Adopt Algorithm for Distributed Constraint Optimization

Pragnesh Jay Modi

Information Sciences Institute & Department of Computer Science
University of Southern California
http://www.isi.edu/~modi

Distributed Optimization Problem

"How do a set of agents optimize over a set of alternatives that have varying degrees of global quality?"

Examples

- allocating resources
- constructing schedules
- planning activities

Difficulties

- No global control/knowledge
- Localized communication
- Quality guarantees required
- Limited time

Approach

- Constraint Based Reasoning
  - Distributed Constraint Optimization Problem (DCOP)
- Adopt algorithm
  - First-ever **distributed, asynchronous, optimal** algorithm for DCOP
  - Efficient, polynomial-space
- Bounded error approximation
  - Principled solution-quality/time-to-solution **tradeoffs**

Constraint Representation

Why constraints for multiagent systems?

- Constraints are natural, general, simple
  - Many successful applications
- Leverage existing work in AI
  - Constraints Journal, Conferences
- Able to model coordination, conflicts, interactions, etc…

Key advances

- **Distributed** constraints
- Constraints have **degrees of violation**

Distributed Constraint Optimization (DCOP)

Given

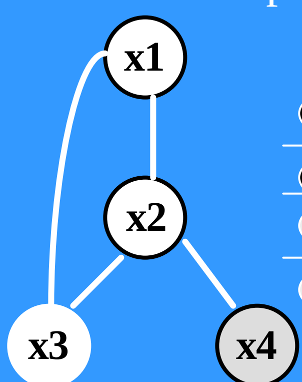
- Variables {x1, x2, …, xn}, each assigned to an agent
- Finite, discrete domains D1, D2, … , Dn,
- For each xi, xj, valued constraint fij: Di x Dj → N.

Goal

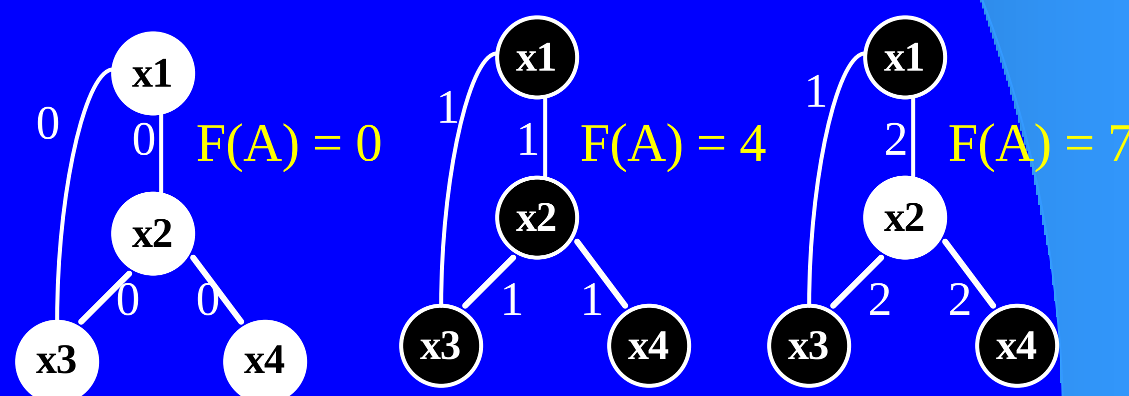
- Find complete assignment A that minimizes F(A) where,

$$F(A) = \Sigma f_{ij}(d_i,d_j), \quad x_i \leftarrow d_i, x_j \leftarrow d_j \text{ in } A$$

Constraint Graph



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



Existing Methods

Theoretical guarantee	Synchronous	Asynchronous
Optimization	**Branch and Bound** (Hirayama97)	**?**
Satisfaction	—	**Asynchronous Backtracking** (Yokoo92)
No guarantee	—	**Iterative Improvement** (Yokoo96)

Execution Model

Desiderata for DCOP

Why is **distributed** *important?*

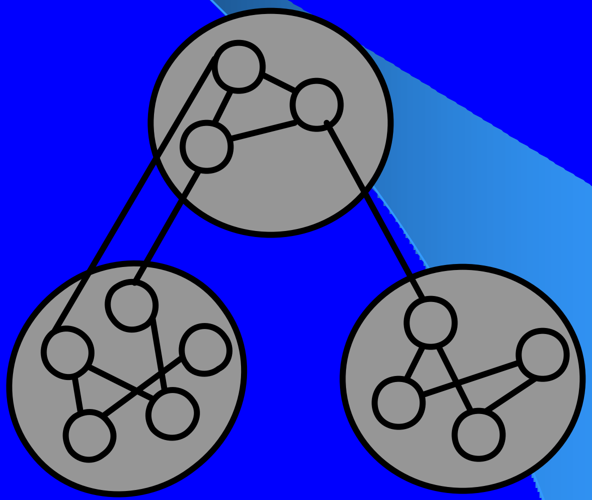
- Autonomy
- Communication cost
- Robustness (central point of failure)
- Privacy

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

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 <u>lower bound</u> gets too high

Why lower bounds?

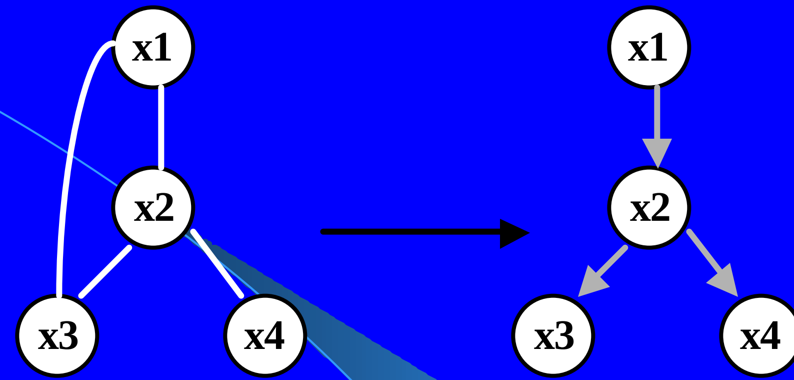
- allows asynchrony
- allows soft constraints
- allows quality guarantees

Any downside?

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

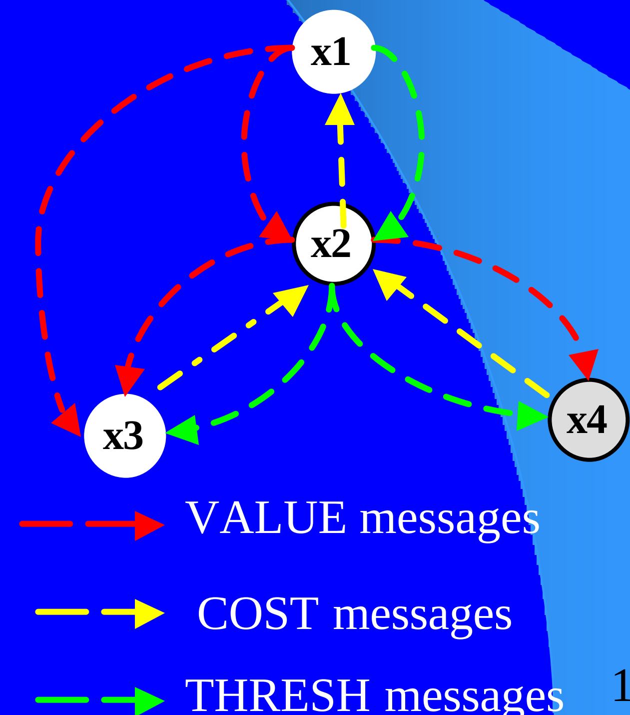
Adopt Algorithm

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



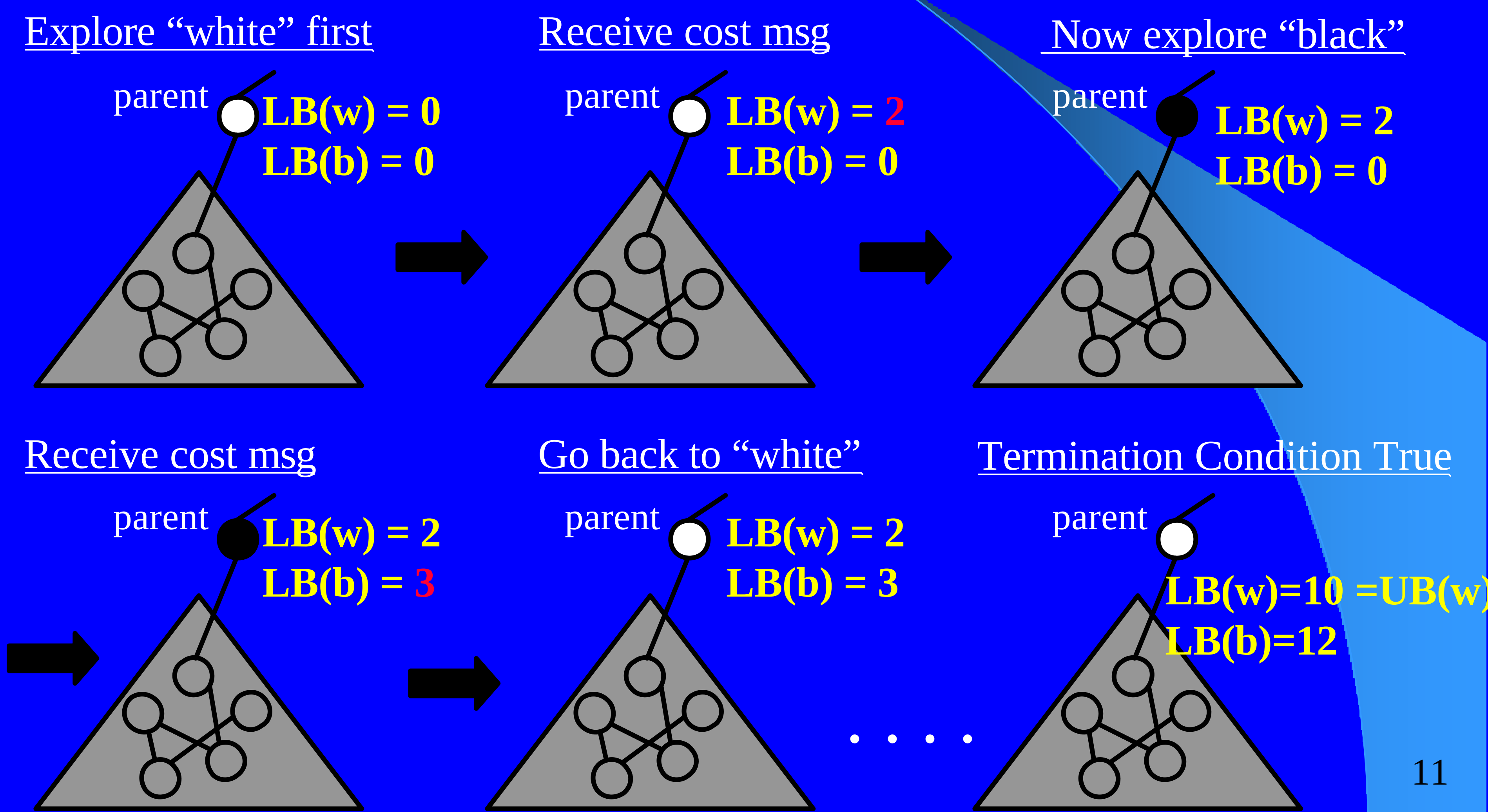
Constraint Graph   Tree Ordering

- <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

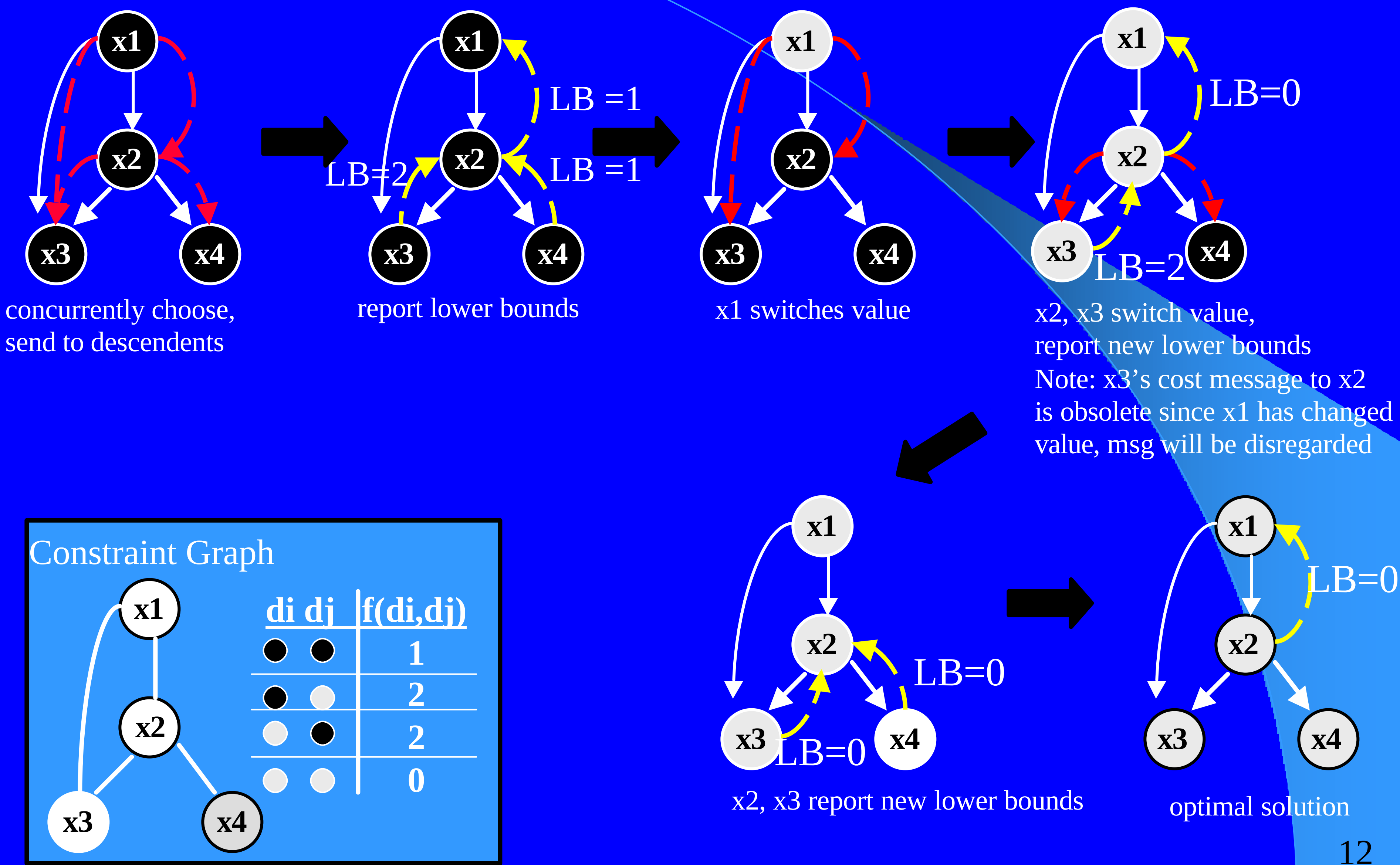


Weak Backtracking

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



Example



x1
x2
x3
x4
concurrently choose,
send to descendents
LB =1
LB =1
LB=2
report lower bounds
x1 switches value
LB=0
LB=2
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
LB=0
LB=0
x2, x3 report new lower bounds
LB=0
optimal solution
Constraint Graph
di dj f(di,dj)
1
2
2
0

Revisiting Abandoned Solutions

Problem

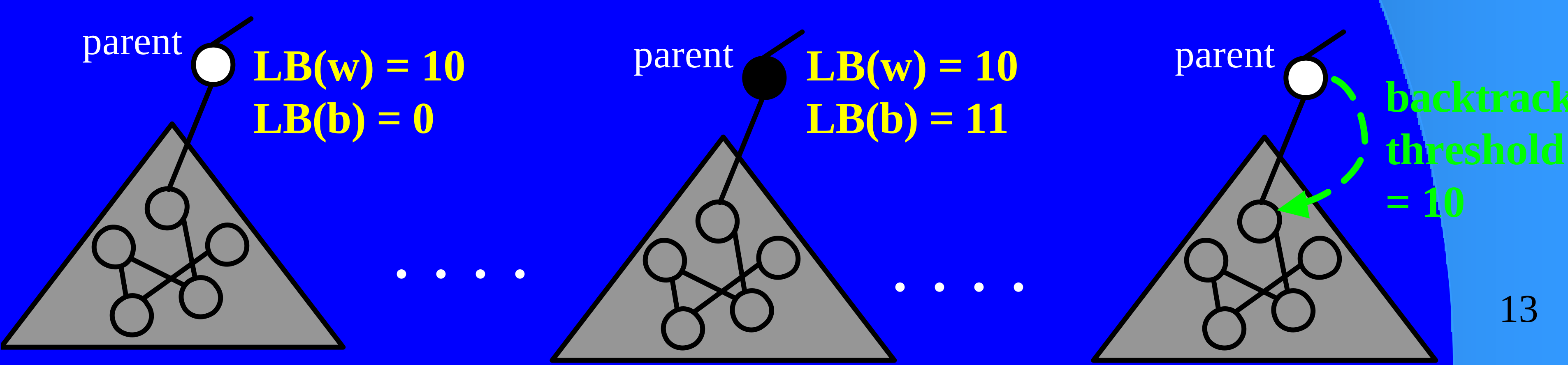
- reconstructing from scratch is **inefficient**
- remembering solutions is **expensive**

Solution

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

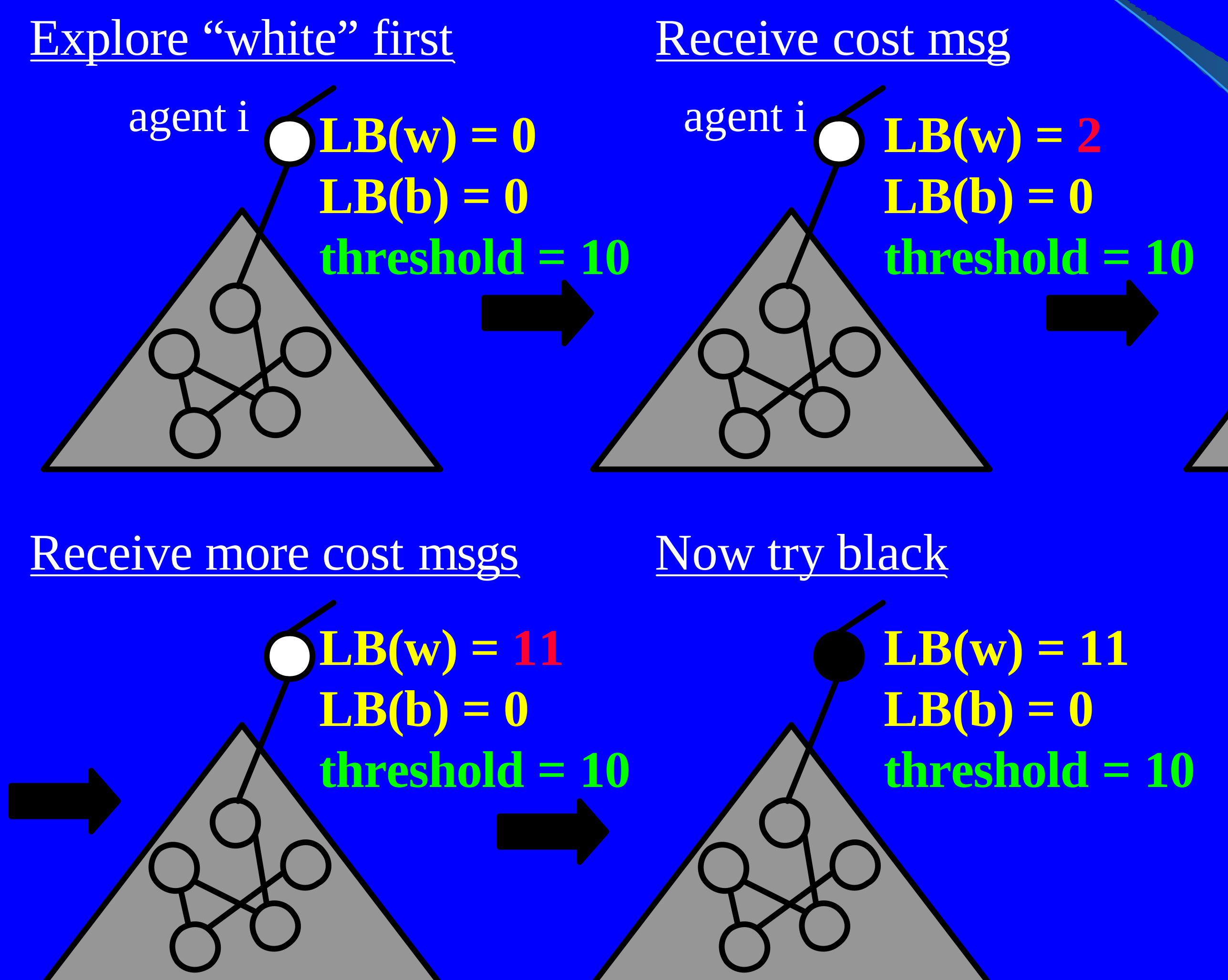
Parent informs child of lower bound:

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



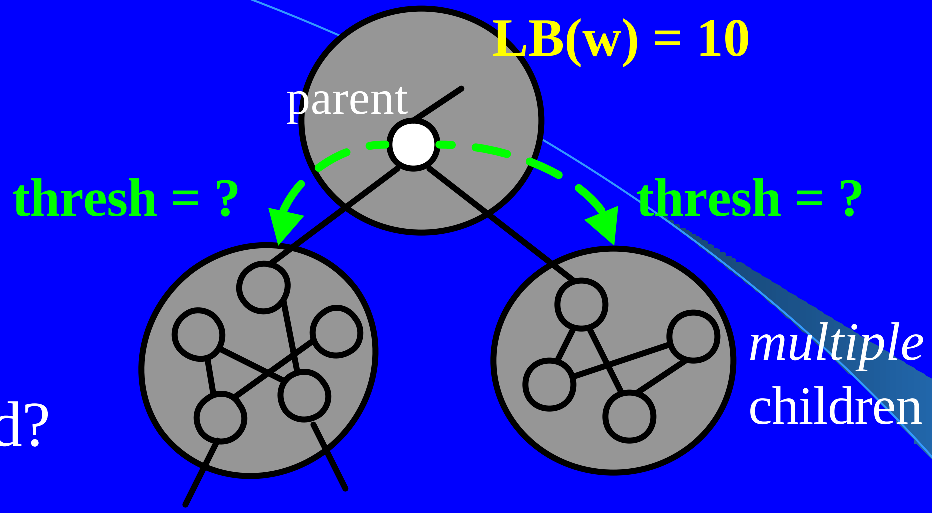
Backtrack Thresholds

- Suppose agent i received threshold = 10 from its parent



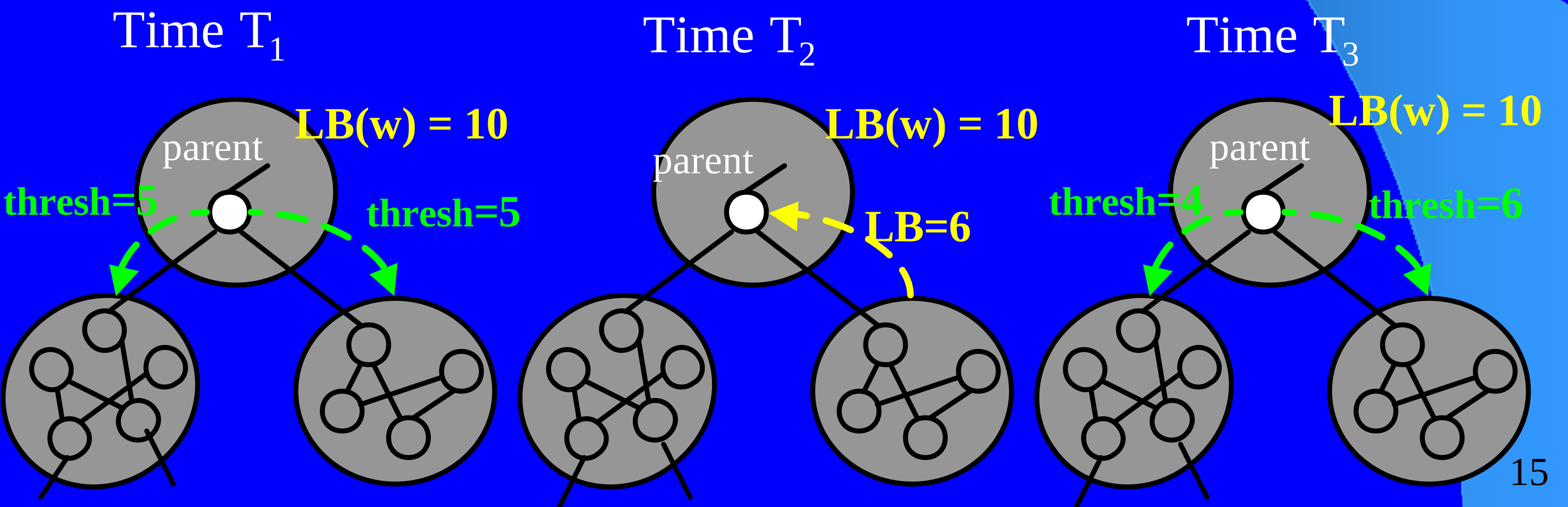
Key Point: Don't change value until LB(current value) > threshold.

Backtrack thresholds with multiple children

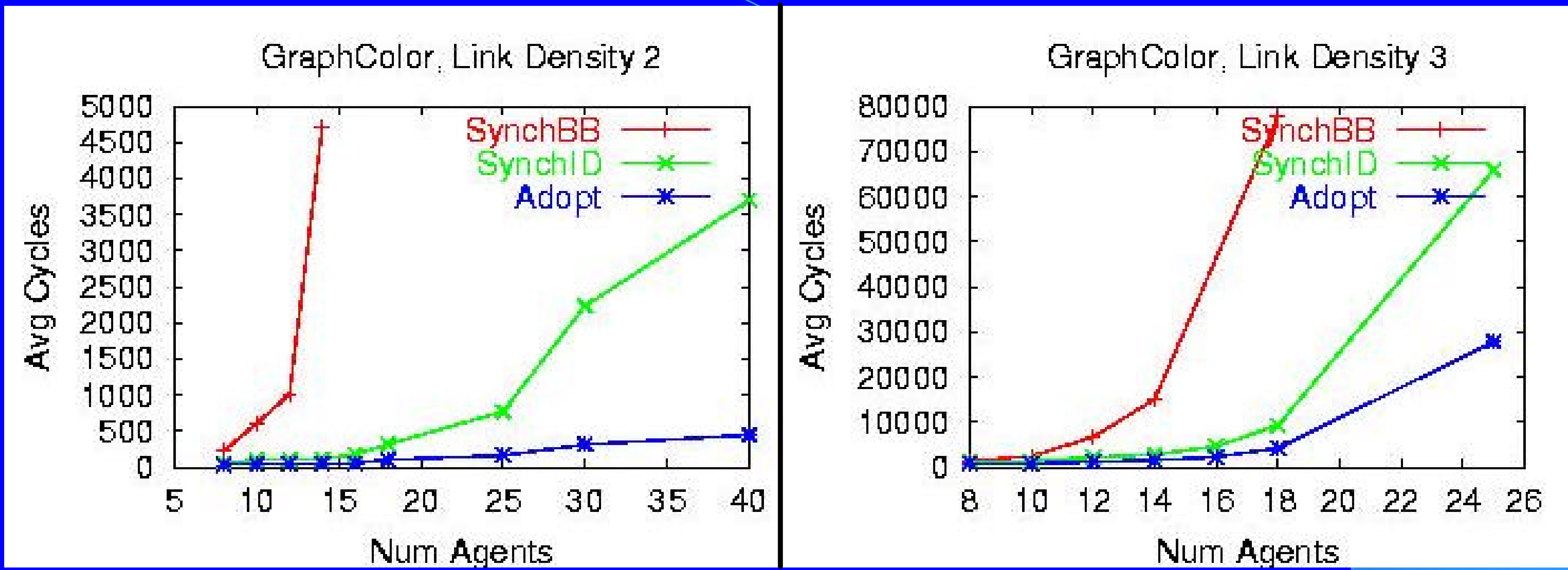


How to correctly subdivide threshold?

Third key idea: **Dynamically rebalance threshold**



Evaluation of Speedups

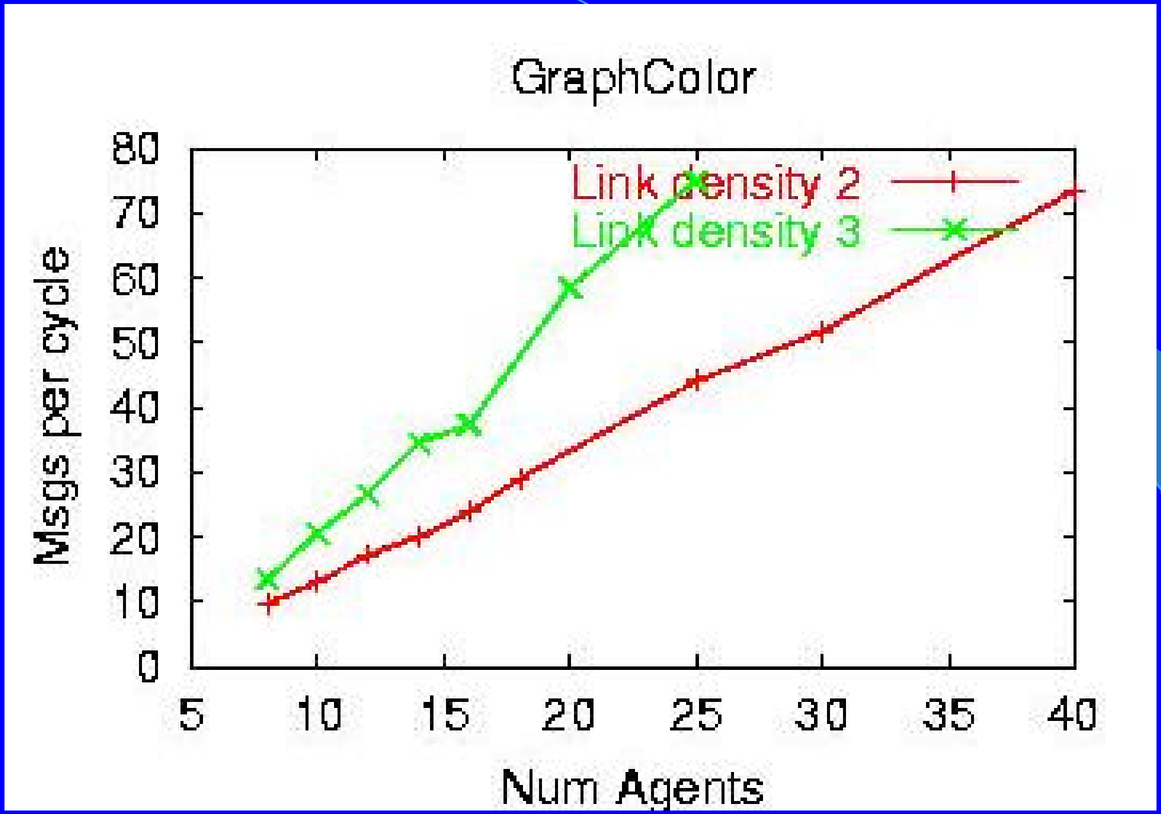


Conclusions

• 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

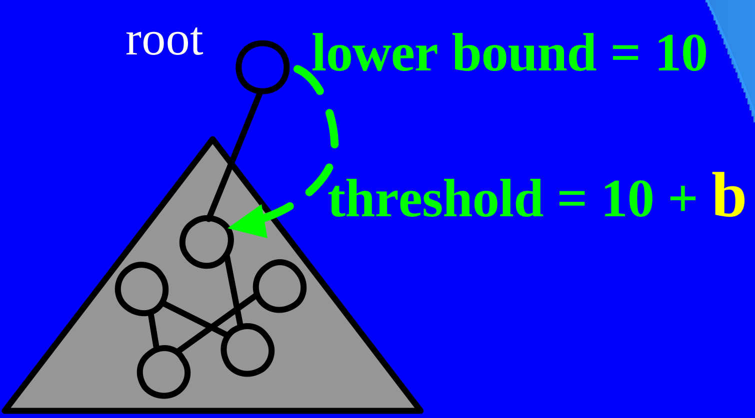
- Communication grows linearly
  - only local communication (no broadcast)

Bounded error approximation

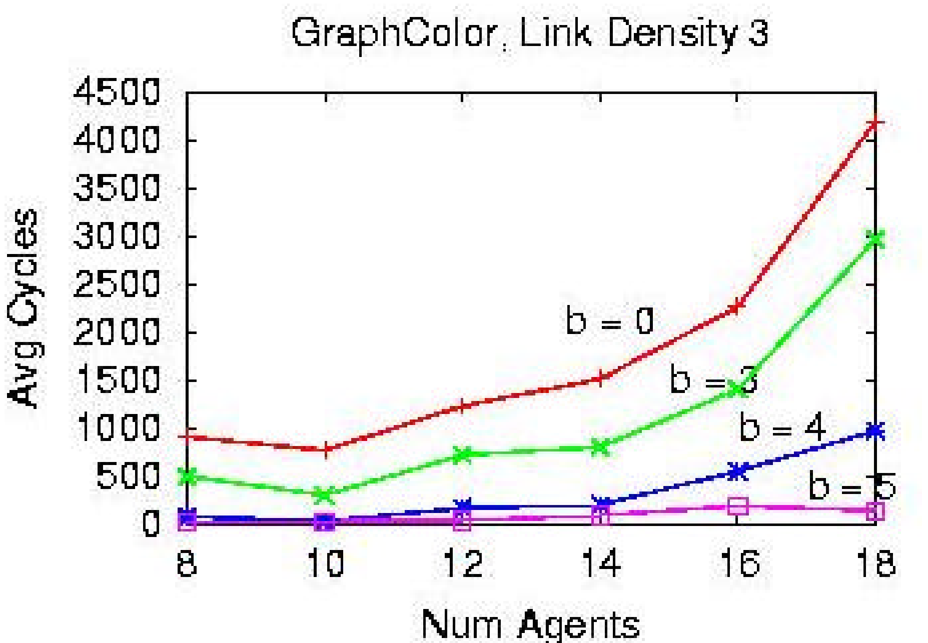
- *Motivation* Quality control for approximate solutions
- *Problem* User provides error bound **b**
- *Goal* Find any solution **S** where

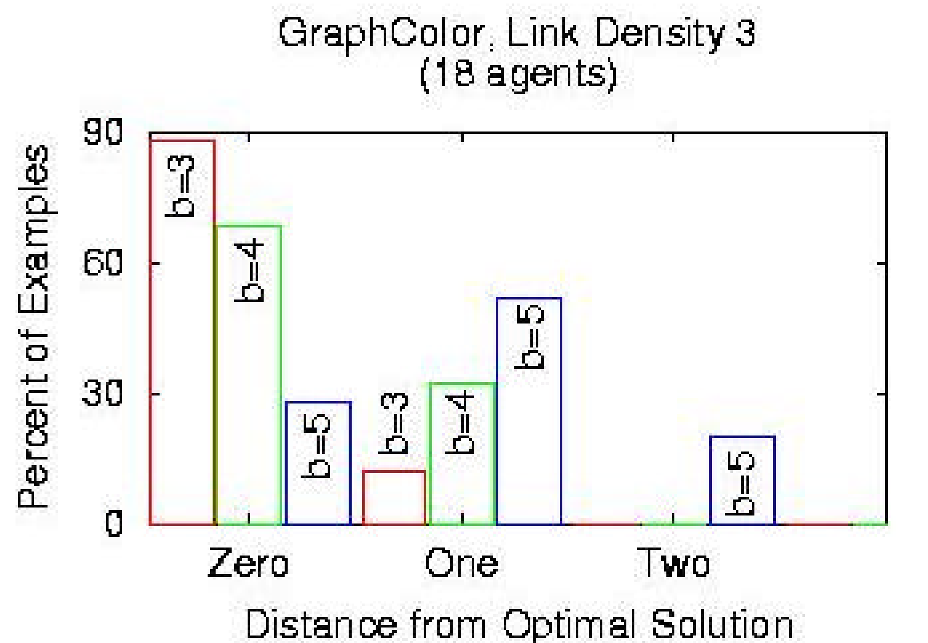
  cost(**S**) $\leq$ cost(optimal soln) + **b**

• 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!

Adopt summary – Key Ideas

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