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 \{x_1, x_2, \ldots, x_n\}, each assigned to an agent
- Finite, discrete domains D1, D2, \ldots, Dn,
- For each xi, xj, valued constraint fij: Di x Dj \rightarrow N.

Goal
- Find complete assignment A that minimizes F(A) where,
  \[ F(A) = \sum f_{ij}(d_i,d_j), \quad x_i \leftarrow d_i, x_j \leftarrow d_j \text{ in } A \]

Constraint Graph

<table>
<thead>
<tr>
<th>di</th>
<th>dj</th>
<th>f(di,dj)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ F(A) = 0 \quad (x_1, x_2) \]
\[ F(A) = 4 \quad (x_1, x_3) \]
\[ F(A) = 7 \quad (x_1, x_4) \]
<table>
<thead>
<tr>
<th>Optimization</th>
<th>Satisfaction</th>
<th>Execution Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theoretical guarantee</td>
<td>No guarantee</td>
<td>Synchronous</td>
</tr>
<tr>
<td>Branch and Bound (Hirayama97)</td>
<td>Asynchronous Backtracking (Yokoo92)</td>
<td>Iterative Improvement (Yokoo96)</td>
</tr>
</tbody>
</table>
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 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/descendants
  - no constraints between siblings

- **Basic Algorithm:**
  - 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”.

Explore “white” first
- \( LB(w) = 0 \)
- \( LB(b) = 0 \)

Receive cost msg
- \( LB(w) = 2 \)
- \( LB(b) = 0 \)

Now explore “black”
- \( LB(w) = 2 \)
- \( LB(b) = 0 \)

Receive cost msg
- \( LB(w) = 2 \)
- \( LB(b) = 3 \)

Go back to “white”
- \( LB(w) = 2 \)
- \( LB(b) = 3 \)

Termination Condition True
- \( LB(w) = 10 = UB(w) \)
- \( LB(b) = 12 \)
Example

Concurrently choose, send to descendents

- x1
- x2
- x3
- x4

Report lower bounds

- x1
- x2
- x3
- x4

LB = 1

x1 switches value

Note: x3’s cost message to x2 is obsolete since x1 has changed value, msg will be disregarded

Constraint Graph

<table>
<thead>
<tr>
<th>di</th>
<th>dj</th>
<th>f(di,dj)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.0</td>
</tr>
</tbody>
</table>

LB = 0

x2, x3 report new lower bounds

Optimal solution
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**
- parent
  - LB(w) = 10
  - LB(b) = 0

**Now explore “black”**
- parent
  - LB(w) = 10
  - LB(b) = 11

**Return to “white”**
- parent
  - backtrack threshold = 10
Backtrack Thresholds

Suppose agent $i$ received threshold $= 10$ from its parent

1. Explore “white” first
   - $\text{LB}(w) = 0$
   - $\text{LB}(b) = 0$
   - threshold $= 10$

2. Receive cost msg
   - $\text{LB}(w) = 2$
   - $\text{LB}(b) = 0$

3. Stick with “white”
   - $\text{LB}(w) = 2$
   - $\text{LB}(b) = 0$

4. Receive more cost msgs
   - $\text{LB}(w) = 11$
   - $\text{LB}(b) = 0$

5. Now try black
   - $\text{LB}(w) = 11$
   - $\text{LB}(b) = 0$

Key Point: Don’t change value until $\text{LB}$(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 $T_3$
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
  \[
  \text{cost}(S) \leq \text{cost}(\text{optimal soln}) + b
  \]

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