptyset)$
return g^*

BRANCH-AND-BOUND-COP-HELPER(i, g)

if $i = n$
    then if $P(g) < c^*$
            then $g^* \leftarrow g$
                    $c^* \leftarrow P(g)$
        return
for $v \in D_i$
    do $g' \leftarrow g + \{x_i \leftarrow v\}$
        if $P(g) < c^*$
            then BRANCH-AND-BOUND-COP-HELPER$(i + 1, g')$

Desiderata for DCOP

Why is **distributed** *important?*

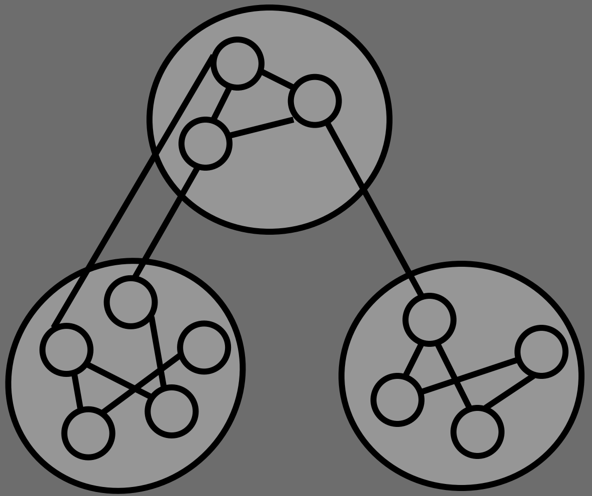
- Autonomy
- Communication cost
- Robustness (central point of failure)
- Privacy

Why is **asynchrony** *important?*

- Parallelism
- Robust to communication delays
- No global clock

Why are **theoretical guarantees** *important?*

- Optimal solutions feasible for special classes
- Bound on worst-case performance



loosely connected communities

State of the Art in DCOP (~2004)

Why have previous distributed methods failed to provide asynchrony + optimality?

- Branch and Bound
  - Backtrack condition - when cost exceeds upper bound
  - Problem – sequential, synchronous
- Asynchronous Backtracking
  - Backtrack condition - when constraint is unsatisfiable
  - Problem - only hard constraints allowed

- Observation Previous approaches backtrack *only* when sub-optimality is proven

Adopt: Asynchronous Distributed Optimization

First key idea -- Weak backtracking

- Adopt's backtrack condition – when lower bound gets too high

Why lower bounds?

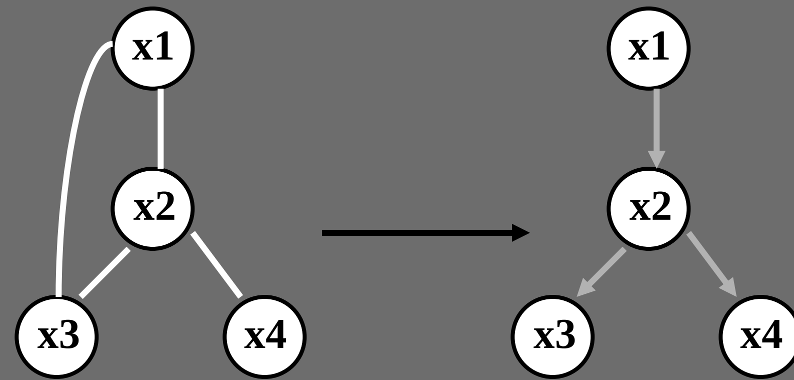
- allows asynchrony
- allows soft constraints
- allows quality guarantees

Any downside?

- backtrack *before* sub-optimality is proven
- solutions need revisiting
  - Second key idea -- Efficient reconstruction of abandoned solutions

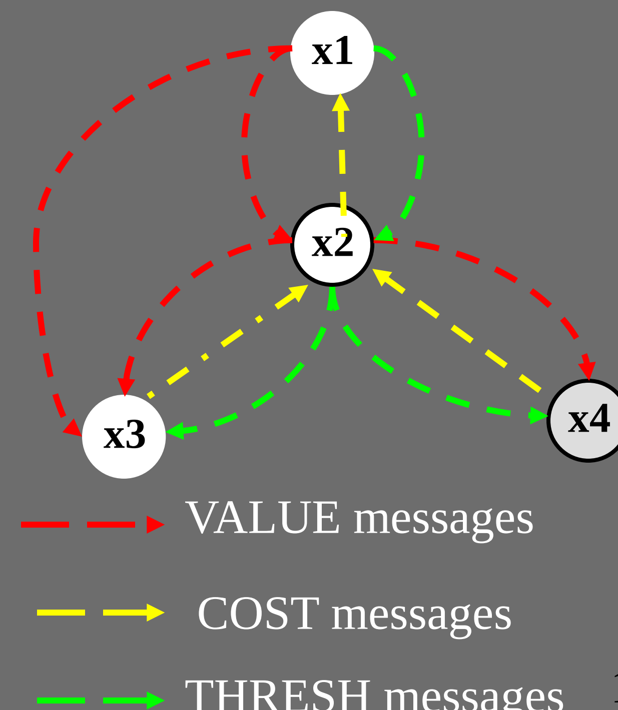
Adopt Algorithm

- Agents are ordered in a tree
  - constraints between ancestors/descendents
  - no constraints between siblings



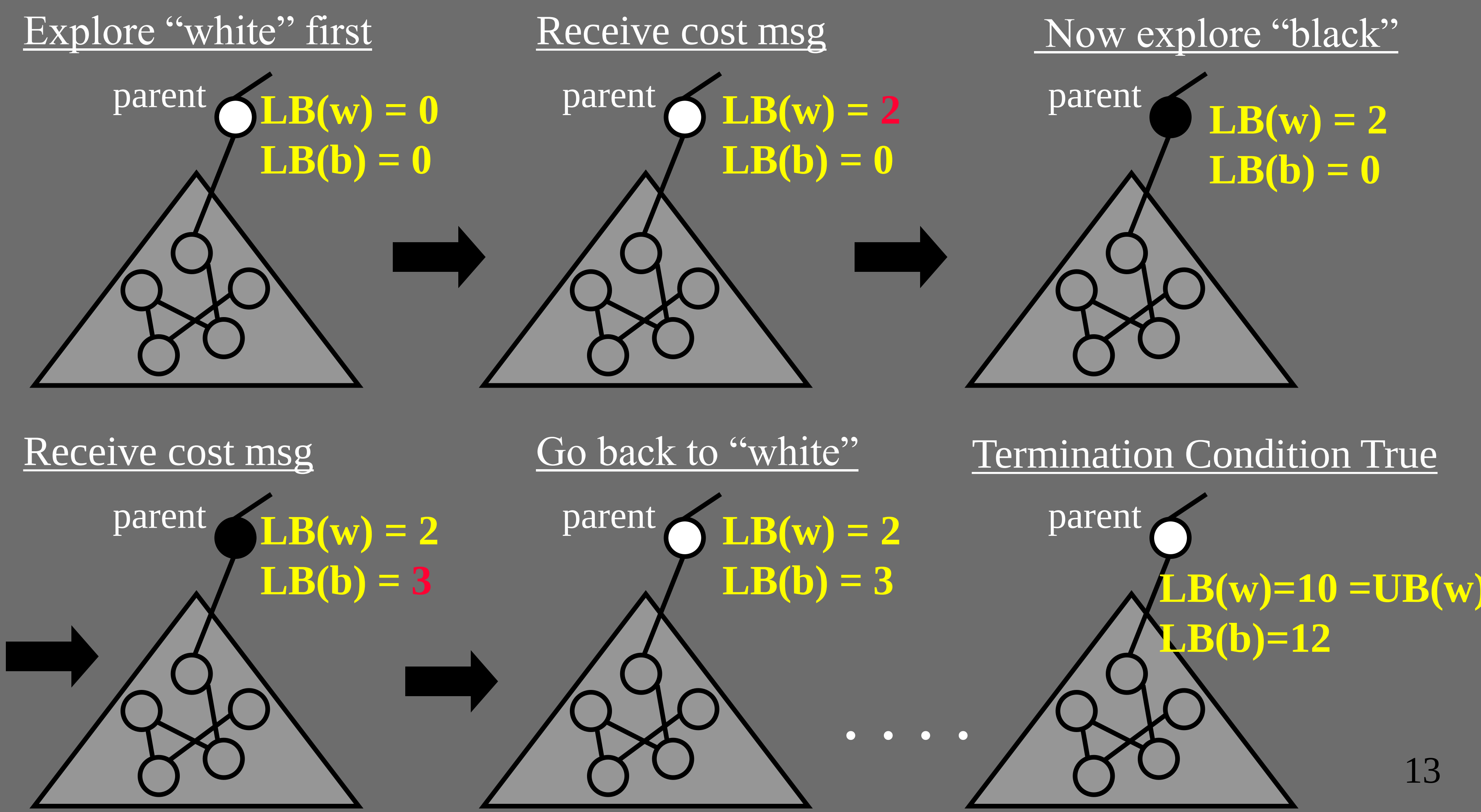
Constraint Graph Tree Ordering

- <u>Basic Algorithm:</u>
  - choose value with min cost
  - Loop until termination-condition true:
    - When receive message:
      - choose value with min cost
      - send **VALUE** message to descendents
      - send **COST** message to parent
      - send **THRESHOLD** message to child

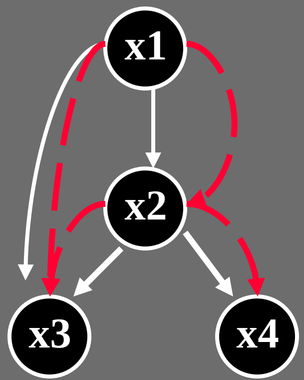


Weak Backtracking

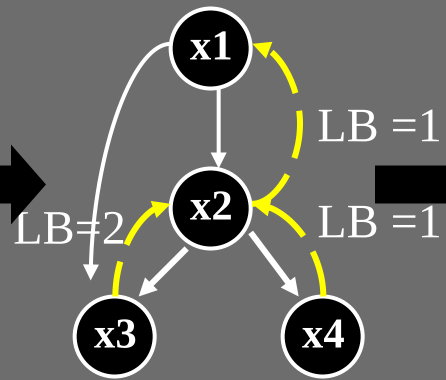
- Suppose parent has two values, “white” and “black”



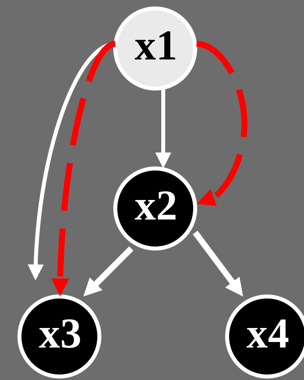
Example



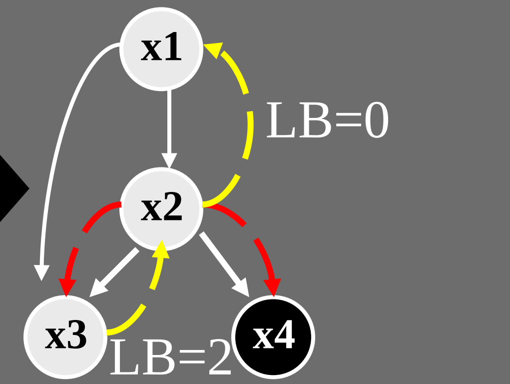
concurrently choose,
send to descendents



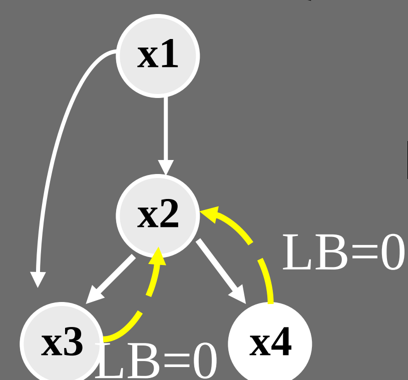
report lower bounds



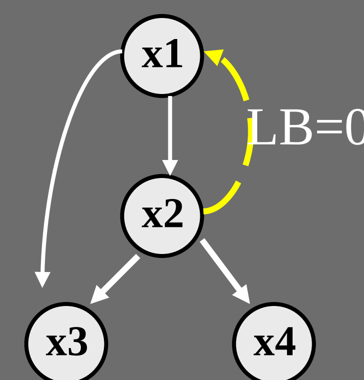
x1 switches value



x2, x3 switch value,
report new lower bounds
Note: x3's cost message to x2
is obsolete since x1 has changed
value, msg will be disregarded

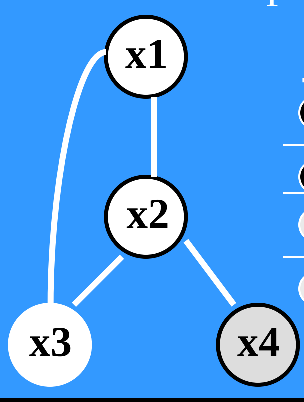


x2, x3 report new lower bounds



optimal solution

Constraint Graph



di	dj	f(di,dj)
●	●	1
●	○	2
○	●	2
○	○	0

Revisiting Abandoned Solutions

Problem

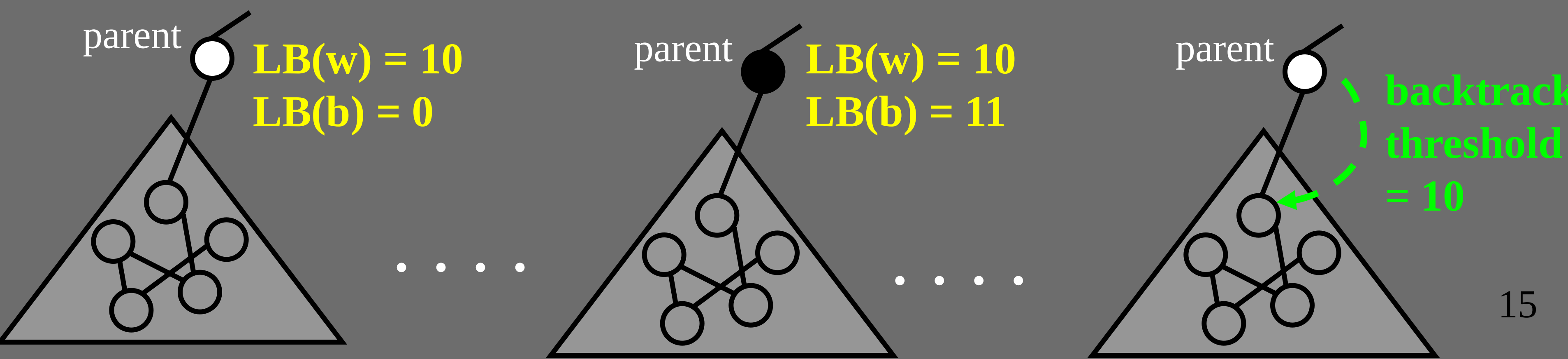
- reconstructing from scratch is **inefficient**
- remembering solutions is **expensive**

Solution

- *backtrack thresholds* – **polynomial space**
- control backtracking to **efficiently** re-search

Parent informs child of lower bound:

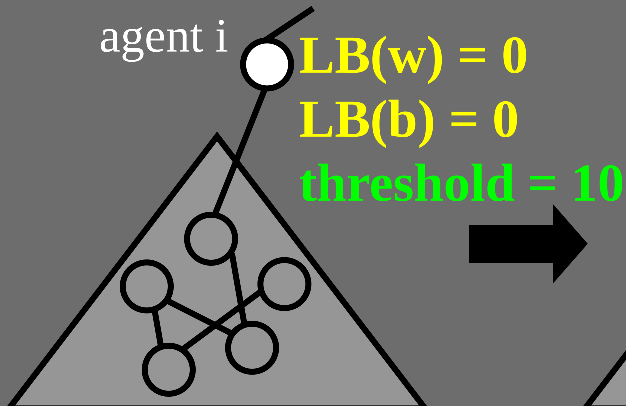
Explore "white" first | Now explore "black" | Return to "white"



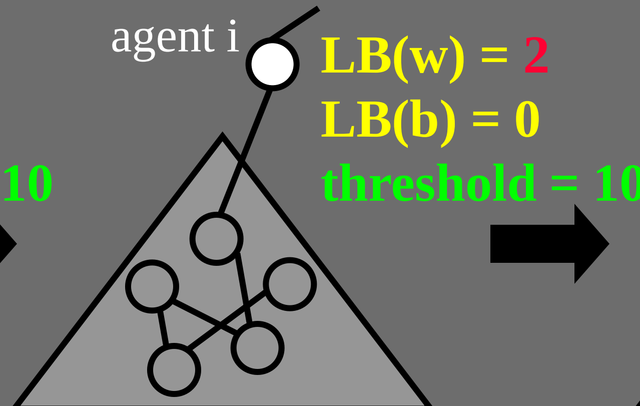
Backtrack Thresholds

- Suppose agent i received threshold = 10 from its parent

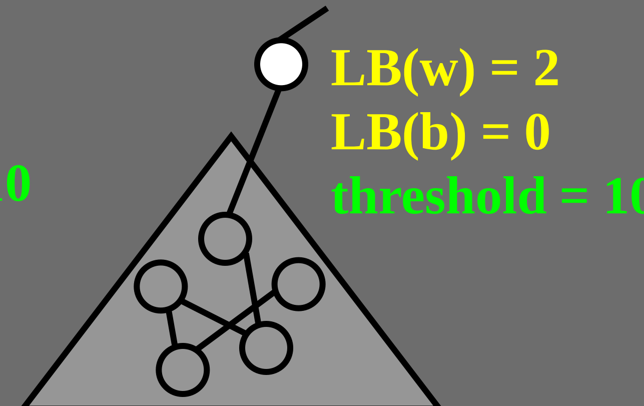
Explore "white" first



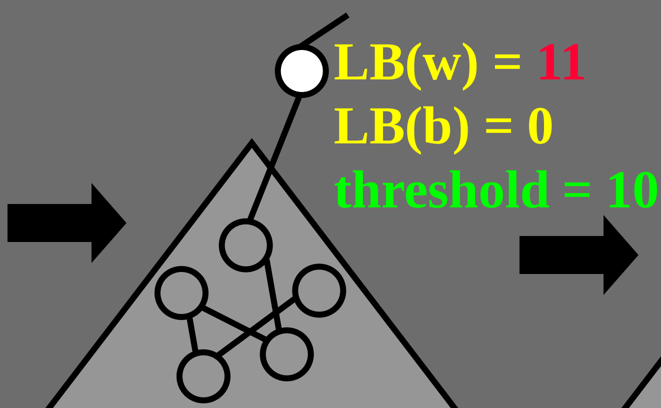
Receive cost msg



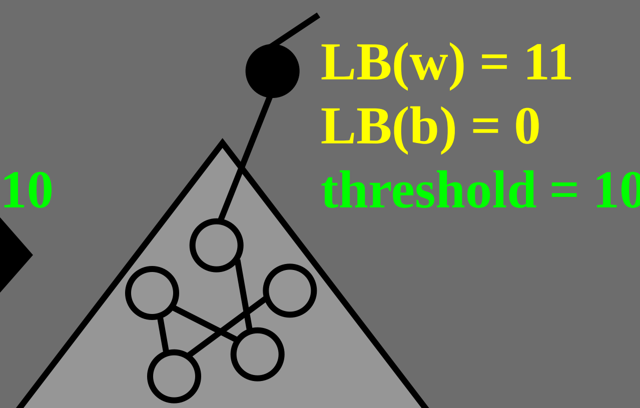
Stick with "white"



Receive more cost msgs

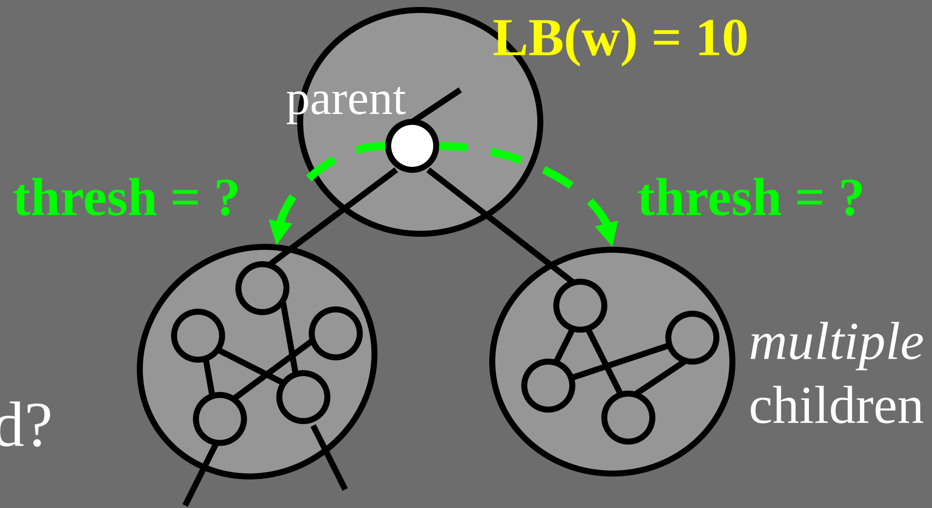


Now try black



Key Point: Don't change value until LB(current value) > threshold.

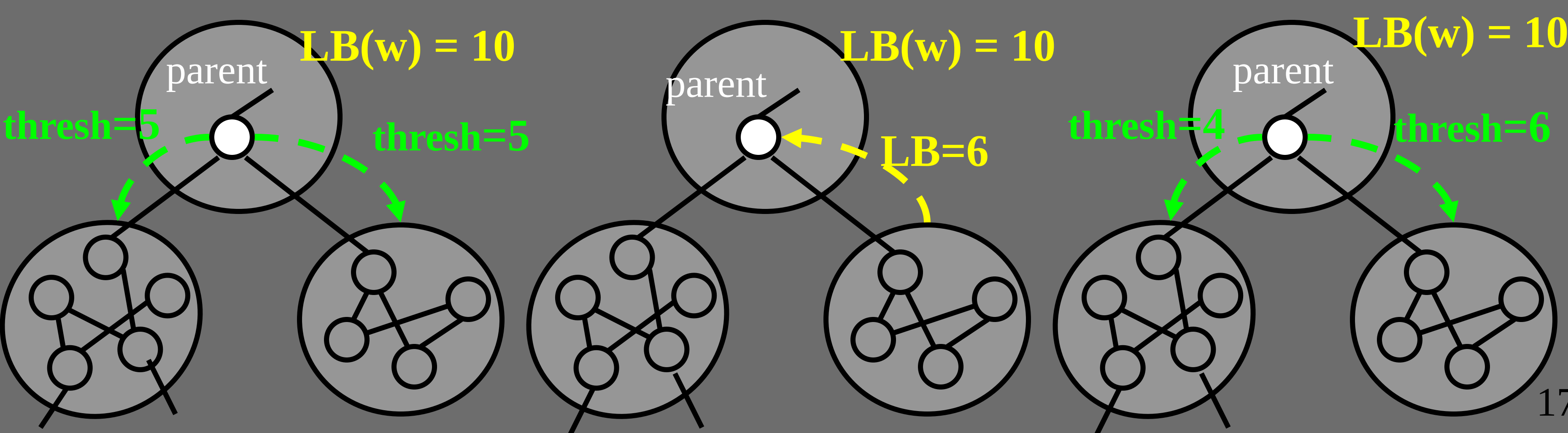
Backtrack thresholds with multiple children



How to correctly subdivide threshold?

Third key idea: **Dynamically rebalance threshold**

Time T_1 Time T_2 Time T_3



ADOPT (1)

Reset-Variables(d, c)

$lower\text{-}bound[d, c] \leftarrow 0$
$t[d, c] \leftarrow 0$
$upper\text{-}bound[d, c] \leftarrow \infty$
$context[d, c] \leftarrow \{\}$

Initialize$()$

$threshold \leftarrow 0$
$received\text{-}terminate \leftarrow$ FALSE
$current\text{-}context \leftarrow \{\}$
$\forall_{d \in D_i, c \in children}$ Reset-Variables(d, c)
$x_i \leftarrow d \in D_i$ which minimizes: my cost plus $\sum_{c \in children} lower\text{-}bound[d, c]$
Backtrack$()$

ADOPT(2)

```
HANDLE-VALUE(j, x_j)
if ¬ received-terminate
   then current-context[j] ← x_j
        for d ∈ D_i, c ∈ children such that context[d, c] is
                                   incompatible with current-context
            do RESET-VARIABLES(d, c)
        MAINTAIN-THRESHOLD-INVARIANT()
        BACKTRACK()

HANDLE-COST(k, context, lb, ub)
d ← context[i]
delete context[i]
if ¬ received-terminate
   then for (j, x_j) ∈ context and j is not my neighbor
            do current-context[j] ← x_j
        for d' ∈ D_i, c ∈ children such that context[d, c] is
                                   incompatible with current-context
            do RESET-VARIABLES(d', c)
        if context compatible with current-context
           then lower-bound[d, k] ← lb
                upper-bound[d, k] ← ub
                context[d, k] ← context
                MAINTAIN-CHILD-THRESHOLD-INVARIANT()
                MAINTAIN-THRESHOLD-INVARIANT()
        BACKTRACK()
```

ADOPT (3)

BACKTRACK()

if *threshold* $= \min_{d \in D_i} \text{cost}(d) + \sum_{c \in children} \textit{upper-bound}[d, c]$
    then $x_i \leftarrow \arg\min_{d \in D_i} \text{cost}(d) + \sum_{c \in children} \textit{upper-bound}[d, c]$
elseif *threshold* $< \text{cost}(x_i) + \sum_{c \in children} \textit{lower-bound}[x_i, c]$
    then $x_i \leftarrow \arg\min_{d \in D_i} \text{cost}(d) + \sum_{c \in children} \textit{lower-bound}[d, c]$
$\forall_{k \in neighbors \wedge k \text{ has lower priority}} k.$HANDLE-VALUE$(i, x_i)$
MAINTAIN-ALLOCATION-INVARIANT()
if *threshold* $= \min_{d \in D_i} \text{cost}(d) + \sum_{c \in children} \textit{upper-bound}[d, c]$ and
    (*received-terminate* or I am root)
    then *current-context*$[i] \leftarrow x_i$
        $\forall_{c \in children} c.$HANDLE-TERMINATE(*current-context*)
        exit
parent .HANDLE-COST(*current-context*,
    $\min_{d \in D_i} \text{cost}(d) + \sum_{c \in children} \textit{lower-bound}[d, c]$,
    $\min_{d \in D_i} \text{cost}(d) + \sum_{c \in children} \textit{upper-bound}[d, c]$)

ADOPT (4)

MAINTAIN-THRESHOLD-INVARIANT()

if $\textit{threshold} < \min_{d \in D_i} \text{cost}(d) + \sum_{c \in \textit{children}} \textit{lower-bound}[d, c]$
  then $\textit{threshold} \leftarrow \min_{d \in D_i} \text{cost}(d) + \sum_{c \in \textit{children}} \textit{lower-bound}[d, c]$
if $\textit{threshold} > \min_{d \in D_i} \text{cost}(d) + \sum_{c \in \textit{children}} \textit{upper-bound}[d, c]$
  then $\textit{threshold} \leftarrow \min_{d \in D_i} \text{cost}(d) + \sum_{c \in \textit{children}} \textit{upper-bound}[d, c]$

MAINTAIN-ALLOCATION-INVARIANT()

while $\textit{threshold} > \text{cost}(x_i) + \sum_{c \in \textit{children}} t[x_i, c]$
  do $\textit{chosen} \leftarrow c' \in \textit{children}$ such that $\textit{upper-bound}[x_i, c'] > t[x_i, c']$
    $t[x_i, \textit{chosen}] \leftarrow t[x_i, \textit{chosen}] + 1$
while $\textit{threshold} < \text{cost}(x_i) + \sum_{c \in \textit{children}} t[x_i, c]$
  do $\textit{chosen} \leftarrow c' \in \textit{children}$ such that $\textit{lower-bound}[x_i, c'] < t[x_i, c']$
    $t[x_i, \textit{chosen}] \leftarrow t[x_i, \textit{chosen}] - 1$
$\forall_{c \in \textit{children}}$ c.HANDLE-THRESHOLD($t[x_i, \textit{chosen}], \textit{current-context}$)

MAINTAIN-CHILD-THRESHOLD-INVARIANT()

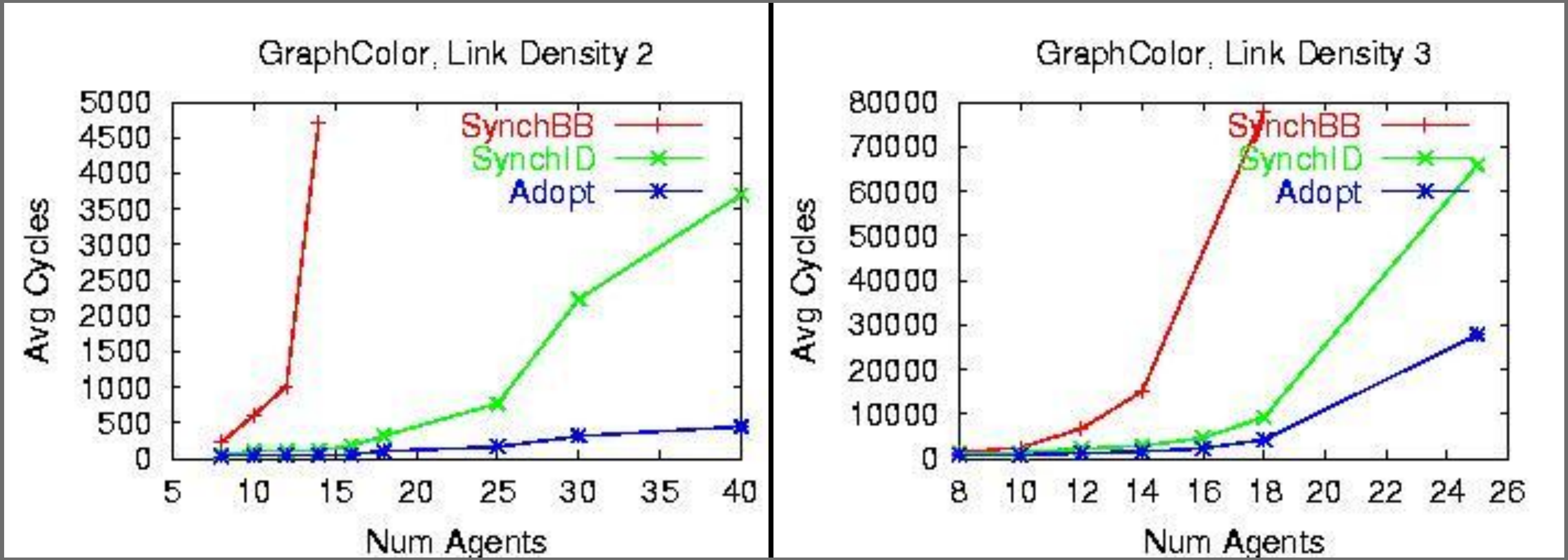
for $d \in D_i, c \in \textit{children}$
  do if $\textit{lower-bound}[d, c] > t[d, c]$
    then $t[d, c] \leftarrow \textit{lower-bound}[d, c]$
for $d \in D_i, c \in \textit{children}$
  do if $\textit{upper-bound}[d, c] < t[d, c]$
    then $t[d, c] \leftarrow \textit{upper-bound}[d, c]$

ADOPT (5)

```
HANDLE-THRESHOLD(t, context)
if context is compatible with current-context
    then threshold ← t
         MAINTAIN-THRESHOLD-INVARIANT()
         BACKTRACK()
```

```
HANDLE-TERMINATE(context)
received-terminate ← TRUE
current-context ← context
BACKTRACK()
```

Evaluation of Speedups

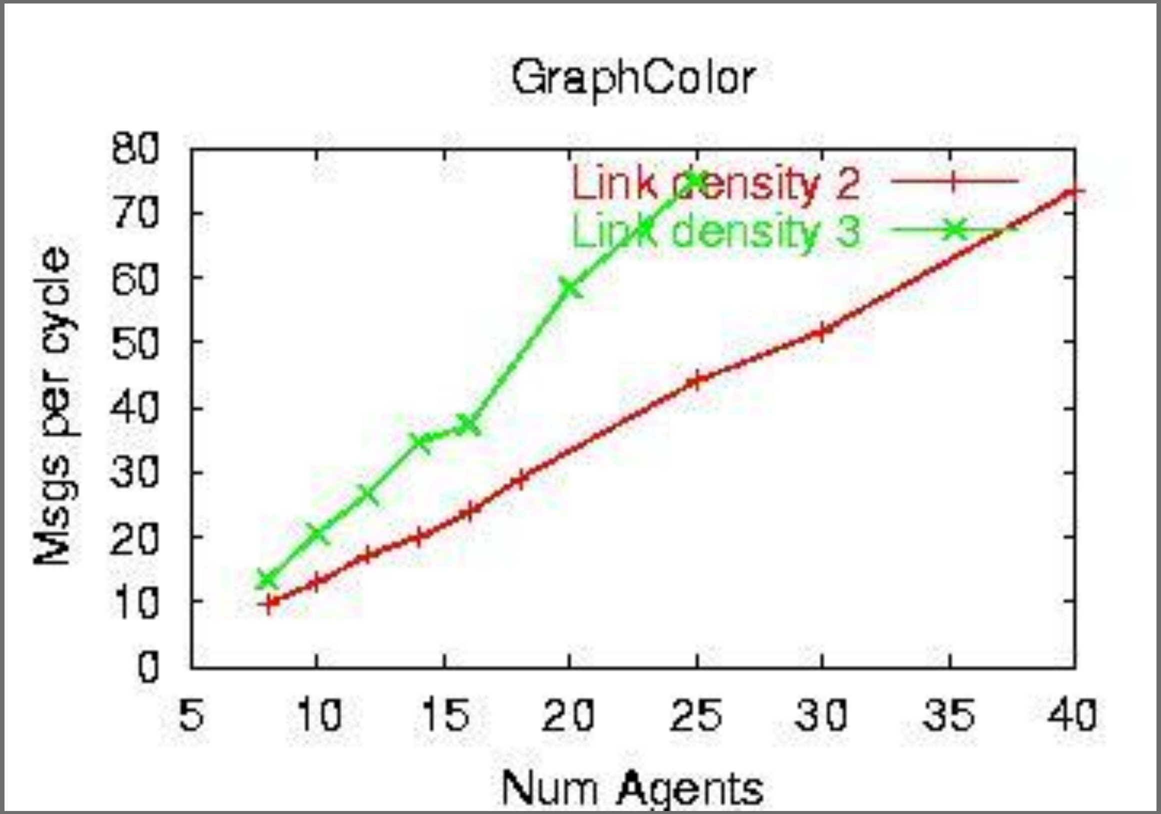


Conclusions

• Adopt's lower bound search method and parallelism yields significant efficiency gains

• Sparse graphs (density 2) solved **optimally, efficiently** by Adopt.

Number of Messages



Conclusion

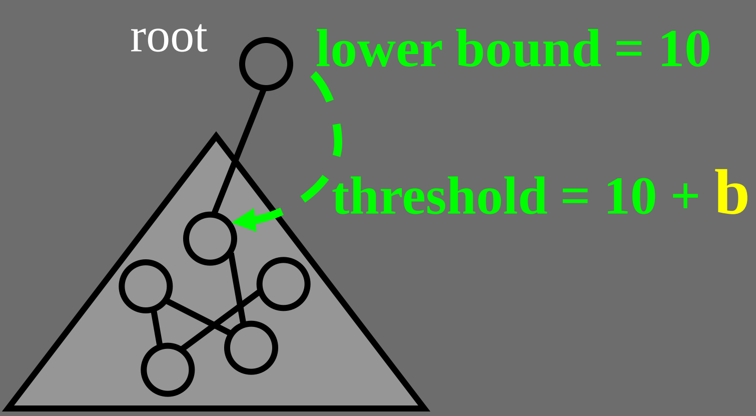
- Communication grows linearly
  - only local communication (no broadcast)

Bounded error approximation

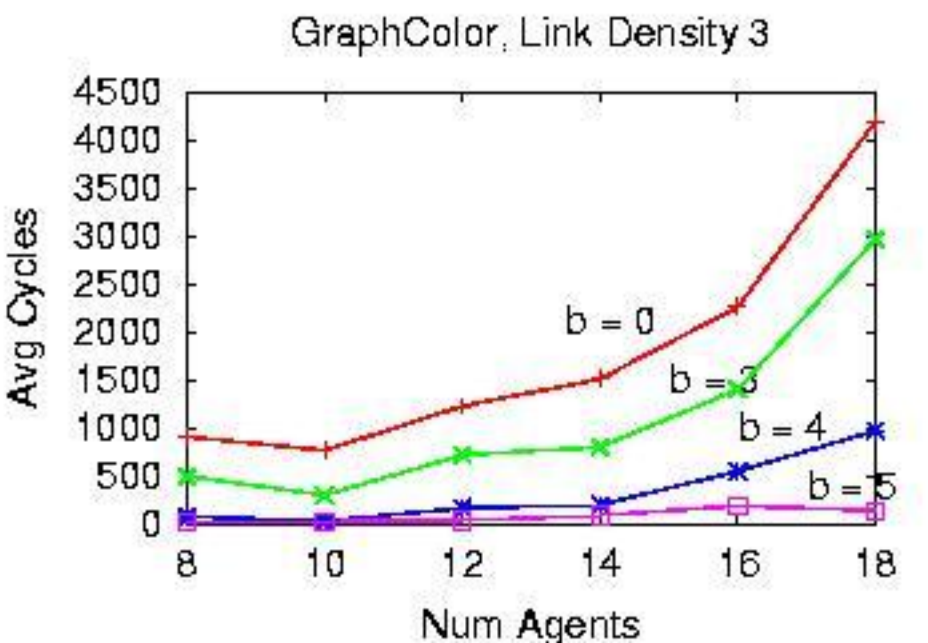
- *Motivation* Quality control for approximate solutions
- *Problem* User provides error bound **b**
- *Goal* Find any solution **S** where

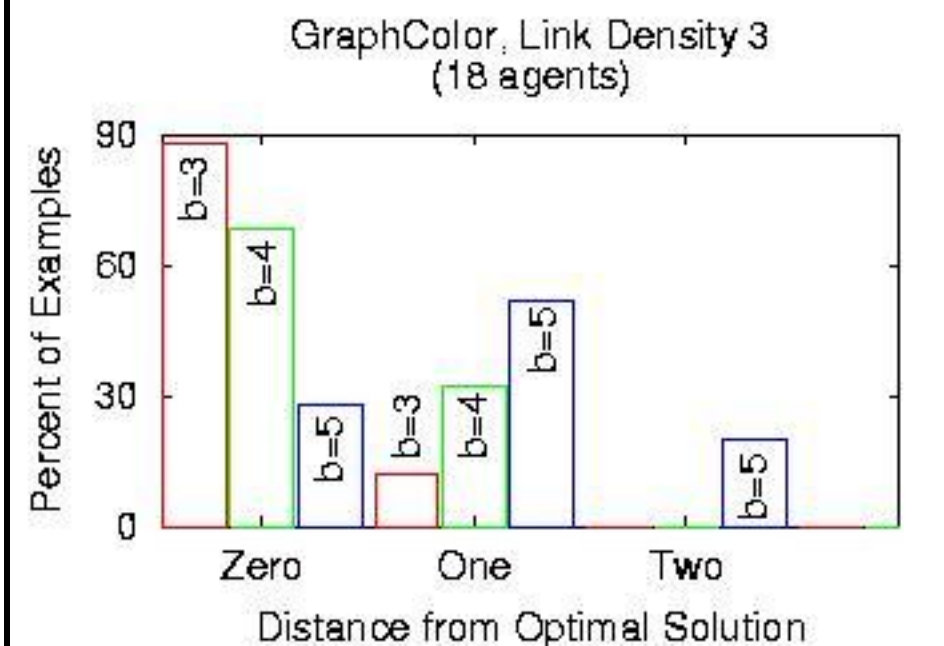
  cost(**S**) $\leq$ cost(optimal soln) + **b**

• Fourth key idea: Adopt's lower-bound based search method naturally leads to bounded error approximation!



Evaluation of Bounded Error





Conclusion

- Time-to-solution decreases as **b** is increased.
- Plus: Guaranteed worst-case performance!

Optimality of Adopt

- For finite DCOPs with binary non-negative constraints, Adopt is **guaranteed to terminate** with the **globally optimal solution**

Adopt summary – Key Ideas

- **Optimal**, **asynchronous** algorithm for DCOP
  - polynomial space at each agent
- Weak Backtracking
  - **lower bound** based search method
  - Parallel search in independent subtree
- Efficient reconstruction of abandoned solutions
  - **backtrack** thresholds to control backtracking
- Bounded error approximation
  - sub-optimal solutions **faster**
  - **bound** on worst-case performance

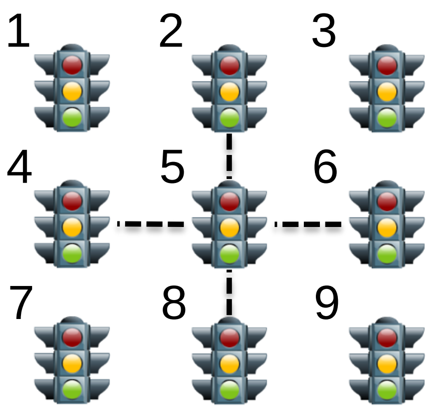
Traffic Light Control DCOP Formulation

$\rho_{i,j}$ density of vehicle in the lane $i \to j$

$\beta_{i,j} = {}^{\rho_{i,j}}/_{\sum_k \rho_{k,j}}$ fraction of traffic at intersection *j* coming from *I*

$\gamma_{i,j}$ 1.5 (if *i* and *j* synchronized), 2 otherwise

$\tau_{i,j} \in \{0, 1, 1.5, 2\}$ degree of alignment of synchronization with traffic



Plan run by agent i	Plan run by agent j	τ
$\oplus$	$\oplus$	0
$\ominus$	$\oplus$	1
$\oplus$	$\ominus$	1.5
$\ominus$	$\ominus$	2

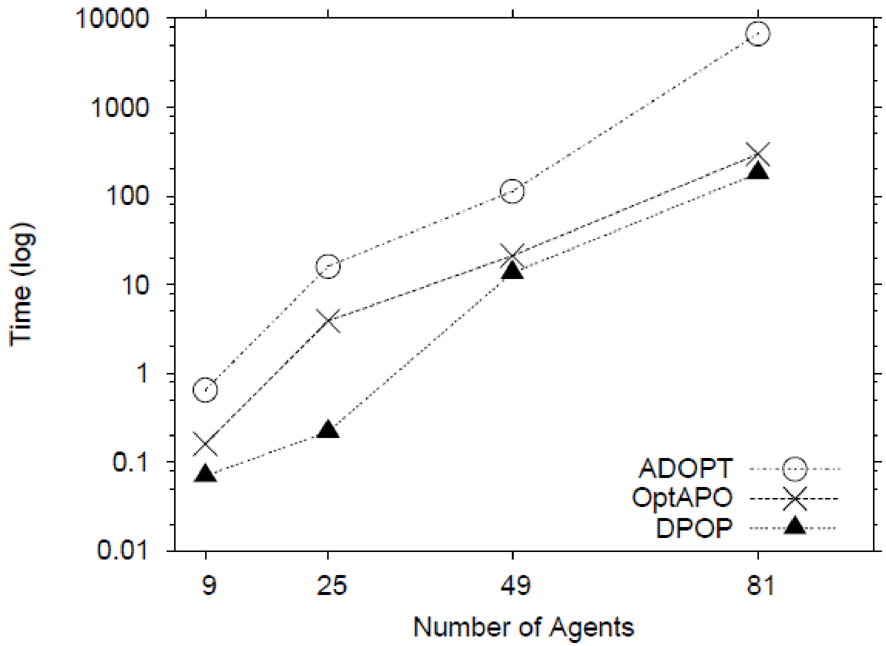
+ synchronized in direction of max traffic
- synchronized in other directions

Constraint cost

$$f_{i,j} = \beta_{i,j}\, \tau_{i,j}\, \gamma_{i,j}$$

from: R. *Junges* and A. L. C. Bazzan. *Evaluating the performance of DCOP algorithms* in a *real world*, *dynamic problem*. In AAMAS 2008, pages 599–606, 2008

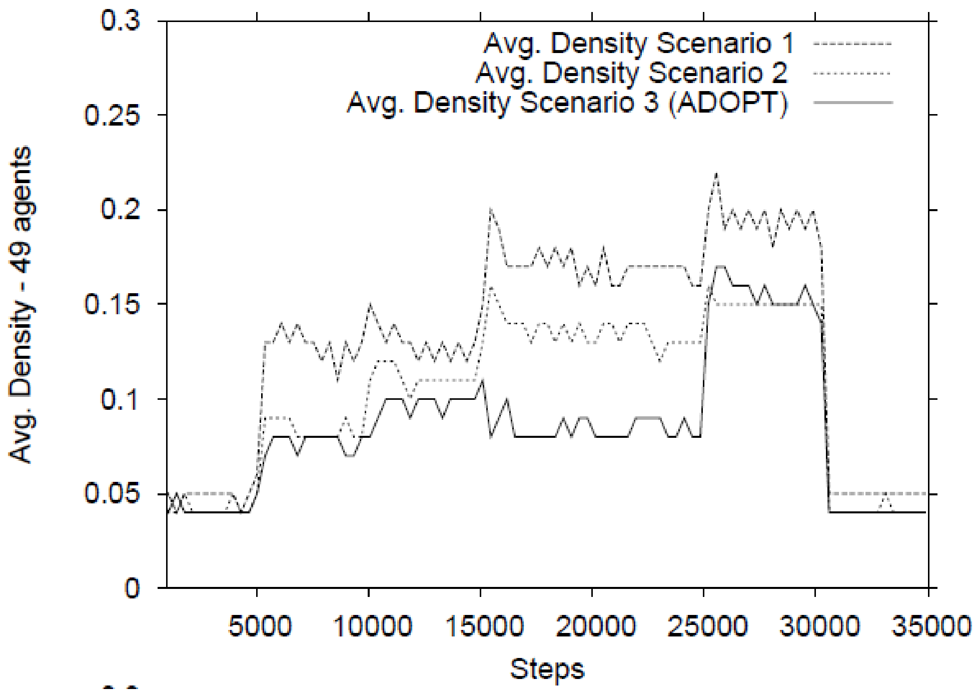
Results - Complexity



Alg.	Nb. of Msgs	Msgs size (bytes)	Total (MB)
ADOPT	358962.4 ± 1690.4	125.40 ± 20.79	43958
OptAPO	4015.66 ± 68.70	93.45 ± 8.15	366
DPOP	322	69286.78 ± 721.12	21787

Results - Quality

Net.	Experiment	Stopped Veh.	Density
3x3	Case 1	10.21 ± 2.28	0.29 ± 0.06
	Case 2	6.72 ± 1.37	0.23 ± 0.03
	Case 3 (ADOPT)	1.88 ± 0.53	0.11 ± 0.02
	Case 3 (OptApo)	1.97 ± 0.60	0.11 ± 0.01
	Case 3 (DPOP)	2.42 ± 0.46	0.13 ± 0.02
5x5	Case 1	9.22 ± 2.36	0.28 ± 0.04
	Case 2	6.95 ± 1.82	0.23 ± 0.03
	Case 3 (ADOPT)	1.78 ± 0.80	0.11 ± 0.01
	Case 3 (OptApo)	1.90 ± 0.59	0.12 ± 0.01
	Case 3 (DPOP)	3.08 ± 0.40	0.14 ± 0.02
7x7	Case 1	8.91 ± 2.29	0.27 ± 0.04
	Case 2	5.82 ± 1.31	0.21 ± 0.02
	Case 3 (ADOPT)	1.97 ± 0.50	0.12 ± 0.01
	Case 3 (OptApo)	1.91 ± 0.49	0.12 ± 0.01
	Case 3 (DPOP)	2.90 ± 0.62	0.14 ± 0.01
9x9	Case 1	8.05 ± 2.17	0.14 ± 0.01
	Case 2	5.35 ± 0.97	0.10 ± 0.01
	Case 3 (ADOPT)	2.04 ± 0.77	0.12 ± 0.01
	Case 3 (OptAPO)	1.98 ± 0.85	0.11 ± 0.01
	Case 3 (DPOP)	2.88 ± 0.72	0.13 ± 0.02



Case 1 = worst case
Case 2 = static optimum
Case 3 = dynamic optimization

Conclusion

- Distributed constraint optimization is a general widely applicable model
- **Optimal (complete) asynchronous** algorithms exist for DCOPs with **binary, non-negative** constraints
  - ADOPT, OptAPO, DPOP
  - Different computation and communication complexity profiles
- ADOPT
  - optimal, asynchronous algorithm for DCOP
  - polynomial space at each agent
- Reading:
  - [Vidal] – Chapter 2
  - P. J. Modi et al. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence Journal*, 161(1-2):149–180, January 2005.