Parsimony and Local Search Algorithms

Georgy Gimel'farb

(with basic contributions by Michael J. Dinneen)

COMPSCI 369 Computational Science

1. Maximum Parsimony
2. Small Parsimony Problem (Fitch's Dynamic Programming)
3. Local Search

Learning outcomes:

- Understand that **heuristic optimisation strategies** must be used when no good exact algorithm is known
- Understand dynamic programming (DP)

Recommended reading:

- D. Gusfield: *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. N.Y.: Cambridge Univ. Press, 1997.
- Ch. Semple and M. Steel: *Phylogenetics*. N.Y.: Oxford Univ. Press, 2003

Parsimony

Occam's razor: When the models have the same explanatory power, tend towards a simpler one http://en.wikipedia.org/wiki/Occam's_razor

Parsimony: the use of the least complex model for observations
Maximum parsimony: the smallest number of evolutionary changes in the preferred phylogenetic tree to explain observed data

- Statistics: the preferred math models involve the smallest number of parameters (i.e. the lowest uncertainty)

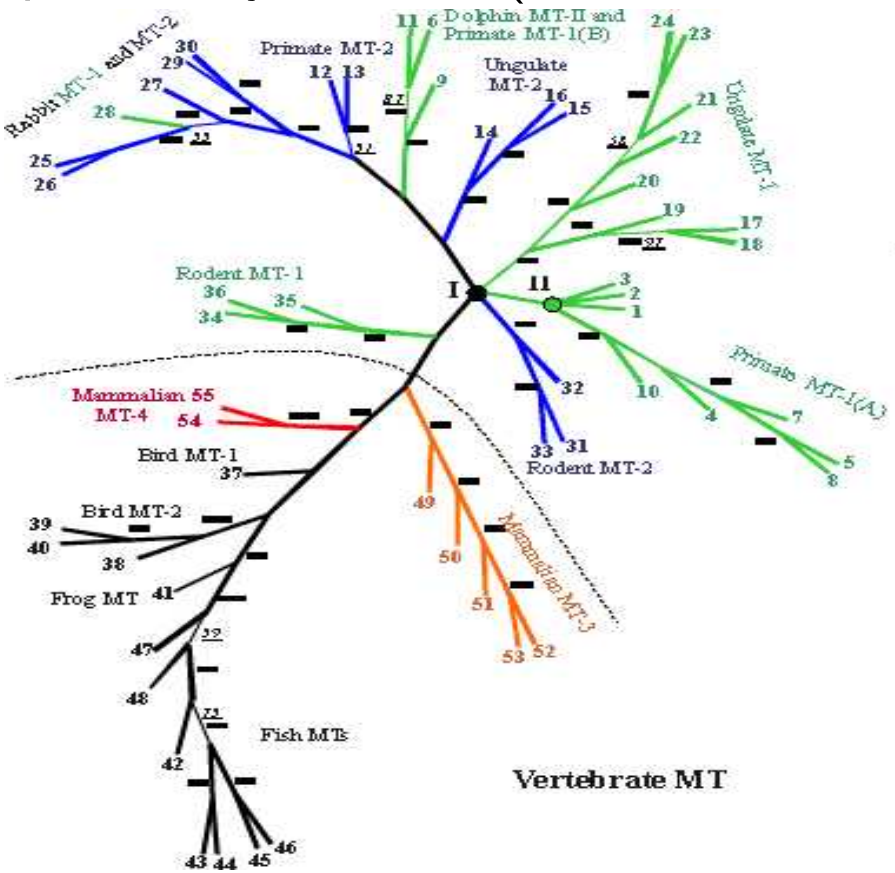
Parsimony is a heuristic guide in developing theoretical models

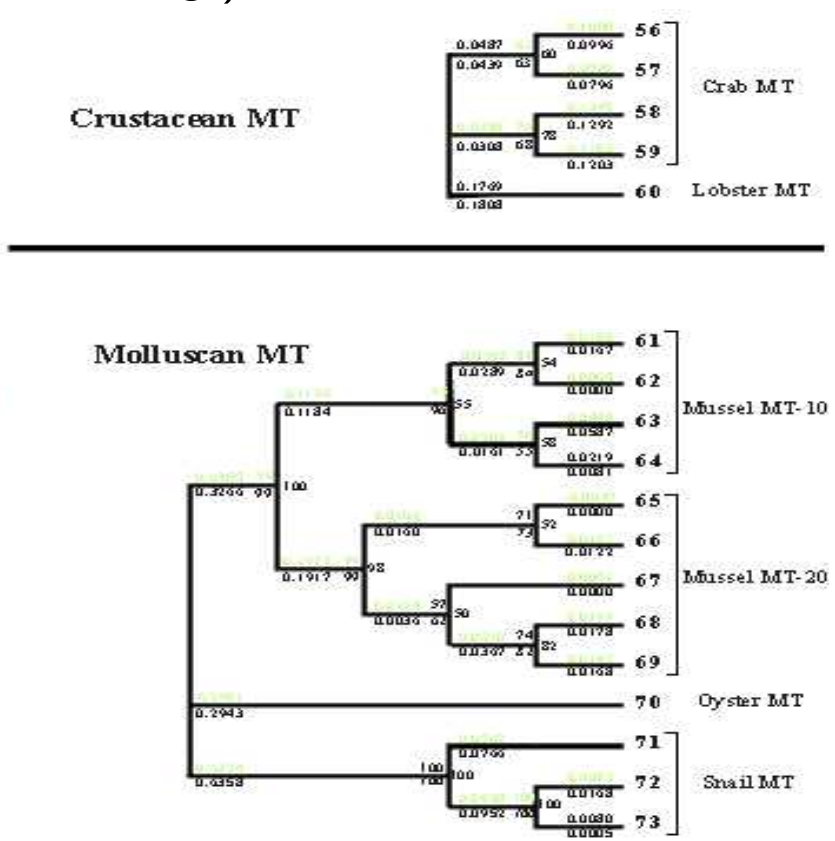
- It does not guarantee a correct conclusion
- It may even strongly support a false conclusion

http://en.wikipedia.org/wiki/Parsimony

Maximum Parsimony & Phylogenetic Trees: An Example

Metallothioneins phylogenetic trees obtained with the maximum parsimony method (P.-A. Binz, J.H.R. Kägi):





http://www.bioc.uzh.ch/mtpage/poster/posterevol.html

Maximum Parsimony Problem

Minimum Steiner Tree

Given a collection C of n strings of the same length k over a given alphabet A, that is, $C \subseteq A^k$,

Find for a tree with C as leaf nodes, a set of $n-2$ internal nodes from A^k, each of degree 3, such that the tree found has the minimal *parsimony score*

The *Hamming distance* $H(x, y)$ between two strings x and y in A^k, is the number of positions in which x and y differ

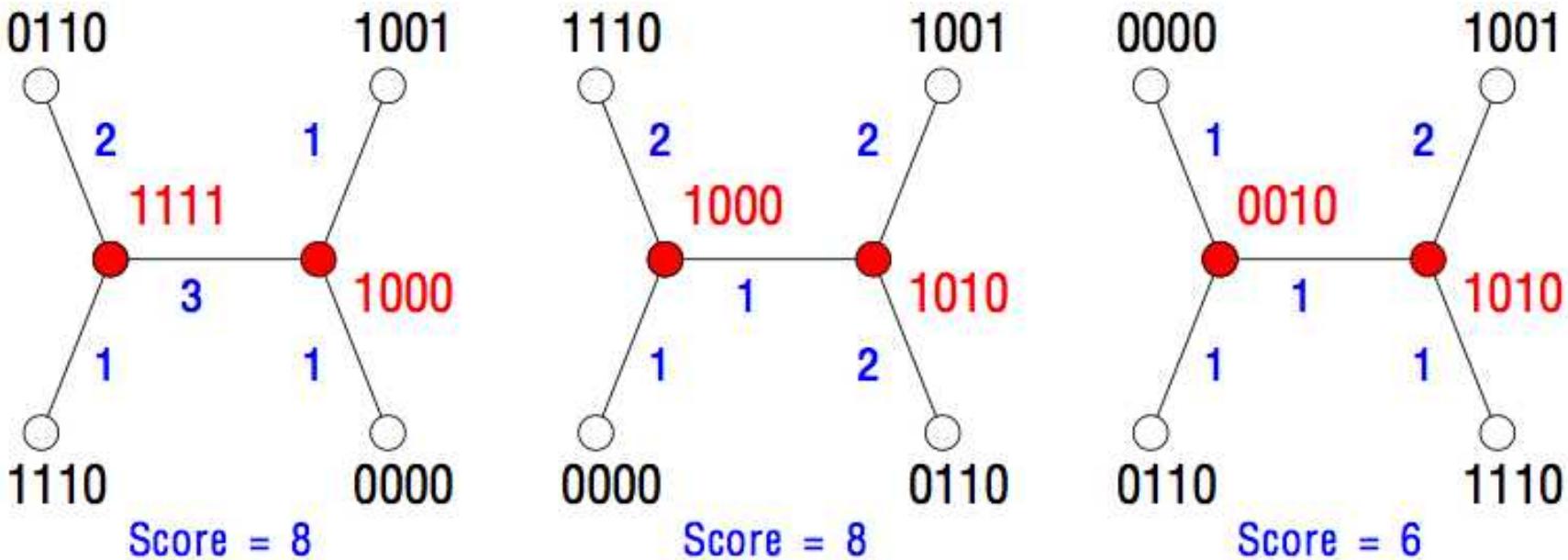
- $H(\text{swing}, \text{twist}) = 3$; $H(\text{class}, \text{class}) = 0$; $H(\text{class}, \text{truck}) = 5$

The parsimony score of a tree $T = (V, E)$ with nodes V labelled by strings in A^k is $\sum_{(u,v)\in E} H(u, v)$

- $A = \{\text{a}, \text{b}\}$; $V = \{\text{aaa}, \text{aab}, \text{abb}\}$; $E = \{(\text{aaa}, \text{aab}); (\text{aaa}, \text{abb})\} \Rightarrow$ Parsimony score: $1 + 2 = 3$

Maximum Parsimony Example

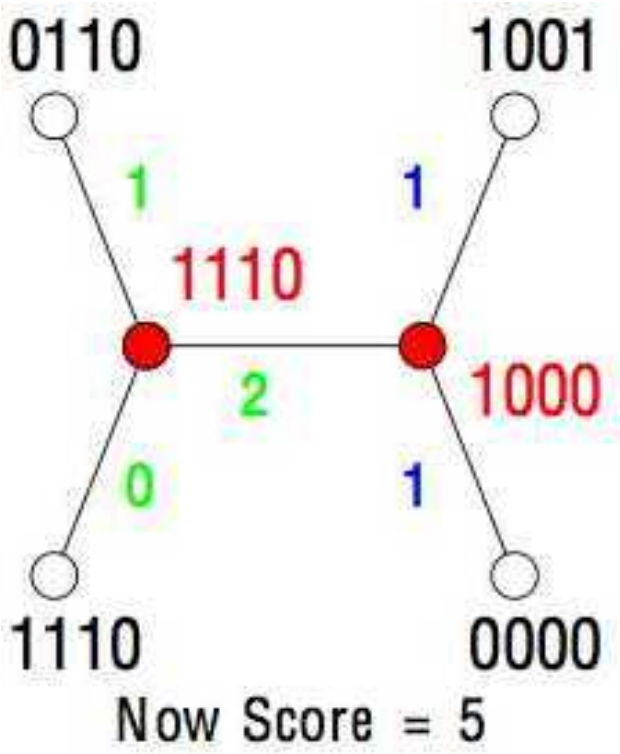
Suppose $A = \{0, 1\}$; $n = 4$, $k = 4$, $C = \{0000, 0110, 1001, 1110\}$



Some possible evolutionary trees and their parsimony scores
(red nodes: inserted internal nodes – ancestors)

Maximum Parsimony Example

Suppose $A = \{0, 1\}$; $n = 4$, $k = 4$, $C = \{0000, 0110, 1001, 1110\}$



⇐ A possible better evolutionary tree (red nodes: inserted internal nodes – ancestors)

- If internal nodes with the same labels as leaves are allowed, the score may decrease
- However, in either physical model the search for the best underlying tree structure is intractable

Computing Best Score for a Fixed Tree

Small parsimony problem: Find the parsimony score for a given tree $T = \{V, E\}$

- Dynamic programming algorithm by *Walter M. Fitch* (1971)

W. Fitch: Toward defining the course of evolution: .... *Systematic Zoology*, vol.20, pp.406–416,1971

Let a collection C of n strings of length k over some alphabet A of size $s = |A|$ be assigned as leaves of some tree T

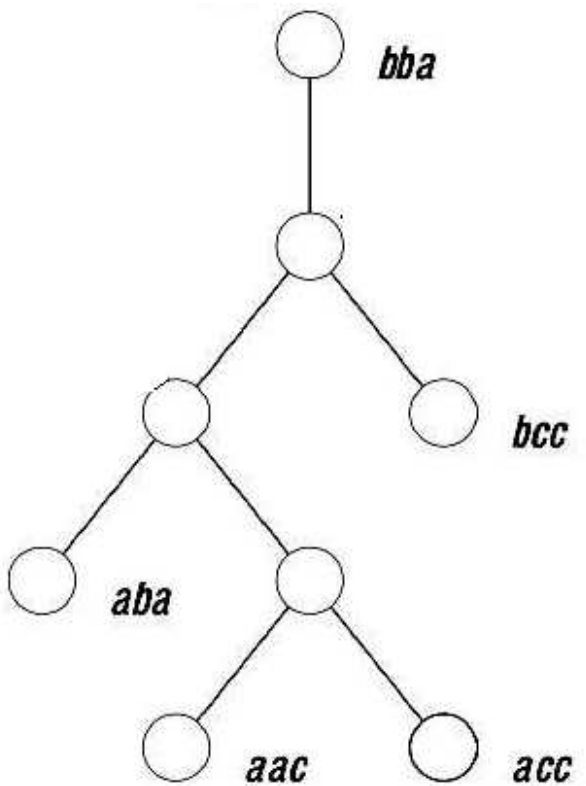
1. Select (arbitrarily) the root of T either at a leaf, or by subdividing an edge
2. **Forward pass**: from bottom(leaf)-up compute a set of (best) candidate strings for each internal (parent) node
3. **Backward pass**: from top(root)-down pick a specific label for each internal node while computing an optimal parsimony score

This dynamic programming algorithm runs in time $O(nsk)$

http://www-ab.informatik.uni-tuebingen.de/teaching/ws06/script/phylogeny_31Jan2007.pdf

Fitch's Algorithm Illustrated

Consider a collection $C = \{aba, bba, bcc, acc, aac\}$ over the alphabet $\{a, b, c\}$ and the following tree T

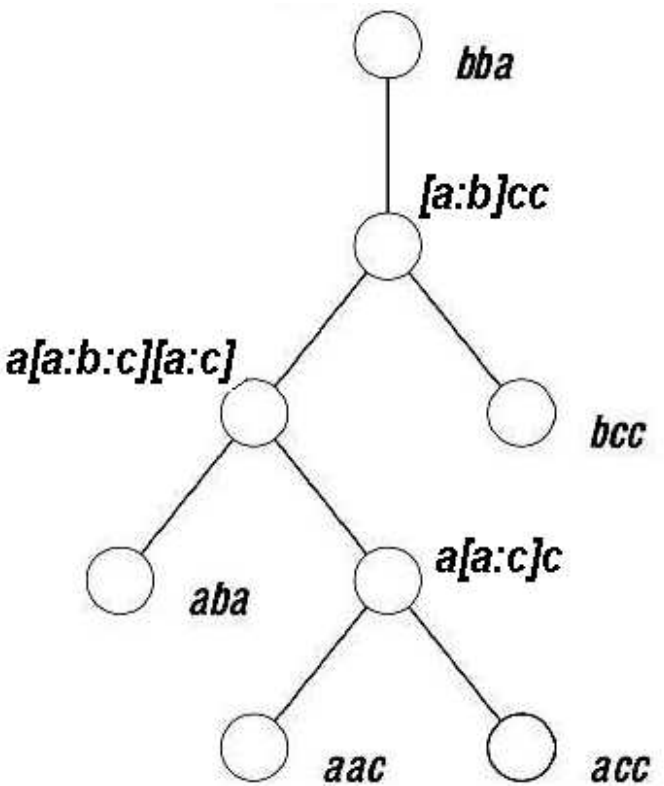


- Select the leaf bba as the root of T
- The forward pass: compute the candidate (*potentially optimal*) states at each of the internal nodes
- The backward pass: assign strings to all internal nodes and calculate the optimal tree score:

$$2 + 1 + 0 + 2 + 0 + 1 + 0 = 6$$

Fitch's Algorithm Illustrated

Consider a collection $C = \{aba, bba, bcc, acc, aac\}$ over the alphabet $\{a, b, c\}$ and the following tree T

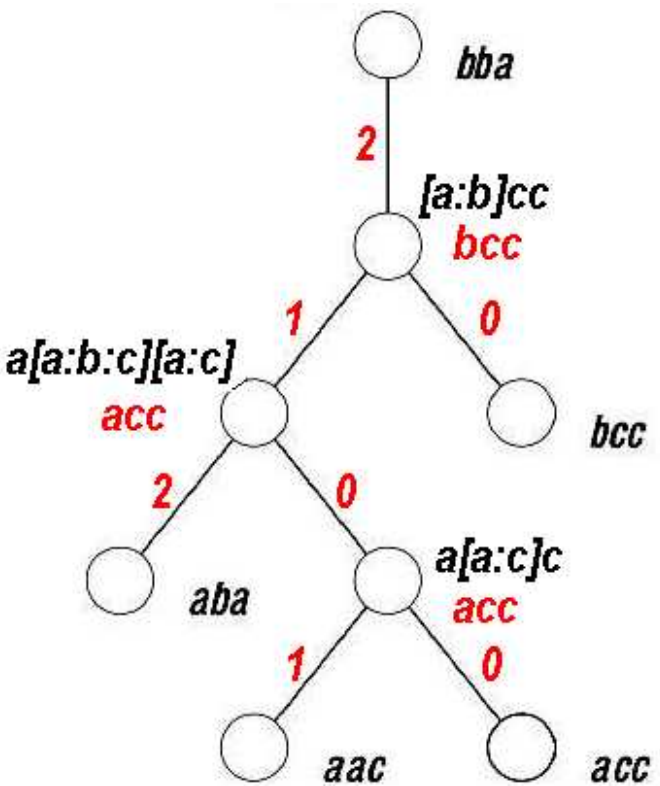


- Select the leaf bba as the root of T
- The forward pass: compute the candidate (*potentially optimal*) states at each of the internal nodes
- The backward pass: assign strings to all internal nodes and calculate the optimal tree score:

$$2 + 1 + 0 + 2 + 0 + 1 + 0 = 6$$

Fitch's Algorithm Illustrated

Consider a collection $C = \{aba, bba, bcc, acc, aac\}$ over the alphabet $\{a, b, c\}$ and the following tree T



- Select the leaf bba as the root of T
- The forward pass: compute the candidate (*potentially optimal*) states at each of the internal nodes
- The backward pass: assign strings to all internal nodes and calculate the optimal tree score:

$$2 + 1 + 0 + 2 + 0 + 1 + 0 = 6$$

Coping with difficult problems

Question: Suppose we need to solve an NP-hard problem like Maximum Parsimony. What should we do?

Answer: Theory says (unless P=NP) you must sacrifice one of three desired features:

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

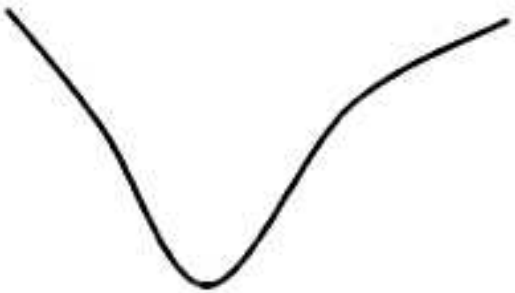
Practical implications:

1. Exponential complexity of the solution, or
2. Only particular instances of the problem are solved, or
3. Only a sub-optimal solution can be found

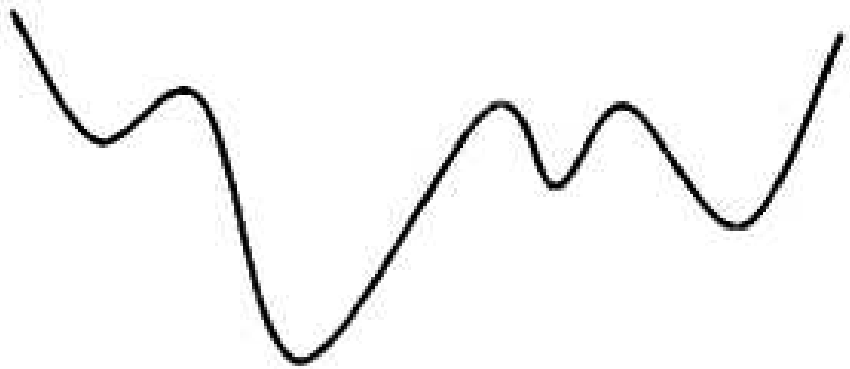
Local Search: Main Idea

Exploring the space of possible solutions in sequential mode, moving from a current solution to a "nearby" one.

- Neighbour relation for the problem: $S \sim S'$
- Gradient descent: Let S denote current solution
  1. If there is a neighbor S' of S with strictly lower cost, replace S with the neighbor whose cost is as small as possible
  2. Otherwise, terminate the algorithm



A funnel search space



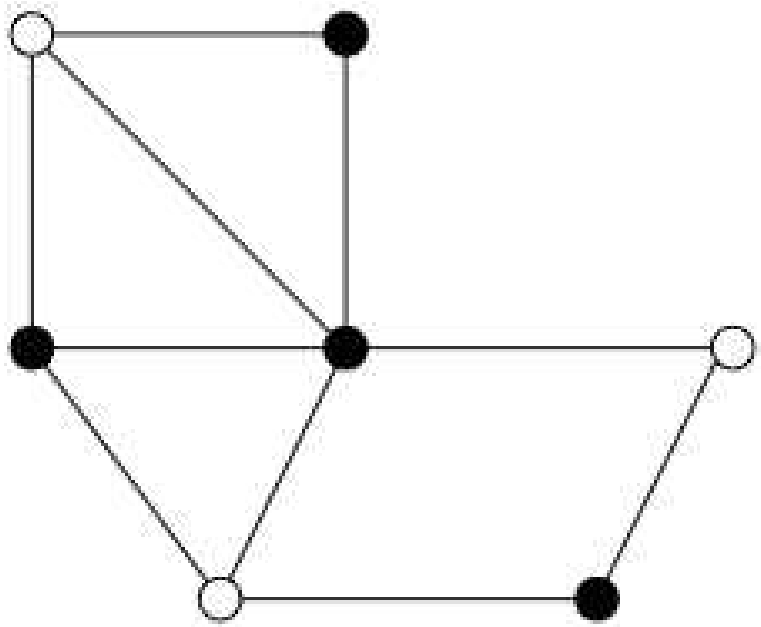
A more realistic situation

Example 1: Vertex Cover Problem

Input: Graph $G = (V, E)$ and a positive integer $k \leq |V|$

Vertex cover: A subset $V' \subseteq V$ with $|V'| \leq k$ such that V' contains at least one vertex from every edge in E

Optimization problem: Find the smallest vertex cover



Vertex Cover Local Search

For a graph $G = (V, E)$, let $S \subseteq V$ be a potential solution.

Then:

Neighbour 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 $n = |S|$ neighbours

Gradient descent:

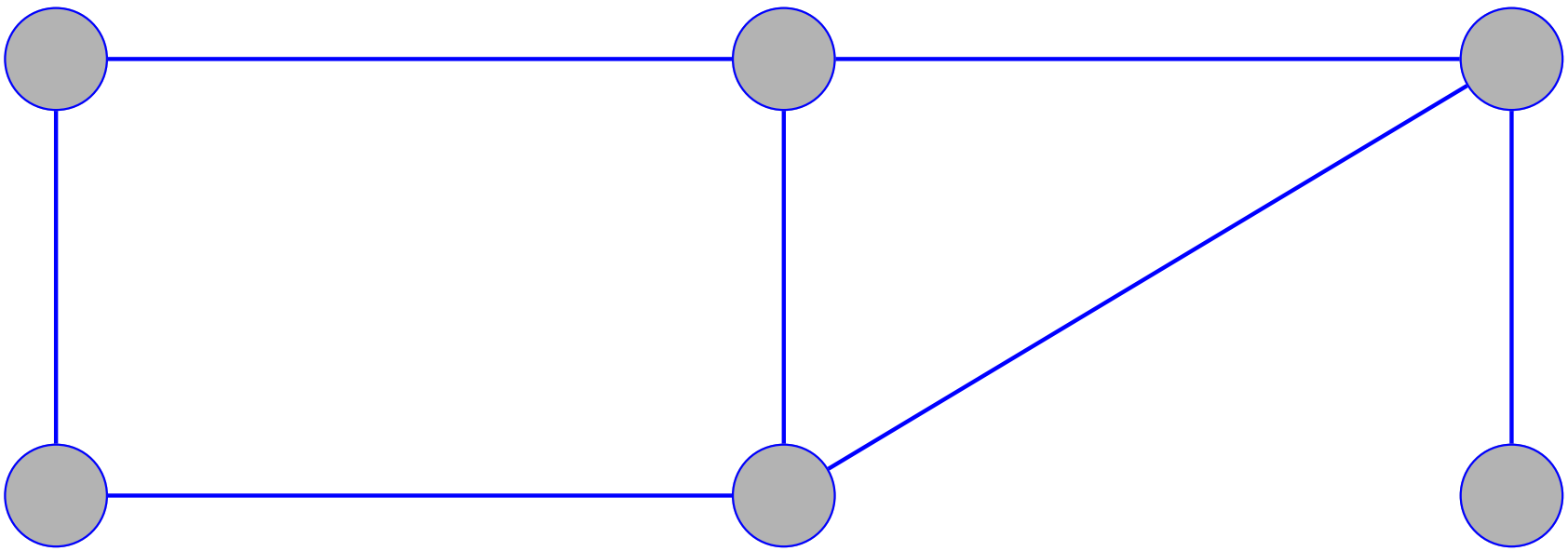
1. Start with $S = V$
2. If there is a neighbor S' that is a vertex cover and has lower cardinality, replace S with S'

Remark: This greedy algorithm terminates after at most $n = |V|$ steps since each update decreases the size of the cover by one.

Vertex Cover Heuristic Illustrated

Vertex status in the graph $G = (V, E)$:

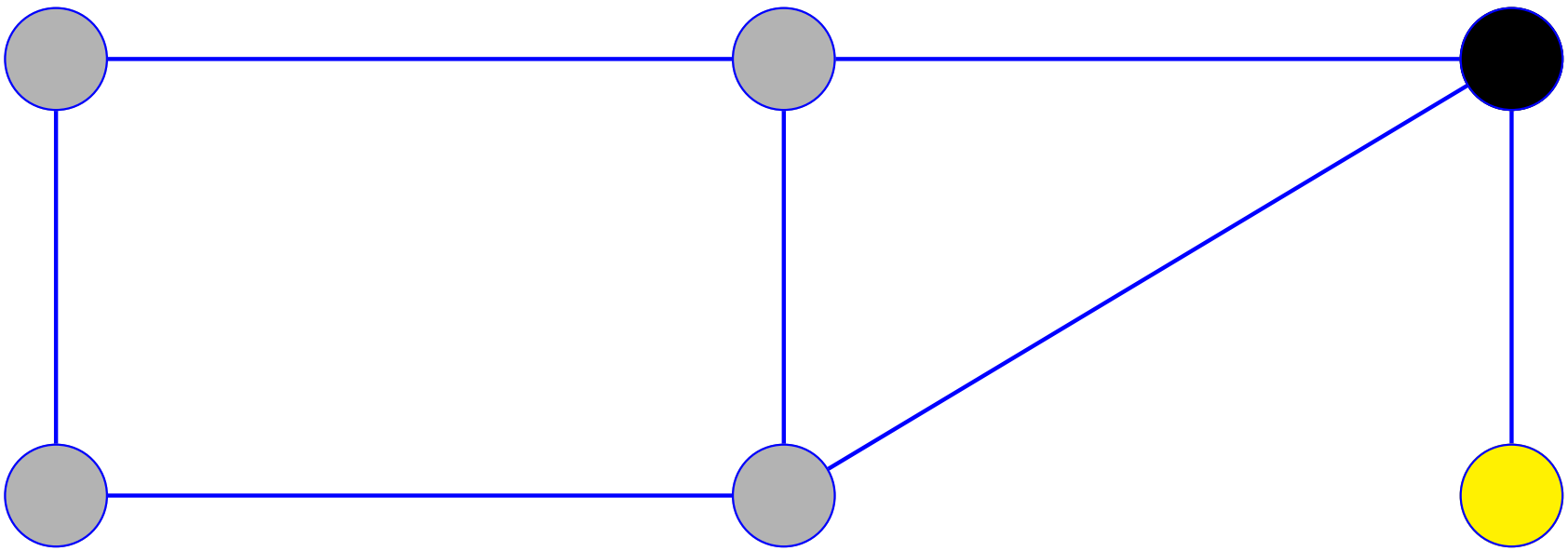
●	Cannot be removed
●	Can be removed
●	Not in the current cover $V' \subseteq V$



Vertex Cover Heuristic Illustrated

Vertex status in the graph $G = (V, E)$:

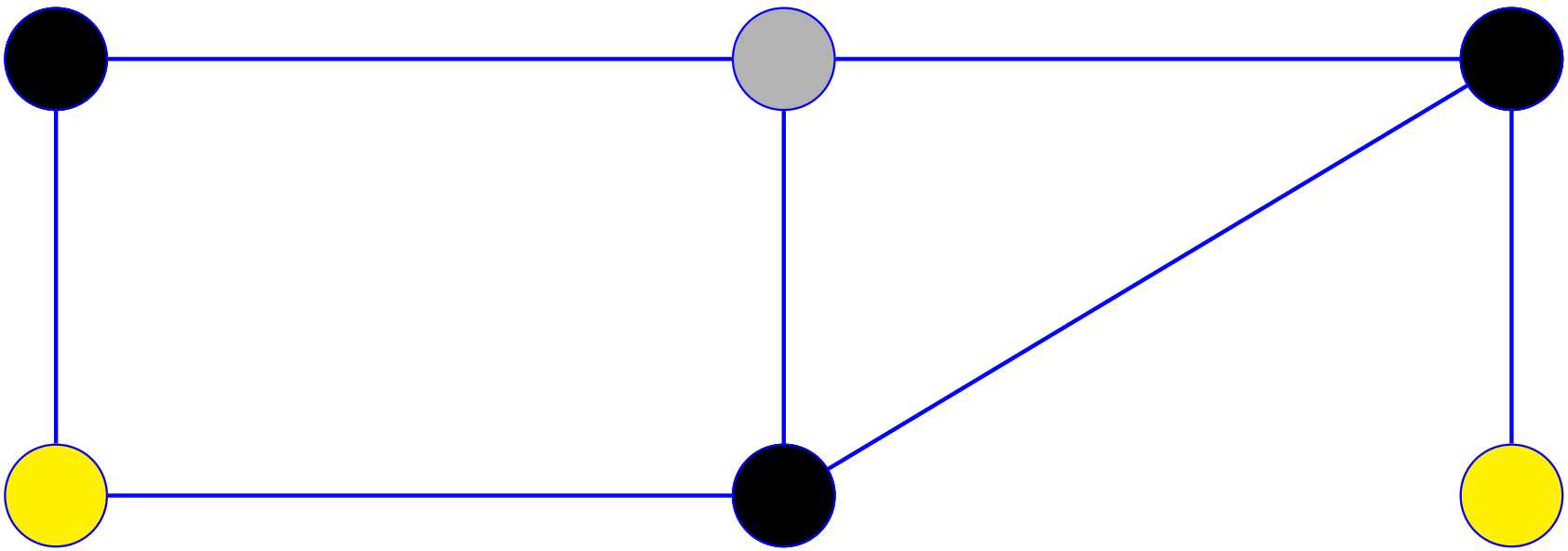
●	Cannot be removed
●	Can be removed
●	Not in the current cover $V' \subseteq V$



Vertex Cover Heuristic Illustrated

Vertex status in the graph $G = (V, E)$:

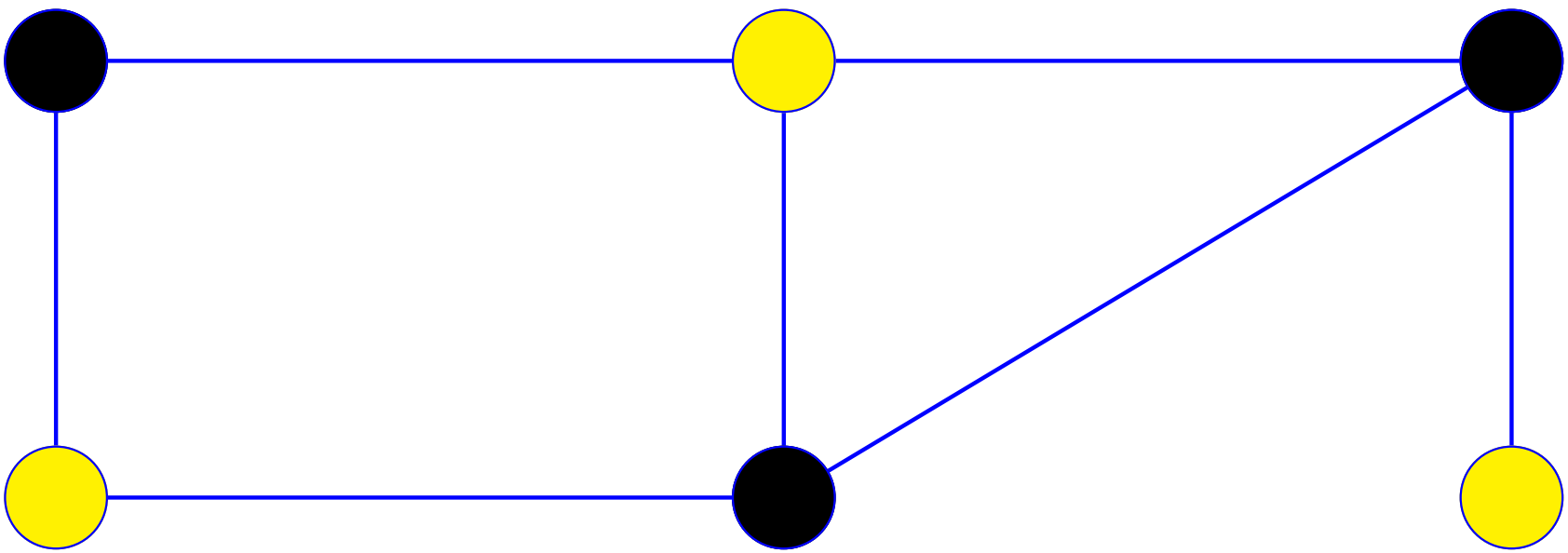
●	Cannot be removed
●	Can be removed
●	Not in the current cover $V' \subseteq V$



Vertex Cover Heuristic Illustrated

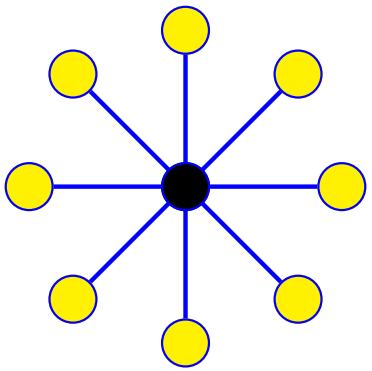
Vertex status in the graph $G = (V, E)$:

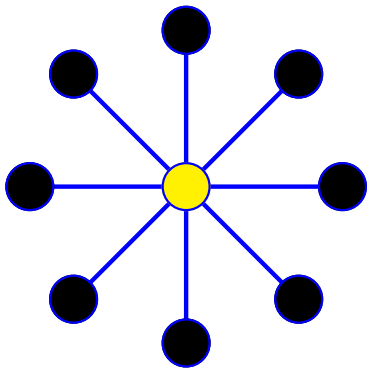
●	Cannot be removed
●	Can be removed
●	Not in the current cover $V' \subseteq V$



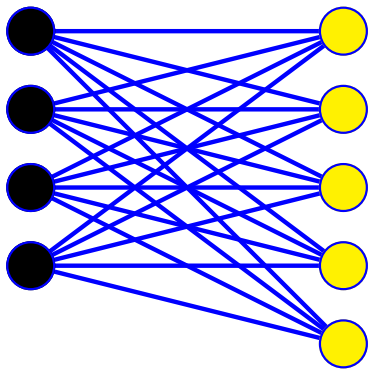
Vertex Cover: Bad Local Optimum

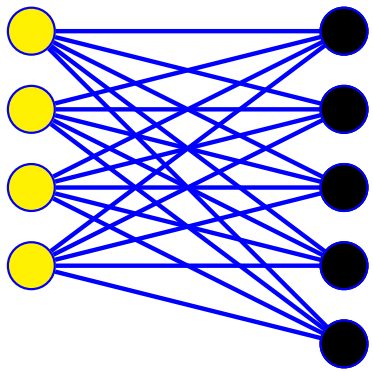
Pure "gradient descent" may get stuck with non-global optimal solutions



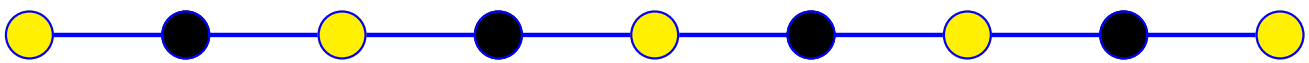


- Optimum: Centre node only
- Local optimum: All other nodes





- Optimum: All nodes on left side
- Local optimum: All nodes on right side



- Optimum: Even nodes
- Local optimum: Omit every third node

Example 2: Maximum Cut Problem

Input: Graph $G = (V, E)$ with positive edge weights $w : E \to \mathbb{R}^+$

Cut: A partition of the vertices V into two disjoint subsets A and B, such that $A \cup B = V$ and $A \cap B = \emptyset$

Cut-set of the cut: The set of edges whose end points are in different subsets of the partition

Edges are said to be *crossing the cut* if they are in its cut-set

Weight of a cut: $\sum_{(u,v) \in E : u \in A, v \in B} w(u, v)$

Optimisation problem

Find a vertex partition with the maximum total weight of edges crossing the cut

Application to n activities (vertices) and m people (edges):
Each person wants to participate in two of the activities
Schedule two activities to maximise number of people that can enjoy both

http://en.wikipedia.org/wiki/Cut_(graph_theory)

Maximum Cut Local Search

Neighbour relation: Given a partition $A \cup B = V$, move one vertex from A to B, or one from B to A if it improves the solution.

Gradient descent algorithm:

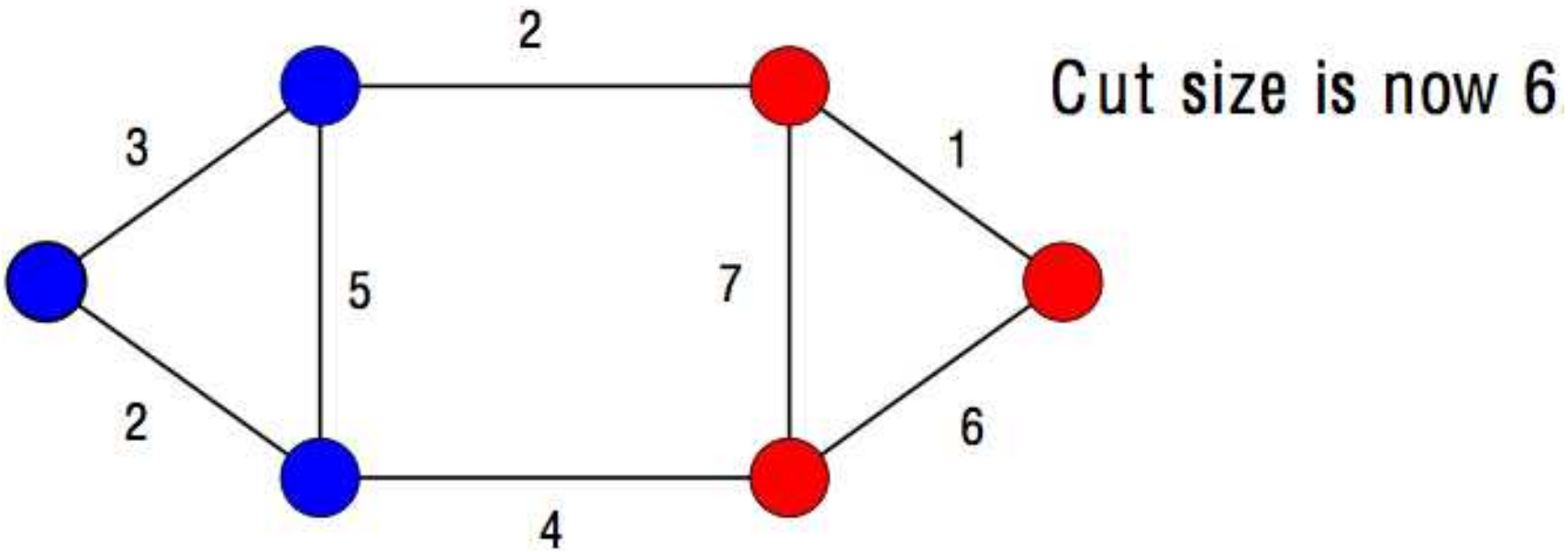
```
Max-Cut-Local (G, w) {
    Pick a random node partition (A, B)

    while (∃ improving node v) {
        if (v is in A) move v to B
        else           move v to A
    }

    return (A, B)
}
```

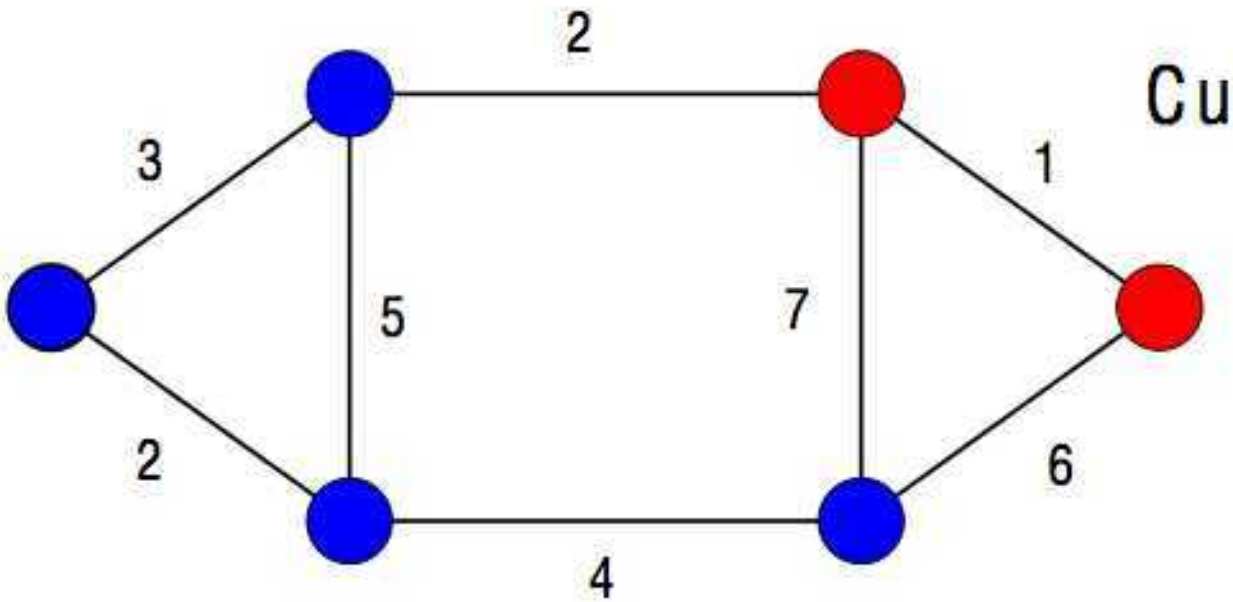
Maximum Cut Cover Heuristic Illustrated

The red or blue vertex colour in the following graph $G = (V, E)$ denotes which part of the cut the vertex is in:



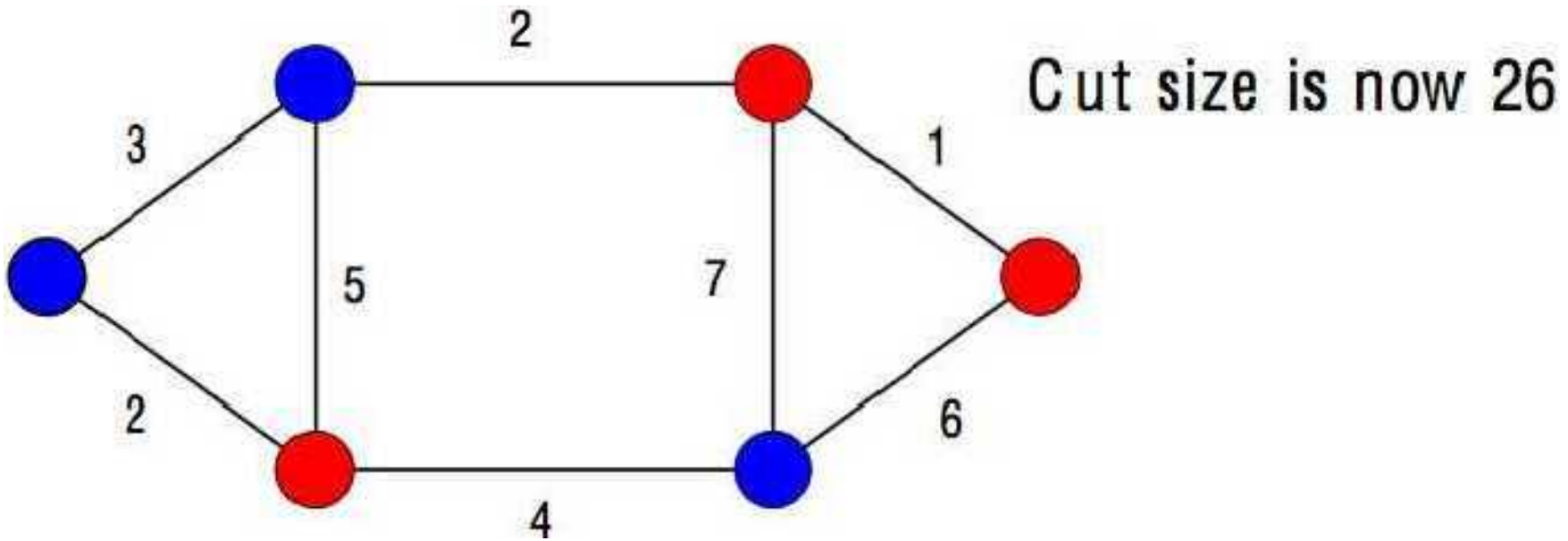
Maximum Cut Cover Heuristic Illustrated

The red or blue vertex colour in the following graph $G = (V, E)$ denotes which part of the cut the vertex is in:



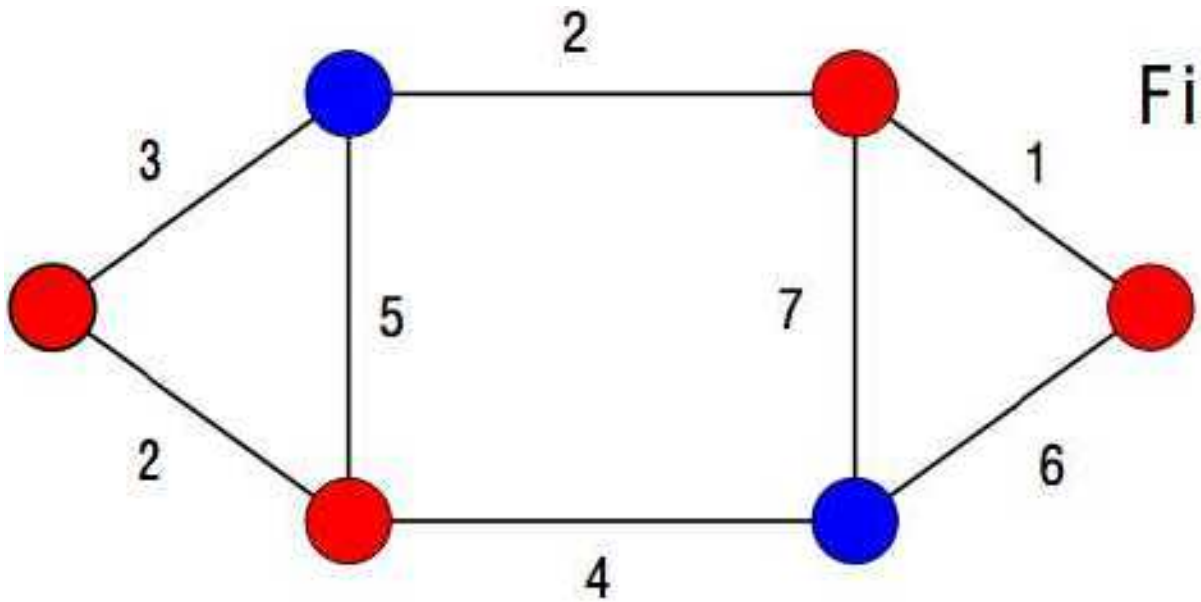
Maximum Cut Cover Heuristic Illustrated

The red or blue vertex colour in the following graph $G = (V, E)$ denotes which part of the cut the vertex is in:



Maximum Cut Cover Heuristic Illustrated

The red or blue vertex colour in the following graph $G = (V, E)$ denotes which part of the cut the vertex is in:



How to Improve Search

Hill-climb with hill-jumps

Standard technique to avoid getting stuck in local optimal solutions that are not global optimal (i.e. in the case the search space is not funnel shaped)

Climb different hills for maximization problems or descend different funnels for minimization problems:

- Need to avoid jumping back to a neighbor state that has already been explored
- Do **hill-jumping** with low-priority, that is, most of the time climb (descend) the same hill (funnel) within the search space
- If possible, do a random/big jump in the search space to increase the likelihood of finding a global optimal solution

Hill-Climbing Approach for Parsimony

The general structure for heuristic searches for approximating maximum parsimony (minimum Steiner tree):

- Find an initial tree (or set of trees) that is obtained by some greedy algorithm
- For each tree in the initial set, initiate a search in which this tree is modified (using a transformation that modifies trees), and the new tree is then scored
- Repeat until a local optimum is found, that is, a tree is found which has no neighbour with a better score, or stop if a time bound is reached
- For all of the local optima, find the set of trees that have the best parsimony score