

Beating the Worst Case

Practical Course

Max, Jean-Pierre

About us



Max Göttlicher



Jean-Pierre von der Heydt

About us



Max Göttlicher



Jean-Pierre von der Heydt



You

About us



Max Göttlicher



Jean-Pierre von der Heydt



You

Links

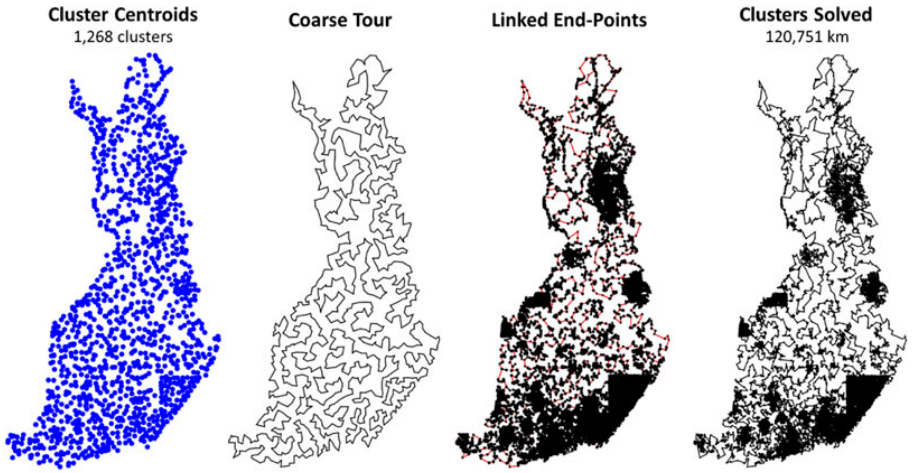
- Website: `scale.iti.kit.edu/teaching/2026ws/beating_wc/start`
- Discord: `discord.gg/uGf52pB6y`

Motivation

- Big-O notation is our primary tool for algorithm analysis
- Such upper bounds are often highly pessimistic

Motivation

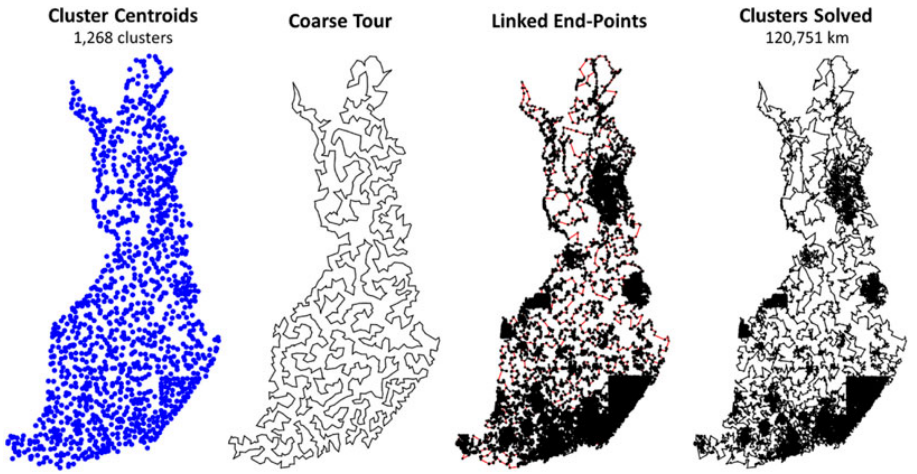
- Big-O notation is our primary tool for algorithm analysis
- Such upper bounds are often highly pessimistic



Source: www.frontiersin.org/articles/10.3389/frobt.2021.689908/full

Motivation

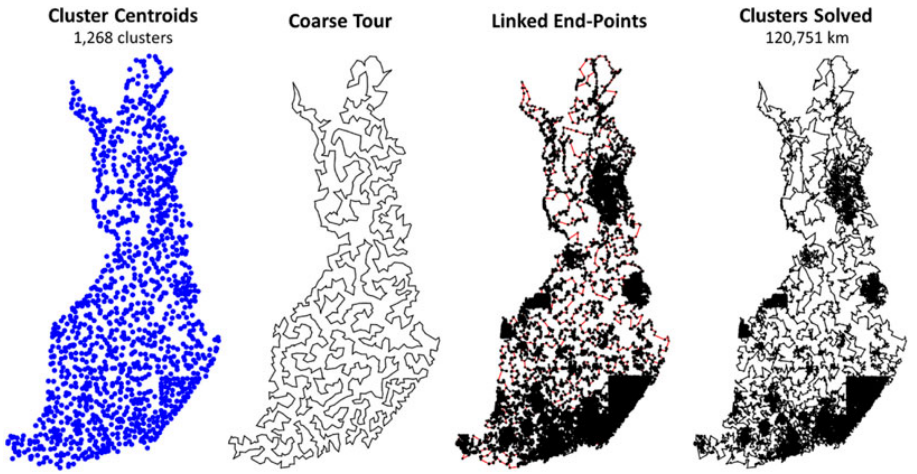
- Big-O notation is our primary tool for algorithm analysis
- Such upper bounds are often highly pessimistic
- TSP-solvers find efficient routes on graphs with millions of vertices



Source: www.frontiersin.org/articles/10.3389/frobt.2021.689908/full

Motivation

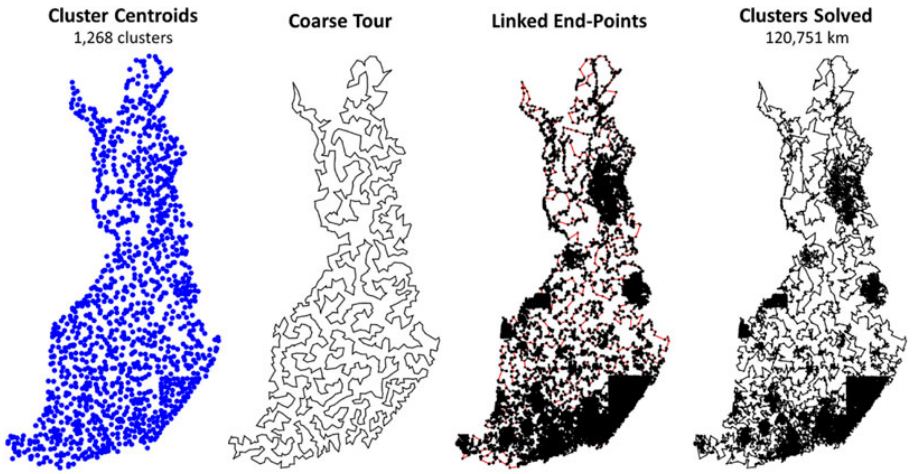
- Big-O notation is our primary tool for algorithm analysis
- Such upper bounds are often highly pessimistic
- TSP-solvers find efficient routes on graphs with millions of vertices
- SAT-solvers solve instances with over millions of variables and clauses



Source: www.frontiersin.org/articles/10.3389/frobt.2021.689908/full

Motivation

- Big-O notation is our primary tool for algorithm analysis
- Such upper bounds are often highly pessimistic
- TSP-solvers find efficient routes on graphs with millions of vertices
- SAT-solvers solve instances with over millions of variables and clauses



Source: www.frontiersin.org/articles/10.3389/frobt.2021.689908/full

Observations

- The actual running times are much better than suggested by worst-case bounds in $2^{O(n)}$
- TSP and SAT are not the only problems where this happens

Motivation

- running times in practice can be much better than in theoretical analysis
- depends on both algorithm and (distribution of) inputs

Motivation

- running times in practice can be much better than in theoretical analysis
- depends on both algorithm and (distribution of) inputs

Our goal

- We consider different algorithms / problems
- Search answers: Why are the algorithms so fast? What kind of inputs do not work?

Motivation

- running times in practice can be much better than in theoretical analysis
- depends on both algorithm and (distribution of) inputs

Our goal

- We consider different algorithms / problems
- Search answers: Why are the algorithms so fast? What kind of inputs do not work?
- How to build a useful experiment setup?
- How to visualize data well?

Plan

Groups

- Work is carried out in teams of three

Plan

Groups

- Work is carried out in teams of three

First phase

- Four exercise sheets (each 2 weeks)
- Setting up an experimental environment, visualizing data, analyzing algorithms

Plan

Groups

- Work is carried out in teams of three

First phase

- Four exercise sheets (each 2 weeks)
- Setting up an experimental environment, visualizing data, analyzing algorithms

"Lecture" in week $i \in \{1, \ldots, 8\}$

- odd i: explain new sheet
  - if $i > 1$: 5min presentations from each group
- even i: discuss progress with sheet, get help

Plan

Groups

- Work is carried out in teams of three

First phase

- Four exercise sheets (each 2 weeks)
- Setting up an experimental environment, visualizing data, analyzing algorithms

"Lecture" in week $i \in \{1, \ldots, 8\}$

- odd i: explain new sheet
  - if $i > 1$: 5min presentations from each group
- even i: discuss progress with sheet, get help

Second phase

- Independent project on a new topic (~7 weeks)
- Planning and implementation of an experimental setup
- weekly meetings with each team
- final presentation + report

Plan

Groups

- Work is carried out in teams of three

First phase

- Four exercise sheets (each 2 weeks)
- Setting up an experimental environment, visualizing data, analyzing algorithms

"Lecture" in week $i \in \{1, \ldots, 8\}$

- odd i: explain new sheet
  - if $i > 1$: 5min presentations from each group
- even i: discuss progress with sheet, get help

Second phase

- Independent project on a new topic (~7 weeks)
- Planning and implementation of an experimental setup
- weekly meetings with each team
- final presentation + report

Evaluation
Grading is based on your work during the second phase. This includes **regular meetings**, the **report** and the **final presentation**.

Usage of AI

We strongly encourage you to solve problems without AI

- You are here to learn and we believe that you do this best, if you work closely with the problems and tools we give you. Relying too much on AI, will lock you out of this learning opportunity.

Usage of AI is permitted

- Both during the exercise phase and the final assignment. You must clearly indicate which parts of your work were generated by AI (and by which model).

You have to be able to justify all of your decisions

- We expect that you can explain all the details of your solution and design decisions. We will put a special emphasis on this during grading.

Framework

- Fork the repository and get familiar with it
- The READMEs should help you understand most things

Beating the Worst Case -- Framework

This small framework serves as a starting point for the practical course *"Beating the Worst Case in Practice: Unerwartet effiziente Algorithmen"*. As participant, you probably want to fork this repository. As the repository contains submodules, you want to clone it with the `--recurse` flag.

```
git clone --recurse git@link_to_your_repository
```

The core of the framework is in framework. The ext_libs folder contains the CLI11 library for parsing command line options and the GIRG library for generating graphs. Unit test are located in tests and use gtest. Moreover, instances includes some small instances for initial testing. The folder evaluation contains example data and an R script to visualize it. You can use this directory to creat your own plots.

Using the Framework

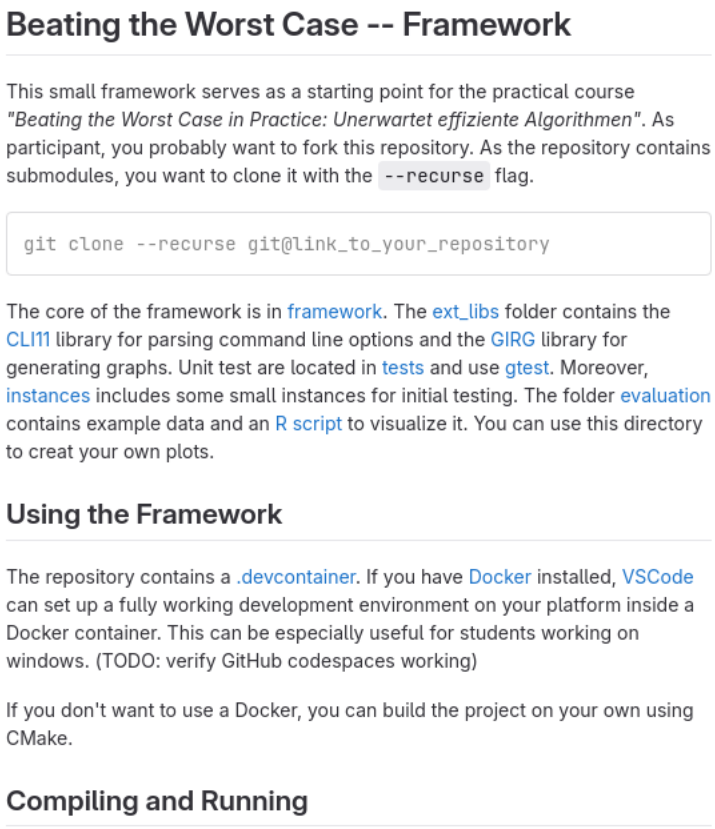
The repository contains a .devcontainer. If you have Docker installed, VSCode can set up a fully working development environment on your platform inside a Docker container. This can be especially useful for students working on windows. (TODO: verify GitHub codespaces working)

If you don't want to use a Docker, you can build the project on your own using CMake.

Compiling and Running

Framework

- Fork the repository and get familiar with it
- The READMEs should help you understand most things
- Graph I/O and some useful algorithms are in ‘`framework`’
- Example command-line applications can be found in ‘`cli`’
- For data analysis, we recommend using *R*, see ‘`evaluation`’ for examples



Beating the Worst Case -- Framework

This small framework serves as a starting point for the practical course *"Beating the Worst Case in Practice: Unerwartet effiziente Algorithmen"*. As participant, you probably want to fork this repository. As the repository contains submodules, you want to clone it with the `--recurse` flag.

```
git clone --recurse git@link_to_your_repository
```

The core of the framework is in framework. The ext_libs folder contains the CLI11 library for parsing command line options and the GIRG library for generating graphs. Unit test are located in tests and use gtest. Moreover, instances includes some small instances for initial testing. The folder evaluation contains example data and an R script to visualize it. You can use this directory to creat your own plots.

Using the Framework

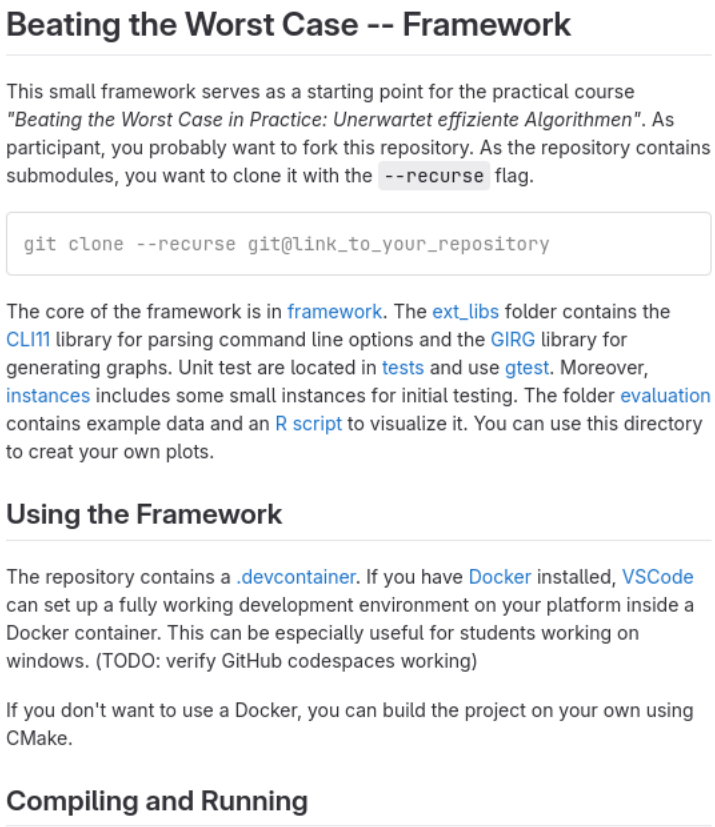
The repository contains a .devcontainer. If you have Docker installed, VSCode can set up a fully working development environment on your platform inside a Docker container. This can be especially useful for students working on windows. (TODO: verify GitHub codespaces working)

If you don't want to use a Docker, you can build the project on your own using CMake.

Compiling and Running

Framework

- Fork the repository and get familiar with it
- The READMEs should help you understand most things
- Graph I/O and some useful algorithms are in ‘`framework`’
- Example command-line applications can be found in ‘`cli`’
- For data analysis, we recommend using *R*, see ‘`evaluation`’ for examples
- First step: try building the repository with CMake and running the program ‘`run`’.



Beating the Worst Case -- Framework

This small framework serves as a starting point for the practical course *"Beating the Worst Case in Practice: Unerwartet effiziente Algorithmen"*. As participant, you probably want to fork this repository. As the repository contains submodules, you want to clone it with the `--recurse` flag.

```
git clone --recurse git@link_to_your_repository
```

The core of the framework is in framework. The ext_libs folder contains the CLI11 library for parsing command line options and the GIRG library for generating graphs. Unit test are located in tests and use gtest. Moreover, instances includes some small instances for initial testing. The folder evaluation contains example data and an R script to visualize it. You can use this directory to creat your own plots.

Using the Framework

The repository contains a .devcontainer. If you have Docker installed, VSCode can set up a fully working development environment on your platform inside a Docker container. This can be especially useful for students working on windows. (TODO: verify GitHub codespaces working)

If you don't want to use a Docker, you can build the project on your own using CMake.

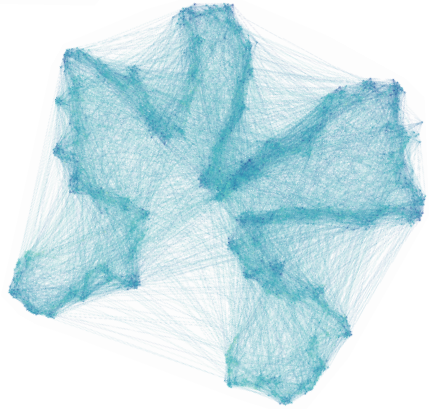
Compiling and Running

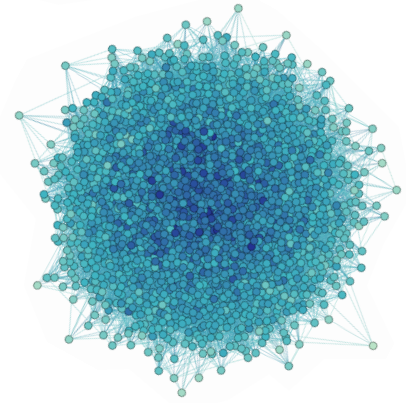
Exercise Sheet 0

- You need to implement **two algorithms**
- We compiled a data-set of 672 graphs for you



ANIMAL SOCIAL NETWORKS	816	INTERACTION NETWORKS	29	SCIENTIFIC COMPUTING	11
BIOLOGICAL NETWORKS	37	INFRASTRUCTURE NETWORKS	8	SOCIAL NETWORKS	77
BRAIN NETWORKS	116	LABELED NETWORKS	105	FACEBOOK NETWORKS	114
COLLABORATION NETWORKS	20	MASSIVE NETWORK DATA	21	TECHNOLOGICAL NETWORKS	12
CHEMINFORMATICS	646	MISCELLANEOUS NETWORKS	2669	WEB GRAPHS	36
CITATION NETWORKS	4	POWER NETWORKS	8	DYNAMIC NETWORKS	115
ECOLOGY NETWORKS	6	PROXIMITY NETWORKS	13	TEMPORAL REACHABILITY	38
ECONOMIC NETWORKS	16	GENERATED GRAPHS	221	BHOSLIB	36
EMAIL NETWORKS	6	RECOMMENDATION NETWORKS	36	DIMACS	78
GRAPH 500	8	ROAD NETWORKS	15	DIMACS10	84
HETEROGENEOUS NETWORKS	15	RETWEET NETWORKS	34	NON-RELATIONAL ML DATA	211

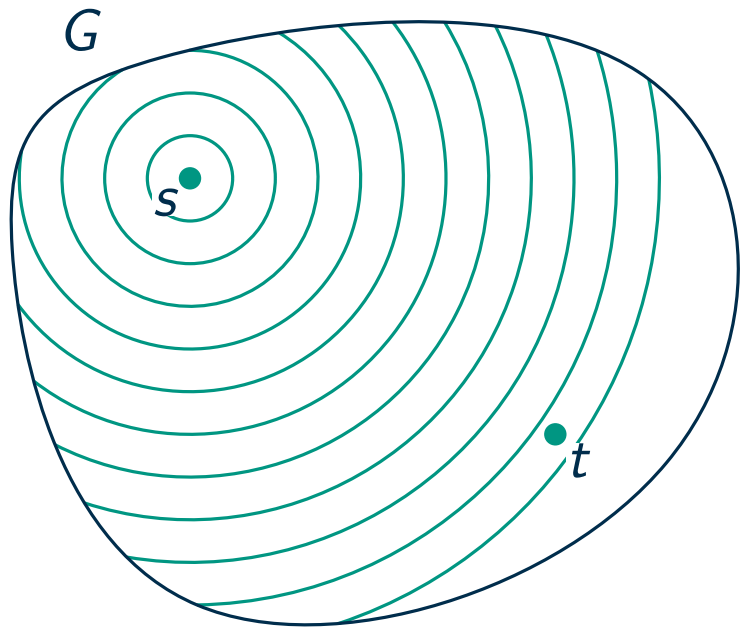




Your Task
Find out on which graphs the algorithms work well and on which they don’t.

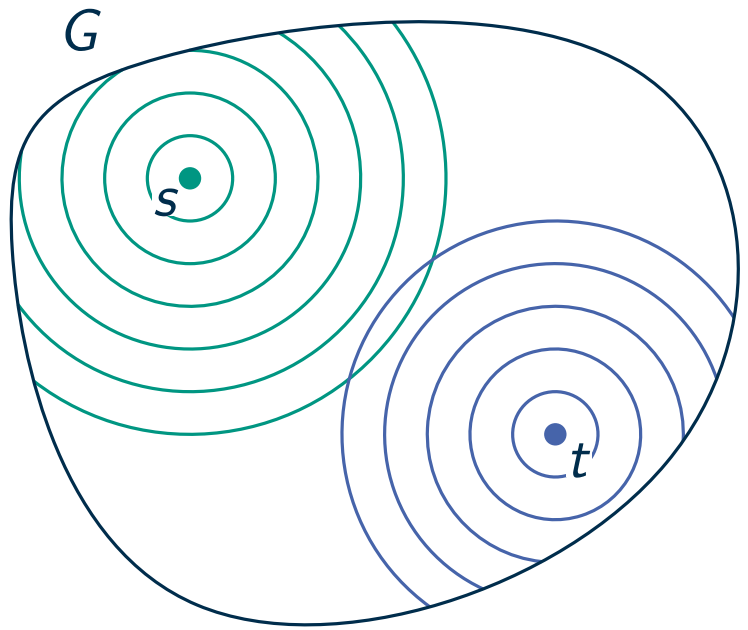
Bi-directional BFS

- BFS is one of the simplest graph algorithms



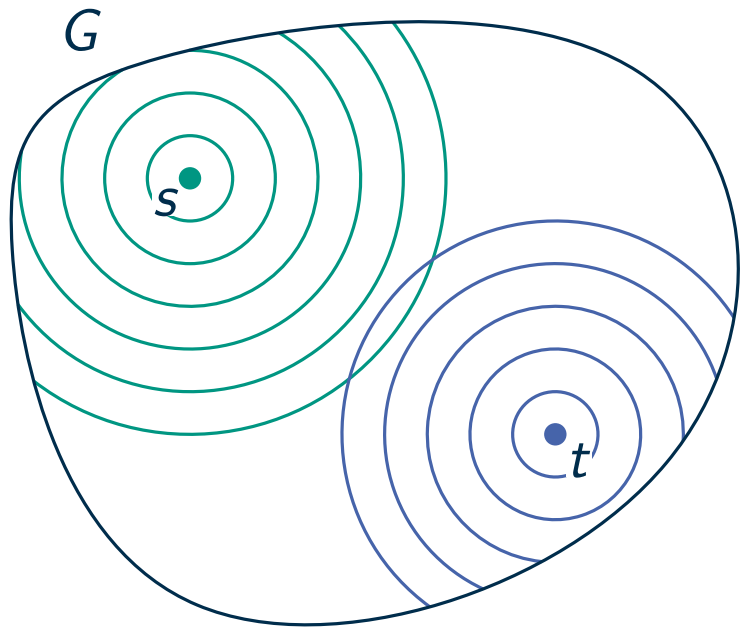
Bi-directional BFS

- BFS is one of the simplest graph algorithms
- bi-directional BFS is one of the simplest optimizations
- search steps are performed alternately from s and from t



Bi-directional BFS

- BFS is one of the simplest graph algorithms
- bi-directional BFS is one of the simplest optimizations
- search steps are performed alternately from s and from t
- variant: estimate the size of the next layer,
  explore the smaller one



Vertex Cover

- let $G = (V, E)$ be a graph
- $X \subseteq V(G)$ is *vertex cover* if for any edge $\{u, v\} \in E$ we have $u \in X$ or $v \in X$
- vertex v *dominates* its neighbor w if $N[w] \subseteq N[v]$
- *dominance rule:* a dominating vertex can be included in the vertex cover

