Local Search

CS 3AC3

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Acknowledgments: Material based on *Algorithm Design* by Jon Kleinberg and Éva Tardos (Chapter 12)

Coping with NP-completeness

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

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

We must sacrifice one of three desired features.

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

Local Search

- Local search is a very general technique; it describes any algorithm that "explores" the space of possible solutions in a sequential fashion, moving in one step from a current solution to a "nearby" one.
- The generality and flexibility of this notion has the advantage that it is not difficult to design a local search approach to almost any computationally hard problem.
- The counterbalancing disadvantage is that it is often very difficult to say anything precise or provable about the quality of the solutions that a local search algorithm finds, and consequently very hard to tell whether one is using a good local search algorithm or a poor one.

Local Search

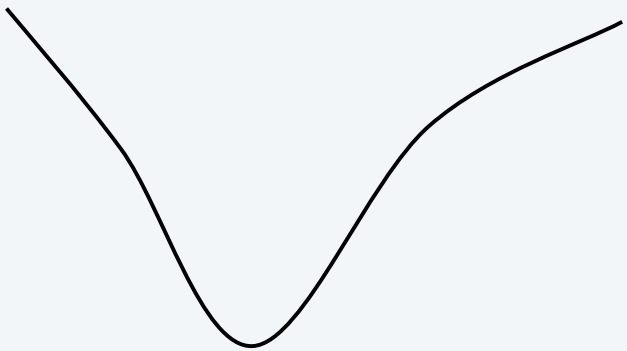
- Our discussion of local search will reflect these trade-offs.
- Local search algorithms are generally heuristics designed to find good, but not necessarily optimal, solutions to computational problems, and we begin by talking about what the search for such solutions looks like at a global level.
- Much of the core of local search was developed by people thinking in terms of analogies with physics.
- Physical systems are performing minimization all the time, when they seek to minimize their potential energy.
- What can we learn from the ways in which nature performs minimization?
- Does it suggest new kinds of algorithms?

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

Gradient descent: vertex cover

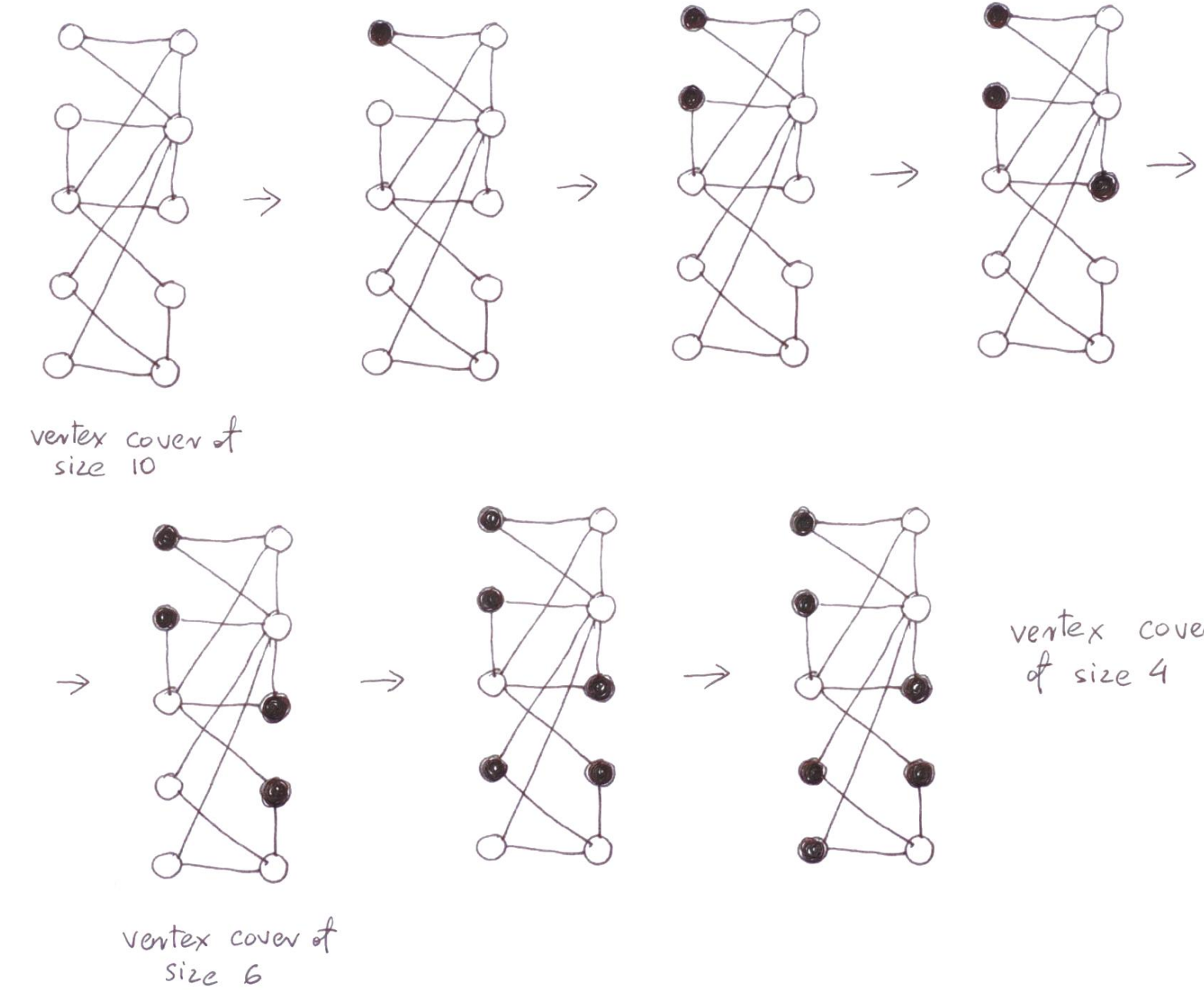
Vertex cover. Given a graph $G = (V, E)$, find a subset of nodes S of minimal cardinality such that for each $(u, v) \in E$, either u or v (or both) are in S.

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'.

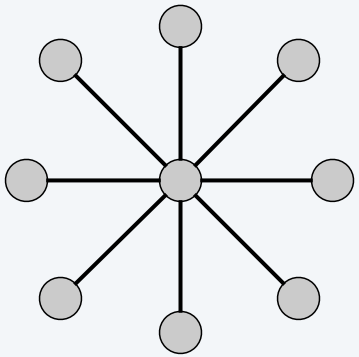
Remark. Algorithm terminates after at most n steps since each update decreases the size of the cover by one.

Gradient descent: vertex cover



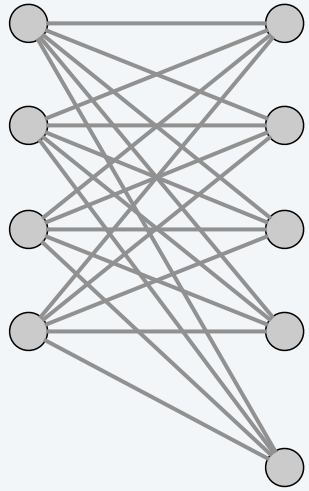
Local Optimum for Graphs

Local optimum. No neighbor is strictly better.



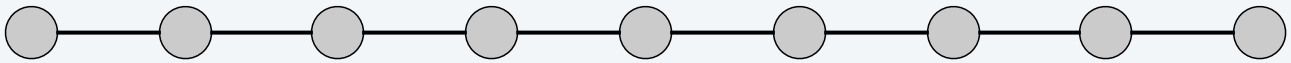
optimum = center node only

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

Hopfield neural networks

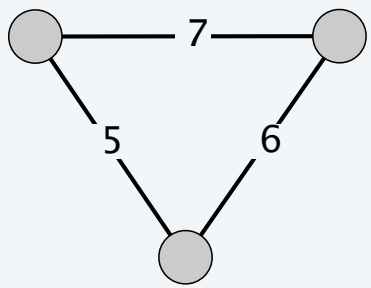
Hopfield networks. Simple model of an associative memory, in which a large collection of units are connected by an underlying network, and neighboring units try to correlate their states.

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.



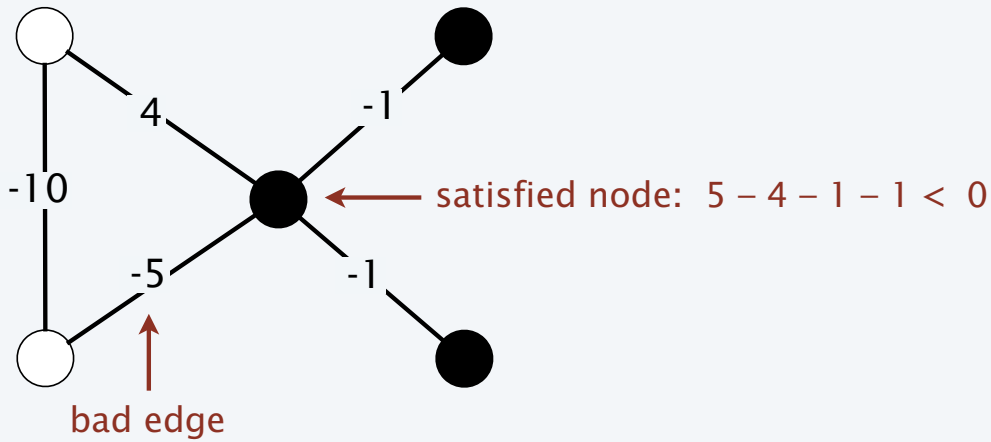
Hopfield neural networks

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 ≥ weight of incident bad edges.

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

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



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.

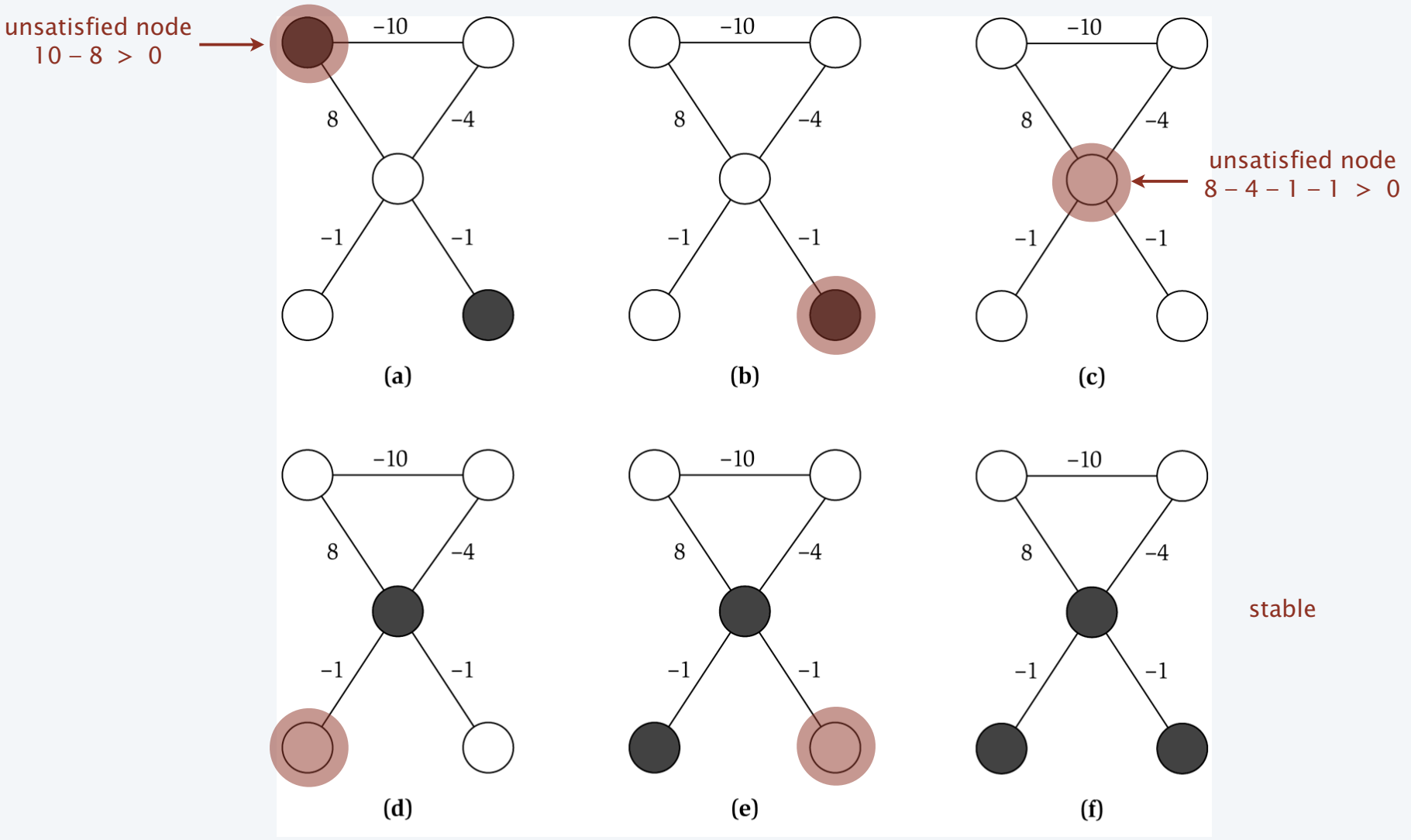
WHILE (current configuration is not stable)

  $u \leftarrow$ unsatisfied node.

  $s_u \leftarrow -s_u$.

RETURN S.

State-flipping algorithm example



Complexity of Hopfield neural network

Theorem

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

- **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 (i.e. polynomial in n and $\log W$) algorithm for the search problem.