Vertex Cover

1. Introduction

A **Vertex Cover** (VC) of a connected undirected (un)weighted graph **G** is a **subset of vertices V** of **G** such that **for every edge** in **G**, **at least one of its endpoints is in V**. A **Minimum Vertex Cover (MVC)** (**Minimum Weight Vertex Cover (MWVC)** for the weighted variant) of **G** is a VC that has the smallest cardinality (if unweighted) or total weight (if weighted) among all possible VCs. A graph can have multiple VC but the cardinality/total weight of its MVC/MWVC is unique.

There is another problem called **Maximum Independent Set** (MIS) that attempts to find the **largest** subset of vertices in a (un)weighted graph **G** without any adjacent vertices in the subset. Interestingly, the **complement of an MVC of a graph is an MIS**.

At the end of every visualization, when an algorithm highlights an MVC solution to a graph, it will also highlight its MIS (which is its complement) with light blue color.

1-1. Two Modes

There are two available modes: Unweighted (default) and Weighted. You can switch between the two modes by clicking the respective tab.

There are algorithms that work in both modes and there are algorithms that only work in a certain mode.

2. Visualization

View the visualisation of the selected MVC algorithms here.

Originally, all vertices and edges in the input graph are colored with the standard black outline. As the visualization goes on, the color light blue will be used to denote covered edges and the color orange on edge will be used to show traversed edges.

At the end of the selected MVC algorithm, if it finds a **minimum** VC, it will highlight the MVC vertices with orange color and the non MVC vertices (a.k.a. the MIS vertices) with lightblue color. Otherwise, if the found vertex cover is not proven to be the minimal one (e.g. the algorithm used is an approximation algorithm), it will highlight the vertices that belong to the found vertex cover with orange color without highlighting the MIS vertices.

3. Input

There are two different sources for specifying an input graph:

1. **Draw Graph**: You can draw **any** connected (un)directed weighted graph as the input graph.
2. **Example Graphs**: You can select from the list of example connected undirected weighted graphs to get you started.

4. Bruteforce

Bruteforce: It tries all possible 2^**V** subsets of vertices. In every iteration, it checks whether the currently selected subset of vertices is a valid vertex cover by iterating over all **E** edges and checking whether there is any edge that is not covered by the vertices in the currently selected subset. This bruteforce algorithm keeps the smallest size of the valid vertex cover as the answer.

This bruteforce algorithm is available in both weighted and unweighted version.

Its time complexity is O(2^**V** × **E**), i.e., very slow.

Discussion: But there is an alternative O(2^**k** × **E**) parameterized solution if we are told that **k** is 'not-that-large'.

4-1. Parameterized Solution

[This is a hidden slide]

5. DP on Tree

DP on Tree: If the graph is a **tree**, the MVC problem can be formulated as a Dynamic Programming problem where the states are (position, take_current_vertex).

Then, it can be seen that:
DP(u, take) = cost[u] + sum(min(DP(v, take), DP(v, not_take))) ∀child v of u, and
DP(u, not take) = sum(DP(v, take)) ∀child v of u

This DP algorithm is available in both weighted and unweighted version.

Its time complexity is O(**V**), i.e., very fast, if the input graph is a tree.

6. Greedy MVC on Tree

Greedy MVC on Tree: Again, if the graph is an **unweighted tree**, it can be solved greedily by observing that if there is any MVC solution that takes a leaf vertex, we can obtain a "not worse" solution by taking the parent of that leaf vertex instead. After removing all covered vertices, we can apply the same observation and repeat it until every vertex is covered.

This greedy MVC algorithm is only available in unweighted mode.

Its time complexity is O(**V**), i.e., very fast, if the input graph is an unweighted tree.

7. Kőnig's Theorem

Kőnig's Theorem: From Kőnig's Theorem, the size of MVC in an **unweighted bipartite** graph is equal to the cardinality of the maximum matching of the bipartite graph. In the case of **weighted bipartite** graph, we can see that this theorem also holds true, with a tweak in how we construct the graph. In this visualization, we use a reduction to max flow problem to get the value of the MVC.

This algorithm is available in both weighted and unweighted version.

Its time complexity is O(**V** × **E**) (for unweighted version; can be smaller with pre-processing) or O(**E**^2 × **V**)/O(**V**^2 × **E**) (for weighted version, depending on the max flow algorithm used).

8. Approximation Algorithms

There are several known approximation algorithms for MVC:

1. For unweighted version, we have either the deterministic 2-approximation or probabilistic 2-approximation (in expectation),
2. For weighted version we have the Bar-Yehuda and Even's 2-approximation algorithm.

Note that these algorithms only yield an "approximated" MVC, meaning that they are not a true **minimum** vertex cover, but a good enough one.

Vertex Cover

1. Introduction

A **Vertex Cover** (VC) of a connected undirected (un)weighted graph **G** is a **subset of vertices V** of **G** such that **for every edge** in **G**, **at least one of its endpoints is in V**. A **Minimum Vertex Cover (MVC)** (**Minimum Weight Vertex Cover (MWVC)** for the weighted variant) of **G** is a VC that has the smallest cardinality (if unweighted) or total weight (if weighted) among all possible VCs. A graph can have multiple VC but the cardinality/total weight of its MVC/MWVC is unique.

There is another problem called **Maximum Independent Set** (MIS) that attempts to find the **largest** subset of vertices in a (un)weighted graph **G** without any adjacent vertices in the subset. Interestingly, the **complement of an MVC of a graph is an MIS**.

At the end of every visualization, when an algorithm highlights an MVC solution to a graph, it will also highlight its MIS (which is its complement) with light blue color.

1-1. Two Modes

There are two available modes: Unweighted (default) and Weighted. You can switch between the two modes by clicking the respective tab.

There are algorithms that work in both modes and there are algorithms that only work in a certain mode.

2. Visualization

View the visualisation of the selected MVC algorithms here.

Originally, all vertices and edges in the input graph are colored with the standard black outline. As the visualization goes on, the color light blue will be used to denote covered edges and the color orange on edge will be used to show traversed edges.

At the end of the selected MVC algorithm, if it finds a **minimum** VC, it will highlight the MVC vertices with orange color and the non MVC vertices (a.k.a. the MIS vertices) with lightblue color. Otherwise, if the found vertex cover is not proven to be the minimal one (e.g. the algorithm used is an approximation algorithm), it will highlight the vertices that belong to the found vertex cover with orange color without highlighting the MIS vertices.

3. Input

There are two different sources for specifying an input graph:

1. **Draw Graph**: You can draw **any** connected (un)directed weighted graph as the input graph.
2. **Example Graphs**: You can select from the list of example connected undirected weighted graphs to get you started.

4. Bruteforce

Bruteforce: It tries all possible 2^**V** subsets of vertices. In every iteration, it checks whether the currently selected subset of vertices is a valid vertex cover by iterating over all **E** edges and checking whether there is any edge that is not covered by the vertices in the currently selected subset. This bruteforce algorithm keeps the smallest size of the valid vertex cover as the answer.

This bruteforce algorithm is available in both weighted and unweighted version.

Its time complexity is O(2^**V** × **E**), i.e., very slow.

Discussion: But there is an alternative O(2^**k** × **E**) parameterized solution if we are told that **k** is 'not-that-large'.

4-1. Parameterized Solution

[This is a hidden slide]

5. DP on Tree

DP on Tree: If the graph is a **tree**, the MVC problem can be formulated as a Dynamic Programming problem where the states are (position, take_current_vertex).

Then, it can be seen that:
DP(u, take) = cost[u] + sum(min(DP(v, take), DP(v, not_take))) ∀child v of u, and
DP(u, not take) = sum(DP(v, take)) ∀child v of u

This DP algorithm is available in both weighted and unweighted version.

Its time complexity is O(**V**), i.e., very fast, if the input graph is a tree.

6. Greedy MVC on Tree

Greedy MVC on Tree: Again, if the graph is an **unweighted tree**, it can be solved greedily by observing that if there is any MVC solution that takes a leaf vertex, we can obtain a "not worse" solution by taking the parent of that leaf vertex instead. After removing all covered vertices, we can apply the same observation and repeat it until every vertex is covered.

This greedy MVC algorithm is only available in unweighted mode.

Its time complexity is O(**V**), i.e., very fast, if the input graph is an unweighted tree.

7. Kőnig's Theorem

Kőnig's Theorem: From Kőnig's Theorem, the size of MVC in an **unweighted bipartite** graph is equal to the cardinality of the maximum matching of the bipartite graph. In the case of **weighted bipartite** graph, we can see that this theorem also holds true, with a tweak in how we construct the graph. In this visualization, we use a reduction to max flow problem to get the value of the MVC.

This algorithm is available in both weighted and unweighted version.

Its time complexity is O(**V** × **E**) (for unweighted version; can be smaller with pre-processing) or O(**E**^2 × **V**)/O(**V**^2 × **E**) (for weighted version, depending on the max flow algorithm used).

8. Approximation Algorithms

There are several known approximation algorithms for MVC:

1. For unweighted version, we have either the deterministic 2-approximation or probabilistic 2-approximation (in expectation),
2. For weighted version we have the Bar-Yehuda and Even's 2-approximation algorithm.

Note that these algorithms only yield an "approximated" MVC, meaning that they are not a true **minimum** vertex cover, but a good enough one.