Algorithm Design XX

Coping with NP-Completeness III: Local Search

Guoqiang Li
School of Software

Shanghai Jiao Tong University

Coping with NP-completeness

Q. Suppose I need to solve an **NP**-complete problem. What should I do?

A. Theory says you're unlikely to find polynomial time algorithm.

Must sacrifice one of three desired features.

- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.

Think About: What features have been sacrificed in today's topic?

Gradient Descent

Gradient Descent: Vertex Cover

VERTEX COVER. Given a graph $G = (V, E)$, find a subset of nodes S of minimal cardinality such that for each $(u, v) \in E$, either u or v are in S.

Neighbor relation. $S \sim S'$ if S' can be obtained from S by adding or deleting a single node.

Note. Each vertex cover S has at most n neighbors.

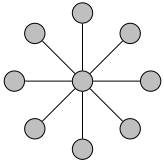
Gradient descent. Start with $S = V$. If there is a neighbor S' that is a vertex cover and has lower cardinality, replace S with S'.

Remark

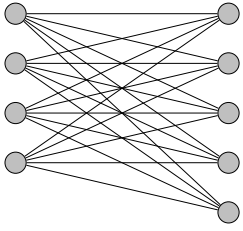
Algorithm terminates after at most n steps since each update decreases the size of the cover by one.

Gradient Descent: Vertex Cover

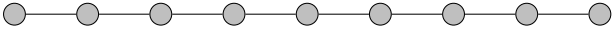
Local optima. No neighbor is strictly better.



optimum = center node only
local optimum = all other nodes



optimum = all nodes on left side
local optimum = all nodes on right side



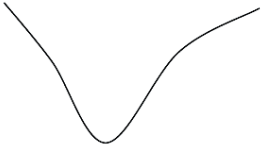
optimum = even nodes
local optimum = omit every third node

Local Search

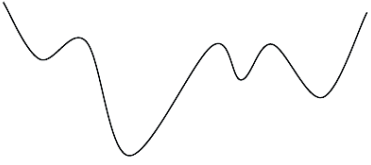
Local search. Algorithm that explores the space of possible solutions in sequential fashion, moving from a current solution to a "nearby" one.

Neighbor relation. Let $S \sim S'$ be a neighbor relation for the problem.

Gradient descent. Let S denote current solution. If there is a neighbor S' of S with strictly lower cost, replace S with the neighbor whose cost is as small as possible. Otherwise, terminate the algorithm.



A funnel



a jagged funnel

Local Search


```
let s be any initial solution
while there is some solution s' in the neighborhood of s
   for which cost(s') < cost(s):  replace s by s'
return s
```

Further Example: TSP

Assume we have all interpoint distances between n cities, giving a search space of $(n-1)!$ different tours. What is a good notion of neighborhood?

Two tours are close if they differ in just a few edges. They can't differ in just one edge, so we will consider differences of two edges.

We define the 2-change neighborhood of tour s as being the set of tours that can be obtained by removing two edges of s and then putting in two other edges.

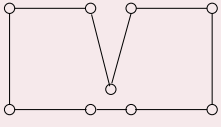
Evaluation of 2-Change Neighborhood

Q: What is its overall running time, and does it always return the best solution?

Neither of these questions has a satisfactory answer:

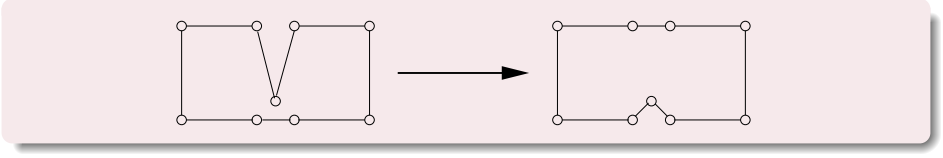
Each iteration is certainly fast, because a tour has only $O(n^2)$ neighbors. There might be an exponential number of them.

We can easily say about the final tour is that it is locally optimal. There might be better solutions further away.



3-Change Neighborhood

We may try a more generous neighborhood, for instance 3-change, consisting of tours that differ on up to three edges.



The size of a neighborhood becomes $O(n^3)$, making each iteration more expensive. Moreover, there may still be suboptimal local minima, although fewer than before.

To avoid these, we would have to go up to 4-change, or higher.

Efficiency demands neighborhoods that can be searched quickly, but smaller neighborhoods can increase the abundance of low-quality local optima.

The appropriate compromise is typically determined by experimentation.

Further Example: Graph Partitioning

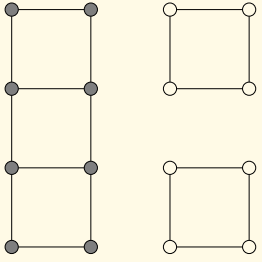
Definition (Graph Partitioning)

Input: An undirected graph $G = (V, E)$ with nonnegative edge weights, a real number $\alpha \in (0, 1/2]$.

Output: A partition of the vertices into two groups A and B, each of size at least $\alpha|V|$.

Goal: Minimize the capacity of the cut (A, B).

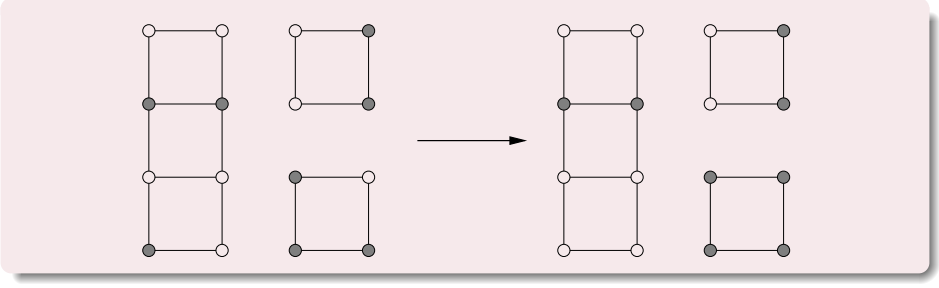
An Example



16 nodes, all edge weights are 0 or 1, $\alpha = 1/2$.

Neighbourhood

Let (A, B), with $|A| = |B|$, be a candidate solution; we will define its neighbors to be all solutions obtainable by swapping one pair of vertices across the cut, that is, all solutions of the form $(A - \{a\} + \{b\}, B - \{b\} + \{a\})$ where $a \in A$ and $b \in B$.



The search space includes some local optima that are quite far from the global solution.

Metropolis Algorithm

Metropolis Algorithm

Metropolis algorithm.

- Simulate behavior of a physical system according to principles of statistical mechanics.
- Globally biased toward "downhill" steps, but occasionally makes "uphill" steps to break out of local minima.

Gibbs-Boltzmann Function

Gibbs-Boltzmann Function

The probability of finding a physical system in a state with energy E is proportional to $e^{-E/(kT)}$, where $T > 0$ is temperature and k is a constant.

- For any temperature $T > 0$, function is monotone decreasing function of energy E.
- System more likely to be in a lower energy state than higher one.
  - T large: high and low energy states have roughly same probability
  - T small: low energy states are much more probable

Metropolis Algorithm

Metropolis algorithm.

- Given a fixed temperature T, maintain current state S.
- Randomly perturb current state S to new state $S' \in N(S)$.
- If $E(S') \leq E(S)$, update current state to S'.
- Otherwise, update current state to S' with probability $e^{-\Delta E/(KT)}$, where $\Delta E = E(S') - E(S) > 0$.

Theorem

Let $f_S(t)$ be fraction of first t steps in which simulation is in state S. Then, assuming some technical conditions, with probability 1:

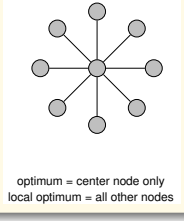
$$\lim_{t \to \infty} f_S(t) = \frac{1}{Z} e^{-E(S)/(kT)} \text{ where } Z = \sum_{S \in N(S)} e^{-E(S)/(kT)}$$

Intuition. Simulation spends roughly the right amount of time in each state, according to Gibbs-Boltzmann equation.

Metropolis Algorithm in Minimum Problems

Start with an initial solution S_0, and constants k and T
In one step:
  Let S be the current solution
  Let S' be chosen uniformly at random from the neighbors of S
  If $c(S') \leq c(S)$ then
    Update $S \leftarrow S'$
  Else
    With probability $e^{-(c(S')-c(S))/(kT)}$
      Update $S \leftarrow S'$
    Otherwise
      Leave S unchanged
  EndIf

Strength of Metropolis Algorithm



On the VERTEX COVER instance consisting of the star graph, the Metropolis algorithm will quickly bounce out of the local minimum that arises when the central point is deleted.

The neighboring solution in which the central point is put back in will be generated and will be accepted with positive probability.

Weakness of Metropolis Algorithm

Consider a graph G with no edges, gradient descent solves this instance with no trouble, deleting nodes in sequence until none are left.

While the Metropolis algorithm will start out this way, it begins to go astray as it nears the global optimum.

Consider the situation in which the current solution contains only c nodes, where c is much smaller than the total number of nodes, n.

With very high probability, the neighboring solution generated by the Metropolis Algorithm will have size $c + 1$, rather than $c - 1$, Thus it gets harder and harder to shrink the size of the vertex cover as the algorithm proceeds.

It is exhibiting a sort of flinching reaction near the bottom of the funnel.

Simulated Annealing

Simulated annealing.

- T large $\Rightarrow$ probability of accepting an uphill move is large.
- T small $\Rightarrow$ uphill moves are almost never accepted.
- Idea: turn knob to control T.
- Cooling schedule: $T = T(i)$ at iteration i.

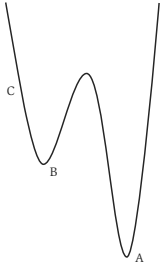
Physical analog.

- Take solid and raise it to high temperature, we do not expect it to maintain a nice crystal structure.
- Take a molten solid and freeze it very abruptly, we do not expect to get a perfect crystal either.
- Annealing: cool material gradually from high temperature, allowing it to reach equilibrium at succession of intermediate lower temperatures.

Simulated Annealing

Simulated annealing works by running the Metropolis algorithm while gradually decreasing the value of T over the course of the execution.

The exact way in which T is updated is called a cooling schedule,which is a function τ from $\{1, 2, 3, \ldots\}$ to the positive real numbers; in iteration i, we use the temperature $T = \tau(i)$.



Physical systems reach a minimum energy state via annealing, the simulated annealing has no guarantee to find an optimal solution.

If the two funnels take equal area, then at high temperatures the system is essentially equally likely to be in either funnel.

Once cooling the temperature, it will become harder and harder to switch between the two funnels.

Hopfield Neural Networks

Hopfield Neural Networks

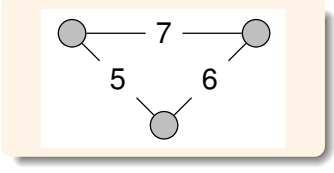
Hopfield networks. Simple model of an associative memory, in which a large collection of units are connected by an underlying network, and neighboring units try to correlate their states.

Input: Graph $G = (V, E)$ with integer (positive or negative) edge weights w.

Configuration. Node assignment $s_u = \pm 1$.

Intuition. If $w_{uv} < 0$, then u and v want to have the same state; if $w_{uv} > 0$ then u and v want different states.

Note. In general, no configuration respects all constraints.



Definition (Good edge)

With respect to a configuration S, edge $e = (u, v)$ is good if $w_e \times s_u \times s_v < 0$. That is, if $w_e < 0$ then $s_u = s_v$; if $w_e > 0$, then $s_u \neq s_v$.

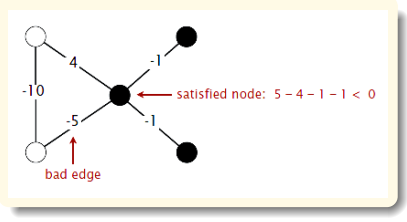
Definition (Satisfied node)

With respect to a configuration S, a node u is satisfied if the weight of incident good edges $\geq$ weight of incident bad edges.

$$\sum_{v:e=(u,v)\in E} w_e s_u s_v \leq 0$$

Definition (Stable configuration)

A configuration is stable if all nodes are satisfied.



Goal. Find a stable configuration, if such a configuration exists.

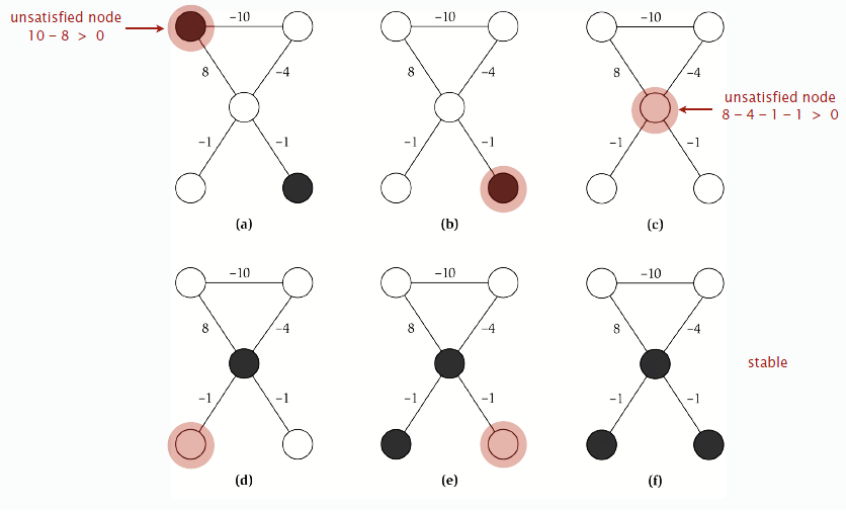
State-Flipping Algorithm

State-flipping algorithm. Repeated flip state of an unsatisfied node.

HOPFIELD-FLIP(G, w)

$S \leftarrow$ arbitrary configuration;
while *current configuration is not stable* **do**
    $u \leftarrow$ unsatisfied node;
    $s_u \leftarrow -s_u$;
end
return S;

State-Flipping Algorithm Example



State-Flipping Algorithm: Proof of Correctness

Theorem

The state-flipping algorithm terminates with a stable configuration after at most $W = \sum_e |w_e|$ *iterations.*

Proof [Hint.] Consider measure of progress $\Phi(S) = \#$ satisfied nodes.

State-Flipping Algorithm: Proof of Correctness

Theorem

The state-flipping algorithm terminates with a stable configuration after at most $W = \sum_e |w_e|$ *iterations.*

Proof. Consider measure of progress $\Phi(\mathrm{S}) = \sum_{\mathrm{e\ good}} |\mathrm{w_e}|$.

- Clearly $0 \leq \Phi(\mathrm{S}) \leq \mathrm{W}$.
- We show $\Phi(\mathrm{S})$ increase by at least 1 after each flip.
  When u flips state:
  - all good edges incident to u become bad
  - all bad edges incident to u become good
  - all other edges remain the same

$$\Phi\left(\mathrm{S}'\right) = \Phi(\mathrm{S}) - \sum_{\substack{e\,:\,e=(u,v)\in E \\ e \text{ is bad}}} |\mathrm{w_e}| + \sum_{\substack{e\,:\,e=(u,v)\in E \\ e \text{ is good}}} |\mathrm{w_e}| \geq \Phi(\mathrm{S}) + 1$$

Maximum Cut

Maximum Cut

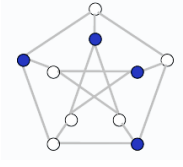
Maximum cut. Given an undirected graph $G = (V, E)$ with positive integer edge weights w_e, find a $cut(A, B)$ such that the total weight of edges crossing the cut is maximized.

$$w(A, B) := \sum_{u \in A, v \in B} w_{uv}$$

Toy application.

- n activities, m people.
- Each person wants to participate in two of the activities.
- Schedule each activity in the morning or afternoon to maximize number of people that can enjoy both activities.

Real applications. Circuit layout, statistical physics.



Single-flip neighborhood. Given a cut (A, B), move one node from A to B, or one from B to A if it improves the solution.

Greedy algorithm.

```
MAX-CUT-LOCAL(G, w)
while there exists an improving node v do
    if v ∉ A then
        A ← A ∪ {v}; B ← B − {v};
    end
    else
        B ← B ∪ {v}; A ← A − {v};
    end
end
return(A,B)
```

Theorem

Let (A, B) be a locally optimal cut and let (A^*, B^*) be an optimal cut. Then $w(A, B) \geq 1/2 \sum_e w_e \geq 1/2 w(A^*, B^*)$.

Proof.

- Local optimality implies that for all $u \in A : \sum_{v \in A} w_{uv} \leq \sum_{v \in B} w_{uv}$.
- Adding up all these inequalities yields: $2 \sum_{\{u,v\} \subseteq A} w_{uv} \leq \sum_{u \in A, v \in B} w_{uv} = w(A, B)$
- Similarly $2 \sum_{\{u,v\} \subseteq B} w_{uv} \leq \sum_{u \in A, v \in B} w_{uv} = w(A, B)$
- Now,

$$\sum_{e \in E} w_e = \underbrace{\sum_{\{u,v\} \subseteq A} w_{uv}}_{\leq \frac{1}{2} w(A,B)} + \underbrace{\sum_{u \in A, v \in B} w_{uv}}_{w(A,B)} + \underbrace{\sum_{\{u,v\} \subseteq A} w_{uv}}_{\leq \frac{1}{2} w(A,B)} \leq 2w(A, B)$$

Maximum Cut: Big Improvement Flips

Local search. Within a factor of 2 for MAX-CUT, but not polynomial time!

Big-improvement-flip algorithm. Only choose a node which, when flipped, increases the cut value by at least $\frac{2\varepsilon}{n} w(A, B)$

Claim

Upon termination, big-improvement-flip algorithm returns a cut (A, B) such that $(2 + \varepsilon)w(A, B) \geq w(A^*, B^*)$

Proof idea. Add $\frac{2\varepsilon}{n} w(A, B)$ to each inequality in original proof.

Maximum Cut: Big Improvement Flips

Claim

Big-improvement-flip algorithm terminates after $O\left(\varepsilon^{-1} n \log W\right)$ *flips, where* $W = \sum_e w_e$.

Proof sketch.

Each flip improves cut value by at least a factor of $(1 + \varepsilon/n)$.

After n/ε iterations the cut value improves by a factor of 2.

- $(1 + 1/x)^x \geq 2$ for $x \geq 1$.

Cut value can be doubled at most $\log_2 W$ times.

Maximum Cut: Context

SHANGHAI JIAO TONG UNIVERSITY

Theorem (Sahni-Gonzales 1976)

There exists a $1/2$-approximation algorithm for MAX-CUT.

Theorem

There exists an 0.878-approximation algorithm for MAX-CUT.

Theorem

*Unless **P = NP**, no 0.942-approximation algorithm for MAX-CUT.*

Neighbor Relations for Max Cut

1-flip neighborhood. Cuts (A, B) and (A', B') differ in exactly one node.

k-flip neighborhood. Cuts (A, B) and (A', B') differ in at most k nodes.

KL-neighborhood.[Kernighan-Lin 1970]

- To form neighborhood of (A, B):
  - Iteration 1: flip node from (A, B) that results in best cut value (A_1, B_1), and mark that node
  - Iteration i: flip node from (A_{i-1}, B_{i-1}) that results in best cut value (A_i, B_i) among all nodes not yet marked.
- Neighborhood of $(A, B) = (A_1, B_1), \ldots, (A_{n-1}, B_{n-1})$.
- Neighborhood includes some very long sequences of flips, but without the computational overhead of a k-flip neighborhood.
- Practice: powerful and useful framework.
- Theory: explain and understand its success in practice.

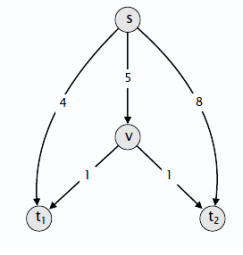
Nash Equilibria

Multicast Routing

Multicast routing. Given a directed graph $G = (V, E)$ with edge costs $c_e \geq 0$, a source node s, and k agents located at terminal nodes $t_1, \ldots, t_k$. Agent j must construct a path P_j from node s to its terminal t_j.

Fair share. If x agents use edge e, they each pay c_e/x.

1	2	1 pays	2 pays
outer	outer	4	8
outer	middle	4	5+1
middle	outer	5+1	8
middle	middle	5/2 + 1	5/2 + 1



Best response dynamics. Each agent is continually prepared to improve its solution in response to changes made by other agents.

Nash equilibrium. Solution where no agent has an incentive to switch.

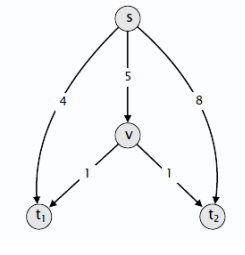
Fundamental question. When do Nash equilibria exist?

Two agents start with outer paths.

Agent 1 has no incentive to switch paths, since $4 < 5 + 1$. But agent 2 does since $8 > 5 + 1$.

Once this happen, agent 1 prefers middle path (since $4 > 5/2 + 1$)

Both agents using middle path is a Nash equilibrium.



Local search algorithm. Each agent is continually prepared to improve its solution in response to changes made by other agents.

Analogies.

- Nash equilibrium : local search.
- Best response dynamics : local search algorithm.
- Unilateral move by single agent : local neighborhood.

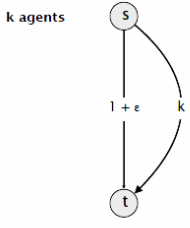
Contrast. Best-response dynamics need not terminate since no single objective function is being optimized.

Socially Optimum

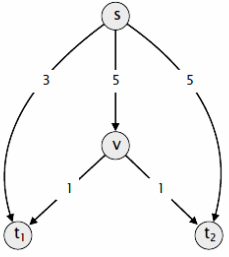
Social optimum. Minimizes total cost to all agent.

Observation. In general, there can be many Nash equilibria.

Even when its unique, it does not necessarily equal the social optimum.



social optimum = 1 + ε
Nash equilibrium A = 1 + ε
Nash equilibrium B = k



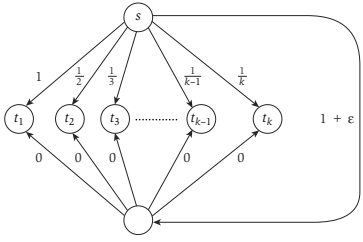
social optimum = 7
unique Nash equilibrium = 8

Price of Stability

Price of stability. Ratio of best Nash equilibrium to social optimum.

Fundamental question. What is price of stability?

Example. Price of stability = $\Theta(\log k)$.



Social optimum. Everyone takes bottom paths.

Unique Nash equilibrium. Everyone takes top paths.

Price of stability. $H(k)/(1+\varepsilon)$

Finding a Nash Equilibrium

Theorem

The following algorithm terminates with a Nash equilibrium.

BEST-RESPONSE-DYNAMICS (G, c, k)

for $j = 1$ *to* k **do**
  $P_j \leftarrow$ any path for agent j;
end
while *not a Nash equilibrium* **do**
  $j \leftarrow$ some agent who can improve by switching paths;
  $P_j \leftarrow$ better path for agent i;
end
return $P_1, P_2, \ldots P_k$

Proof. Consider a set of $P_1, \ldots, P_k$

- Let x_e denote the number of paths that use edge e.
- Let $\Phi(P_1, P_2, \ldots P_k) = \sum_{e \in E} c_e \cdot H(x_e)$ be a potential function, where

$$H(0) = 0$$
$$H(k) = \sum_{i=1}^{k} \frac{1}{i}$$

- Since there are only finitely many sets of paths, it suffices to show that Φ strictly decreases in each step.

Proof. [continued]

- Consider agent j switching from path P_j to path P_j'.
- Agent j switches because

$$\underbrace{\sum_{f \in P_j' - P_j} \frac{c_f}{x_f + 1}}_{\text{newly incurred cost}} < \underbrace{\sum_{e \in P_j - P_j'} \frac{c_e}{x_e}}_{\text{cost saved}}$$

- Φ increase by $\sum_{f \in P_j' - P_j} c_f \left[H\left(x_f + 1\right) - H\left(x_f\right) \right] = \sum_{f \in P_j' - P_j} \frac{c_f}{x_f + 1}$.
- Φ decrease by $\sum_{e \in P_j - P_j'} c_e \left[H\left(x_e\right) - H\left(x_e - 1\right) \right] = \sum_{e \in P_j - P_j'} \frac{c_c}{x_e}$
- Thus, net change in Φ is negative.

Lemma

Let $C(P_1, \ldots, P_k)$ denote the total cost of selecting paths $P_1, \ldots, P_k$. For any set of paths $P_1, \ldots, P_k$, we have

$$C\left(P_1, \ldots, P_k\right) \leq \Phi\left(P_1, \ldots, P_k\right) \leq H(k) \cdot C\left(P_1, \ldots, P_k\right)$$

Proof.

Let x_e denote the number of paths containing edge e.

- Let E^+ denote set of edges that belong to at least one of the paths. Then,

$$C\left(P_1, \ldots, P_k\right) = \sum_{e \in E^+} c_e \leq \underbrace{\sum_{e \in E^+} c_e H\left(x_e\right)}_{\Phi(P_1, \ldots, P_k)} \leq \sum_{e \in E^+} c_e H(k) = H(k) C\left(P_1, \ldots, P_k\right)$$

Bounding the Price of Stability

Theorem

There is a Nash equilibrium for which the total cost to all agents exceeds that of the social optimum by at most a factor of $H(k)$.

Proof.

- Let $(P_1^*, \ldots, P_k^*)$ denote a set of socially optimal paths.
- Run best-response dynamics algorithm starting from P^*.
- Since Φ is monotone decreasing $\Phi(P_1, \ldots, P_k) \leq \Phi(P_1^*, \ldots, P_k^*)$,

$$C(P_1, \ldots, P_k) \leq \Phi(P_1, \ldots, P_k) \leq \Phi(P_1^*, \ldots, P_k^*) \leq H(k) \cdot C(P_1^*, \ldots, P_k^*)$$

Summary

Existence. Nash equilibria always exist for k-agent multicast routing with fair sharing.

Price of stability. Best Nash equilibrium is never more than a factor of $H(k)$ worse than the social optimum.

Fundamental open problem. Find any Nash equilibria in polynomial time.