Distributed Combinatorial Optimization

An Introduction

Evan A. Sultanik

Evan.Sultanik@jhuapl.edu

Outline

Outline

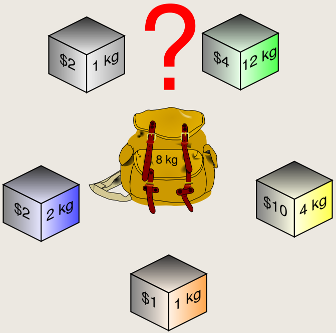
> "…it's supposed to be a panacea that satisfies everybody, at the risk of satisfying nobody."
>
> —Donald Knuth
> Page v of The TEX Book

Combinatorial Optimization

Definition

Combinatorial Optimization is the process of finding an optimal subset of objects from within a finite set of objects.

Example: the *Knapsack Problem*



Integer Programming Encoding

Given a finite set of n objects each with a value $v_1, v_2, \ldots, v_n$ and a weight $w_1, w_2, \ldots, w_n$, the knapsack problem asks to find a subset of the objects whose combined weight does not exceed a given maximum, w_{max}, and whose combined value is maximized:

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

subject to:

$$\sum_{j=1}^{n} w_j x_j \leq w_{\text{max}},$$

$$x_k \in \{0, 1\}, \qquad k = 1 \ldots n,$$

where the chosen set of objects is $S = \{i \in \{1, 2, \ldots, n\} : x_i = 1\}$.

Integer Programming Encoding

Given a finite set of n objects each with a value $v_1, v_2, \ldots, v_n$ and a weight $w_1, w_2, \ldots, w_n$, the knapsack problem asks to find a subset of the objects whose combined weight does not exceed a given maximum, w_{max}, and whose combined value is maximized:

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

subject to:

$$\sum_{j=1}^{n} w_j x_j \leq w_{\text{max}},$$

$$x_k \in \{0, 1\}, \qquad k = 1 \ldots n,$$

where the chosen set of objects is $S = \{i \in \{1, 2, \ldots, n\} : x_i = 1\}$.

Integer Programming Encoding

Given a finite set of n objects each with a value $v_1, v_2, \ldots, v_n$ and a weight $w_1, w_2, \ldots, w_n$, the knapsack problem asks to find a subset of the objects whose combined weight does not exceed a given maximum, w_{max}, and whose combined value is maximized:

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

subject to:

$$\sum_{j=1}^{n} w_j x_j \leq w_{\mathsf{max}},$$

$$x_k \in \{0, 1\}, \qquad k = 1 \ldots n,$$

where the chosen set of objects is $S = \{i \in \{1, 2, \ldots, n\} : x_i = 1\}$.

Integer Programming Encoding

Given a finite set of n objects each with a value $v_1, v_2, \ldots, v_n$ and a weight $w_1, w_2, \ldots, w_n$, the knapsack problem asks to find a subset of the objects whose combined weight does not exceed a given maximum, w_{max}, and whose combined value is maximized:

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

subject to:

$$\sum_{j=1}^{n} w_j x_j \leq w_{\text{max}},$$

$$x_k \in \{0, 1\}, \qquad k = 1 \ldots n,$$

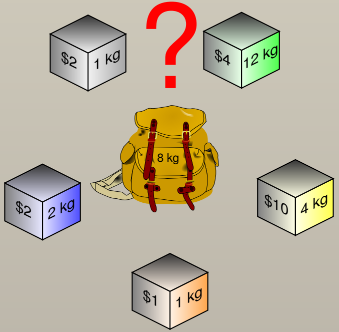
where the chosen set of objects is $S = \{i \in \{1, 2, \ldots, n\} : x_i = 1\}$.

Example

maximize $x_1\$2 + x_2\$4 + x_3\$10 + x_4\$1 + x_5\$2$

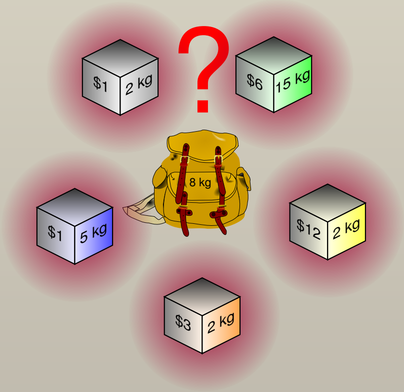
subject to:

$$1x_1 + 12x_2 + 4x_3 + 1x_4 + 2x_5 \leq 8\,\text{kg},$$
$$x_k \in \{0, 1\}, \qquad k = 1 \ldots n.$$



Relevance to *Streaming*?

- Usually these problems are solved *once*. But what if the problem itself changes over time?



- There is a *stream* of *modification events* to which we must react and re-optimize.

Dynamic Optimization

maximize $x_1\$2 + x_2\$4 + x_3\$10 + x_4\$1 + x_5\$2$

subject to:

$$\begin{aligned} &1x_1 + 12x_2 + 4x_3 + 1x_4 + 2x_5 \leq 8\,\text{kg}, \\ &x_k \in \{0, 1\}, && k = 1 \ldots n. \end{aligned}$$

Optimal Solution

$x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1,\ x_5 = 1$ (Payoff $= \$15$)

Dynamic Optimization

maximize $x_1\$1 + x_2\$4 + x_3\$10 + x_4\$1 + x_5\$2$

subject to:

$$1x_1 + 12x_2 + 4x_3 + 1x_4 + 2x_5 \leq 8\,\text{kg},$$
$$x_k \in \{0,1\}, \qquad k = 1\ldots n.$$

Optimal Solution

$x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1,\ x_5 = 1$ (Payoff $= \$14$)

Events (Time $\longrightarrow$)

$\boxed{\$2 \mapsto \$1}$

Dynamic Optimization

maximize $x_1\$1 + x_2\$4 + x_3\$10 + x_4\$1 + x_5\$2$

subject to:

$$2x_1 + 12x_2 + 4x_3 + 1x_4 + 2x_5 \leq 8\,\text{kg},$$
$$x_k \in \{0, 1\}, \qquad k = 1 \ldots n.$$

Optimal Solution

$x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 0,\ x_5 = 1$ (Payoff $= \$13$)

Events (Time $\longrightarrow$)

$\boxed{\$2 \mapsto \$1}$ $\boxed{1\,\text{kg} \mapsto 2\,\text{kg}}$

Dynamic Optimization

maximize $x_1\$1 + x_2\$6 + x_3\$10 + x_4\$1 + x_5\$2$

subject to:

$$2x_1 + 12x_2 + 4x_3 + 1x_4 + 2x_5 \leq 8\,\text{kg},$$
$$x_k \in \{0, 1\}, \qquad k = 1 \ldots n.$$

Optimal Solution

$x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 0,\ x_5 = 1$ (Payoff $= \$13$)

Events (Time $\longrightarrow$)

$\$2 \mapsto \$1$ | $1\,\text{kg} \mapsto 2\,\text{kg}$ | $\$4 \mapsto \$6$

Dynamic Optimization

maximize $x_1\$1 + x_2\$6 + x_3\$10 + x_4\$1 + x_5\$2$

subject to:

$$2x_1 + 15x_2 + 4x_3 + 1x_4 + 2x_5 \leq 8\,\text{kg},$$
$$x_k \in \{0, 1\}, \qquad k = 1 \ldots n.$$

Optimal Solution

$x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 0,\ x_5 = 1$ (Payoff $= \$13$)

Events (Time $\longrightarrow$)

$\$2 \mapsto \$1$ | $1\,\text{kg} \mapsto 2\,\text{kg}$ | $\$4 \mapsto \$6$ | $12\,\text{kg} \mapsto 15\,\text{kg}$

Dynamic Optimization

maximize $x_1\$1 + x_2\$6 + x_3\$10 + x_4\$1 + x_5\$2$

subject to:

$$
\begin{aligned}
&2x_1 + 15x_2 + 4x_3 + 1x_4 + 2x_5 \leq 20\,\text{kg}, & \\
&x_k \in \{0, 1\}, & k = 1 \dots n.
\end{aligned}
$$

Optimal Solution

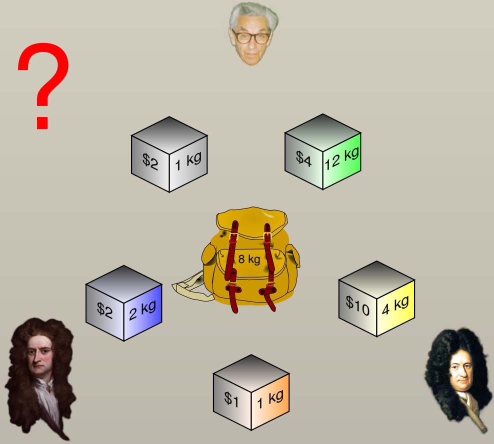
$x_1 = 0,\ x_2 = 1,\ x_3 = 1,\ x_4 = 1,\ x_5 = 0$ (Payoff $= \$17$)

Events (Time $\longrightarrow$)

$\$2 \mapsto \$1$	$1\,\text{kg} \mapsto 2\,\text{kg}$	$\$4 \mapsto \$6$	$12\,\text{kg} \mapsto 15\,\text{kg}$	$= 20\,\text{kg}$

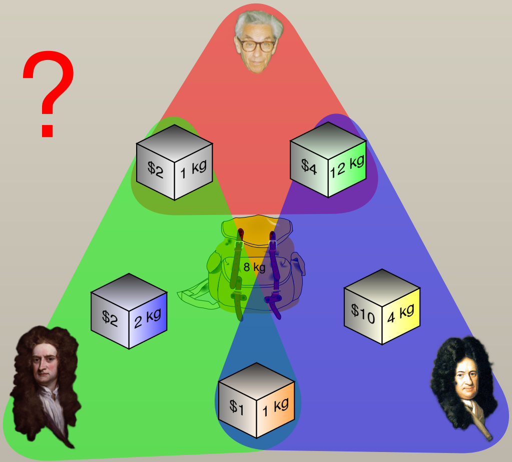
Distributed Optimization

A set of *agents* distributedly decide which objects to choose.



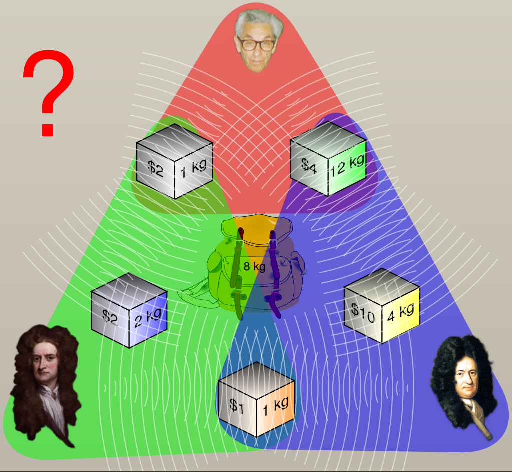
Distributed Optimization

Each agent only knows about a subset of the objects.



Distributed Optimization

They will have to *negotiate* to solve the problem.

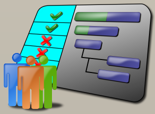


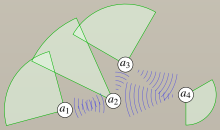
Why Distribute?

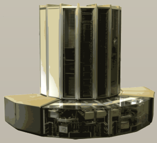
Privacy No single agent knows the entire world-state.
Example: *Meeting Scheduling*

Locality The problem is naturally distributed; extra effort is required to centralize the world-state for a centralized optimization algorithm.
Example: *Sensor Networks*

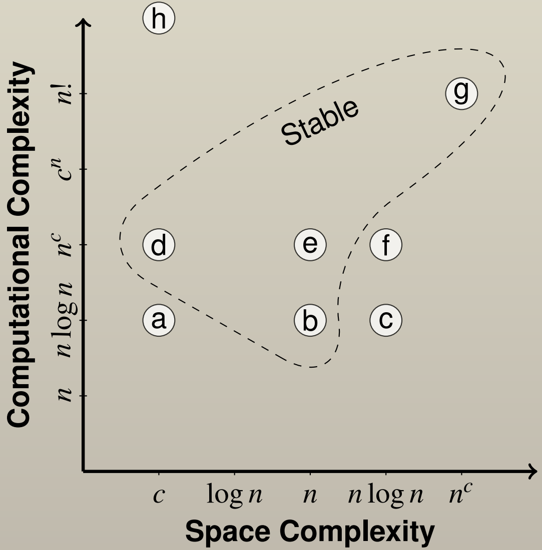
Efficiency Each agent is, in effect, its own processor, so we might achieve a speedup from parallelism.
Example: *Cloud Computing*







APL



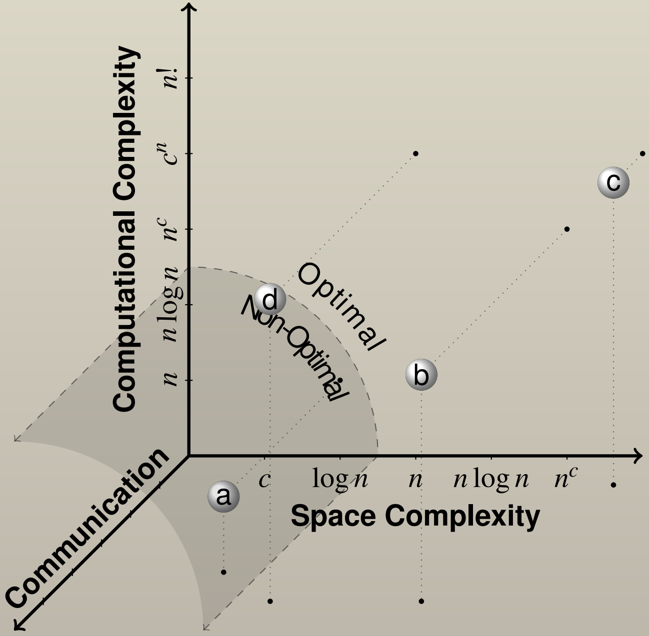
- a Heapsort
- b Merge sort
- c Introsort
- d Bubblesort
- e Strand sort
- f Quicksort†
- g Brute force (DFS)
- h Bogosort

† Assuming that memory pointers require logarithmic space.

Efficient Sequential Algorithms

- **Moore's Law:** processor speed doubles about every two years.
- If an algorithm has computational complexity $O(n^c)$ and current hardware can only solve problems of size n, then we will only have to wait $O(\log n)$ years[1] until hardware can solve a problem of size $n + 1$.
- Conclusion: polynomial runtime is desirable in sequential algorithms!

[1] More precisely, about $c \times \log_2\left(\frac{n+1}{n}\right)$ years.

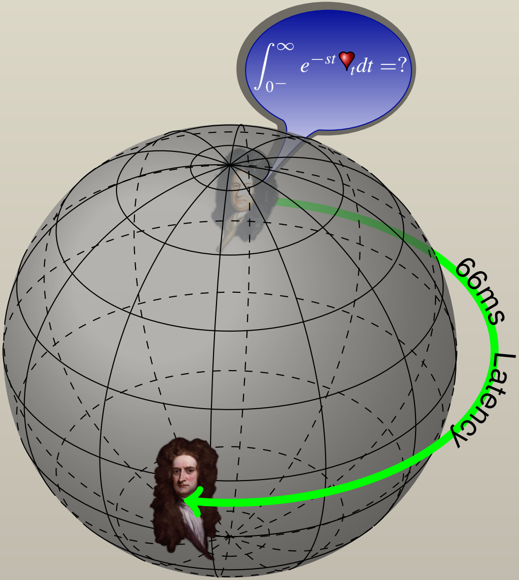


a DSA

b (BnB-)Adopt

c DPOP

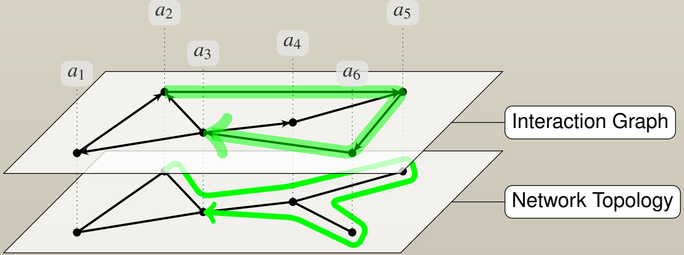
d MB-DPOP(1)



Conclusion

In distributed algorithms, there is no equivalent to Moore's law! Number of different metrics to optimize (*e.g.*, rounds, messages, latency, *&c.*).

When to distribute?



- the problem itself is naturally distributed;
- local properties of the problem seem to allow for speedups from distributed processing;
- in certain environments, such as sensor networks, hardware restrictions might necessitate decentralization in order to save memory/power/&c.;
- privacy (no central node can be trusted); and
- ultimately need $O(n)$ messaging rounds.

Constraint Reasoning

(a.k.a. "Constraint Programming")

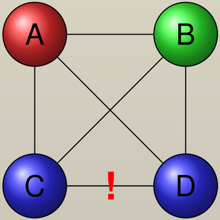
Idea: Model problems as systems of constraints.

- Set of *variables*: $V = \{v_1, v_2, \ldots, v_n\}$
- Each variable has an associated *domain* from which it can be assigned a value: $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$.
- There are a set of constraints that dictate costs for certain variable assignments:

$$f : \bigcup_{S \in 2^V} \prod_{v_i \in S} (\{v_i\} \times D_i) \to \mathbb{R}.$$

Example: Graph Coloring

Graph $G = \langle V, E \rangle$:



$$
\begin{aligned}
V &= \{v_A, v_B, v_C, v_D\} \\
D_A = D_B = D_C &= \{{\color{red}\blacksquare}, {\color{green}\blacksquare}, {\color{blue}\blacksquare}\} \\
f(\langle v_i, d_k\rangle, \langle v_j, d_\ell\rangle) \mapsto 1 \quad &\text{if} \quad \langle v_i, v_j\rangle \in E \wedge d_k = d_\ell.
\end{aligned}
$$

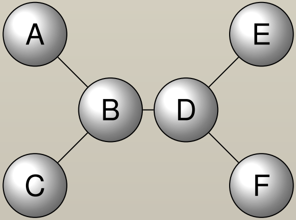
(incur a cost of 1) (two neighboring vertices are assigned the same color)

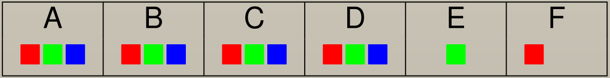
Goal

Find a mapping from variables to domains that minimizes f.

Acyclic Constraint Graphs

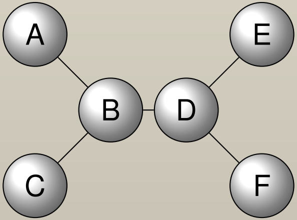
Consider this graph coloring problem.





Acyclic Constraint Graphs

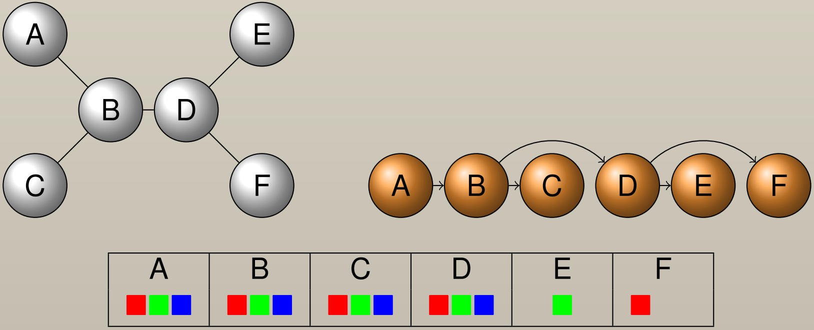
Note that E and F have unary constraints dictating their colors.



A	B	C	D	E	F

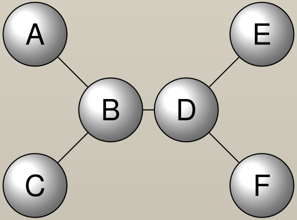
Acyclic Constraint Graphs

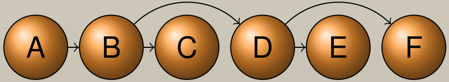
Perform a DFS traversal of the constraint graph. . .

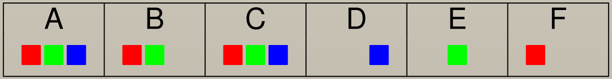


Acyclic Constraint Graphs

In reverse order, remove inconsistent entries in the domains.

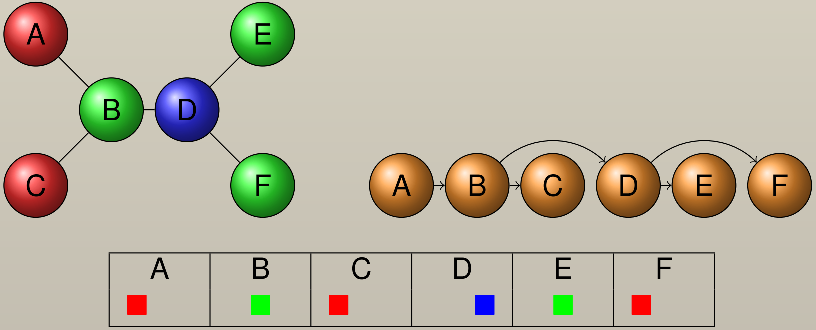






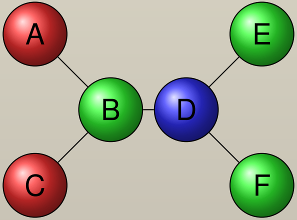
Acyclic Constraint Graphs

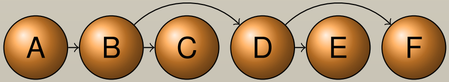
Working in order, choose values remaining in the domains.



Acyclic Constraint Graphs

This algorithm runs in $O(|\mathcal{D}|^2|V|)$ time.





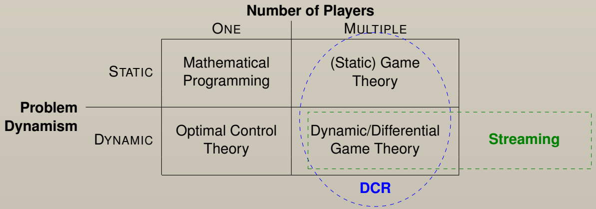
A	B	C	D	E	F

A Common Approximation:

Remove constraints until the constraint graph becomes acyclic!

Distributed Constraint Reasoning

Idea: Model inherently distributed problems as systems of constraints.

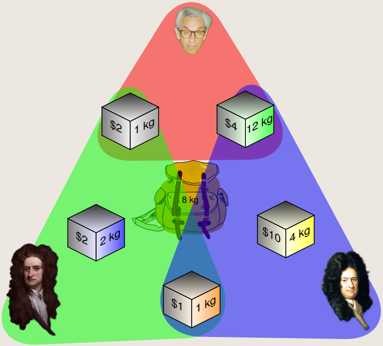


Definition

An "**Agent**" is a situated computational process with one or more of the following properties: *autonomy*, *proactivity* and *interactivity*.

Distributed Constraint Reasoning

Variables are assigned by agents:



Idea: Mo ... ystems of

Streaming

Definition

An “**Agent**” is a situated computational process with one or more of the following properties: *autonomy*, *proactivity* and *interactivity*.

Algorithms: DisCSP

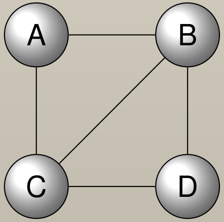
Early DCR research focused on DisCSP:

- Asynchronous Backtracking (1992)
- Asynchronous Weak-Commitment (1994)
- Distributed Breakout (1995) $\leftarrow$ local search
- Distributed Forward Checking (2000)

Distributed Stochastic Algorithm

- **Idea:** Perform a greedy, local search.
- Very fast!
- No guarantee of optimality (can get stuck in local minima).

Example: Graph Coloring

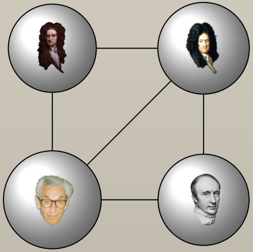


W. Zhang, G. Wang, and L. Wittenburg
Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance
In *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*, 2002.

Distributed Stochastic Algorithm

- **Idea:** Perform a greedy, local search.
- Very fast!
- No guarantee of optimality (can get stuck in local minima).

Each agent controls a vertex.



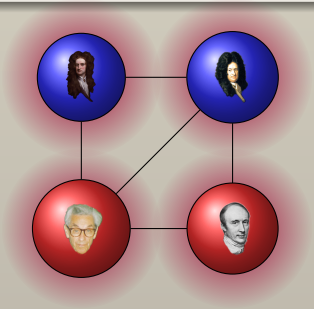
W. Zhang, G. Wang, and L. Wittenburg
Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance
In *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*, 2002.

APL

Distributed Stochastic Algorithm

- **Idea:** Perform a greedy, local search.
- Very fast!
- No guarantee of optimality (can get stuck in local minima).

Agents randomly choose a value from their domain...



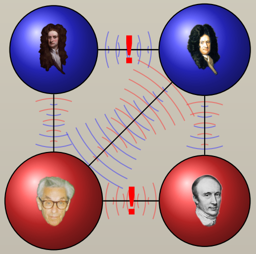
W. Zhang, G. Wang, and L. Wittenburg
Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance
In *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*, 2002.

APL

Distributed Stochastic Algorithm

- **Idea:** Perform a greedy, local search.
- Very fast!
- No guarantee of optimality (can get stuck in local minima).

. . . then broadcast their choices to neighbors.

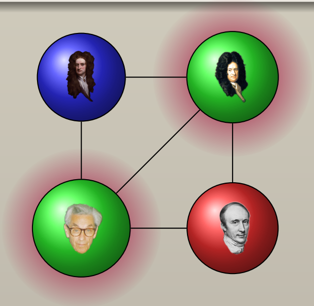


W. Zhang, G. Wang, and L. Wittenburg
Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance
In *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*, 2002.

Distributed Stochastic Algorithm

- **Idea:** Perform a greedy, local search.
- Very fast!
- No guarantee of optimality (can get stuck in local minima).

If conflict, choose another value given neighbors' choices.

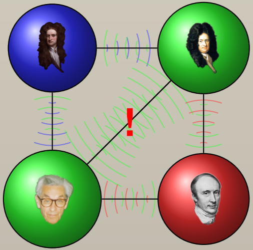


W. Zhang, G. Wang, and L. Wittenburg
Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance
In *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*, 2002.

Distributed Stochastic Algorithm

- **Idea:** Perform a greedy, local search.
- Very fast!
- No guarantee of optimality (can get stuck in local minima).

Re-broadcast to neighbors.

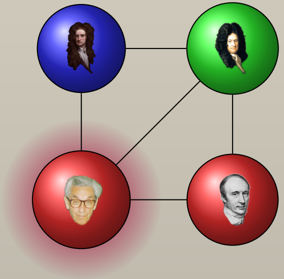


W. Zhang, G. Wang, and L. Wittenburg
Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance
In *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*, 2002.

Distributed Stochastic Algorithm

- **Idea:** Perform a greedy, local search.
- Very fast!
- No guarantee of optimality (can get stuck in local minima).

Resolve conflicts.



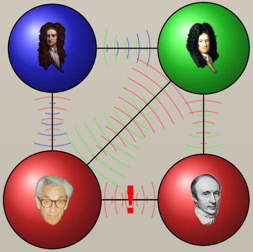
W. Zhang, G. Wang, and L. Wittenburg
Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance
In *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*, 2002.

APL

Distributed Stochastic Algorithm

- **Idea:** Perform a greedy, local search.
- Very fast!
- No guarantee of optimality (can get stuck in local minima).

Re-broadcast to neighbors.

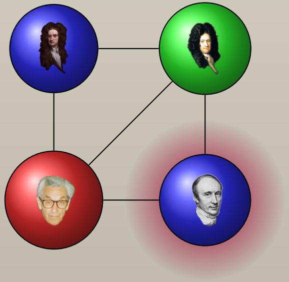


W. Zhang, G. Wang, and L. Wittenburg
Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance
In *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*, 2002.

Distributed Stochastic Algorithm

- **Idea:** Perform a greedy, local search.
- Very fast!
- No guarantee of optimality (can get stuck in local minima).

Resolve conflicts.



W. Zhang, G. Wang, and L. Wittenburg
Distributed Stochastic Search for Constraint Satisfaction and Optimization: Parallelism, Phase Transitions and Performance
In *Proceedings of the AAAI Workshop on Probabilistic Approaches in Search*, 2002.

Provably Optimal DCOP Algorithms

Idea: Maintain a structure (like a spanning tree) to organize the problem.

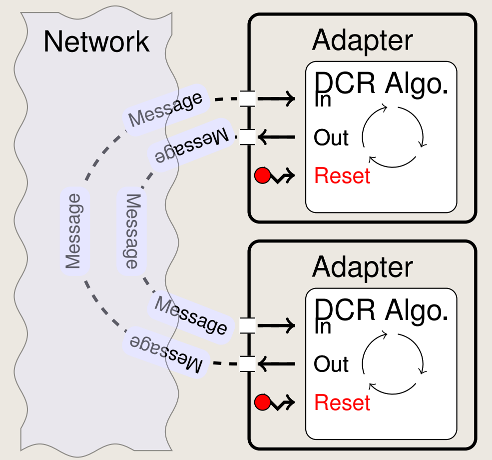
- parallel asynchronous exploration of disjoint subproblems, reminiscent of iterative A* search (**ADOPT**, 2003);
- incremental partial centralization (**OptAPO**, 2004);
- dynamic programming (**DPOP**, 2006); and
- distributed branch-and-bound, both synchronous (**NCBB**, 2006) and asynchronous (**BnB-ADOPT**, 2007);
- hybrid additionally using local search (**ADOPT-ing**, 2007).

Dynamic (*i.e.*, Streaming) DCR

- **DSA:** Every time a modification event occurs, simply re-resolve conflicts!
- **Pseudotree-Based Algorithms:** Need a method to dynamically maintain a depth-first spanning tree (*e.g.*, Superstabilizing DFS [Collin & Dolev, 1994] or Mobed [Sultanik, *et al.*, 2010]).
- Alternative: Use an *adapter* that automatically resets the algorithm whenever an event occurs that invalidates the current state.

Dynamic (*i.e.*, Streaming) DCOPs

The Dynamic DCR "Adapter"



R. Lass, E. Sultanik, and W. Regli
Dynamic Distributed Constraint Reasoning.
In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 2008.

If the adapter detects an event that invalidates the current state of the algorithm, re-solve from scratch.

General Approximation Schema

Optimization Problem

↓

Integer Programming Formulation (IP)

↓

Relaxed to a Continuous Optimization Formulation

↓

Solution to the Continuous Problem

↓

Approximated Solution to (IP)

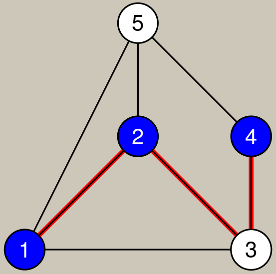
The Primal-Dual Formulation

For any linear program there is a dual linear program:

$$\begin{array}{rl} \max & c^t x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0 \end{array} \iff \begin{array}{rl} \min & b^t y \\ \text{s.t.} & A^t y \leq c \\ & y \leq 0 \end{array}$$

The Steiner Forest Problem

Example



Objective: Find a minimum weight forest (*e.g.*, —) that connects all ● nodes to each other, possibly utilizing ○ nodes.

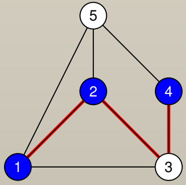
Example: Encoding Steiner Forest

Whether or not an edge e will be in the forest: $x_e \in \{0, 1\}$.

$$\min \sum_{e \in E} w(e) x_e$$

$$\text{s.t.} \sum_{e \in \delta(S)} x_e \geq f(S), \quad \forall S \subset V : S \neq \emptyset$$

$$x_e \geq 0, \quad \forall e \in E,$$



$$\max \sum_{S \subset V} f(S) y_S$$

$$\text{s.t.} \sum_{S : e \in \delta(S)} y_S \leq w_e, \quad \forall e \in E$$

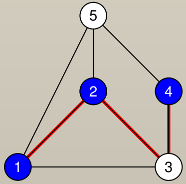
$$y_S \geq 0, \quad \forall S \subset V : S \neq \emptyset.$$

M. Aggarwal and N. Garg
A Scaling Technique for Better Network Design.
In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

Example: Encoding Steiner Forest

The weight of edge e: $w(e)$.

$$
\begin{aligned}
\min \quad & \sum_{e \in E} w(e) x_e \\
\text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq f(S), && \forall S \subset V : S \neq \emptyset \\
& x_e \geq 0, && \forall e \in E,
\end{aligned}
$$



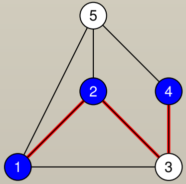
$$
\begin{aligned}
\max \quad & \sum_{S \subset V} f(S) y_S \\
\text{s.t.} \quad & \sum_{S : e \in \delta(S)} y_S \leq w_e, && \forall e \in E \\
& y_S \geq 0, && \forall S \subset V : S \neq \emptyset.
\end{aligned}
$$

M. Aggarwal and N. Garg
A Scaling Technique for Better Network Design.
In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

Example: Encoding Steiner Forest

$f(S) = 1$ iff $\emptyset \neq S \cap \{1, 2, 4\} \neq \{1, 2, 4\}$

$$\begin{aligned}
\min \quad & \sum_{e \in E} w(e) x_e \\
\text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq f(S), && \forall S \subset V : S \neq \emptyset \\
& x_e \geq 0, && \forall e \in E,
\end{aligned}$$



$$\begin{aligned}
\max \quad & \sum_{S \subset V} f(S) y_S \\
\text{s.t.} \quad & \sum_{S : e \in \delta(S)} y_S \leq w_e, && \forall e \in E \\
& y_S \geq 0, && \forall S \subset V : S \neq \emptyset.
\end{aligned}$$

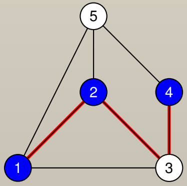
M. Aggarwal and N. Garg
A Scaling Technique for Better Network Design.
In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

Example: Encoding Steiner Forest

Each variable in the primal becomes a constraint in the dual

$$\begin{aligned} \min \quad & \sum_{e \in E} w(e) x_e \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq f(S), & \forall S \subset V : S \neq \emptyset \\ & x_e \geq 0, & \forall e \in E, \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{S \subset V} f(S) y_S \\ \text{s.t.} \quad & \sum_{S : e \in \delta(S)} y_S \leq w_e, & \forall e \in E \\ & y_S \geq 0, & \forall S \subset V : S \neq \emptyset. \end{aligned}$$



M. Aggarwal and N. Garg
A Scaling Technique for Better Network Design.
In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

Example: Encoding Steiner Forest

$$\min \sum_{e \in E} w(e) x_e$$

$$\text{s.t.} \sum_{e \in \delta(S)} x_e \geq f(S), \quad \forall S \subset V : S \neq \emptyset$$

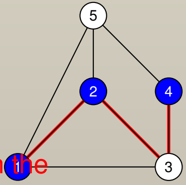
$$x_e \geq 0, \quad \forall e \in E,$$

Each constraint in the primal becomes a variable in the dual

$$\max \sum_{S \subset V} f(S) y_S$$

$$\text{s.t.} \sum_{S : e \in \delta(S)} y_S \leq w_e, \quad \forall e \in E$$

$$y_S \geq 0, \quad \forall S \subset V : S \neq \emptyset.$$



M. Aggarwal and N. Garg
A Scaling Technique for Better Network Design.
In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

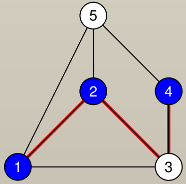
Example: Encoding Steiner Forest

This is a mechanical process!

$$\min \sum_{e \in E} w(e) x_e$$

$$\text{s.t.} \sum_{e \in \delta(S)} x_e \geq f(S), \quad \forall S \subset V : S \neq \emptyset$$

$$x_e \geq 0, \quad \forall e \in E,$$



$$\max \sum_{S \subset V} f(S) y_S$$

$$\text{s.t.} \sum_{S : e \in \delta(S)} y_S \leq w_e, \quad \forall e \in E$$

$$y_S \geq 0, \quad \forall S \subset V : S \neq \emptyset.$$

M. Aggarwal and N. Garg
A Scaling Technique for Better Network Design.
In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

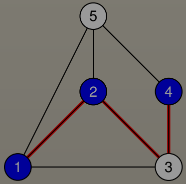
Example: Encoding Steiner Forest

Note: *looks like* exponential constraints!

$$\min \sum_{e \in E} w(e) x_e$$

$$\text{s.t.} \sum_{e \in \delta(S)} x_e \geq f(S), \quad \forall S \subset V : S \neq \emptyset$$

$$x_e \geq 0, \quad \forall e \in E,$$



$$\max \sum_{S \subset V} f(S) y_S$$

$$\text{s.t.} \sum_{S : e \in \delta(S)} y_S \leq w_e, \quad \forall e \in E$$

$$y_S \geq 0, \quad \forall S \subset V : S \neq \emptyset.$$

M. Aggarwal and N. Garg
A Scaling Technique for Better Network Design.
In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

Properties of the Schema

TERM	MEANING
OP	Optimization Problem
IP	Integer Programming Formulation of OP
LP	Continuous Optimization Relaxation of IP
D	The Dual of LP
$Z^*_{LP}/Z^*_{D}/Z^*_{IP}$	Cost of the Optimal Solution to LP/D/IP

NAME	PROPERTY
Weak Duality	The cost of any feasible solution to D is a lower bound on the solution to LP.
Strong Duality	$Z^*_{D} = Z^*_{LP} \leq Z^*_{IP}$
Complementary Slackness	A primal variable can be positive iff its associated dual constraint is tight.

Algorithmic Form of the Primal-Dual Schema

procedure PRIMAL-DUAL(IP)

  Let (CO) be the continuous optimization relaxation of (IP).

  Let (D) be the dual to (CO).

  Initialize vectors $x = 0$ and $y = 0$ which are, respectively, the solutions for (CO) and (D). /* Note that y will initially be dual feasible, but x will not necessarily be primal feasible. */

  while x is primal infeasible **do**

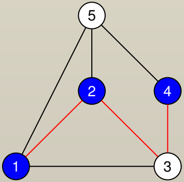
    While maintaining dual feasibility, deterministically increase the dual values y_i until one dual constraint becomes tight (*i.e.*, that variable cannot be increased any more without breaking a dual constraint).

    For a subset of the tight dual constraints, increase the primal variable corresponding to them by an integral amount.

  The cost of the dual solution is used as a lower bound on OPT.

V. Vazirani
Approximation Algorithms.
Springer-Verlag, Berlin, 2001.

The (Sequential) Algorithm



(We will assume unit edge weights for simplicity.)

$$\begin{aligned} d(1) &= 1.5 & F &= \{\{1,2\},\{1,3\},\{3,4\}\} \\ d(2) &= 1.5 & \mathcal{C} &= \{\{1,2,3,4\},\{5\}\} \\ d(3) &= 0.5 & e &= \langle 3,4 \rangle \\ d(4) &= 1.5 & \epsilon &= 0.5 \\ d(5) &= 0 & & \end{aligned}$$

procedure CONSTRAINED-FOREST(G, w, f)
- $F \leftarrow \emptyset$ /* Implicitly set y_S for all $S \subset V$ */
- $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
- **for all** $v \in V$ **do**
  - $d(v) \leftarrow 0$
- **while** $\exists C \in \mathcal{C} : f(C) = 1$ **do**
  - Find an edge $e = \langle i, j \rangle$ such that $\mu(i) \neq \mu(j)$ and $\epsilon = \dfrac{w(e) - d(i) - d(j)}{f(\mu(i)) + f(\mu(j))}$ is minimized.
  - $F \leftarrow F \cup \{e\}$.
  - **for all** $v \in V$ **do**
    - $d(v) \leftarrow d(v) + \epsilon \cdot f(\mu(v))$ /* Implicitly set $y_C \leftarrow y_C + \epsilon \cdot f(C)$ for all $C \in \mathcal{C}$. */
  - $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mu(i) \cup \mu(j)\} - \{\mu(i)\} - \{\mu(j)\}$
- $F \leftarrow \{e \in F :$ For some connected component N of $(V, F - \{e\}), f(N) = 1)\}$

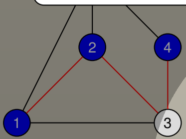
D. Williamson, M. Goemans, M. Mihail, and V. Vazirani
A primal-dual approximation algorithm for generalized steiner network problems.
Combinatorica, **15**(3):435–454, 1995.

APL

The (Sequential) Algorithm

Can solve other problems for different f!

More on this in a bit. . .



(We will assume unit edge weights for simplicity.)

$\{1,2\},\{1,3\},\{3,4\}\}$

$d(2) = 1.5$ $\quad \mathcal{C} = \{\{1,2,3,4\},\{5\}\}$

$d(3) = 0.5$ $\quad e = \langle 3,4\rangle$

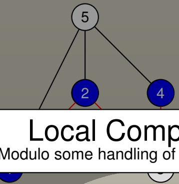
$d(4) = 1.5$ $\quad \epsilon = 0.5$

$d(5) = 0$

procedure CONSTRAINED-FOREST(G, w, f)
    $F \leftarrow \emptyset$ /* Implicitly set y_S for all $S \subset V$ */
    $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
    for all $v \in V$ **do**
        $d(v) \leftarrow 0$
    while $\exists C \in \mathcal{C} : f(C) = 1$ **do**
        Find an edge $e = \langle i,j\rangle$ such that $\mu(i) \neq \mu(j)$ and $\epsilon = \dfrac{w(e) - d(i) - d(j)}{f(\mu(i)) + f(\mu(j))}$ is minimized.
        $F \leftarrow F \cup \{e\}$.
        for all $v \in V$ **do**
            $d(v) \leftarrow d(v) + \epsilon \cdot f(\mu(v))$ /* Implicitly set $y_C \leftarrow y_C + \epsilon \cdot f(C)$ for all $C \in \mathcal{C}$. */
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mu(i) \cup \mu(j)\} - \{\mu(i)\} - \{\mu(j)\}$
    $F \leftarrow \{e \in F :$ For some connected component N of $(V, F - \{e\}), f(N) = 1)\}$

D. Williamson, M. Goemans, M. Mihail, and V. Vazirani
A primal-dual approximation algorithm for generalized steiner network problems.
Combinatorica, **15**(3):435–454, 1995.

The (Sequential) Algorithm



(We will assume unit edge weights for

$$
\begin{aligned}
d(1) &= 1.5 & F &= \{\{1,2\},\{1,3\},\{3,4\}\}\\
d(2) &= 1.5 & \mathcal{C} &= \{\{1,2,3,4\},\{5\}\}\\
d(3) &= 0.5 & e &= \langle 3,4\rangle\\
d(4) &= 1.5 & \epsilon &= 0.5\\
d(5) &= 0 & &
\end{aligned}
$$

Local Computation!

Modulo some handling of race conditions...

```
procedure CONSTRAINED-FOREST(G, w, f)
    F ← ∅    /* Implicitly set y_S for all S ⊂ V */
    C ← {{v} : v ∈ V}
    for all v ∈ V do
        d(v) ← 0
    while ∃C ∈ C : f(C) = 1 do
        Find an edge e = ⟨i, j⟩ such that μ(i) ≠ μ(j) and ε = (w(e) − d(i) − d(j)) / (f(μ(i)) + f(μ(j))) is minimized.
        F ← F ∪ {e}.
        for all v ∈ V do
            d(v) ← d(v) + ε·f(μ(v))    /* Implicitly set y_C ← y_C + ε·f(C) for all C ∈ C. */
        C ← C ∪ {μ(i) ∪ μ(j)} − {μ(i)} − {μ(j)}
    F ← {e ∈ F : For some connected component N of (V, F − {e}), f(N) = 1)}
```


D. Williamson, M. Goemans, M. Mihail, and V. Vazirani
A primal-dual approximation algorithm for generalized steiner network problems.
Combinatorica, **15**(3):435–454, 1995.

Generalizing the Indicator Function

$$\begin{aligned} \min \quad & \sum_{e \in E} w(e) x_e & \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq f(S), & \forall S \subset V : S \neq \emptyset \\ & x_e \geq 0, & \forall e \in E, \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{S \subset V} f(S) y_S & \\ \text{s.t.} \quad & \sum_{S : e \in \delta(S)} y_S \leq w_e, & \forall e \in E \\ & y_S \geq 0, & \forall S \subset V : S \neq \emptyset. \end{aligned}$$

Steiner Forest: $f(S) = 1$ iff $\emptyset \neq S \cap \{1, 2, 4\} \neq \{1, 2, 4\}$

"Wouldn't it be *totally radical* if we could solve a seemingly completely different problem simply by tweaking the definition of f?*"

* May not be a direct quote.

M. Goemans and D. Williamson
A General Approximation Technique for Constrained Forest Problems.
SIAM Journal on Computing, **24**:296–317, 1995.

Constrained Forest Problems

$$\min \sum_{e \in E} w(e) x_e \qquad\qquad \max \sum_{S \subset V} f(S) y_S$$

$$\text{s.t.} \sum_{e \in \delta(S)} x_e \geq f(S), \quad \forall S \subset V : S \neq \emptyset \qquad \text{s.t.} \sum_{S : e \in \delta(S)} y_S \leq w_e, \quad \forall e \in E$$

$$x_e \geq 0, \quad \forall e \in E, \qquad y_S \geq 0, \quad \forall S \subset V : S \neq \emptyset.$$

NAME	PROBLEM	$f(S) = 1$ iff . . .
Minimum-weight perfect matching	Find a minimum-cost set of non-adjacent edges that cover all vertices.	$\lvert S\rvert$ is odd.
T-join	Given an even subset T of vertices, find a minimum-cost set of edges that has odd degree at vertices in T and even degree at vertices not in T.	$\lvert S \cap T\rvert$ is odd.
Minimum spanning tree/forest	Find a minimum weight forest that maximizes connectivity between vertices.	$\exists u \in S, v \notin S : u \rightsquigarrow v \in G$
Generalized Steiner tree	Find a minimum-cost forest that connects all vertices in T_i for $i = 1, \ldots, p$.	$\exists i \in \{1, \ldots, p\} : \emptyset \neq S \cap T_i \neq T_i$.
Point-to-point connection	Given a set $C = \{c_1, \ldots, c_p\}$ of sources and a set $D = \{d_1, \ldots, d_p\}$ of destinations in a graph $G = \langle V, E\rangle$, find a minimum-cost set F of edges such that each source-destination pair is connected in F.	$\lvert S \cap C\rvert \neq \lvert S \cap D\rvert$.
Partitioning (w/triangle inequality)	Find a minimum-cost collection of vertex-disjoint trees, paths, or cycles that cover all vertices.	$S \not\equiv 0 (\text{mod } k)$.
Location design/routing	Select depots among a subset D of vertices of a graph $G = \langle V, E\rangle$ and cover all vertices in V with a set of cycles, each containing a selected depot, while minimizing the sum of the fixed costs of open-	$\emptyset \neq S \subseteq V$

Proper Functions

$$\min \sum_{e \in E} w(e) x_e$$
$$\text{s.t.} \sum_{e \in \delta(S)} x_e \geq f(S), \quad \forall S \subset V : S \neq \emptyset$$
$$x_e \geq 0, \quad \forall e \in E,$$

$$\max \sum_{S \subset V} f(S) y_S$$
$$\text{s.t.} \sum_{S : e \in \delta(S)} y_S \leq w_e, \quad \forall e \in E$$
$$y_S \geq 0, \quad \forall S \subset V : S \neq \emptyset.$$

A function on the powerset of a set of vertices, $f : 2^V \to \{0, 1\}$, is said to be *proper* if the following are true:

PROPERTY NAME	RULE
Null	$f(\emptyset) = 0$
Symmetry	$\forall S \subseteq V : f(S) = f(V - S)$
Disjointness	$\forall A, B \subseteq V : (A \cap B = \emptyset) \implies f(A \cup B) \leq \max\{f(A), f(B)\}.$

Proper Functions (Continued)

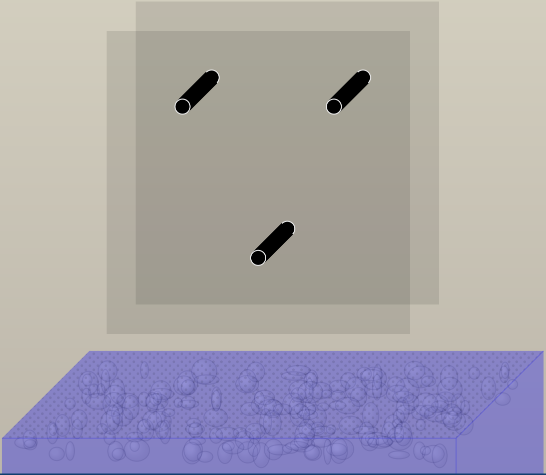
- If f is proper then the sequential algorithm will. . .
  - . . .run in polynomial time; and
  - . . .produce a solution that is 2-*OPT*
    (*i.e.*, the cost will be no more than two times the cost of the optimal solution).
- "*Constrained Forest Problems*"
- Many constrained forest problems are **NP**-HARD.
- Various extensions (*e.g.*, supermodular, well spaced, *&c.*).

Surprise!

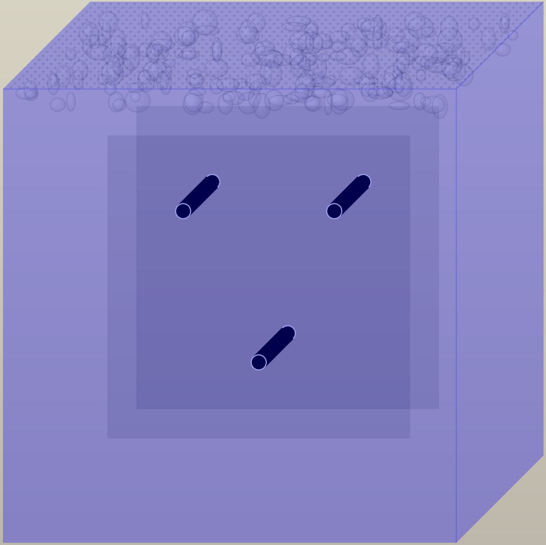
The sequential 2-approximation result can be generalized to a large family of functions and efficiently distributed for optimizing over streams.

“Natural” Organization

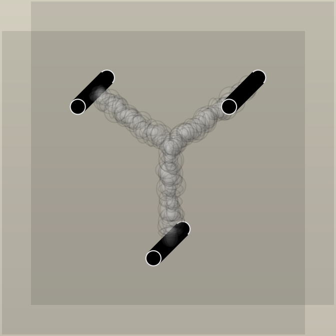
Scott Aaronson
NP-complete Problems and Physical Reality
SIGACT News **36**(1):30–52, 2005.



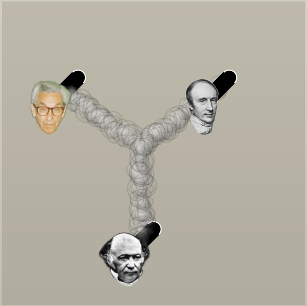
“Natural” Organization



"Natural" Organization

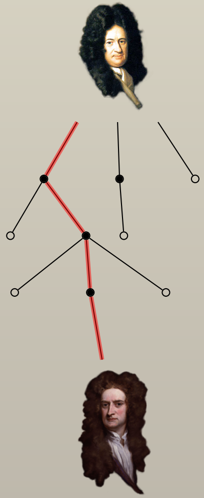


“Natural” Organization



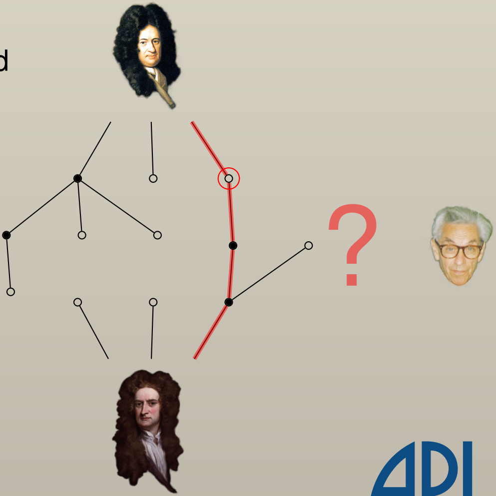
Search
(Foreshadowing!)

- **DFS**: Stack
- **BFS**: Queue
- **Best-first**: Priority Queue
- **A***: Priority Queue with Heuristic

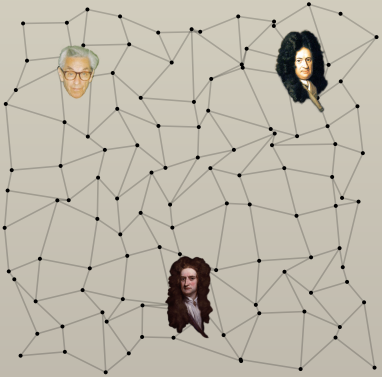


Bidirectional Search

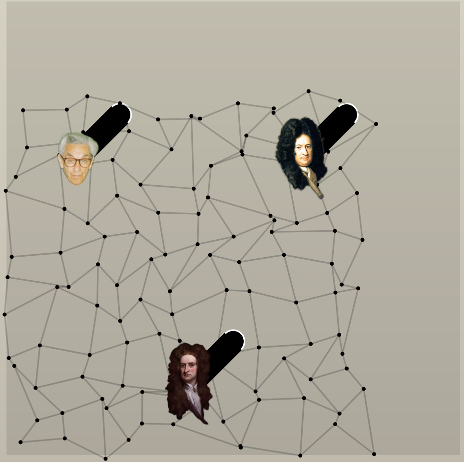
- Modified GOAL-TEST and an optimal search $\rightsquigarrow$ guaranteed optimality.
- Speedup from parallelism.
- Question: What if Erdős wants to join the party?



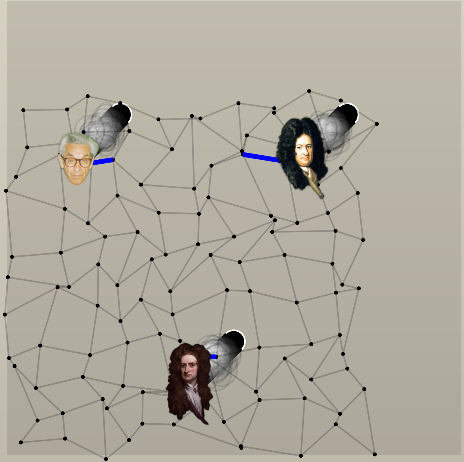
Multidirectional Graph Search?



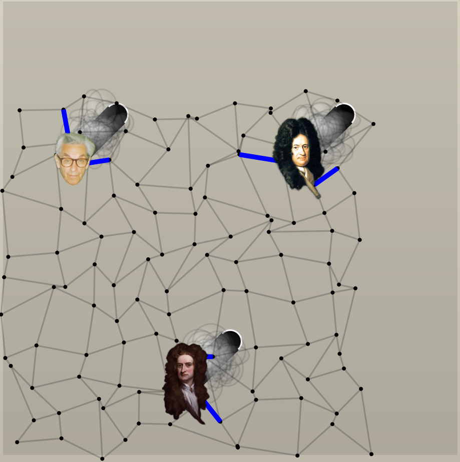
Multidirectional Graph Search?



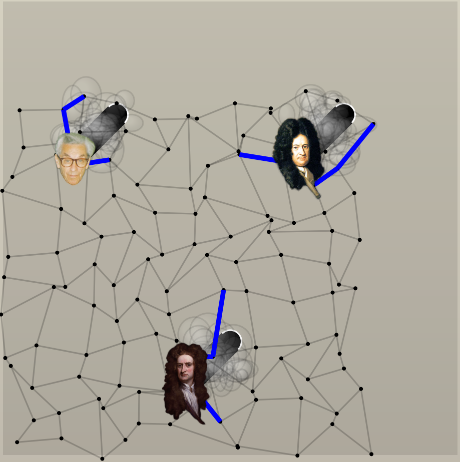
Multidirectional Graph Search?



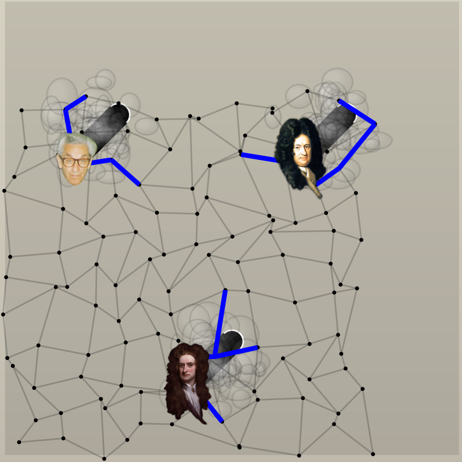
Multidirectional Graph Search?



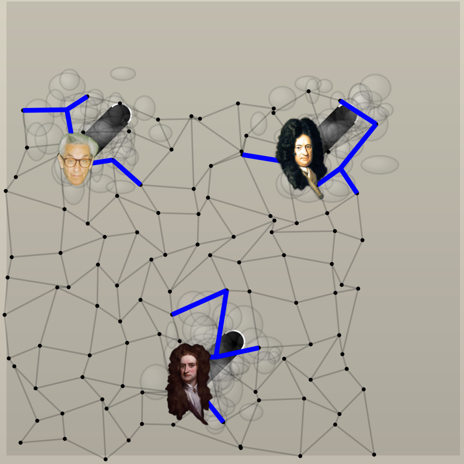
Multidirectional Graph Search?



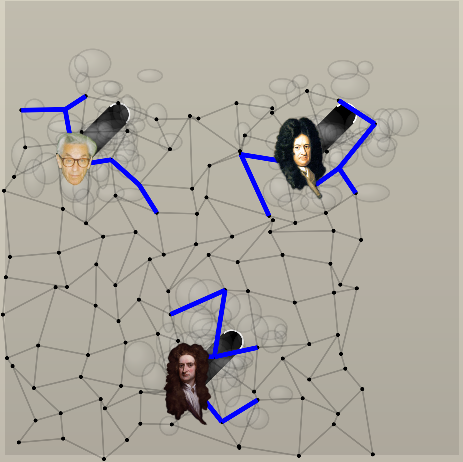
Multidirectional Graph Search?



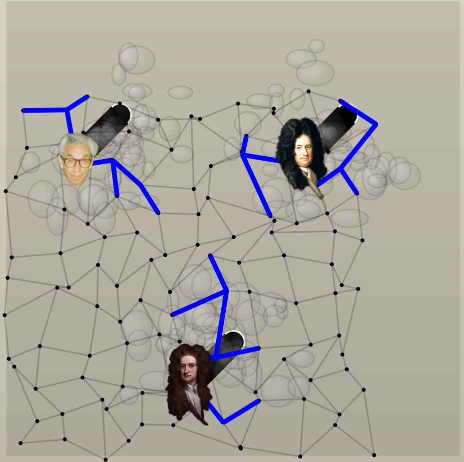
Multidirectional Graph Search?



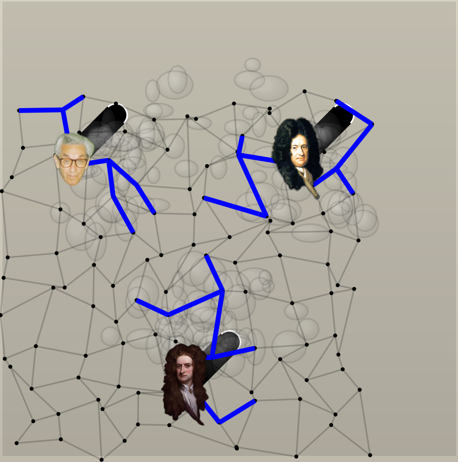
Multidirectional Graph Search?



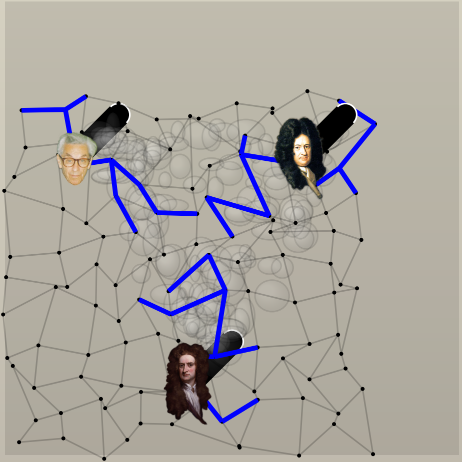
Multidirectional Graph Search?



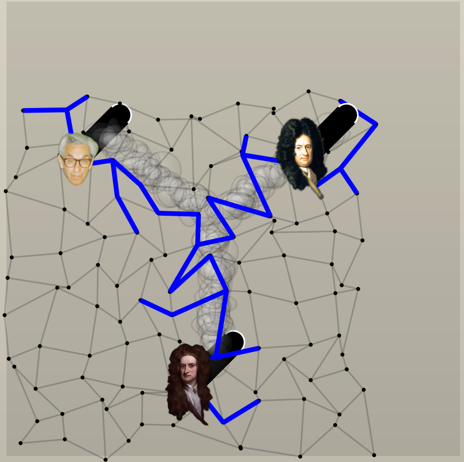
Multidirectional Graph Search?



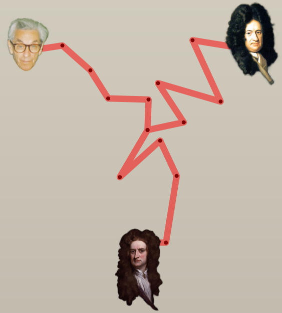
Multidirectional Graph Search?



Multidirectional Graph Search?



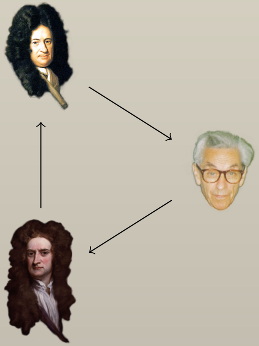
Multidirectional Graph Search?



Multidirectional Graph Search

Challenges

- How do we prevent cycles?



- How do we ensure correctness/completeness?
- Optimality?

Generalized Distributed Constrained Forest Algorithm

for Multidirectional Graph Search

procedure MULTIDIRECTIONAL-GRAPH-SEARCH(v)
Require: v is the start vertex running this search.
Ensure: $H = \langle \tilde{V}, \tilde{E} \rangle$ is the resulting forest.
    $\tilde{V} \leftarrow \{v\}$
    $\tilde{E} \leftarrow \emptyset$
    $F \leftarrow$ our neighbors /* The fringe of our search */
    $g(v) \leftarrow 0$ for all $v \in V$ /* Initialize the path-cost function to 0 */
    while Our interaction constraints are still unsatisfied **do**
        Find an edge $e = \langle u, v \rangle$ in the fringe that minimizes $\varepsilon = w(e) - g(u) - g(v)$
        if u either is being or already was expanded by another search **then**
            Merge our execution with u's search.
            if The other search also expanded the edge $\langle v, u \rangle$ in this round **then**
                $\epsilon \leftarrow \frac{\epsilon}{2}$
        for all $k \in \tilde{V} : k$ is incident to an edge in the fringe **do**
            $g(k) \leftarrow g(k) + \varepsilon$ /* Update the path-cost */
        $F \leftarrow (F \setminus \{e\}) \cup \delta(\{u\})$ /* Update the fringe with e's successors */
        $\tilde{V} \leftarrow \tilde{V} \cup \{u\}$ /* Add u to the final forest */
        $\tilde{E} \leftarrow \tilde{E} \cup \{e\}$ /* Add e to the final forest */

Generalized Distributed Constrained Forest Algorithm

for Multidirectional Gra

Initialization

Set up the fringe and path-cost functions.

```
procedure MULT
Require: v is the start vertex running this search.
Ensure: H = ⟨Ṽ, Ẽ⟩ is the resulting forest.
    Ṽ ← {v}
    Ẽ ← ∅
    F ← our neighbors   /* The fringe of our search */
    g(v) ← 0 for all v ∈ V   /* Initialize the path-cost function to 0 */
    while Our interaction constraints are still unsatisfied do
        Find an edge e = ⟨u, v⟩ in the fringe that minimizes ε = w(e) − g(u) − g(v)
        if u either is being or already was expanded by another search then
            Merge our execution with u's search.
            if The other search also expanded the edge ⟨v, u⟩ in this round then
                ϵ ← ϵ/2
        for all k ∈ Ṽ : k is incident to an edge in the fringe do
            g(k) ← g(k) + ε   /* Update the path-cost */
        F ← (F \ {e}) ∪ δ({u})   /* Update the fringe with e's successors */
        Ṽ ← Ṽ ∪ {u}   /* Add u to the final forest */
        Ẽ ← Ẽ ∪ {e}   /* Add e to the final forest */
```

Generalized Distributed Constrained Forest Algorithm

for Multidirectional Graph Search

procedure MULT
Require: v is the star
Ensure: $H = \langle \tilde{V}, \tilde{E} \rangle$ i
$\tilde{V} \leftarrow \{v\}$
$\tilde{E} \leftarrow \emptyset$
$F \leftarrow$ our neighbors /* The fringe of our search */
$g(v) \leftarrow 0$ for all $v \in V$ /* Initialize the path-cost function to 0 */
while Our interaction constraints are still unsatisfied **do**
Find an edge $e = \langle u, v \rangle$ in the fringe that minimizes $\varepsilon = w(e) - g(u) - g(v)$
if u either is being or already was expanded by another search **then**
Merge our execution with u's search.
if The other search also expanded the edge $\langle v, u \rangle$ in this round **then**
$\epsilon \leftarrow \frac{\epsilon}{2}$
for all $k \in \tilde{V} : k$ is incident to an edge in the fringe **do**
$g(k) \leftarrow g(k) + \varepsilon$ /* Update the path-cost */
$F \leftarrow (F \setminus \{e\}) \cup \delta(\{u\})$ /* Update the fringe with e's successors */
$\tilde{V} \leftarrow \tilde{V} \cup \{u\}$ /* Add u to the final forest */
$\tilde{E} \leftarrow \tilde{E} \cup \{e\}$ /* Add e to the final forest */

Goal-Test Function

Keep on searching until all of the constraints are satisfied.

Generalized Distributed Constrained Forest Algorithm

for Multidirectional Graph Search

procedure MULTIDIRECTIONAL-GRAPH-SEARCH(v)

Require: v is the star

Ensure: $H = \langle \tilde{V}, \tilde{E} \rangle$ i

$\tilde{V} \leftarrow \{v\}$

$\tilde{E} \leftarrow \emptyset$

$F \leftarrow$ our neighbors /* The fringe of our search */

$g(v) \leftarrow 0$ for all $v \in V$ /* Initialize the path-cost function to 0 */

while Our interaction constraints are still unsatisfied **do**

  Find an edge $e = \langle u, v \rangle$ in the fringe that minimizes $\varepsilon = w(e) - g(u) - g(v)$

  if u either is being or already was expanded by another search **then**

    Merge our execution with u's search.

    if The other search also expanded the edge $\langle v, u \rangle$ in this round **then**

      $\epsilon \leftarrow \frac{\epsilon}{2}$

  for all $k \in \tilde{V}$: k is incident to an edge in the fringe **do**

    $g(k) \leftarrow g(k) + \varepsilon$ /* Update the path-cost */

  $F \leftarrow (F \setminus \{e\}) \cup \delta(\{u\})$ /* Update the fringe with e's successors */

  $\tilde{V} \leftarrow \tilde{V} \cup \{u\}$ /* Add u to the final forest */

  $\tilde{E} \leftarrow \tilde{E} \cup \{e\}$ /* Add e to the final forest */

Remove Node from Fringe

The fringe is prioritized using a special potential function heuristic.

Generalized Distributed Constrained Forest Algorithm

for Multidirectional Graph Search

procedure MULTIDIRECTIONAL-GRAPH-SEARCH(v)

Require: v is the start vertex running this search.

Ensure: $H = \langle \tilde{V}, \tilde{E} \rangle$ is the resulting forest.

- $\tilde{V} \leftarrow \{v\}$
- $\tilde{E} \leftarrow \emptyset$
- $F \leftarrow$ our neighbors /* The fringe of our search */
- $g(v) \leftarrow 0$ for a … to 0 */
- **while** Our int…
  - Find an ed… $= w(e) - g(u) - g(v)$
  - **if** u either is being or already was expanded by another search **then**
    - Merge our execution with u's search.
    - **if** The other search also expanded the edge $\langle v, u \rangle$ in this round **then**
      - $\epsilon \leftarrow \frac{\epsilon}{2}$
  - **for all** $k \in \tilde{V} : k$ is incident to an edge in the fringe **do**
    - $g(k) \leftarrow g(k) + \varepsilon$ /* Update the path-cost */
  - $F \leftarrow (F \setminus \{e\}) \cup \delta(\{u\})$ /* Update the fringe with e's successors */
  - $\tilde{V} \leftarrow \tilde{V} \cup \{u\}$ /* Add u to the final forest */
  - $\tilde{E} \leftarrow \tilde{E} \cup \{e\}$ /* Add e to the final forest */

Path-Cost Update

Update the open nodes.

Generalized Distributed Constrained Forest Algorithm

for Multidirectional Graph Search

procedure MULTIDIRECTIONAL-GRAPH-SEARCH(v)

Require: v is the start vertex running this search.

Ensure: $H = \langle \tilde{V}, \tilde{E} \rangle$ is the resulting forest.

```
    Ṽ ← {v}
    Ẽ ← ∅
    F ← our neighbors  /* The fringe of our search */
    g(v) ← 0 for all v ∈ V  /* Initialize the path-cost function to 0 */
    while Our interaction constraints are still unsatisfied do
        Find an edge e = ⟨u, v⟩ in the fringe that minimizes ε = w(e) − g(u) − g(v)
        if u either is being or already was expanded by another search then
            Merge our execution with u's search
            if The ... in this round then
                ϵ ← ...
        for all k ∈ V : k is incident to an edge in the fringe do
            g(k) ← g(k) + ε  /* Update the path-cost */
        F ← (F \ {e}) ∪ δ({u})  /* Update the fringe with e's successors */
        Ṽ ← Ṽ ∪ {u}  /* Add u to the final forest */
        Ẽ ← Ẽ ∪ {e}  /* Add e to the final forest */
```

Successors

Add successors to the fringe.

Generalized Distributed Constrained Forest Algorithm

for Multidirectional Graph Search

```
procedure MULTIDIRECTIONAL-GRAPH-SEARCH(
Require: v is the start vertex running this search.
Ensure: H = ⟨Ṽ, Ẽ⟩ is the resulting forest.
    Ṽ ← {v}
    Ẽ ← ∅
    F ← our neighbors   /* The fringe of our search */
    g(v) ← 0 for all v ∈ V   /* Initialize the path-cost function to 0 */
    while Our interaction constraints are still unsatisfied do
        Find an edge e = ⟨u, v⟩ in the fringe that minimizes ε = w(e) − g(u) − g(v)
        if u either is being or already was expanded by another search then
            Merge our execution with u's search.
            if The other search also expanded the edge ⟨v, u⟩ in this round then
                ϵ ← ϵ/2
        for all k ∈ Ṽ : k is incident to an edge in the fringe do
            g(k) ← g(k) + ε   /* Update the path-cost */
        F ← (F \ {e}) ∪ δ({u})   /* Update the fringe with e's successors */
        Ṽ ← Ṽ ∪ {u}   /* Add u to the final forest */
        Ẽ ← Ẽ ∪ {e}   /* Add e to the final forest */
```

Gradient!

This is the potential function that will ensure 2-OPT

Generalized Distributed Constrained Forest Algorithm

fo

Dual Variables

The path-cost implicitly initializes the dual variables.

procedure MULTIDIRECTIONAL-GRAPH-SEARCH(v)
Require: v is the start vertex running this search.
Ensure: $H = \langle \tilde{V}, \tilde{E} \rangle$ is the resulting forest.
  $\tilde{V} \leftarrow \{v\}$
  $\tilde{E} \leftarrow \emptyset$
  $F \leftarrow$ our neighbors /* The fringe of our search */
  $g(v) \leftarrow 0$ for all $v \in V$ /* Initialize the path-cost function to 0 */
  while Our interaction constraints are still unsatisfied **do**
    Find an edge $e = \langle u, v \rangle$ in the fringe that minimizes $\varepsilon = w(e) - g(u) - g(v)$
    if u either is being or already was expanded by another search **then**
      Merge our execution with u's search.
      if The other search also expanded the edge $\langle v, u \rangle$ in this round **then**
        $\epsilon \leftarrow \frac{\epsilon}{2}$
    for all $k \in \tilde{V} : k$ is incident to an edge in the fringe **do**
      $g(k) \leftarrow g(k) + \varepsilon$ /* Update the path-cost */
    $F \leftarrow (F \setminus \{e\}) \cup \delta(\{u\})$ /* Update the fringe with e's successors */
    $\tilde{V} \leftarrow \tilde{V} \cup \{u\}$ /* Add u to the final forest */
    $\tilde{E} \leftarrow \tilde{E} \cup \{e\}$ /* Add e to the final forest */

Generalized Distributed Constrained Forest Algorithm

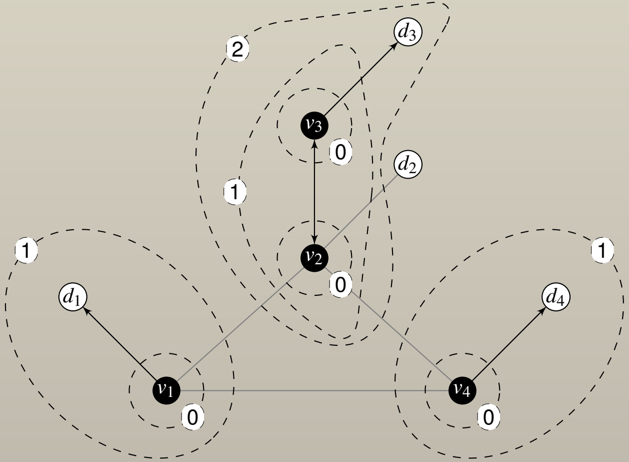
for Multidirectional Graph Search

procedure MULTIDIRECTIONAL-GRAPH-SEARCH(v)
Require: v is the start vertex running this search.
Ensure: $H = \langle \tilde{V}, \tilde{E} \rangle$ is the resulting forest.
    $\tilde{V} \leftarrow \{v\}$
    $\tilde{E} \leftarrow \emptyset$
    … our search $*/$
    … path-cost function to 0 $*/$
    … still unsatisfied **do**
        Find an edge $e = \langle u, v \rangle$ in the fringe that minimizes $\varepsilon = w(e) - g(u) - g(v)$
        if u either is being or already was expanded by another search **then**
            Merge our execution with u's search.
            if The other search also expanded the edge $\langle v, u \rangle$ in this round **then**
                $\epsilon \leftarrow \frac{\epsilon}{2}$
        for all $k \in \tilde{V}$: k is incident to an edge in the fringe **do**
            $g(k) \leftarrow g(k) + \varepsilon$ $/*$ Update the path-cost $*/$
        $F \leftarrow (F \setminus \{e\}) \cup \delta(\{u\})$ $/*$ Update the fringe with e's successors $*/$
        $\tilde{V} \leftarrow \tilde{V} \cup \{u\}$ $/*$ Add u to the final forest $*/$
        $\tilde{E} \leftarrow \tilde{E} \cup \{e\}$ $/*$ Add e to the final forest $*/$

Pushing Up the Duals

Implicitly sets $y_{\tilde{V}} \leftarrow y_{\tilde{V}} + \epsilon$.

Technical Sketch



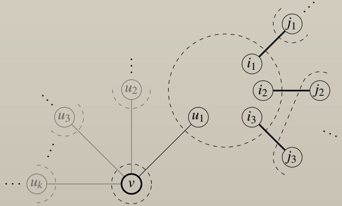
Technical Sketch

Assumptions

- No knowledge of global state.
- Only need to know one's neighbors.
- Guaranteed message delivery, but arbitrary latency.

Dynamic/Streaming Problems

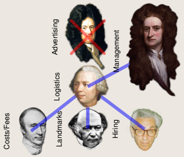
1. In certain well-defined cases, after a *modification event* the algorithm can be continued and is still guaranteed to produce a 2-optimal solution.
   - If the weights of all of v's incident edges are greater than or equal to the slack of all of their neighboring vertices' fringe nodes:



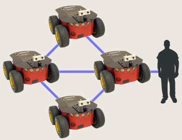
2. In all other cases, we can backtrack to the most recent round during which the conditions allowed for the dynamic modification.
3. Worst case: backtrack to the start, which is only $O(n)$ rounds.
4. Backtracking only increases memory/computation polynomially.

Example Domains

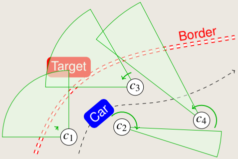
Corporations



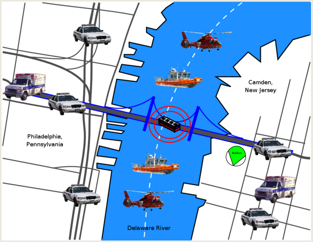
Robotic Teaming



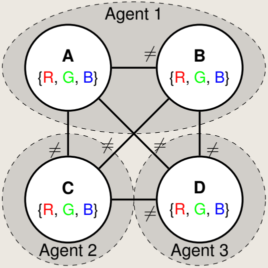
Sensor Networks



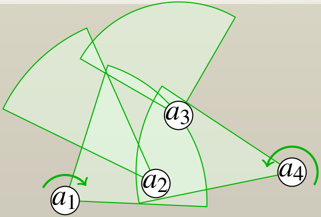
Service Composition

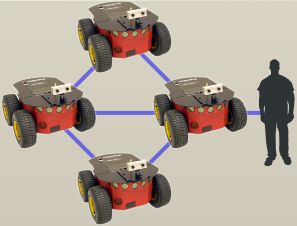


DCR



Robot Teaming





- Group of mobile robots each equipped with a wireless access point.
  - **Objective of the robots**: maximally cover an area with the wireless network.
  - **In order to save power**: Choose a maximum subset of robots that can lower their transmit power while still retaining coverage.

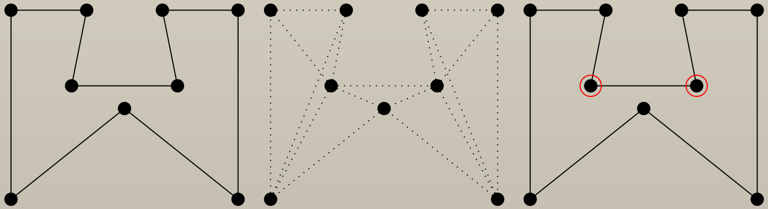
E. Sultanik, A. Shokoufandeh, and W. Regli
Dominating Sets of Agents in Visibility Graphs: Distributed Algorithms for Art Gallery Problems.
In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, May 2010.

APL

Art Gallery Problems

Example

Find the minimum number of guards required to observe the interior of a polygonal area.

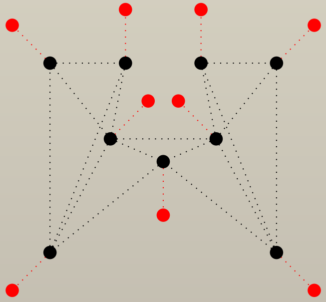


Variants

- Guards in the interior.
- Treasures.
- Non-uniform cost for stationing a guard.
- **NP**-COMPLETE.

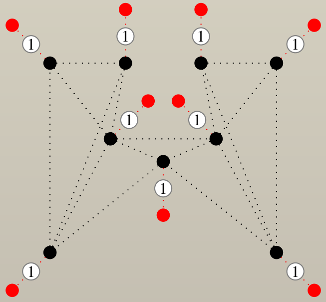
Equiv. as a Connectivity Problem

Augment each vertex with a special guard vertex ("●").



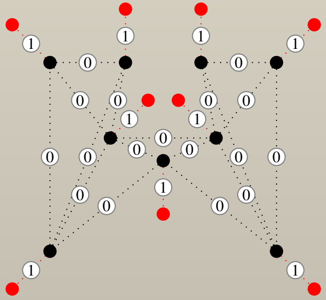
Equiv. as a Connectivity Problem

Weight the new edges with the cost of guarding from there.

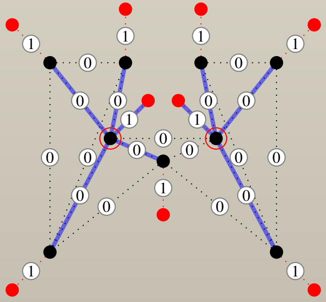


Equiv. as a Connectivity Problem

Weight the original visibility graph edges 0.



Equiv. as a Connectivity Problem

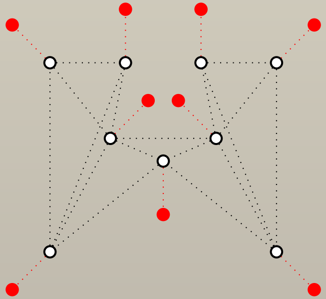


Connectivity Problem: Find an acyclic subgraph such that:

1. every ● is connected to a ● by a path of length ≤ 2; and
2. the subgraph's weight is minimized.

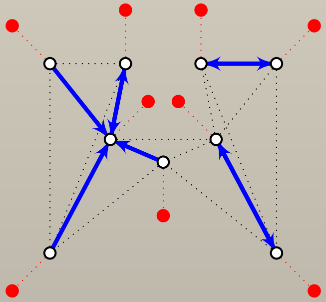
Technical Sketch

Round 0: All Vertices are Unguarded ("$\circ$")



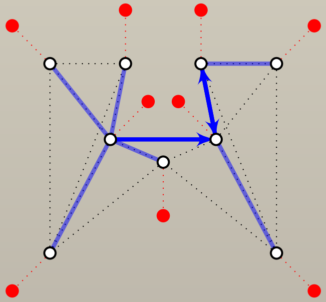
Technical Sketch

Round 1**:** Unguarded components add cut-edge of min. potential.



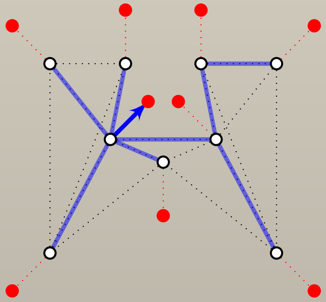
Technical Sketch

Round 2**:** Unguarded components add cut-edge of min. potential.



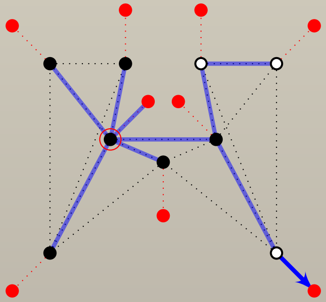
Technical Sketch

Round 3**:** Unguarded components add cut-edge of min. potential.



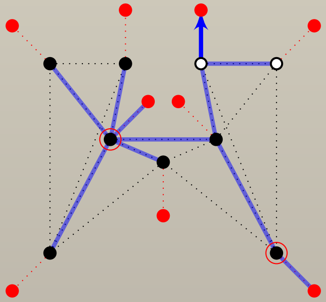
Technical Sketch

Round 4**:** Unguarded components add cut-edge of min. potential.



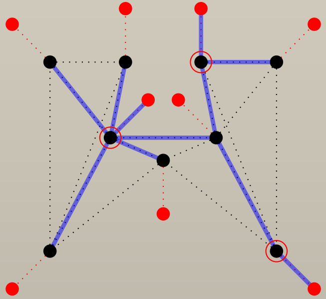
Technical Sketch

Round 5**:** Unguarded components add cut-edge of min. potential.



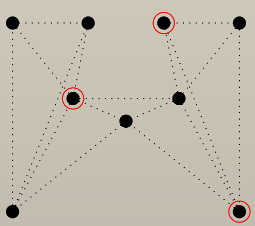
Technical Sketch

Round 6**:** All nodes are guarded, so we terminate.



Technical Sketch

Round 6**:** All nodes are guarded, so we terminate.



Conclusions

- Certain streams data processing problems can be cast as a dynamic/distributed multiagent optimization problem.
- Measuring the performance of distributed algorithms is hard.
- In most cases, we want $O(n)$ communication rounds.
- Distributed constraint reasoning is a powerful model that is useful for many problems.
- Approximation algorithms are often *useful* and sometimes *necessary*.
- The Primal-Dual Schema is a very powerful tool for approximation.

Thank you for your time and attention.

Questions?

Evan A. Sultanik
Evan.Sultanik@jhuapl.edu
http://www.sultanik.com/