

12. Local Search

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

Coping with NP-completeness

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

A. Sacrifice one of three desired features.

- 1. Solve *arbitrary instances* of the problem.
- 2. Solve problem to *optimality*.
- 3. Solve problem in *polynomial time*.

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 (or both) are in S .

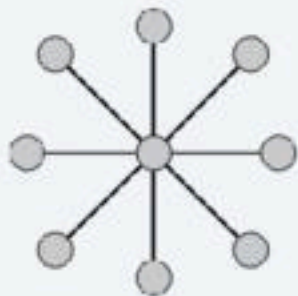
Neighbor relation. $S \sim S'$ if S' can be obtained from S by adding or deleting a single node. Each vertex cover S has at most n neighbors.

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' .

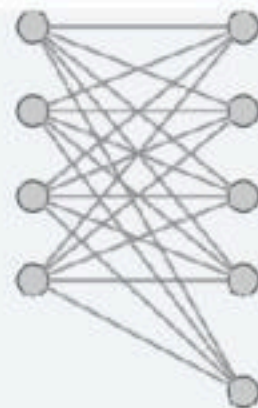
Remark. Algorithm terminates after at most n steps since each update decreases the size of the cover by one.

Gradient descent: vertex cover

Local optimum. 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.

Problem. Stuck into local optimal 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:

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

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

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.

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: 1-bit of memory.

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.

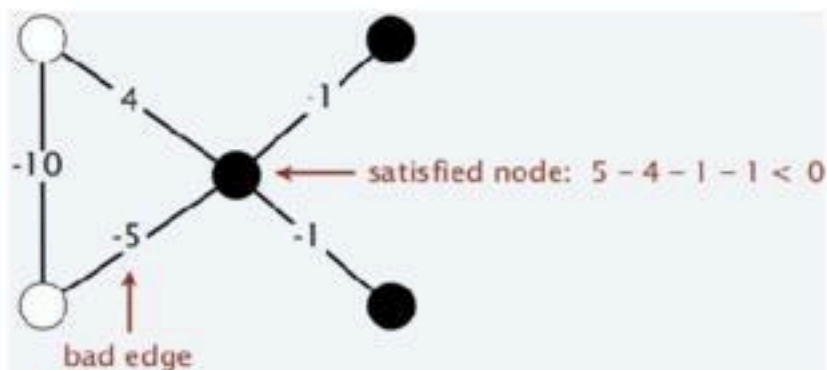
Hopfield neural networks

Def. 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 $s_u \neq s_v$.

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

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

Def. A configuration is **stable** if all nodes are satisfied.



Hopfield neural networks

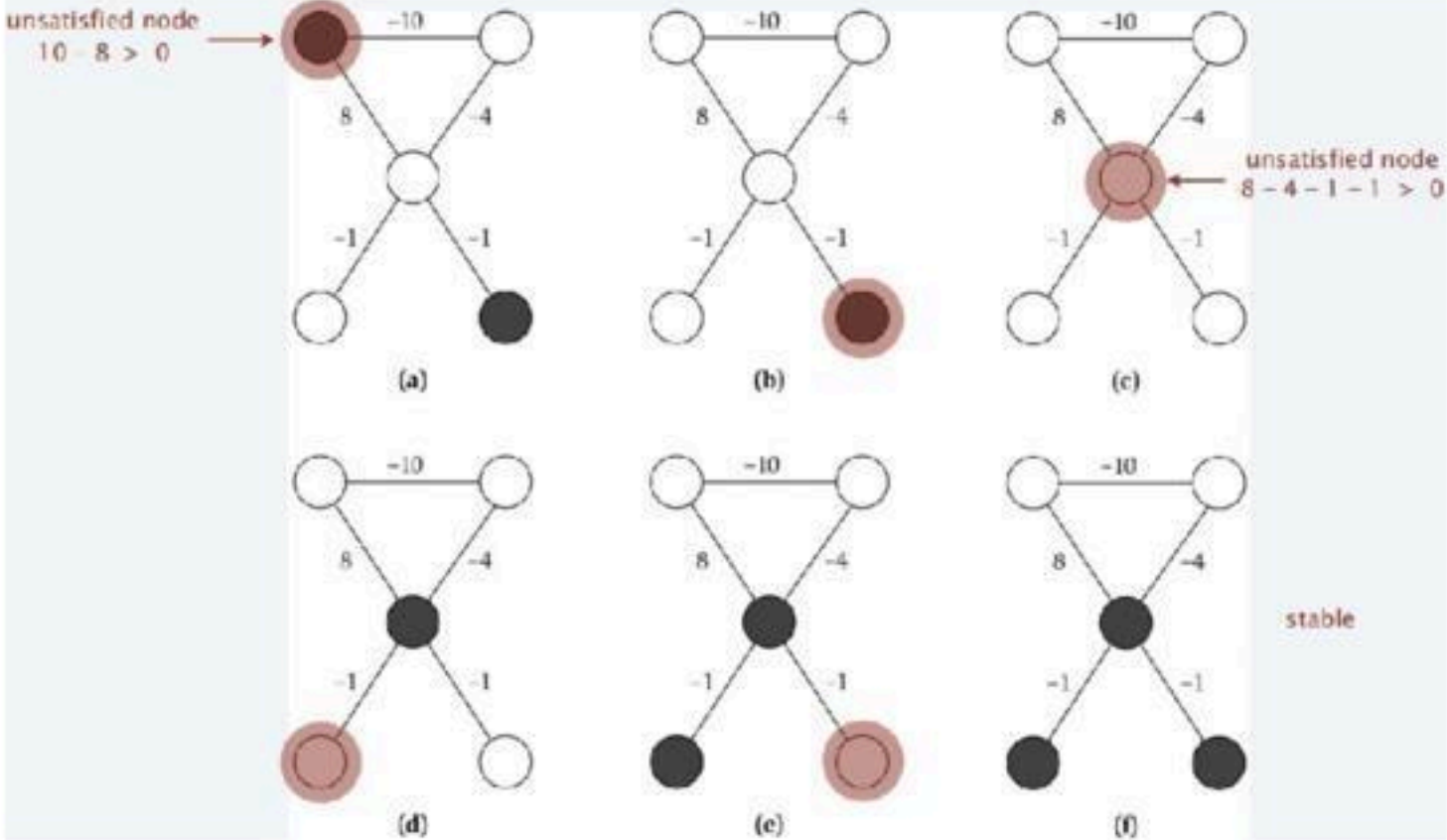
Goal. Find a stable configuration, if such a configuration exists.

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

HOPFIELD-FLIP (G, w)

1. S = arbitrary configuration;
2. WHILE (current configuration is not stable):
 1. u = unsatisfied node;
 2. $s_u = -s_u$;
3. RETURN S ;

State-flipping: example



State-flipping: correctness

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

Pf attempt. Consider measure of progress $\Phi(S) = \#$ satisfied nodes.

State-flipping: correctness

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

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

- Clearly $0 \leq \Phi(S) \leq W$.
- We show $\Phi(S)$ increases 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

$$\Phi(S') = \Phi(S) - \sum_{e \text{ bad}} |w_e| + \sum_{e \text{ good}} |w_e| \geq \Phi(S) + 1$$

Complexity of Hopfield neural network

Hopfield network search problem. Given a weighted graph, find a stable configuration if one exists.

Hopfield network decision problem. Given a weighted graph, does there exist a stable configuration?

Remark. The decision problem is trivially solvable (always yes), but there is no known poly-time algorithm for the search problem.

- polynomial in n and $\log W$

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.

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.

Maximum cut

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)

1. $(A, B) =$ random cut;
2. WHILE (there exists an improving node v):
 1. IF $v \notin A$:
 1. $A = A \cup \{v\}$;
 2. $B = B - \{v\}$;
 2. ELSE $v \in A$:
 1. $B = B \cup \{v\}$;
 2. $A = A - \{v\}$;
3. RETURN (A, B) ;

Maximum cut: local search analysis

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

Pf.

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

Maximum cut: big improvement flips

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

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

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

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

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

- Each flip improves cut value by at least a factor of $(1 + \epsilon/n)$.
- After n/ϵ iterations the cut value improves by a factor of 2.
- Cut value can be doubled at most $\log_2 W$ times.

Maximum cut: context

Theorem. [Sahni-Gonzales 1976] There exists a $\frac{1}{2}$ -approximation algorithm for MAX-CUT.

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

Theorem. Unless $\mathcal{P} = \mathcal{NP}$, 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 $(A, B) = (A_1, B_1), \dots, (A_{i-1}, B_{i-1})$.
- 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.

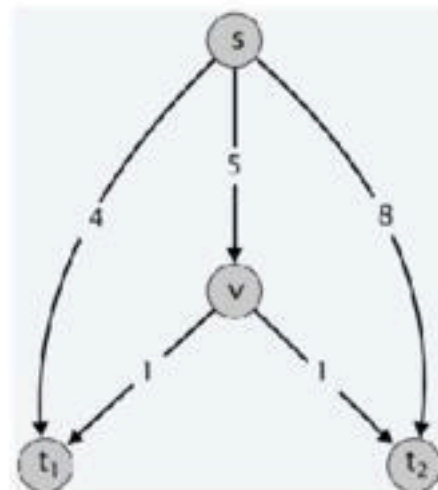
Nash equilibria

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, \dots, 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	cost 1	cost 2
outer	outer	4	8
outer	middle	4	$5 + 1$
middle	outer	$5 + 1$	8
middle	middle	$5/2 + 1$	$5/2 + 1$



Multicast routing

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?

Multicast routing

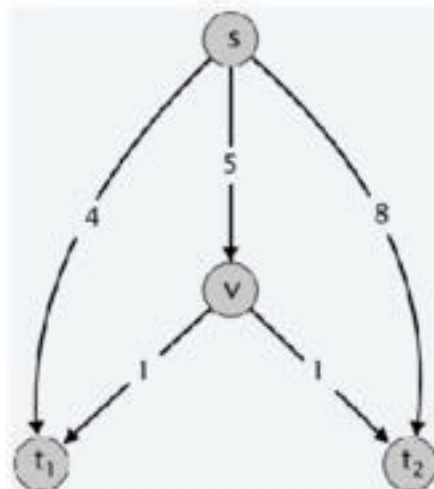
Best response dynamics. Each agent is continually prepared to improve its solution in response to changes made by other agents.

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

Fundamental question. When do Nash equilibria exist?

Ex:

1. Two agents start with outer paths.
2. Agent 1 has no incentive to switch paths (since $4 < 5 + 1$), but agent 2 does (since $8 > 5 + 1$).
3. Once this happens, agent 1 prefers middle path (since $4 > 5/2 + 1$).
4. Both agents using middle path is a Nash equilibrium.



Nash equilibrium and local search

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

Analogies.

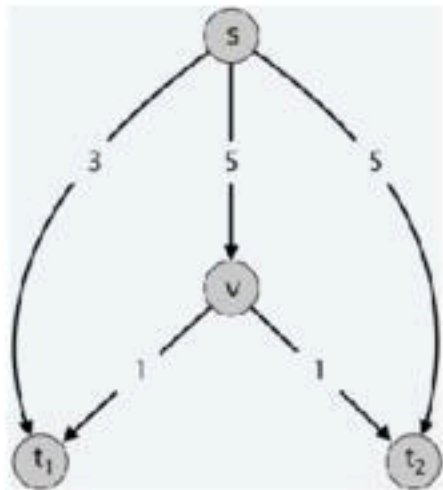
- Nash equilibrium : local search.
- Best response dynamics : local search algorithm.
- Unilateral move by single agent : local neighborhood.

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

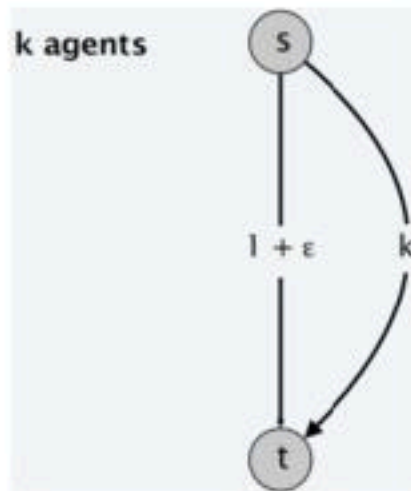
Socially optimum

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 = 7
unique Nash equilibrium = 8



social optimum = $(1 + \epsilon)/k$
Nash equilibrium A = $(1 + \epsilon)/k$
Nash equilibrium B = 1

Price of stability

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

Fundamental question. What is price of stability?

Price of stability

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

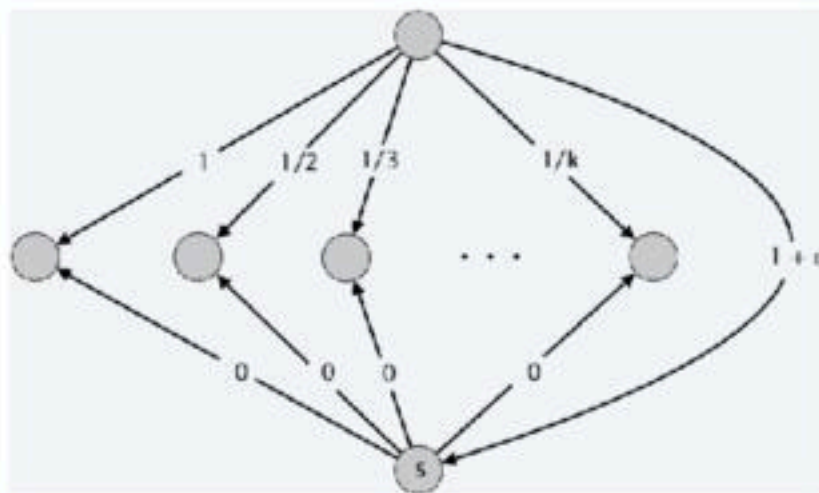
Fundamental question. What is price of stability?

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

Social optimum. Everyone takes bottom paths.

Unique Nash equilibrium. Everyone takes top paths.

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



Finding a Nash equilibrium

Theorem. The following algorithm terminates with a Nash equilibrium.

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

1. FOR $j = 1..k$:
 1. $P_j =$ any path for agent j ;
2. WHILE (not a Nash equilibrium):
 1. $j =$ some agent who can improve by switching paths;
 2. $P_j =$ better path for agent i ;
3. RETURN (P_1, \dots, P_k) ;

Pf. Consider a set of paths P_1, \dots, P_k .

- Let x_e denote the number of paths that use edge e .
- Let $\Phi(P_1, \dots, P_k) = \sum_{e \in E} c_e H(x_e)$ be a potential function, where $H(0) = 0$, $H(k) = \sum_{i=1..k} \frac{1}{i}$

Finding a Nash equilibrium

Pf. [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 increases by $\sum_{f \in P'_j - P_j} c_f [H(x_f + 1) - H(x_f)] = \sum_{f \in P'_j - P_j} \frac{c_f}{x_f + 1}.$

- Φ decreases by $\sum_{e \in P_j - P'_j} c_e [H(x_e) - H(x_e - 1)] = \sum_{e \in P_j - P'_j} \frac{c_e}{x_e}.$

- Thus, net change in Φ is negative.

Bounding the price of stability

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

$$C(P_1, \dots, P_k) \leq \Phi(P_1, \dots, P_k) \leq H(k) \cdot C(P_1, \dots, P_k)$$

Pf. 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,

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

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 $H(k)$.

Pf.

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

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

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 poly-time.