

# Chapter 12

## Local Search



Slides by Kevin Wayne.  
Copyright © 2005 Pearson-Addison Wesley.  
All rights reserved.

## Coping With NP-Hardness

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

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

Must sacrifice one of three desired features.

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

## 12.1 Landscape of an Optimization Problem

---

## 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  $S$ .

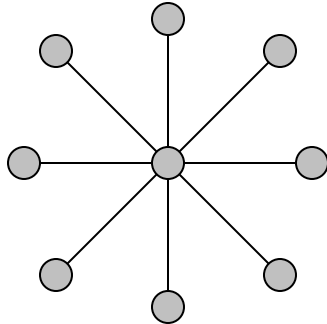
**Neighbor relation.**  $S \sim S'$  if  $S'$  can be obtained from  $S$  by adding or deleting a single node. Each vertex cover  $S$  has at most  $n$  neighbors.

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

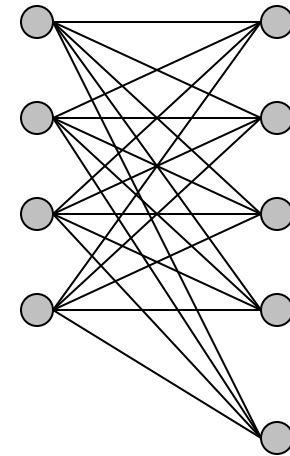
**Remark.** Algorithm terminates after at most  $n$  steps since each update decreases the size of the cover by one.

# Gradient Descent: Vertex Cover

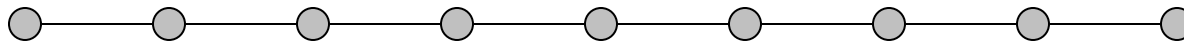
Local optimum. No neighbor is strictly better.



optimum = center node only  
local optimum = all other nodes



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



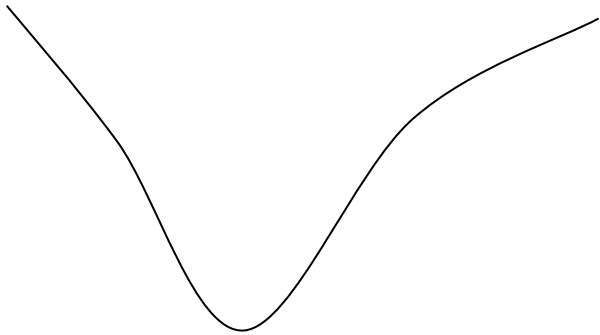
optimum = even nodes  
local optimum = omit every third node

# Local Search

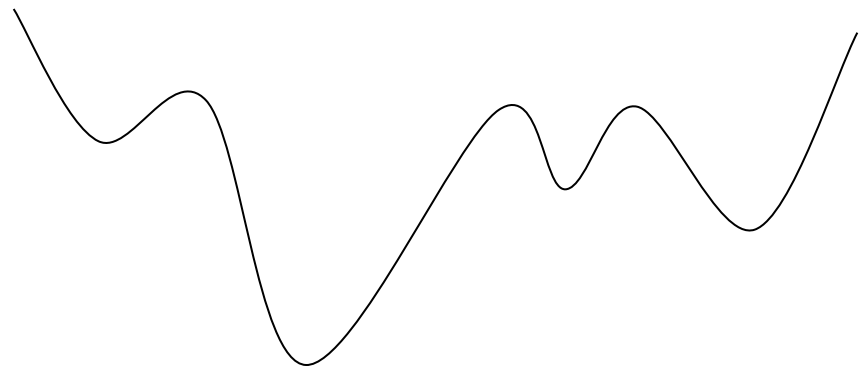
**Local search.** Algorithm that explores the space of possible solutions in sequential fashion, moving from a current solution to a "nearby" one.

**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.



A funnel



A jagged funnel

## 12.2 Metropolis Algorithm

---

# Metropolis Algorithm

**Metropolis algorithm.** [Metropolis, Rosenbluth, Rosenbluth, Teller, Teller 1953]

- 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.** 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

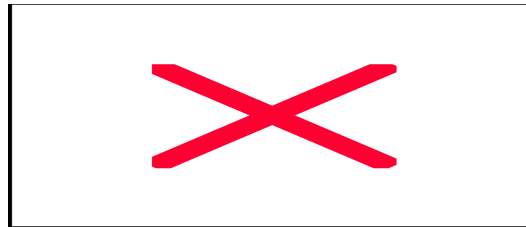


# Metropolis Algorithm

## 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:



**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.

## 12.3 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 edge weights  $w$ .

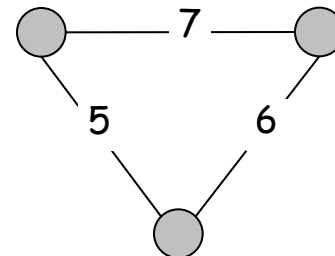
positive or negative

**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.

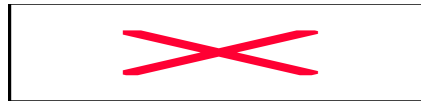
**Note.** In general, no configuration respects all constraints.



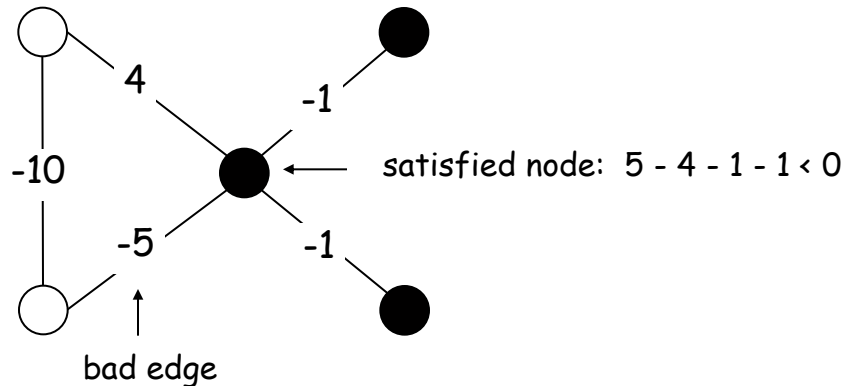
# Hopfield Neural Networks

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

**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.



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



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

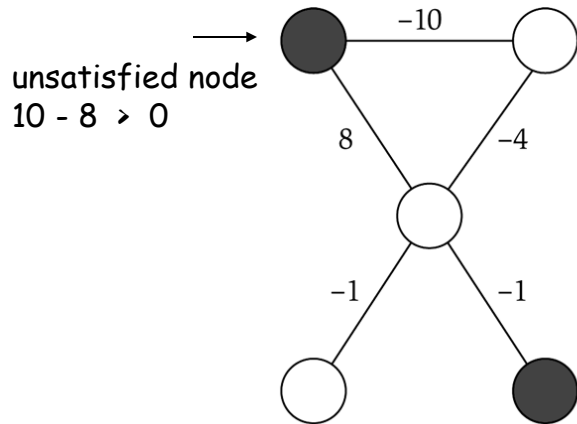
# Hopfield Neural Networks

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

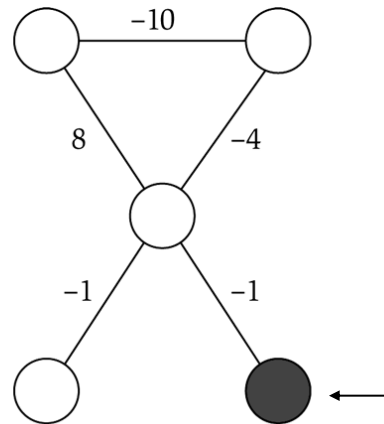
**State-flipping algorithm.** Repeated flip state of an unsatisfied node.

```
Hopfield-Flip( $G, w$ ) {  
     $S \leftarrow$  arbitrary configuration  
  
    while (current configuration is not stable) {  
         $u \leftarrow$  unsatisfied node  
         $s_u = -s_u$   
    }  
  
    return  $S$   
}
```

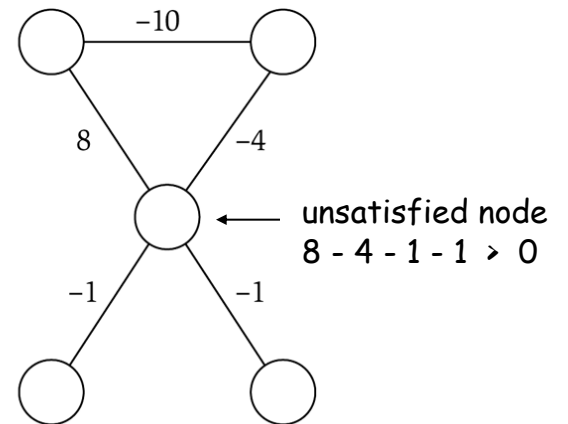
# State Flipping Algorithm



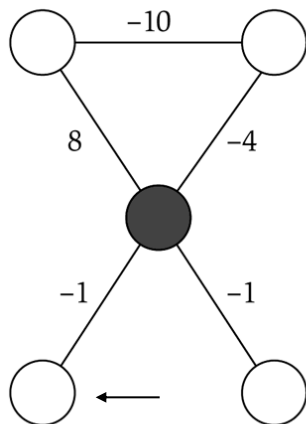
(a)



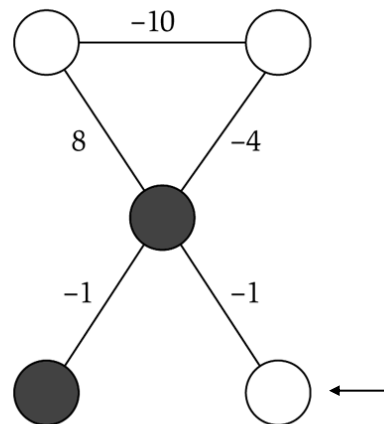
(b)



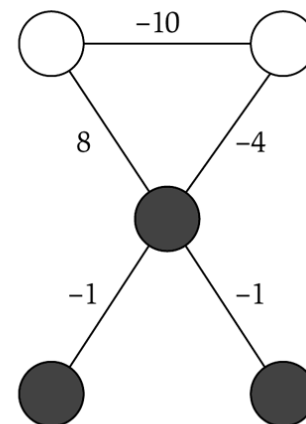
(c)



(d)



(e)



(f)

stable

# Hopfield Neural Networks

**Claim.** 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.



# Hopfield Neural Networks

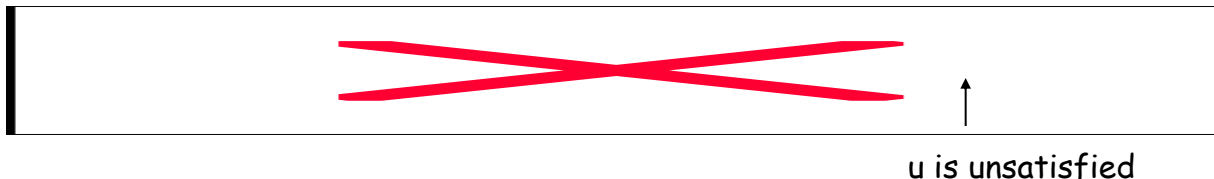
**Claim.** State-flipping algorithm terminates with a stable configuration after at most  $W = \sum_e |w_e|$  iterations.

**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



# 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.

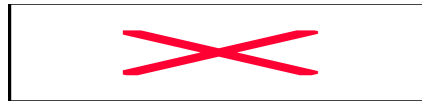
↑  
polynomial in  $n$  and  $\log W$

## 12.4 Maximum Cut

---

# Maximum Cut

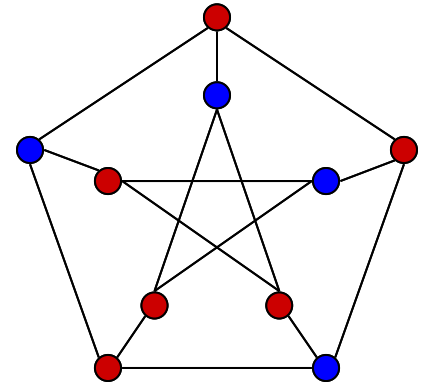
**Maximum cut.** Given an undirected graph  $G = (V, E)$  with positive integer edge weights  $w_e$ , find a node partition  $(A, B)$  such that the total weight of edges crossing the cut is maximized.



## 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 partition  $(A, B)$ , 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) {  
    Pick a random node partition (A, B)  
  
    while ( $\exists$  improving node v) {  
        if (v is in A) move v to B  
        else           move v to A  
    }  
  
    return (A, B)  
}
```

# Maximum Cut: Local Search Analysis

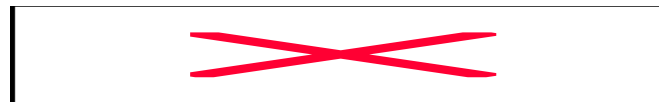
**Theorem.** Let  $(A, B)$  be a locally optimal partition and let  $(A^*, B^*)$  be optimal partition. Then  $w(A, B) \geq \frac{1}{2} \sum_e w_e \geq \frac{1}{2} w(A^*, B^*)$ .

↑  
weights are nonnegative

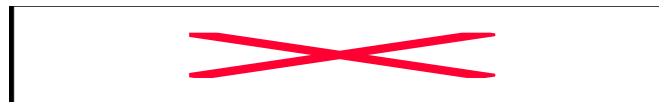
**Pf.**

- Local optimality implies that for all  $u \in A$  : 

Adding up all these inequalities yields:

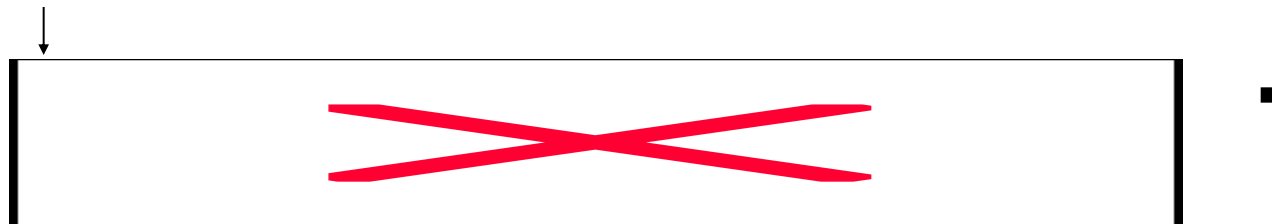


- Similarly




- Now,

each edge counted once



# 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 

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

**Pf idea.** Add  to each inequality in original proof.

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

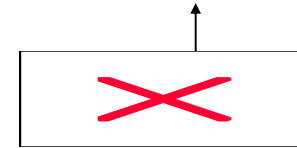
- Each flip improves cut value by at least a factor of  $(1 + \varepsilon/n)$ .
- After  $n/\varepsilon$  iterations the cut value improves by a factor of 2.
- Cut value can be doubled at most  $\log W$  times.

↑  
if  $x \geq 1$ ,  $(1 + 1/x)^x \geq 2$

# Maximum Cut: Context

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

**Theorem.** [Goemans-Williamson 1995] There exists an 0.878567-approximation algorithm for MAX-CUT.



**Theorem.** [Håstad 1997] Unless  $P = NP$ , no  $16/17$  approximation algorithm for MAX-CUT.

↑  
0.941176



## 12.5 Neighbor Relations

---

# Neighbor Relations for Max Cut

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

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

- $\Theta(n^k)$  neighbors.

**KL-neighborhood.** [Kernighan-Lin 1970]

cut value of  $(A_1, B_1)$  may be worse than  $(A, B)$



- 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  $(A, B) = (A_1, B_1), \dots, (A_{n-1}, B_{n-1})$ .
- 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.

## 12.7 Nash Equilibria

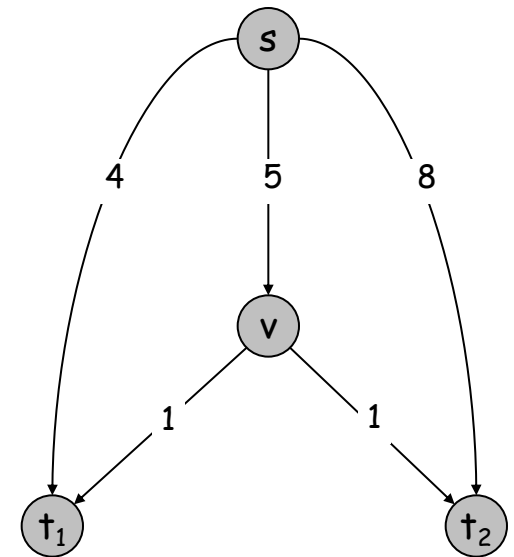
---

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



# Nash Equilibrium

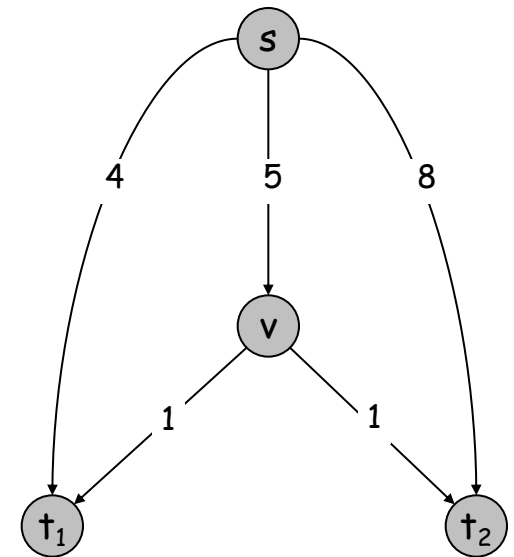
**Best response dynamics.** Each agent is continually prepared to improve its solution in response to changes made by other agents.

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

**Fundamental question.** When do Nash equilibria exist?

**Ex:**

- 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 happens, agent 1 prefers middle path (since  $4 > 5/2 + 1$ ).
- 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.

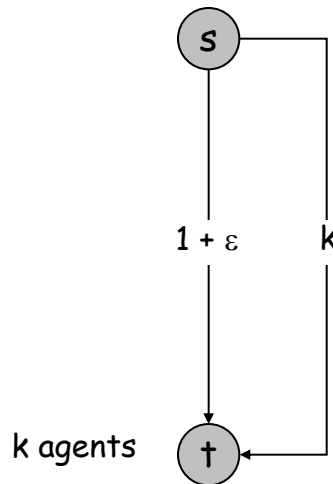
- Nash equilibrium : local search.
- Best response dynamics : local search algorithm.
- Unilateral move by single agent : local neighborhood.

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

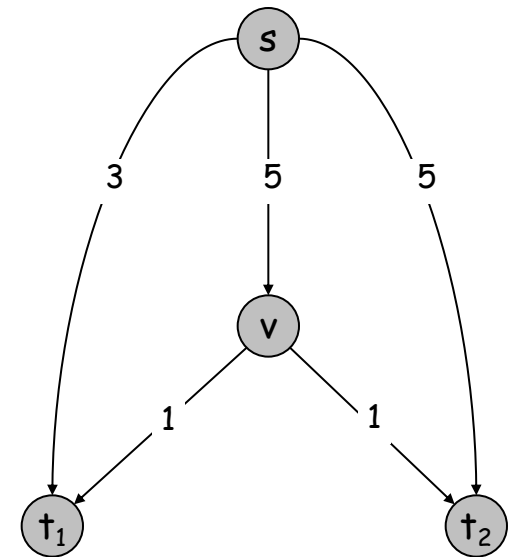
# Socially Optimum

**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 =  $1 + \epsilon$   
Nash equilibrium A =  $1 + \epsilon$   
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.

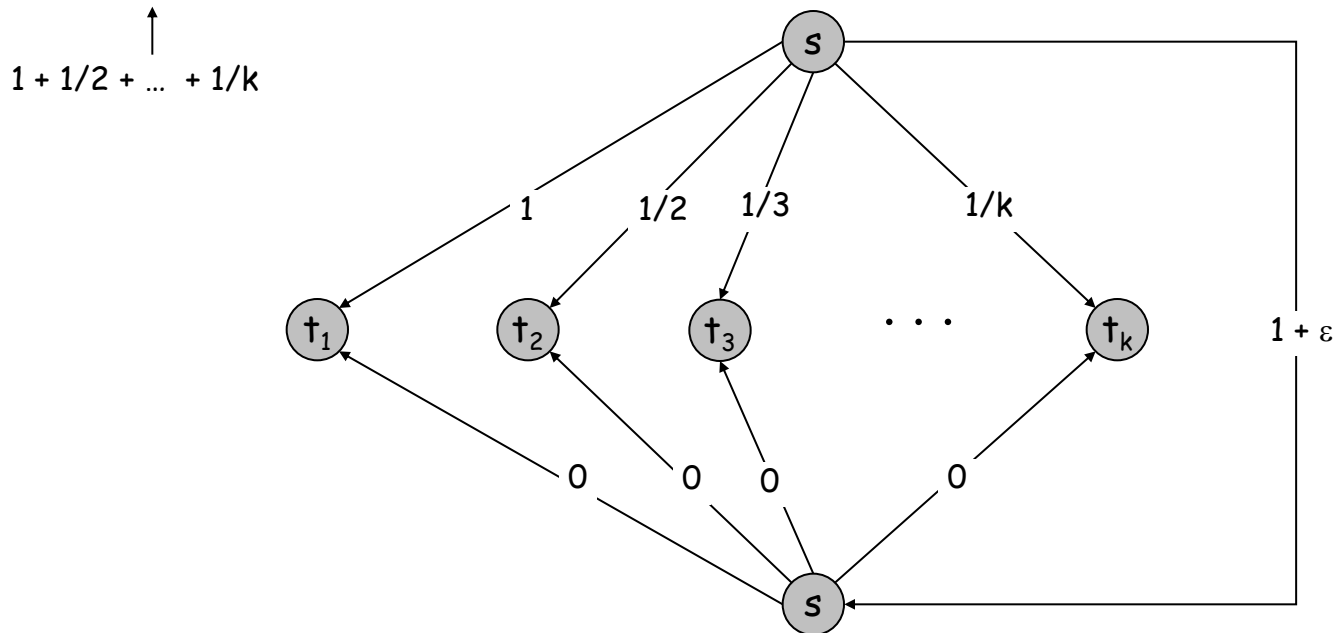
**Fundamental question.** What is price of stability?

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

**Social optimum.** Everyone takes bottom paths.

**Unique Nash equilibrium.** Everyone takes top paths.

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





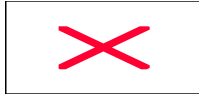
## Finding a Nash Equilibrium

**Theorem.** The following algorithm terminates with a Nash equilibrium.

```
Best-Response-Dynamics(G, c) {  
    Pick a path for each agent  
  
    while (not a Nash equilibrium) {  
        Pick an agent i who can improve by switching paths  
        Switch path of agent i  
    }  
}
```

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

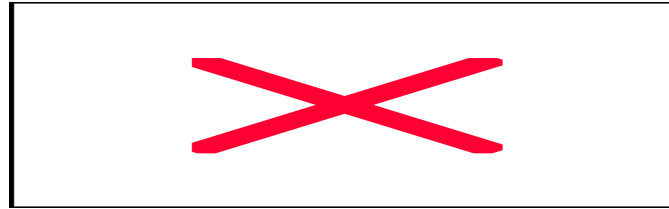
- Let  $x_e$  denote the number of paths that use edge  $e$ .
- Let  $\Phi(P_1, \dots, P_k) = \sum_{e \in E} c_e \cdot H(x_e)$  be a potential function.
- Since there are only finitely many sets of paths, it suffices to show that  $\Phi$  strictly decreases in each step.

$H(0) = 0,$  

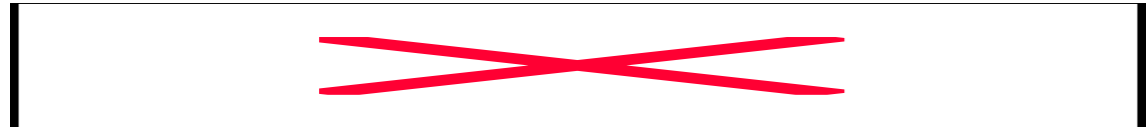
# Finding a Nash Equilibrium

Pf. (continued)

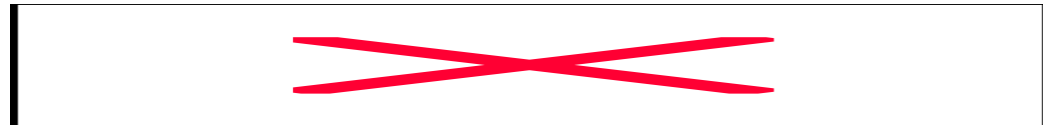
- Consider agent  $j$  switching from path  $P_j$  to path  $P_j'$ .
- Agent  $j$  switches because



- $\Phi$  increases by



- $\Phi$  decreases by



- Thus, net change in  $\Phi$  is negative. ▪

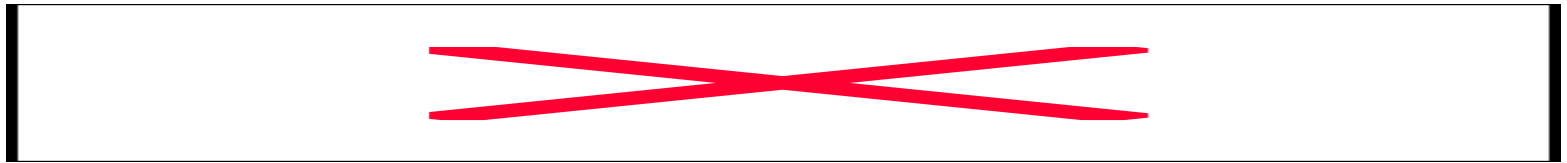
## Bounding the Price of Stability

**Claim.** 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



**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.

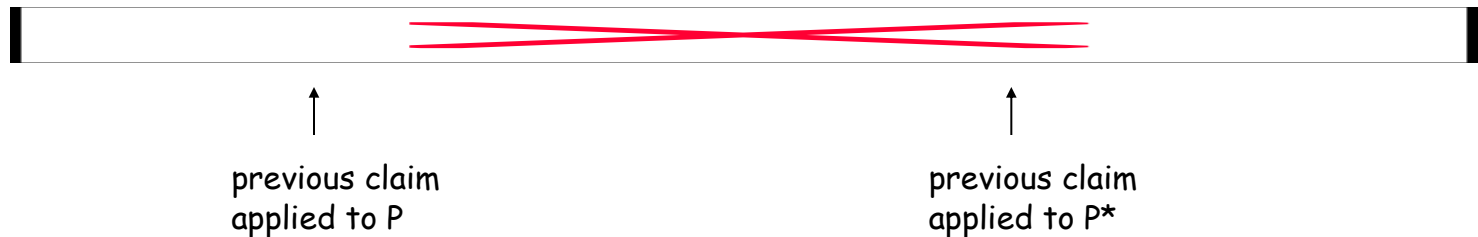


# Bounding the Price of Stability

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



## 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.