Termination

Degeneracy: One iteration of SIMPLEX leaves the objective value unchanged.

Termination

Degeneracy: One iteration of SIMPLEX leaves the objective value unchanged.

$$
\begin{array}{rclllllll}
z & = & & & x_1 & + & x_2 & + & x_3 \\
x_4 & = & 8 & - & x_1 & - & x_2 & & \\
x_5 & = & & & & & x_2 & - & x_3
\end{array}
$$

Termination

Degeneracy: One iteration of SIMPLEX leaves the objective value unchanged.

$$
\begin{array}{rcrcrcrcr}
z & = & & & x_1 & + & x_2 & + & x_3 \\
\boxed{x_4} & = & 8 & \boxed{- \quad x_1} & & - & x_2 & & \\
x_5 & = & & & & & x_2 & - & x_3
\end{array}
$$

Pivot with x_1 entering and x_4 leaving

Termination

Degeneracy: One iteration of SIMPLEX leaves the objective value unchanged.

$$
\begin{array}{rcrcrcrcr}
z & = & & & x_1 & + & x_2 & + & x_3 \\
x_4 & = & 8 & - & x_1 & - & x_2 & & \\
x_5 & = & & & & & x_2 & - & x_3
\end{array}
$$

Pivot with x_1 entering and x_4 leaving

$$
\begin{array}{rcrcrcrcr}
z & = & 8 & & & \boxed{+ \quad x_3} & & - & x_4 \\
x_1 & = & 8 & - & x_2 & & & - & x_4 \\
\boxed{x_5} & = & \boxed{} & & x_2 & \boxed{- \quad x_3} & & &
\end{array}
$$

since $b_5 = 0$ next basic solution will be identical (in particular, objective value remains the same)

Termination

Degeneracy: One iteration of SIMPLEX leaves the objective value unchanged.

$$
\begin{array}{rcrcrcrcr}
z & = & & & x_1 & + & x_2 & + & x_3 \\
x_4 & = & 8 & - & x_1 & - & x_2 & & \\
x_5 & = & & & & & x_2 & - & x_3
\end{array}
$$

$\Big\downarrow$ Pivot with x_1 entering and x_4 leaving

$$
\begin{array}{rcrcrcrcr}
z & = & 8 & & & + & x_3 & - & x_4 \\
x_1 & = & 8 & - & x_2 & & & - & x_4 \\
x_5 & = & & & x_2 & - & x_3 & &
\end{array}
$$

$\Big\downarrow$ Pivot with x_3 entering and x_5 leaving

Termination

Degeneracy: One iteration of SIMPLEX leaves the objective value unchanged.

$$\begin{array}{rcrcrcrcr} z & = & & & x_1 & + & x_2 & + & x_3 \\ x_4 & = & 8 & - & x_1 & - & x_2 & & \\ x_5 & = & & & & & x_2 & - & x_3 \end{array}$$

Pivot with x_1 entering and x_4 leaving

$$\begin{array}{rcrcrcrcr} z & = & 8 & & & + & x_3 & - & x_4 \\ x_1 & = & 8 & - & x_2 & & & - & x_4 \\ x_5 & = & & & x_2 & - & x_3 & & \end{array}$$

Pivot with x_3 entering and x_5 leaving

$$\begin{array}{rcrcrcrcr} z & = & 8 & + & x_2 & - & x_4 & - & x_5 \\ x_1 & = & 8 & - & x_2 & - & x_4 & & \\ x_3 & = & & & x_2 & & & - & x_5 \end{array}$$

Pivot with x_2 ent. and x_1 leav.

$\dashrightarrow (x_1, x_2, x_3) = (0, 8, 8)$

$z = 16 + \ldots$

Termination

Degeneracy: One iteration of SIMPLEX leaves the objective value unchanged.

$$\begin{aligned} z &= x_1 + x_2 + x_3 \\ x_4 &= 8 - x_1 - x_2 \\ x_5 &= x_2 - x_3 \end{aligned}$$

Pivot with x_1 entering and x_4 leaving

$$\begin{aligned} z &= 8 + x_3 - x_4 \\ x_1 &= 8 - x_2 - x_4 \\ x_5 &= x_2 - x_3 \end{aligned}$$

Pivot with x_3 entering and x_5 leaving

Cycling: Slack forms at two iterations are identical, and SIMPLEX fails to terminate!

$$\begin{aligned} z &= 8 + x_2 - x_4 - x_5 \\ x_1 &= 8 - x_2 - x_4 \\ x_3 &= x_2 - x_5 \end{aligned}$$

Termination and Running Time

Cycling: SIMPLEX may fail to terminate.

Termination and Running Time

It is theoretically possible, but very rare in practice.

Cycling: SIMPLEX may fail to terminate.

Termination and Running Time

It is theoretically possible, but very rare in practice.

Cycling: SIMPLEX may fail to terminate.

Anti-Cycling Strategies

Termination and Running Time

It is theoretically possible, but very rare in practice.

Cycling: SIMPLEX may fail to terminate.

Anti-Cycling Strategies

among j with $c_j > 0$

1. Bland's rule: Choose entering variable with smallest index

Termination and Running Time

It is theoretically possible, but very rare in practice.

Cycling: SIMPLEX may fail to terminate.

Anti-Cycling Strategies

1. Bland's rule: Choose entering variable with smallest index
2. Random rule: Choose entering variable uniformly at random

Termination and Running Time

It is theoretically possible, but very rare in practice.

Cycling: SIMPLEX may fail to terminate.

Anti-Cycling Strategies

1. Bland's rule: Choose entering variable with smallest index
2. Random rule: Choose entering variable uniformly at random
3. Perturbation: Perturb the input slightly so that it is impossible to have two solutions with the same objective value

Termination and Running Time

It is theoretically possible, but very rare in practice.

Cycling: SIMPLEX may fail to terminate.

Anti-Cycling Strategies

1. Bland's rule: Choose entering variable with smallest index
2. Random rule: Choose entering variable uniformly at random
3. Perturbation: Perturb the input slightly so that it is impossible to have two solutions with the same objective value

Replace each b_i by $\widehat{b}_i = b_i + \boxed{\epsilon_i}$ where $\epsilon_i \gg \epsilon_{i+1}$ are all small.

"random" noise

Termination and Running Time

It is theoretically possible, but very rare in practice.

Cycling: SIMPLEX may fail to terminate.

Anti-Cycling Strategies

1. Bland's rule: Choose entering variable with smallest index
2. Random rule: Choose entering variable uniformly at random
3. Perturbation: Perturb the input slightly so that it is impossible to have two solutions with the same objective value

Replace each b_i by $\widehat{b_i} = b_i + \epsilon_i$, where $\epsilon_i \gg \epsilon_{i+1}$ are all small.

Lemma 29.7

Assuming INITIALIZE-SIMPLEX returns a slack form for which the basic solution is feasible, SIMPLEX either reports that the program is unbounded or returns a feasible solution in at most $\boxed{\binom{n+m}{m} \text{ iterations.}}$

Termination and Running Time

It is theoretically possible, but very rare in practice.

Cycling: SIMPLEX may fail to terminate.

Anti-Cycling Strategies

1. Bland's rule: Choose entering variable with smallest index
2. Random rule: Choose entering variable uniformly at random
3. Perturbation: Perturb the input slightly so that it is impossible to have two solutions with the same objective value

Replace each b_i by $\widehat{b}_i = b_i + \epsilon_i$, where $\epsilon_i \gg \epsilon_{i+1}$ are all small.

Lemma 29.7

Assuming INITIALIZE-SIMPLEX returns a slack form for which the basic solution is feasible, SIMPLEX either reports that the program is unbounded or returns a feasible solution in at most $\binom{n+m}{m}$ iterations.

Every set B of basic variables uniquely determines a slack form, and there are at most $\binom{n+m}{m}$ unique slack forms.

Outline

Introduction

Standard and Slack Forms

Formulating Problems as Linear Programs

Simplex Algorithm

Finding an Initial Solution

Finding an Initial Solution

$$\begin{array}{lrcrcr} \text{maximize} & 2x_1 & - & x_2 & & \\ \text{subject to} & & & & & \\ & 2x_1 & - & x_2 & \leq & 2 \\ & x_1 & - & 5x_2 & \leq & \boxed{-4} \\ & & x_1, x_2 & & \geq & 0 \end{array}$$

Finding an Initial Solution

$$
\begin{array}{lrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
& 2x_1 & - & x_2 & \leq & 2 \\
& x_1 & - & 5x_2 & \leq & -4 \\
& & x_1, x_2 & & \geq & 0
\end{array}
$$

$$\Big\downarrow \ \text{Conversion into slack form}$$

Finding an Initial Solution

$$
\begin{array}{lrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
& 2x_1 & - & x_2 & \leq & 2 \\
& x_1 & - & 5x_2 & \leq & -4 \\
& & x_1, x_2 & & \geq & 0
\end{array}
$$

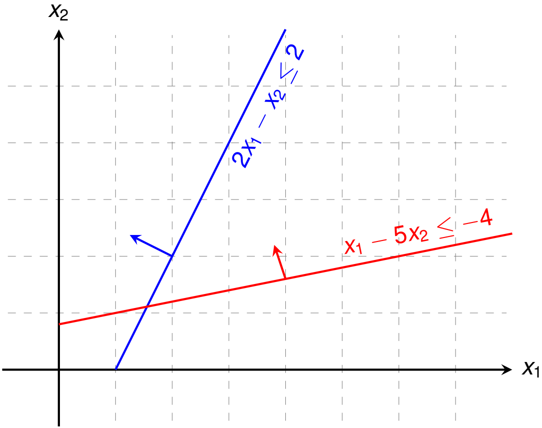
Conversion into slack form

$$
\begin{array}{rcrcrcr}
z & = & & & 2x_1 & - & x_2 \\
x_3 & = & 2 & - & 2x_1 & + & x_2 \\
x_4 & = & -4 & - & x_1 & + & 5x_2
\end{array}
$$

Basic solution $(x_1, x_2, x_3, x_4) = (0, 0, 2, -4)$ is not feasible!

Geometric Illustration

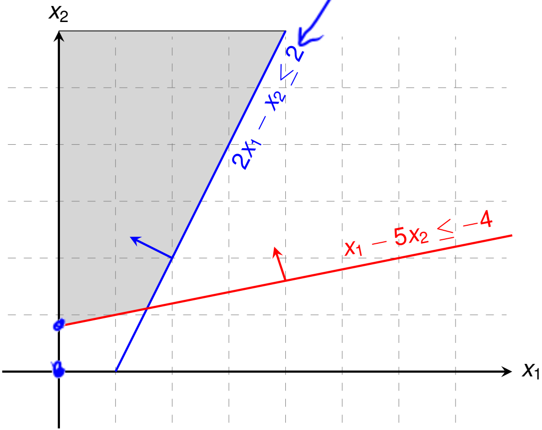
$$
\begin{array}{lrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
& 2x_1 & - & x_2 & \leq & 2 \\
& x_1 & - & 5x_2 & \leq & -4 \\
& & x_1, x_2 & & \geq & 0
\end{array}
$$



Geometric Illustration

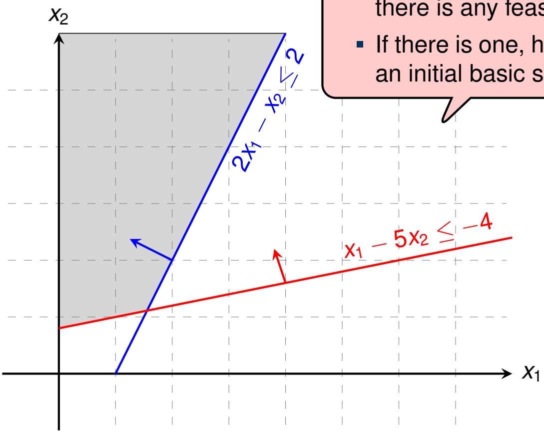
$$\begin{array}{lrcrcr} \text{maximize} & \boxed{2x_1 & - & x_2} & & \\ \text{subject to} & & & & & \\ & \boxed{2x_1 & - & x_2} & \leq & 2 \\ & x_1 & - & 5x_2 & \leq & -4 \\ & & x_1, x_2 & & \geq & 0 \end{array}$$

all points (x_1, x_2) with $2x_1 - x_2 = 2$ are optimal



Geometric Illustration

$$
\begin{array}{lrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
& 2x_1 & - & x_2 & \leq & 2 \\
& x_1 & - & 5x_2 & \leq & -4 \\
& & x_1, x_2 & & \geq & 0
\end{array}
$$



Questions:

- How to determine whether there is any feasible solution?
- If there is one, how to determine an initial basic solution?

Formulating an Auxiliary Linear Program

maximize $\sum_{j=1}^{n} c_j x_j$

subject to

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n \end{array}$$

Formulating an Auxiliary Linear Program

maximize $\sum_{j=1}^{n} c_j x_j$

subject to

$$
\begin{array}{rcll}
\sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\
x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n
\end{array}
$$

Formulating an Auxiliary Linear Program

Formulating an Auxiliary Linear Program

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n \end{array}$$

Formulating an Auxiliary Linear Program

"Relax" each constraint

$$\begin{array}{ll} \text{maximize} & \boxed{-x_0} \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j \boxed{- x_0} & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 0, 1, \ldots, n \end{array}$$

minimize x_0, the "distance" from being feasible

Formulating an Auxiliary Linear Program

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n \end{array}$$

Formulating an Auxiliary Linear Program

$$\begin{array}{ll} \text{maximize} & -x_0 \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j - x_0 & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 0, 1, \ldots, n \end{array}$$

Lemma 29.11

Let L_{aux} be the auxiliary LP of a linear program L in standard form. Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Formulating an Auxiliary Linear Program

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n \end{array}$$

Formulating an Auxiliary Linear Program

$$\begin{array}{ll} \text{maximize} & -x_0 \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j - x_0 & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 0, 1, \ldots, n \end{array}$$

Lemma 29.11

Let L_{aux} be the auxiliary LP of a linear program L in standard form. Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Proof.

Formulating an Auxiliary Linear Program

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \boxed{\sum_{j=1}^{n} a_{ij} x_j & \leq & b_i} & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n \end{array}$$

Formulating an Auxiliary Linear Program

$$\begin{array}{ll} \text{maximize} & -x_0 \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j - x_0 & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 0, 1, \ldots, n \end{array}$$

Lemma 29.11

Let L_{aux} be the auxiliary LP of a linear program L in standard form. Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Proof.

- "$\Rightarrow$": Suppose L has a feasible solution $\overline{x} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$

Formulating an Auxiliary Linear Program

$$\begin{array}{lrcll} \text{maximize} & \sum_{j=1}^{n} c_j x_j & & & \\ \text{subject to} & & & & \\ & \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ & x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n \end{array}$$

Formulating an Auxiliary Linear Program

$$\begin{array}{lrcll} \text{maximize} & -x_0 & & & \\ \text{subject to} & & & & \\ & \sum_{j=1}^{n} a_{ij} x_j - x_0 & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ & x_j & \geq & 0 & \text{for } j = 0, 1, \ldots, n \end{array}$$

Lemma 29.11

Let L_{aux} be the auxiliary LP of a linear program L in standard form. Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Proof.

- "$\Rightarrow$": Suppose L has a feasible solution $\overline{x} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$
  - $\overline{x}_0 = 0$ combined with $\overline{x}$ is a feasible solution to L_{aux} with objective value 0.

Formulating an Auxiliary Linear Program

$$\begin{array}{ll} \text{maximize} & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n \end{array}$$

Formulating an Auxiliary Linear Program

$$\begin{array}{ll} \text{maximize} & -x_0 \\ \text{subject to} & \end{array}$$

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j - x_0 & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 0, 1, \ldots, n \end{array}$$

Lemma 29.11

Let L_{aux} be the auxiliary LP of a linear program L in standard form. Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Proof.

- "$\Rightarrow$": Suppose L has a feasible solution $\overline{x} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$
  - $\overline{x}_0 = 0$ combined with $\overline{x}$ is a feasible solution to L_{aux} with objective value 0.
  - Since $\overline{x}_0 \geq 0$ and the objective is to maximize $-x_0$, this is optimal for L_{aux}

Formulating an Auxiliary Linear Program

$$
\begin{array}{ll}
\text{maximize} & \sum_{j=1}^{n} c_j x_j \\
\text{subject to} &
\end{array}
$$

$$
\begin{array}{rcll}
\sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\
x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n
\end{array}
$$

$\Downarrow$ Formulating an Auxiliary Linear Program

$$
\begin{array}{ll}
\text{maximize} & \underline{-x_0} = 0 \\
\text{subject to} &
\end{array}
$$

$$
\begin{array}{rcll}
\sum_{j=1}^{n} a_{ij} x_j \cancel{- x_0} & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\
x_j & \geq & 0 & \text{for } j = 0, 1, \ldots, n
\end{array}
$$

Lemma 29.11

Let L_{aux} be the auxiliary LP of a linear program L in standard form. Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Proof.

- "$\Rightarrow$": Suppose L has a feasible solution $\overline{x} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$
  - $\overline{x}_0 = 0$ combined with $\overline{x}$ is a feasible solution to L_{aux} with objective value 0.
  - Since $\overline{x}_0 \geq 0$ and the objective is to maximize $-x_0$, this is optimal for L_{aux}
- "$\Leftarrow$": Suppose that the optimal objective value of L_{aux} is 0

Formulating an Auxiliary Linear Program

maximize $\sum_{j=1}^{n} c_j x_j$

subject to

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 1, 2, \ldots, n \end{array}$$

$\Big\downarrow$ Formulating an Auxiliary Linear Program

maximize $-x_0$

subject to

$$\begin{array}{rcll} \sum_{j=1}^{n} a_{ij} x_j - x_0 & \leq & b_i & \text{for } i = 1, 2, \ldots, m, \\ x_j & \geq & 0 & \text{for } j = 0, 1, \ldots, n \end{array}$$

Lemma 29.11

Let L_{aux} be the auxiliary LP of a linear program L in standard form. Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Proof.

- "$\Rightarrow$": Suppose L has a feasible solution $\overline{x} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$
  - $\overline{x}_0 = 0$ combined with $\overline{x}$ is a feasible solution to L_{aux} with objective value 0.
  - Since $\overline{x}_0 \geq 0$ and the objective is to maximize $-x_0$, this is optimal for L_{aux}
- "$\Leftarrow$": Suppose that the optimal objective value of L_{aux} is 0
  - Then $\overline{x}_0 = 0$, and the remaining solution values $(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$ satisfy L.

Formulating an Auxiliary Linear Program

maximize $\sum_{j=1}^{n} c_j x_j$

subject to

$$\begin{aligned} \textstyle\sum_{j=1}^{n} a_{ij} x_j &\leq b_i \quad \text{for } i = 1, 2, \ldots, m, \\ x_j &\geq 0 \quad \text{for } j = 1, 2, \ldots, n \end{aligned}$$

Formulating an Auxiliary Linear Program

maximize $-x_0$

subject to

$$\begin{aligned} \textstyle\sum_{j=1}^{n} a_{ij} x_j - x_0 &\leq b_i \quad \text{for } i = 1, 2, \ldots, m, \\ x_j &\geq 0 \quad \text{for } j = 0, 1, \ldots, n \end{aligned}$$

Lemma 29.11

Let L_{aux} be the auxiliary LP of a linear program L in standard form. Then L is feasible if and only if the optimal objective value of L_{aux} is 0.

Proof.

- "$\Rightarrow$": Suppose L has a feasible solution $\overline{x} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$
  - $\overline{x}_0 = 0$ combined with $\overline{x}$ is a feasible solution to L_{aux} with objective value 0.
  - Since $\overline{x}_0 \geq 0$ and the objective is to maximize $-x_0$, this is optimal for L_{aux}
- "$\Leftarrow$": Suppose that the optimal objective value of L_{aux} is 0
  - Then $\overline{x}_0 = 0$, and the remaining solution values $(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$ satisfy L. $\square$

INITIALIZE-SIMPLEX

INITIALIZE-SIMPLEX(A, b, c)

let k be the index of the minimum b_i
if $b_k \geq 0$ // is the initial basic solution feasible?
    return $(\{1, 2, \ldots, n\}, \{n+1, n+2, \ldots, n+m\}, A, b, c, 0)$
form L_{aux} by adding $-x_0$ to the left-hand side of each constraint
    and setting the objective function to $-x_0$
let (N, B, A, b, c, ν) be the resulting slack form for L_{aux}
$l = n + k$
// L_{aux} has $n + 1$ nonbasic variables and m basic variables.
$(N, B, A, b, c, \nu) = \text{PIVOT}(N, B, A, b, c, \nu, l, 0)$
// The basic solution is now feasible for L_{aux}.
iterate the **while** loop of lines 3–12 of SIMPLEX until an optimal solution
    to L_{aux} is found
if the optimal solution to L_{aux} sets $\bar{x}_0$ to 0
    if $\bar{x}_0$ is basic
        perform one (degenerate) pivot to make it nonbasic
    from the final slack form of L_{aux}, remove x_0 from the constraints and
        restore the original objective function of L, but replace each basic
        variable in this objective function by the right-hand side of its
        associated constraint
    return the modified final slack form
else return "infeasible"

Test solution with $N = \{1, 2, \ldots, n\}$, $B = \{n+1, n+2, \ldots, n+m\}$, $\overline{x}_i = b_i$ for $i \in B$, $\overline{x}_i = 0$ otherwise.

INITIALIZE-SIMPLEX(A, b, c)

let k be the index of the minimum b_i
if $b_k \geq 0$ // is the initial basic solution feasible?
    return $(\{1, 2, \ldots, n\}, \{n+1, n+2, \ldots, n+m\}, A, b, c, 0)$
form L_{aux} by adding $-x_0$ to the left-hand side of each constraint
    and setting the objective function to $-x_0$
let (N, B, A, b, c, ν) be the resulting slack form for L_{aux}
$l = n + k$
// L_{aux} has $n+1$ nonbasic variables and m basic variables.
$(N, B, A, b, c, \nu) = \text{PIVOT}(N, B, A, b, c, \nu, l, 0)$
// The basic solution is now feasible for L_{aux}.
iterate the **while** loop of lines 3–12 of SIMPLEX until an optimal solution
    to L_{aux} is found
if the optimal solution to L_{aux} sets $\bar{x}_0$ to 0
    if $\bar{x}_0$ is basic
        perform one (degenerate) pivot to make it nonbasic
    from the final slack form of L_{aux}, remove x_0 from the constraints and
        restore the original objective function of L, but replace each basic
        variable in this objective function by the right-hand side of its
        associated constraint
    return the modified final slack form
else return "infeasible"

Test solution with $N = \{1, 2, \ldots, n\}$, $B = \{n+1, n+2, \ldots, n+m\}$, $\overline{x}_i = b_i$ for $i \in B$, $\overline{x}_i = 0$ otherwise.

INITIALIZE-SIMPLEX(A, b, c)

1 let k be the index of the minimum b_i
2 **if** $b_k \geq 0$ // is the initial basic solution feasible?
3   **return** $(\{1, 2, \ldots, n\}, \{n+1, n+2, \ldots, n+m\}, A, b, c, 0)$
4 form L_{aux} by adding $-x_0$ to the left-hand side of each constraint and setting the objective function to $-x_0$
5 let (N, B, A, b, c, ν) be the resulting slack form for L_{aux}
6 $\boxed{l = n + k}$
7 // L_{aux} has $n + 1$ nonbasic variables and m basic variables.
8 $\boxed{(N, B, A, b, c, \nu) = \text{PIVOT}(N, B, A, b, c, \nu, l, 0)}$
9 // The basic solution is now feasible for L_{aux}.
10 iterate the **while** loop of lines 3–12 of SIMPLEX until an optimal solution to L_{aux} is found
11 **if** the optimal solution to L_{aux} sets $\bar{x}_0$ to 0
12   **if** $\bar{x}_0$ is basic
13   perform one (degenerate) pivot to make it nonbasic
14   from the final slack form of L_{aux}, remove x_0 from the constraints and restore the original objective function of L, but replace each basic variable in this objective function by the right-hand side of its associated constraint
15   **return** the modified final slack form
16 **else return** "infeasible"

ℓ will be the leaving variable so that x_ℓ has the most negative value.

this ensures basic solution becomes feasible after line 8 (non-negative)

INITIALIZE-SIMPLEX

Test solution with $N = \{1, 2, \ldots, n\}$, $B = \{n+1, n+2, \ldots, n+m\}$, $\overline{x}_i = b_i$ for $i \in B$, $\overline{x}_i = 0$ otherwise.

INITIALIZE-SIMPLEX(A, b, c)

let k be the index of the minimum b_i
if $b_k \geq 0$ // is the initial basic solution feasible?
    return $(\{1, 2, \ldots, n\}, \{n+1, n+2, \ldots, n+m\}, A, b, c, 0)$
form L_{aux} by adding $-x_0$ to the left-hand side of each constraint
    and setting the objective function to $-x_0$
let (N, B, A, b, c, ν) be the resulting slack form for L_{aux}
$l = n + k$
// L_{aux} has $n+1$ nonbasic variables and m basic variables.
$(N, B, A, b, c, \nu) = \text{PIVOT}(N, B, A, b, c, \nu, l, 0)$
// The basic solution is now feasible for L_{aux}.
iterate the **while** loop of lines 3–12 of SIMPLEX until an optimal solution
    to L_{aux} is found
if the optimal solution to L_{aux} sets $\bar{x}_0$ to 0
    if $\bar{x}_0$ is basic
        perform one (degenerate) pivot to make it nonbasic
    from the final slack form of L_{aux}, remove x_0 from the constraints and
        restore the original objective function of L, but replace each basic
        variable in this objective function by the right-hand side of its
        associated constraint
    return the modified final slack form
else return "infeasible"

ℓ will be the leaving variable so that x_ℓ has the most negative value.

Pivot step with x_ℓ leaving and x_0 entering.

INITIALIZE-SIMPLEX

Test solution with $N = \{1, 2, \ldots, n\}$, $B = \{n+1, n+2, \ldots, n+m\}$, $\overline{x}_i = b_i$ for $i \in B$, $\overline{x}_i = 0$ otherwise.

INITIALIZE-SIMPLEX(A, b, c)

let k be the index of the minimum b_i
if $b_k \geq 0$ // is the initial basic solution feasible?
    return $(\{1, 2, \ldots, n\}, \{n+1, n+2, \ldots, n+m\}, A, b, c, 0)$
form L_{aux} by adding $-x_0$ to the left-hand side of each constraint
    and setting the objective function to $-x_0$
let (N, B, A, b, c, ν) be the resulting slack form for L_{aux}
$l = n + k$
// L_{aux} has $n+1$ nonbasic variables and m basic variables.
$(N, B, A, b, c, \nu) = \text{PIVOT}(N, B, A, b, c, \nu, l, 0)$
// The basic solution is now feasible for L_{aux}.
iterate the **while** loop of lines 3–12 of SIMPLEX until an optimal solution
    to L_{aux} is found
if the optimal solution to L_{aux} sets $\bar{x}_0$ to 0
    if $\bar{x}_0$ is basic
        perform one (degenerate) pivot to make it nonbasic
    from the final slack form of L_{aux}, remove x_0 from the constraints and
        restore the original objective function of L, but replace each basic
        variable in this objective function by the right-hand side of its
        associated constraint
    return the modified final slack form
else return "infeasible"

ℓ will be the leaving variable so that x_ℓ has the most negative value.

Pivot step with x_ℓ leaving and x_0 entering.

This pivot step does not change the value of any variable.

because $\overline{x}_0 = 0$!

Example of INITIALIZE-SIMPLEX (1/3)

$$
\begin{array}{llrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
 & 2x_1 & - & x_2 & \leq & 2 \\
 & x_1 & - & 5x_2 & \leq & \boxed{-4} \\
 & & x_1, x_2 & & \geq & 0
\end{array}
$$

→ "canonical" basic solution is not feasible!

Example of INITIALIZE-SIMPLEX (1/3)

$$\begin{array}{lrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
& 2x_1 & - & x_2 & \leq & 2 \\
& x_1 & - & 5x_2 & \leq & -4 \\
& & x_1, x_2 & & \geq & 0
\end{array}$$

Formulating the auxiliary linear program

Example of INITIALIZE-SIMPLEX (1/3)

$$
\begin{array}{llcrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & & & \\
\text{subject to} & & & & & & & \\
 & 2x_1 & - & x_2 & & & \leq & 2 \\
 & x_1 & - & 5x_2 & & & \leq & -4 \\
 & & x_1, x_2 & & & & \geq & 0
\end{array}
$$

$\downarrow$ Formulating the auxiliary linear program

$$
\begin{array}{llcrcrcr}
\text{maximize} & & & & & -x_0 & & \\
\text{subject to} & & & & & & & \\
 & 2x_1 & - & x_2 & - & x_0 & \leq & 2 \\
 & x_1 & - & 5x_2 & - & x_0 & \leq & -4 \\
 & & x_1, x_2, x_0 & & & & \geq & 0
\end{array}
$$

Example of INITIALIZE-SIMPLEX (1/3)

$$\begin{array}{lrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
& 2x_1 & - & x_2 & \leq & 2 \\
& x_1 & - & 5x_2 & \leq & -4 \\
& & x_1, x_2 & & \geq & 0
\end{array}$$

Formulating the auxiliary linear program

$$\begin{array}{lrcrcrcr}
\text{maximize} & & & & & -x_0 & & \\
\text{subject to} & & & & & & & \\
& 2x_1 & - & x_2 & - & x_0 & \leq & 2 \\
& x_1 & - & 5x_2 & - & x_0 & \leq & -4 \\
& x_1, x_2, x_0 & & & & & \geq & 0
\end{array}$$

Converting into slack form

Example of INITIALIZE-SIMPLEX (1/3)

$$\begin{array}{lrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
 & 2x_1 & - & x_2 & \leq & 2 \\
 & x_1 & - & 5x_2 & \leq & -4 \\
 & & x_1, x_2 & & \geq & 0
\end{array}$$

$\downarrow$ Formulating the auxiliary linear program

$$\begin{array}{lrcrcrcr}
\text{maximize} & & & & & -x_0 & & \\
\text{subject to} & & & & & & & \\
 & 2x_1 & - & x_2 & - & x_0 & \leq & 2 \\
 & x_1 & - & 5x_2 & - & x_0 & \leq & -4 \\
 & & x_1, x_2, x_0 & & & & \geq & 0
\end{array}$$

$\downarrow$ Converting into slack form

$$\begin{array}{rcrcrcrcr}
z & = & & & & & & - & x_0 \\
x_3 & = & 2 & - & 2x_1 & + & x_2 & + & x_0 \\
x_4 & = & -4 & - & x_1 & + & 5x_2 & + & x_0
\end{array}$$

Example of INITIALIZE-SIMPLEX (1/3)

$$\begin{array}{llcrcrcr}
\text{maximize} & 2x_1 & - & x_2 & & \\
\text{subject to} & & & & & \\
& 2x_1 & - & x_2 & \leq & 2 \\
& x_1 & - & 5x_2 & \leq & -4 \\
& & x_1, x_2 & & \geq & 0
\end{array}$$

Formulating the auxiliary linear program

$$\begin{array}{llcrcrcr}
\text{maximize} & & & & & -x_0 & & \\
\text{subject to} & & & & & & & \\
& 2x_1 & - & x_2 & - & x_0 & \leq & 2 \\
& x_1 & - & 5x_2 & - & x_0 & \leq & -4 \\
& & x_1, x_2, x_0 & & & & \geq & 0
\end{array}$$

Basic solution $(0, 0, 0, 2, -4)$ not feasible!

Converting into slack form

$$\begin{array}{rcrcrcrcr}
z & = & & & & & & - & x_0 \\
x_3 & = & 2 & - & 2x_1 & + & x_2 & + & x_0 \\
x_4 & = & -4 & - & x_1 & + & 5x_2 & + & x_0
\end{array}$$

Example of INITIALIZE-SIMPLEX (2/3)

$$
\begin{array}{rcrcrcrcr}
z & = & & & & & & \boxed{-} & \boxed{x_0} \\
x_3 & = & 2 & - & 2x_1 & + & x_2 & + & x_0 \\
x_4 & = & \boxed{-4} & - & x_1 & + & 5x_2 & + & x_0
\end{array}
$$

Example of INITIALIZE-SIMPLEX (2/3)

$$
\begin{array}{rcrcrcrcr}
z & = & & & & & & \boxed{-} & \boxed{x_0} \\
x_3 & = & 2 & - & 2x_1 & + & x_2 & + & x_0 \\
x_4 & = & -4 & - & x_1 & + & 5x_2 & + & x_0
\end{array}
$$

Pivot with x_0 entering and x_4 leaving

(this "degenerate" pivot step ensures all basic variables are non-negative)

Example of INITIALIZE-SIMPLEX (2/3)

$$
\begin{array}{rcrcrcrcr}
z & = & & & & & & - & x_0 \\
x_3 & = & 2 & - & 2x_1 & + & x_2 & + & x_0 \\
x_4 & = & -4 & - & x_1 & + & 5x_2 & + & x_0
\end{array}
$$

Pivot with x_0 entering and x_4 leaving

$$
\begin{array}{rcrcrcrcr}
z & = & -4 & - & x_1 & + & 5x_2 & - & x_4 \\
x_0 & = & \boxed{4} & + & x_1 & - & 5x_2 & + & x_4 \\
x_3 & = & \boxed{6} & - & x_1 & - & 4x_2 & + & x_4
\end{array}
$$

$6 = 2 + 4$

Example of INITIALIZE-SIMPLEX (2/3)

$$
\begin{array}{rcrcrcrcr}
z & = & & & & & & - & x_0 \\
x_3 & = & 2 & - & 2x_1 & + & x_2 & + & x_0 \\
x_4 & = & -4 & - & x_1 & + & 5x_2 & + & x_0
\end{array}
$$

Pivot with x_0 entering and x_4 leaving

$$
\begin{array}{rcrcrcrcr}
z & = & -4 & - & x_1 & + & 5x_2 & - & x_4 \\
x_0 & = & 4 & + & x_1 & - & 5x_2 & + & x_4 \\
x_3 & = & 6 & - & x_1 & - & 4x_2 & + & x_4
\end{array}
$$

Basic solution $(4, 0, 0, 6, 0)$ is feasible!

Example of INITIALIZE-SIMPLEX (2/3)

$$
\begin{array}{rcrcrcrcr}
z & = & & & & & & - & x_0 \\
x_3 & = & 2 & - & 2x_1 & + & x_2 & + & x_0 \\
x_4 & = & -4 & - & x_1 & + & 5x_2 & + & x_0
\end{array}
$$

Pivot with x_0 entering and x_4 leaving

$$
\begin{array}{rcrcrcrcr}
z & = & -4 & - & x_1 & + & 5x_2 & - & x_4 \\
x_0 & = & 4 & + & x_1 & - & 5x_2 & + & x_4 \\
x_3 & = & 6 & - & x_1 & - & 4x_2 & + & x_4
\end{array}
$$

Basic solution $(4, 0, 0, 6, 0)$ is feasible!

Pivot with x_2 entering and x_0 leaving

$$
\begin{array}{rcrcrcrcr}
z & = & & - & x_0 & & & & \\
x_2 & = & \frac{4}{5} & - & \frac{x_0}{5} & + & \frac{x_1}{5} & + & \frac{x_4}{5} \\
x_3 & = & \frac{14}{5} & + & \frac{4x_0}{5} & - & \frac{9x_1}{5} & + & \frac{x_4}{5}
\end{array}
$$

Example of Initialize-Simplex (2/3)

$$\begin{array}{rcrcrcrcr} z & = & & & & & & - & x_0 \\ x_3 & = & 2 & - & 2x_1 & + & x_2 & + & x_0 \\ x_4 & = & -4 & - & x_1 & + & 5x_2 & + & x_0 \end{array}$$

Pivot with x_0 entering and x_4 leaving

$$\begin{array}{rcrcrcrcr} z & = & -4 & - & x_1 & + & 5x_2 & - & x_4 \\ x_0 & = & 4 & + & x_1 & - & 5x_2 & + & x_4 \\ x_3 & = & 6 & - & x_1 & - & 4x_2 & + & x_4 \end{array}$$

Basic solution $(4, 0, 0, 6, 0)$ is feasible!

Pivot with x_2 entering and x_0 leaving

$$\begin{array}{rcrcrcrcr} z & = & & - & x_0 & & & & \\ x_2 & = & \frac{4}{5} & - & \frac{x_0}{5} & + & \frac{x_1}{5} & + & \frac{x_4}{5} \\ x_3 & = & \frac{14}{5} & + & \frac{4x_0}{5} & - & \frac{9x_1}{5} & + & \frac{x_4}{5} \end{array}$$

Optimal solution has $x_0 = 0$, hence the initial problem was feasible!

Example of INITIALIZE-SIMPLEX (3/3)

$$
\begin{array}{rcccccccc}
z & = & & - & x_0 & & & & \\
x_2 & = & \frac{4}{5} & - & \frac{x_0}{5} & + & \frac{x_1}{5} & + & \frac{x_4}{5} \\
x_3 & = & \frac{14}{5} & + & \frac{4x_0}{5} & - & \frac{9x_1}{5} & + & \frac{x_4}{5}
\end{array}
$$

Example of INITIALIZE-SIMPLEX (3/3)

$$
\begin{aligned}
z &= & - & \; x_0 & & & & \\
x_2 &= \frac{4}{5} & - & \frac{x_0}{5} & + & \frac{x_1}{5} & + & \frac{x_4}{5} \\
x_3 &= \frac{14}{5} & + & \frac{4x_0}{5} & - & \frac{9x_1}{5} & + & \frac{x_4}{5}
\end{aligned}
$$

Set $x_0 = 0$ and express objective function by non-basic variables

Example of INITIALIZE-SIMPLEX (3/3)

$$\begin{array}{rcrcrcrcr}
z & = & & - & x_0 & & & & \\
x_2 & = & \frac{4}{5} & - & \frac{x_0}{5} & + & \frac{x_1}{5} & + & \frac{x_4}{5} \\
x_3 & = & \frac{14}{5} & + & \frac{4x_0}{5} & - & \frac{9x_1}{5} & + & \frac{x_4}{5}
\end{array}$$

Set $x_0 = 0$ and express objective function by non-basic variables

$$2x_1 - 2x_2 = 2x_1 - \left(\frac{4}{5} - \cancel{\frac{x_0}{5}} + \frac{x_1}{5} + \frac{x_4}{5}\right)$$

$$\begin{array}{rcrcrcr}
z & = & -\frac{4}{5} & + & \frac{9x_1}{5} & - & \frac{x_4}{5} \\
x_2 & = & \frac{4}{5} & + & \frac{x_1}{5} & + & \frac{x_4}{5} \\
x_3 & = & \frac{14}{5} & - & \frac{9x_1}{5} & + & \frac{x_4}{5}
\end{array}$$

Example of INITIALIZE-SIMPLEX (3/3)

$$\begin{aligned} z &= && - && x_0 \\ x_2 &= \frac{4}{5} && - && \frac{x_0}{5} && + && \frac{x_1}{5} && + && \frac{x_4}{5} \\ x_3 &= \frac{14}{5} && + && \frac{4x_0}{5} && - && \frac{9x_1}{5} && + && \frac{x_4}{5} \end{aligned}$$

Set $x_0 = 0$ and express objective function by non-basic variables

$$2x_1 - 2x_2 = 2x_1 - \left(\frac{4}{5} - \frac{x_0}{5} + \frac{x_1}{5} + \frac{x_4}{5}\right)$$

$$\begin{aligned} z &= -\frac{4}{5} && + && \frac{9x_1}{5} && - && \frac{x_4}{5} \\ x_2 &= \frac{4}{5} && + && \frac{x_1}{5} && + && \frac{x_4}{5} \\ x_3 &= \frac{14}{5} && - && \frac{9x_1}{5} && + && \frac{x_4}{5} \end{aligned}$$

Slack form returned by INITIALIZE-SIMPLEX

Basic solution $(0, \frac{4}{5}, \frac{14}{5}, 0)$, which is feasible!

Main Loop of SIMPLEX can be now executed

Example of INITIALIZE-SIMPLEX (3/3)

$$
\begin{aligned}
z &= & - & \; x_0 & & & & \\
x_2 &= \frac{4}{5} & - & \frac{x_0}{5} & + & \frac{x_1}{5} & + & \frac{x_4}{5} \\
x_3 &= \frac{14}{5} & + & \frac{4x_0}{5} & - & \frac{9x_1}{5} & + & \frac{x_4}{5}
\end{aligned}
$$

Set $x_0 = 0$ and express objective function by non-basic variables

$$2x_1 - 2x_2 = 2x_1 - \left(\tfrac{4}{5} - \tfrac{x_0}{5} + \tfrac{x_1}{5} + \tfrac{x_4}{5}\right)$$

$$
\begin{aligned}
z &= -\frac{4}{5} & + & \frac{9x_1}{5} & - & \frac{x_4}{5} \\
x_2 &= \frac{4}{5} & + & \frac{x_1}{5} & + & \frac{x_4}{5} \\
x_3 &= \frac{14}{5} & - & \frac{9x_1}{5} & + & \frac{x_4}{5}
\end{aligned}
$$

Basic solution $(0, \frac{4}{5}, \frac{14}{5}, 0)$, which is feasible!

Lemma 29.12

If a linear program L has no feasible solution, then INITIALIZE-SIMPLEX returns "infeasible". Otherwise, it returns a valid slack form for which the basic solution is feasible.

Fundamental Theorem of Linear Programming

Theorem 29.13

Any linear program L, given in standard form, either

1. has an optimal solution with a finite objective value,
2. is infeasible, or
3. is unbounded.

If L is infeasible, SIMPLEX returns "infeasible". If L is unbounded, SIMPLEX returns "unbounded". Otherwise, SIMPLEX returns an optimal solution with a finite objective value.

proof non-trivial and requires concept of "dual linear program".
[CLRS3, Chapter 29.4]

Linear Programming and Simplex: Summary

Linear Programming

Linear Programming and Simplex: Summary

Linear Programming

- extremely versatile tool for modelling problems of all kinds

Linear Programming and Simplex: Summary

Linear Programming

- extremely versatile tool for modelling problems of all kinds
- basis of Integer Programming, to be discussed in later lectures

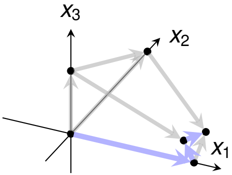
Linear Programming and Simplex: Summary

Linear Programming

- extremely versatile tool for modelling problems of all kinds
- basis of Integer Programming, to be discussed in later lectures

Simplex Algorithm

- In practice: usually terminates in polynomial time, i.e., $O(m+n)$



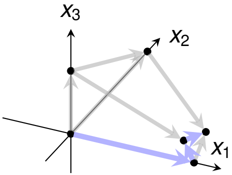
Linear Programming and Simplex: Summary

Linear Programming

- extremely versatile tool for modelling problems of all kinds
- basis of Integer Programming, to be discussed in later lectures

Simplex Algorithm

- In practice: usually terminates in polynomial time, i.e., $O(m+n)$
- In theory: even with anti-cycling may need exponential time



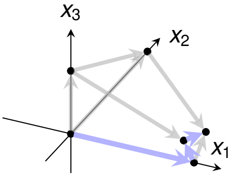
Linear Programming and Simplex: Summary

Linear Programming

- extremely versatile tool for modelling problems of all kinds
- basis of Integer Programming, to be discussed in later lectures

Simplex Algorithm

- In practice: usually terminates in polynomial time, i.e., $O(m+n)$
- In theory: even with anti-cycling may need exponential time



Research Problem: Is there a pivoting rule which makes SIMPLEX a polynomial-time algorithm?

→ different rules lead to different instantiations of SIMPLEX

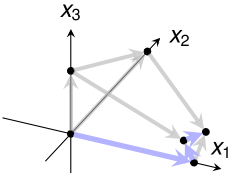
Linear Programming and Simplex: Summary

Linear Programming

- extremely versatile tool for modelling problems of all kinds
- basis of Integer Programming, to be discussed in later lectures

Simplex Algorithm

- In practice: usually terminates in polynomial time, i.e., $O(m+n)$
- In theory: even with anti-cycling may need exponential time



Research Problem: Is there a pivoting rule which makes SIMPLEX a polynomial-time algorithm?

Polynomial-Time Algorithms

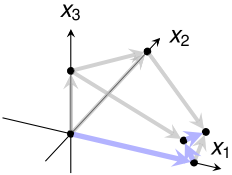
Linear Programming and Simplex: Summary

Linear Programming

- extremely versatile tool for modelling problems of all kinds
- basis of Integer Programming, to be discussed in later lectures

Simplex Algorithm

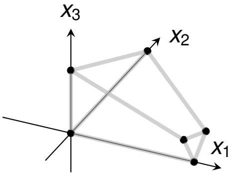
- In practice: usually terminates in polynomial time, i.e., $O(m+n)$
- In theory: even with anti-cycling may need exponential time



Research Problem: Is there a pivoting rule which makes SIMPLEX a polynomial-time algorithm?

Polynomial-Time Algorithms

- Interior-Point Methods: traverses the interior of the feasible set of solutions (not just vertices!)



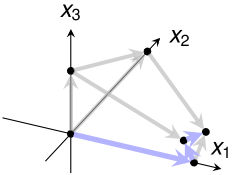
Linear Programming and Simplex: Summary

Linear Programming

- extremely versatile tool for modelling problems of all kinds
- basis of Integer Programming, to be discussed in later lectures

Simplex Algorithm

- In practice: usually terminates in polynomial time, i.e., $O(m+n)$
- In theory: even with anti-cycling may need exponential time

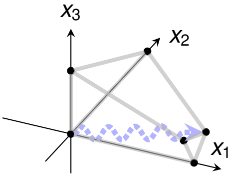


Research Problem: Is there a pivoting rule which makes SIMPLEX a polynomial-time algorithm?

Polynomial-Time Algorithms

- Interior-Point Methods: traverses the interior of the feasible set of solutions (not just vertices!)

more complicated



IV. Approximation Algorithms: Covering Problems

Thomas Sauerwald

Easter 2015

UNIVERSITY OF CAMBRIDGE

Outline

Introduction

Vertex Cover

The Set-Covering Problem

Motivation

Many fundamental problems are **NP-complete**, yet they are too important to be abandoned.

Motivation

Many fundamental problems are **NP-complete**, yet they are too important to be abandoned.

Examples: HAMILTON, 3-SAT, VERTEX-COVER, KNAPSACK,...

Motivation

Many fundamental problems are **NP-complete**, yet they are too important to be abandoned.

Examples: HAMILTON, 3-SAT, VERTEX-COVER, KNAPSACK,...

Strategies to cope with NP-complete problems

1. If inputs (or solutions) are small, an algorithm with exponential running time may be satisfactory.
2. Isolate important special cases which can be solved in polynomial-time.
3. Develop algorithms which find near-optimal solutions in polynomial-time.

Motivation

Many fundamental problems are **NP-complete**, yet they are too important to be abandoned.

Examples: HAMILTON, 3-SAT, VERTEX-COVER, KNAPSACK,...

Strategies to cope with NP-complete problems

1. If inputs (or solutions) are small, an algorithm with exponential running time may be satisfactory.
2. Isolate important special cases which can be solved in polynomial-time.
3. Develop algorithms which find near-optimal solutions in polynomial-time.

Motivation

Many fundamental problems are **NP-complete**, yet they are too important to be abandoned.

Examples: HAMILTON, 3-SAT, VERTEX-COVER, KNAPSACK,. . .

Strategies to cope with NP-complete problems

1. If inputs (or solutions) are small, an algorithm with exponential running time may be satisfactory.
2. Isolate important special cases which can be solved in polynomial-time.
3. Develop algorithms which find near-optimal solutions in polynomial-time.

We will call these **approximation algorithms**.

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

This covers both maximization and minimization problems.

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Maximization problem: $\frac{C^*}{C} \geq 1$

This covers both maximization and minimization problems.

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Maximization problem: $\frac{C^*}{C} \geq 1$
- Minimization problem: $\frac{C}{C^*} \geq 1$

This covers both maximization and minimization problems.

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Maximization problem: $\frac{C^*}{C} \geq 1$
- Minimization problem: $\frac{C}{C^*} \geq 1$

This covers both maximization and minimization problems.

For many problems: tradeoff between runtime and approximation ratio.

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Maximization problem: $\frac{C^*}{C} \geq 1$
- Minimization problem: $\frac{C}{C^*} \geq 1$

This covers both maximization and minimization problems.

For many problems: tradeoff between runtime and approximation ratio.

Approximation Schemes

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Maximization problem: $\frac{C^*}{C} \geq 1$
- Minimization problem: $\frac{C}{C^*} \geq 1$

This covers both maximization and minimization problems.

For many problems: tradeoff between runtime and approximation ratio.

Approximation Schemes

An approximation scheme is an approximation algorithm, which given any input and $\epsilon > 0$, is a $(1 + \epsilon)$-approximation algorithm.

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Maximization problem: $\frac{C^*}{C} \geq 1$
- Minimization problem: $\frac{C}{C^*} \geq 1$

This covers both maximization and minimization problems.

For many problems: tradeoff between runtime and approximation ratio.

Approximation Schemes

An approximation scheme is an approximation algorithm, which given any input and $\epsilon > 0$, is a $(1+\epsilon)$-approximation algorithm.

- It is a polynomial-time approximation scheme (PTAS) if for any fixed $\epsilon > 0$, the runtime is polynomial in n.

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Maximization problem: $\frac{C^*}{C} \geq 1$
- Minimization problem: $\frac{C}{C^*} \geq 1$

This covers both maximization and minimization problems.

For many problems: tradeoff between runtime and approximation ratio.

Approximation Schemes

An approximation scheme is an approximation algorithm, which given any input and $\epsilon > 0$, is a $(1+\epsilon)$-approximation algorithm.

- It is a polynomial-time approximation scheme (PTAS) if for any fixed $\epsilon > 0$, the runtime is polynomial in n. For example, $O(n^{2/\epsilon})$.

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Maximization problem: $\frac{C^*}{C} \geq 1$
- Minimization problem: $\frac{C}{C^*} \geq 1$

This covers both maximization and minimization problems.

For many problems: tradeoff between runtime and approximation ratio.

Approximation Schemes

An approximation scheme is an approximation algorithm, which given any input and $\epsilon > 0$, is a $(1+\epsilon)$-approximation algorithm.

- It is a polynomial-time approximation scheme (PTAS) if for any fixed $\epsilon > 0$, the runtime is polynomial in n. For example, $O(n^{2/\epsilon})$.
- It is a fully polynomial-time approximation scheme (FPTAS) if the runtime is polynomial in both $1/\epsilon$ and n.

Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

- Maximization problem: $\frac{C^*}{C} \geq 1$
- Minimization problem: $\frac{C}{C^*} \geq 1$

This covers both maximization and minimization problems.

For many problems: tradeoff between runtime and approximation ratio.

Approximation Schemes

An approximation scheme is an approximation algorithm, which given any input and $\epsilon > 0$, is a $(1+\epsilon)$-approximation algorithm.

- It is a polynomial-time approximation scheme (PTAS) if for any fixed $\epsilon > 0$, the runtime is polynomial in n. For example, $O(n^{2/\epsilon})$.
- It is a fully polynomial-time approximation scheme (FPTAS) if the runtime is polynomial in both $1/\epsilon$ and n. For example, $O((1/\epsilon)^2 \cdot n^3)$.

Outline

Introduction

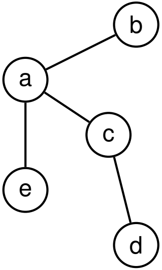
Vertex Cover

The Set-Covering Problem

The Vertex-Cover Problem

Vertex Cover Problem

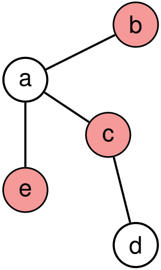
- Given: Undirected graph $G = (V, E)$
- Goal: Find a minimum-cardinality subset $V' \subseteq V$ such that if $(u, v) \in E(G)$, then $u \in V'$ or $v \in V'$.



The Vertex-Cover Problem

Vertex Cover Problem

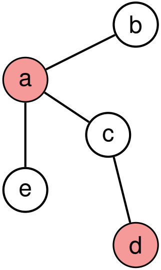
- Given: Undirected graph $G = (V, E)$
- Goal: Find a minimum-cardinality subset $V' \subseteq V$ such that if $(u, v) \in E(G)$, then $u \in V'$ or $v \in V'$.



The Vertex-Cover Problem

Vertex Cover Problem

- Given: Undirected graph $G = (V, E)$
- Goal: Find a minimum-cardinality subset $V' \subseteq V$ such that if $(u, v) \in E(G)$, then $u \in V'$ or $v \in V'$.

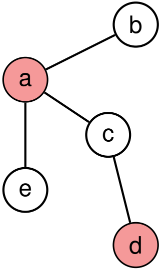


The Vertex-Cover Problem

We are covering edges by picking vertices!

Vertex Cover Problem

- Given: Undirected graph $G = (V, E)$
- Goal: Find a minimum-cardinality subset $V' \subseteq V$ such that if $(u, v) \in E(G)$, then $u \in V'$ or $v \in V'$.



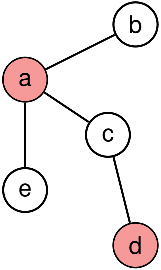
The Vertex-Cover Problem

We are covering edges by picking vertices!

Vertex Cover Problem

- Given: Undirected graph $G = (V, E)$
- Goal: Find a minimum-cardinality subset $V' \subseteq V$ such that if $(u, v) \in E(G)$, then $u \in V'$ or $v \in V'$.

This is an NP-hard problem.



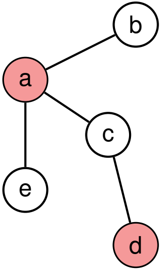
The Vertex-Cover Problem

We are covering edges by picking vertices!

Vertex Cover Problem

- Given: Undirected graph $G = (V, E)$
- Goal: Find a minimum-cardinality subset $V' \subseteq V$ such that if $(u, v) \in E(G)$, then $u \in V'$ or $v \in V'$.

This is an NP-hard problem.



Applications:

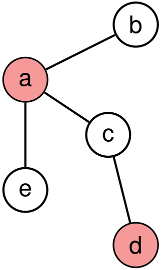
The Vertex-Cover Problem

We are covering edges by picking vertices!

Vertex Cover Problem

- Given: Undirected graph $G = (V, E)$
- Goal: Find a minimum-cardinality subset $V' \subseteq V$ such that if $(u, v) \in E(G)$, then $u \in V'$ or $v \in V'$.

This is an NP-hard problem.



Applications:

- Every edge forms a task, and every vertex represents a person/machine which can execute that task

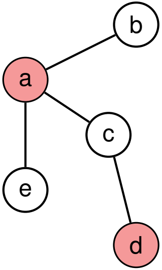
The Vertex-Cover Problem

We are covering edges by picking vertices!

Vertex Cover Problem

- Given: Undirected graph $G = (V, E)$
- Goal: Find a minimum-cardinality subset $V' \subseteq V$ such that if $(u, v) \in E(G)$, then $u \in V'$ or $v \in V'$.

This is an NP-hard problem.



Applications:

- Every edge forms a task, and every vertex represents a person/machine which can execute that task
- Perform all tasks with the minimal amount of resources

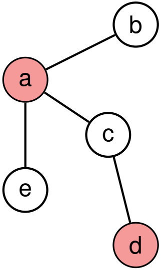
The Vertex-Cover Problem

We are covering edges by picking vertices!

Vertex Cover Problem

- Given: Undirected graph $G = (V, E)$
- Goal: Find a minimum-cardinality subset $V' \subseteq V$ such that if $(u, v) \in E(G)$, then $u \in V'$ or $v \in V'$.

This is an NP-hard problem.



Applications:

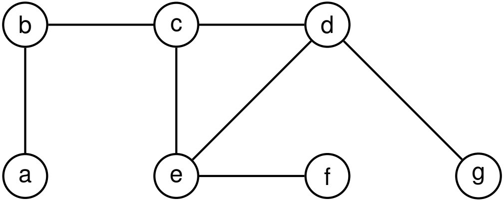
- Every edge forms a task, and every vertex represents a person/machine which can execute that task
- Perform all tasks with the minimal amount of resources
- Extensions: weighted ~~edges~~ vertices or hypergraphs

An Approximation Algorithm based on Greedy

```
APPROX-VERTEX-COVER(G)
C = ∅
E' = G.E
while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'      potentially many choices
    C = C ∪ {u, v}
    remove from E' every edge incident on either u or v
return C
```

An Approximation Algorithm based on Greedy

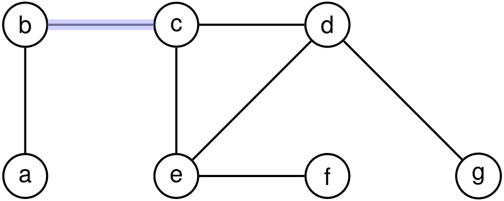
```
APPROX-VERTEX-COVER(G)
C = ∅
E' = G.E
while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'
    C = C ∪ {u, v}
    remove from E' every edge incident on either u or v
return C
```



An Approximation Algorithm based on Greedy

APPROX-VERTEX-COVER(G)

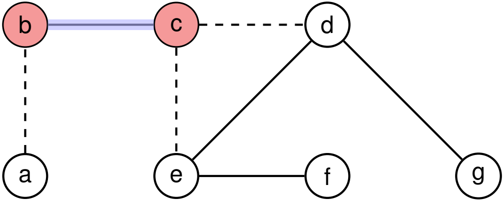
$C = \emptyset$
$E' = G.E$
while $E' \neq \emptyset$
    let (u, v) be an arbitrary edge of E'
    $C = C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
return C



An Approximation Algorithm based on Greedy

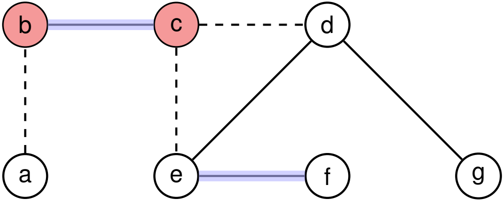
APPROX-VERTEX-COVER(G)

```
C = ∅
E' = G.E
while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'
    C = C ∪ {u, v}
    remove from E' every edge incident on either u or v
return C
```



An Approximation Algorithm based on Greedy

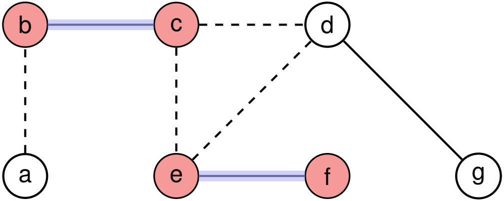
```
APPROX-VERTEX-COVER(G)
C = ∅
E' = G.E
while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'
    C = C ∪ {u, v}
    remove from E' every edge incident on either u or v
return C
```



An Approximation Algorithm based on Greedy

APPROX-VERTEX-COVER(G)

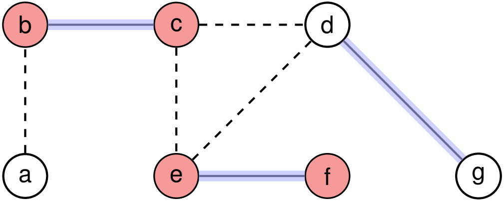
$C = \emptyset$
$E' = G.E$
while $E' \neq \emptyset$
    let (u, v) be an arbitrary edge of E'
    $C = C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
return C



An Approximation Algorithm based on Greedy

APPROX-VERTEX-COVER(G)

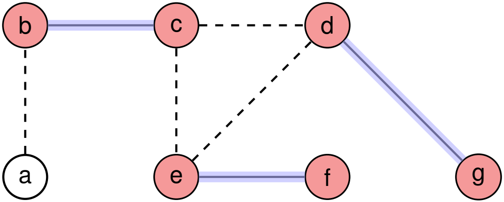
```
C = ∅
E' = G.E
while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'
    C = C ∪ {u, v}
    remove from E' every edge incident on either u or v
return C
```



An Approximation Algorithm based on Greedy

APPROX-VERTEX-COVER(G)

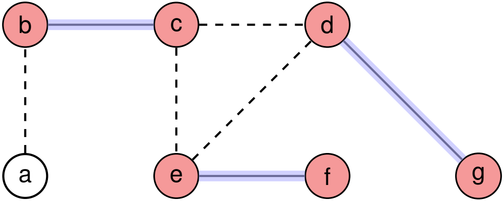
1 $C = \emptyset$
2 $E' = G.E$
3 **while** $E' \neq \emptyset$
4 let (u, v) be an arbitrary edge of E'
5 $C = C \cup \{u, v\}$
6 remove from E' every edge incident on either u or v
7 **return** C



An Approximation Algorithm based on Greedy

APPROX-VERTEX-COVER(G)

$C = \emptyset$
$E' = G.E$
while $E' \neq \emptyset$
    let (u, v) be an arbitrary edge of E'
    $C = C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
return C

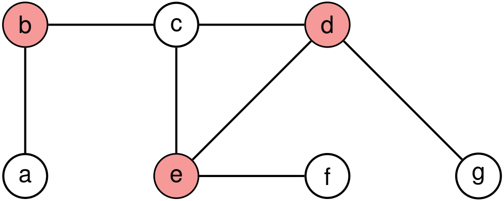


APPROX-VERTEX-COVER produces a set of size 6.

An Approximation Algorithm based on Greedy

APPROX-VERTEX-COVER(G)

$C = \emptyset$
$E' = G.E$
while $E' \neq \emptyset$
    let (u, v) be an arbitrary edge of E'
    $C = C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
return C



The optimal solution has size 3.

Analysis of Greedy for Vertex Cover

```
APPROX-VERTEX-COVER(G)
C = ∅
E' = G.E
while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'
    C = C ∪ {u, v}
    remove from E' every edge incident on either u or v
return C
```

Analysis of Greedy for Vertex Cover

```
APPROX-VERTEX-COVER(G)
C = ∅
E' = G.E
while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'
    C = C ∪ {u, v}
    remove from E' every edge incident on either u or v
return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
C = ∅
E' = G.E
while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'
    C = C ∪ {u, v}
    remove from E' every edge incident on either u or v
return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')
- Let $A \subseteq E$ denote the set of edges picked in line 4

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')
- Let $A \subseteq E$ denote the set of edges picked in line 4
- Every optimal cover C^* must include at least one endpoint of edges in A,

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')
- Let $A \subseteq E$ denote the set of edges picked in line 4
- Every optimal cover C^* must include at least one endpoint of edges in A, and edges in A do not share a common endpoint:

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')
- Let $A \subseteq E$ denote the set of edges picked in line 4
- Every optimal cover C^* must include at least one endpoint of edges in A, and edges in A do not share a common endpoint: $\boxed{|C^*| \geq |A|}$

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')
- Let $A \subseteq E$ denote the set of edges picked in line 4
- Every optimal cover C^* must include at least one endpoint of edges in A, and edges in A do not share a common endpoint: $\boxed{|C^*| \geq |A|}$
- Every edge in A contributes 2 vertices to $|C|$:

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')
- Let $A \subseteq E$ denote the set of edges picked in line 4
- Every optimal cover C^* must include at least one endpoint of edges in A, and edges in A do not share a common endpoint: $\boxed{|C^*| \geq |A|}$
- Every edge in A contributes 2 vertices to $|C|$: $\boxed{|C| = 2|A|}$

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')
- Let $A \subseteq E$ denote the set of edges picked in line 4
- Every optimal cover C^* must include at least one endpoint of edges in A, and edges in A do not share a common endpoint: $\boxed{|C^*| \geq |A|}$
- Every edge in A contributes 2 vertices to $|C|$: $\boxed{|C| = 2|A|}$

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')
- Let $A \subseteq E$ denote the set of edges picked in line 4
- Every optimal cover C^* must include at least one endpoint of edges in A, and edges in A do not share a common endpoint: $\boxed{|C^*| \geq |A|}$
- Every edge in A contributes 2 vertices to $|C|$: $\boxed{|C| = 2|A| \leq 2|C^*|.}$ □

Analysis of Greedy for Vertex Cover

APPROX-VERTEX-COVER(G)

```
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

We can bound the size of the returned solution without knowing the (size of an) optimal solution!

Theorem 35.1

APPROX-VERTEX-COVER is a poly-time 2-approximation algorithm.

Proof:

- Running time is $O(V + E)$ (using adjaency lists to represent E')
- Let $A \subseteq E$ denote the set of edges picked in line 4
- Every optimal cover C^* must include at least one endpoint of edges in A, and edges in A do not share a common endpoint: $\boxed{|C^*| \geq |A|}$
- Every edge in A contributes 2 vertices to $|C|$: $\boxed{|C| = 2|A| \leq 2|C^*|.}$ $\square$