Theory of Algorithms III

Linear Programming

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Linear Programming

Linear Programming

A linear programming problem gives a set of variables,

Linear Programming

A linear programming problem gives a set of variables, and assigns real values to them so as to

Linear Programming

A linear programming problem gives a set of variables, and assigns real values to them so as to

1. satisfy a set of linear equations and/or linear inequalities involving these variables, and
2. maximize or minimize a given linear objective function.

Example: Profit Maximization

A boutique chocolatier has two products:

Example: Profit Maximization

A boutique chocolatier has two products:

- triangular chocolates, called Pyramide,
- and the more decadent and deluxe Pyramide Nuit.

Example: Profit Maximization

A boutique chocolatier has two products:

- triangular chocolates, called Pyramide,
- and the more decadent and deluxe Pyramide Nuit.

Q: How much of each should it produce to maximize profits?

Example: Profit Maximization

A boutique chocolatier has two products:

- triangular chocolates, called Pyramide,
- and the more decadent and deluxe Pyramide Nuit.

Q: How much of each should it produce to maximize profits?

- Every box of Pyramide has a a profit of \$1.
- Every box of Nuit has a profit of \$6.
- The daily demand is limited to at most 200 boxes of Pyramide and 300 boxes of Nuit.
- The current workforce can produce a total of at most 400 boxes of chocolate per day.

LP Formulation

`Objective function`	$\max x_1 + 6x_2$
`Constraints`	$x_1 \leq 200$
	$x_2 \leq 300$
	$x_1 + x_2 \leq 400$
	$x_1, x_2 \geq 0$

LP Formulation

`Objective function`	$\max x_1 + 6x_2$
`Constraints`	$x_1 \leq 200$
	$x_2 \leq 300$
	$x_1 + x_2 \leq 400$
	$x_1, x_2 \geq 0$

A linear equation in x_1 and x_2 defines a line in the two-dimensional (2D) plane, and a linear inequality designates a half-space, the region on one side of the line.

LP Formulation

Objective function	$\max x_1 + 6x_2$
Constraints	$x_1 \leq 200$
	$x_2 \leq 300$
	$x_1 + x_2 \leq 400$
	$x_1, x_2 \geq 0$

A linear equation in x_1 and x_2 defines a line in the two-dimensional (2D) plane, and a linear inequality designates a half-space, the region on one side of the line.

The set of all feasible solutions of this linear program is the intersection of five half-spaces.

LP Formulation

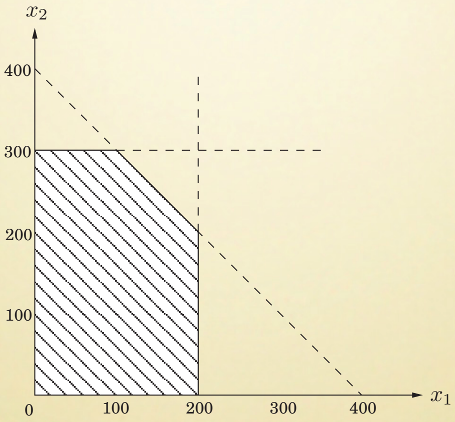
`Objective function`	$\max x_1 + 6x_2$
`Constraints`	$x_1 \leq 200$
	$x_2 \leq 300$
	$x_1 + x_2 \leq 400$
	$x_1, x_2 \geq 0$

A linear equation in x_1 and x_2 defines a line in the two-dimensional (2D) plane, and a linear inequality designates a half-space, the region on one side of the line.

The set of all feasible solutions of this linear program is the intersection of five half-spaces.

It is a convex polygon.

The Convex Polygon



The Optimal Solution

We want to find the point in this polygon at which the objective function is maximized.

The Optimal Solution

We want to find the point in this polygon at which the objective function is maximized.

The points with a profit of c dollars lie on the line $x_1 + 6x_2 = c$, which has a slope of $-1/6$.

The Optimal Solution

We want to find the point in this polygon at which the objective function is maximized.

The points with a profit of c dollars lie on the line $x_1 + 6x_2 = c$, which has a slope of $-1/6$.

As c increases, this "profit line" moves parallel to itself, up and to the right.

The Optimal Solution

We want to find the point in this polygon at which the objective function is maximized.

The points with a profit of c dollars lie on the line $x_1 + 6x_2 = c$, which has a slope of $-1/6$.

As c increases, this "profit line" moves parallel to itself, up and to the right.

Since the goal is to maximize c, we must move the line as far up as possible, while still touching the feasible region.

The Optimal Solution

We want to find the point in this polygon at which the objective function is maximized.

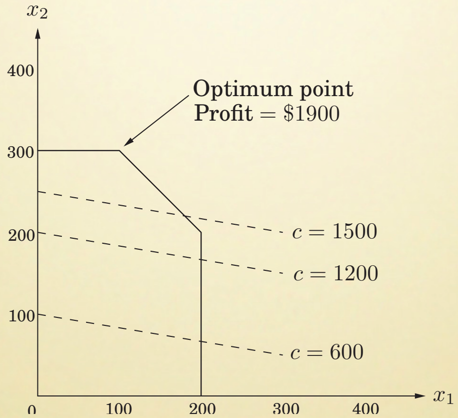
The points with a profit of c dollars lie on the line $x_1 + 6x_2 = c$, which has a slope of $-1/6$.

As c increases, this "profit line" moves parallel to itself, up and to the right.

Since the goal is to maximize c, we must move the line as far up as possible, while still touching the feasible region.

The optimum solution will be the very last feasible point that the profit line sees and must therefore be a vertex of the polygon.

The Convex Polygon



The Optimal Solution

It is a general rule of linear programs that the optimum is achieved at a vertex of the feasible region.

The Optimal Solution

It is a general rule of linear programs that the optimum is achieved at a vertex of the feasible region.

The only exceptions are cases in which there is no optimum; this can happen in two ways:

The Optimal Solution

It is a general rule of linear programs that the optimum is achieved at a vertex of the feasible region.

The only exceptions are cases in which there is no optimum; this can happen in two ways:

1. The linear program is infeasible; that is, the constraints are so tight that it is impossible to satisfy all of them.

The Optimal Solution

It is a general rule of linear programs that the optimum is achieved at a vertex of the feasible region.

The only exceptions are cases in which there is no optimum; this can happen in two ways:

1. The linear program is infeasible; that is, the constraints are so tight that it is impossible to satisfy all of them.
   - For instance, $x \leq 1$, $x \geq 2$.

The Optimal Solution

It is a general rule of linear programs that the optimum is achieved at a vertex of the feasible region.

The only exceptions are cases in which there is no optimum; this can happen in two ways:

1. The linear program is infeasible; that is, the constraints are so tight that it is impossible to satisfy all of them.
   - For instance, $x \leq 1$, $x \geq 2$.
2. The constraints are so loose that the feasible region is unbounded, and it is possible to achieve arbitrarily high objective values.

The Optimal Solution

It is a general rule of linear programs that the optimum is achieved at a vertex of the feasible region.

The only exceptions are cases in which there is no optimum; this can happen in two ways:

1. The linear program is infeasible; that is, the constraints are so tight that it is impossible to satisfy all of them.
   - For instance, $x \leq 1, x \geq 2$.
2. The constraints are so loose that the feasible region is unbounded, and it is possible to achieve arbitrarily high objective values.
   - For instance, $\max x_1 + x_2$
   - $x_1, x_2 \geq 0$

Solving Linear Programs

Linear programs (LPs) can be solved by the simplex method, devised by George Dantzig in 1947.

Solving Linear Programs

Linear programs (LPs) can be solved by the simplex method, devised by George Dantzig in 1947.

This algorithm starts at a vertex,

Solving Linear Programs

Linear programs (LPs) can be solved by the simplex method, devised by George Dantzig in 1947.

This algorithm starts at a vertex, and repeatedly looks for an adjacent vertex of better objective value.

Solving Linear Programs

Linear programs (LPs) can be solved by the simplex method, devised by George Dantzig in 1947.

This algorithm starts at a vertex, and repeatedly looks for an adjacent vertex of better objective value.

It does hill-climbing on the vertices of the polygon,

Solving Linear Programs

Linear programs (LPs) can be solved by the simplex method, devised by George Dantzig in 1947.

This algorithm starts at a vertex, and repeatedly looks for an adjacent vertex of better objective value.

It does hill-climbing on the vertices of the polygon, walking from neighbor to neighbor so as to steadily increase profit along the way.

Solving Linear Programs

Linear programs (LPs) can be solved by the simplex method, devised by George Dantzig in 1947.

This algorithm starts at a vertex, and repeatedly looks for an adjacent vertex of better objective value.

It does hill-climbing on the vertices of the polygon, walking from neighbor to neighbor so as to steadily increase profit along the way.

Upon reaching a vertex that has no better neighbor, simplex declares it to be optimal and halts.

Solving Linear Programs

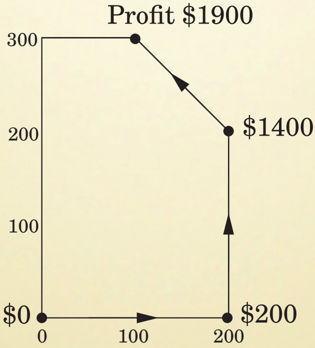
Q: Why does this local test imply global optimality?

Solving Linear Programs

Q: Why does this local test imply global optimality?

By simple geometry. Since all the vertex's neighbors lie below the line, the rest of the feasible polygon must also lie below this line.

The Example



A Notable Result

Smoothed analysis proposed by Daniel Spielman and Shanghua Teng is a way of measuring the complexity of an algorithm. It gives a more realistic analysis of the practical performance of the algorithm. It was used to explain that the simplex algorithm runs in exponential-time in the worst-case and yet in practice it is a very efficient algorithm, which was one of the main motivations for developing smoothed analysis. The authors received the 2008 Gödel Prize and the 2009 Fulkerson Prize.

More Products

The chocolatier introduces a third and even more exclusive chocolates, called Pyramide Luxe.

More Products

The chocolatier introduces a third and even more exclusive chocolates, called Pyramide Luxe. One box of these will bring in a profit of \$13.

More Products

The chocolatier introduces a third and even more exclusive chocolates, called Pyramide Luxe. One box of these will bring in a profit of \$13.

Let x_1, x_2, x_3 denote the number of boxes of each chocolate produced daily, with x_3 referring to Luxe.

More Products

The chocolatier introduces a third and even more exclusive chocolates, called Pyramide Luxe. One box of these will bring in a profit of \$13.

Let x_1, x_2, x_3 denote the number of boxes of each chocolate produced daily, with x_3 referring to Luxe.

The old constraints on x_1 and x_2 persist. The labor restriction now extends to x_3 as well: the sum of all three variables is at most 400.

More Products

The chocolatier introduces a third and even more exclusive chocolates, called Pyramide Luxe. One box of these will bring in a profit of $13.

Let x_1, x_2, x_3 denote the number of boxes of each chocolate produced daily, with x_3 referring to Luxe.

The old constraints on x_1 and x_2 persist. The labor restriction now extends to x_3 as well: the sum of all three variables is at most 400.

Nuit and Luxe require the same packaging machinery. Luxe uses it three times as much, which imposes another constraint
$x_2 + 3x_3 \leq 600$.

LP

$$\max x_1 + 6x_2 + 13x_3$$
$$x_1 \leq 200$$
$$x_2 \leq 300$$
$$x_1 + x_2 + x_3 \leq 400$$
$$x_2 + 3x_3 \leq 600$$
$$x_1, x_2, x_3 \geq 0$$

LP

The space of solutions is now three-dimensional.

LP

The space of solutions is now three-dimensional.

Each linear equation defines a 3D plane, and each inequality a half-space on one side of the plane.

LP

The space of solutions is now three-dimensional.

Each linear equation defines a 3D plane, and each inequality a half-space on one side of the plane.

The feasible region is an intersection of seven half-spaces, a polyhedron.

LP

The space of solutions is now three-dimensional.

Each linear equation defines a 3D plane, and each inequality a half-space on one side of the plane.

The feasible region is an intersection of seven half-spaces, a polyhedron.

A profit of c corresponds to the plane $x_1 + 6x_2 + 13x_3 = c$.

LP

The space of solutions is now three-dimensional.

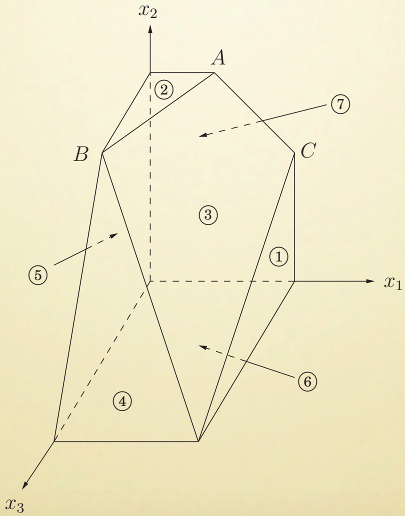
Each linear equation defines a 3D plane, and each inequality a half-space on one side of the plane.

The feasible region is an intersection of seven half-spaces, a polyhedron.

A profit of c corresponds to the plane $x_1 + 6x_2 + 13x_3 = c$.

As c increases, this profit-plane moves parallel to itself, further into the positive orthant until it no longer touches the feasible region.

The Example



LP

The point of final contact is the optimal vertex: $(0, 300, 100)$, with total profit $\$3100$.

LP

The point of final contact is the optimal vertex: $(0, 300, 100)$, with total profit $\$3100$.

Q: How would the simplex algorithm behave on this modified problem?

LP

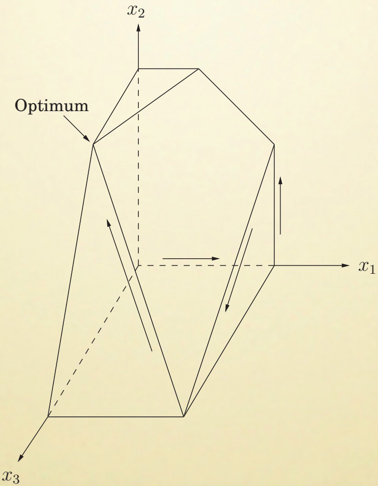
The point of final contact is the optimal vertex: $(0, 300, 100)$, with total profit $\$3100$.

Q: How would the simplex algorithm behave on this modified problem?

A possible trajectory

$$\frac{(0,0,0)}{\$0} \to \frac{(200,0,0)}{\$200} \to \frac{(200,200,0)}{\$1400} \to \frac{(200,0,200)}{\$2800} \to \frac{(0,300,100)}{\$3100}$$

The Example



ILP and Rounding

Example: Production Planning

The company makes handwoven carpets, a product for which the demand is extremely seasonal.

Example: Production Planning

The company makes handwoven carpets, a product for which the demand is extremely seasonal.

Our analyst has just obtained demand estimates for all months of the next calendar year: $d_1, d_2, \ldots, d_{12}$, ranging from 440 to 920.

Example: Production Planning

The company makes handwoven carpets, a product for which the demand is extremely seasonal.

Our analyst has just obtained demand estimates for all months of the next calendar year: $d_1, d_2, \ldots, d_{12}$, ranging from 440 to 920.

Currently with 30 employees, each of whom makes 20 carpets per month and gets a monthly salary of \$2000.

Example: Production Planning

The company makes handwoven carpets, a product for which the demand is extremely seasonal.

Our analyst has just obtained demand estimates for all months of the next calendar year: $d_1, d_2, \ldots, d_{12}$, ranging from 440 to 920.

Currently with 30 employees, each of whom makes 20 carpets per month and gets a monthly salary of \$2000.

With no initial surplus of carpets.

Example: Production Planning

Q: How can we handle the fluctuations in demand? There are three ways:

Example: Production Planning

Q: How can we handle the fluctuations in demand? There are three ways:

1. Overtime. Overtime pay is 80% more than regular pay. Workers can put in at most 30% overtime.

Example: Production Planning

Q: How can we handle the fluctuations in demand? There are three ways:

1. Overtime. Overtime pay is 80% more than regular pay. Workers can put in at most 30% overtime.
2. Hiring and firing, costing \$320 and \$400, respectively, per worker.

Example: Production Planning

Q: How can we handle the fluctuations in demand? There are three ways:

1. Overtime. Overtime pay is 80% more than regular pay. Workers can put in at most 30% overtime.
2. Hiring and firing, costing \$320 and \$400, respectively, per worker.
3. Storing surplus production, costing \$8 per carpet per month. Currently without stored carpets on hand, and without any carpets stored at the end of year.

LP Formulations

$$\begin{aligned}
w_i &= \text{number of workers during } i\text{-th month; } w_0 = 30. \\
x_i &= \text{number of carpets made during } i\text{-th month.} \\
o_i &= \text{number of carpets made by overtime in month } i. \\
h_i, f_i &= \text{number of workers hired and fired, respectively,} \\
&\quad\ \text{at beginning of month } i. \\
s_i &= \text{number of carpets stored at end of month } i\text{; } s_0 = 0.
\end{aligned}$$

LP Formulation

LP Formulation

All variables must be nonnegative:

$$w_i, x_i, o_i, h_i, f_i, s_i \geq 0, i = 1, \ldots, 12$$

LP Formulation

All variables must be nonnegative:

$$w_i, x_i, o_i, h_i, f_i, s_i \geq 0, i = 1, \ldots, 12$$

The total number of carpets made per month consists of regular production plus overtime:

$$x_i = 20w_i + o_i$$

$i = 1, \ldots, 12.$

LP Formulation

All variables must be nonnegative:

$$w_i, x_i, o_i, h_i, f_i, s_i \geq 0, i = 1, \ldots, 12$$

The total number of carpets made per month consists of regular production plus overtime:

$$x_i = 20w_i + o_i$$

$i = 1, \ldots, 12$.

The number of workers can potentially change at the start of each month:

$$w_i = w_{i-1} + h_i - f_i$$

LP Formulation

LP Formulation

The number of carpets stored at the end of each month is what we started with, plus the number we made, minus the demand for the month:

$$s_i = s_{i-1} + x_i - d_i$$

LP Formulation

The number of carpets stored at the end of each month is what we started with, plus the number we made, minus the demand for the month:

$$s_i = s_{i-1} + x_i - d_i$$

And overtime is limited:

$$o_i \leq 6w_i$$

LP Formulation

The objective function is to minimize the total cost:

$$\min 2000 \sum_i w_i + 320 \sum_i h_i + 400 \sum_i f_i + 8 \sum_i s_i + 180 \sum_i o_i$$

Integer Linear Programming

The optimum solution might turn out to be fractional;

Integer Linear Programming

The optimum solution might turn out to be fractional; for instance, it might involve hiring 10.6 workers in the month of March.

Integer Linear Programming

The optimum solution might turn out to be fractional; for instance, it might involve hiring 10.6 workers in the month of March.

This number would have to be rounded to either 10 or 11 in order to make sense, and the overall cost would then increase correspondingly.

Integer Linear Programming

The optimum solution might turn out to be fractional; for instance, it might involve hiring 10.6 workers in the month of March.

This number would have to be rounded to either 10 or 11 in order to make sense, and the overall cost would then increase correspondingly.

In the example, most of the variables take on fairly large values, and thus rounding is unlikely to affect things too much.

Integer Linear Programming

There are other LPs, in which rounding decisions have to be made very carefully to end up with an integer solution of reasonable quality.

Integer Linear Programming

There are other LPs, in which rounding decisions have to be made very carefully to end up with an integer solution of reasonable quality.

There is a tension in linear programming between the ease of obtaining fractional solutions and the desirability of integer ones.

Integer Linear Programming

There are other LPs, in which rounding decisions have to be made very carefully to end up with an integer solution of reasonable quality.

There is a tension in linear programming between the ease of obtaining fractional solutions and the desirability of integer ones.

In NP problems, finding the optimum integer solution of an LP is an important but very hard problem, called integer linear programming.

Quiz

In the shortest path problem, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights, a source vertex s, and destination vertex t. We wish to compute the weight of a shortest path from s to t.

Shortest Path in LP

$$\max d_t$$

$$\begin{aligned} d_v \leq d_u + w(u,v) &\quad (u,v) \in E \\ d_s = 0 & \\ d_i \geq 0 &\quad i \in V \end{aligned}$$

Shortest Path in LP

$$\max d_t$$

$$\begin{aligned} d_v &\leq d_u + w(u,v) & (u,v) \in E \\ d_s &= 0 & \\ d_i &\geq 0 & i \in V \end{aligned}$$

Q: Another formalization?

Shortest Path in LP

Shortest Path in LP

Let $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$; that is, $\mathcal{S}$ is the set of all s-t cuts in the graph. Then we can model the shortest s-t path problem with the following integer program,

$$\min \sum_{e \in E} w_e x_e$$

$$\begin{aligned} \sum_{e \in \delta(S)} x_e \geq 1 \quad & S \in \mathcal{S} \\ x_e \in \{0, 1\} \quad & e \in E \end{aligned}$$

where $\delta(S)$ is the set of all edges that have one endpoint in S and the other endpoint not in S.

Shortest Path in LP

Let $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$; that is, $\mathcal{S}$ is the set of all s-t cuts in the graph. Then we can model the shortest s-t path problem with the following integer program,

$$\min \sum_{e \in E} w_e x_e$$

$$\begin{aligned} \sum_{e \in \delta(S)} x_e \geq 1 \quad & S \in \mathcal{S} \\ x_e \in \{0, 1\} \quad & e \in E \end{aligned}$$

where $\delta(S)$ is the set of all edges that have one endpoint in S and the other endpoint not in S.

- Can we relax the restriction $x_e \in \{0, 1\}$ to $0 \leq x_e \leq 1$?

Shortest Path in LP

Let $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$; that is, $\mathcal{S}$ is the set of all s-t cuts in the graph. Then we can model the shortest s-t path problem with the following integer program,

$$\min \sum_{e \in E} w_e x_e$$

$$\begin{aligned} \sum_{e \in \delta(S)} x_e \geq 1 \quad & S \in \mathcal{S} \\ x_e \in \{0, 1\} \quad & e \in E \end{aligned}$$

where $\delta(S)$ is the set of all edges that have one endpoint in S and the other endpoint not in S.

- Can we relax the restriction $x_e \in \{0, 1\}$ to $0 \leq x_e \leq 1$?
- How about $x_e \geq 0$?

Duality

Product Planning Revisit

Recall:

$$\max x_1 + 6x_2$$
$$x_1 \le 200$$
$$x_2 \le 300$$
$$x_1 + x_2 \le 400$$
$$x_1, x_2, x_3 \ge 0$$

Product Planning Revisit

Recall:

$$\max x_1 + 6x_2$$
$$x_1 \leq 200$$
$$x_2 \leq 300$$
$$x_1 + x_2 \leq 400$$
$$x_1, x_2, x_3 \geq 0$$

Simplex declares the optimum solution to be $(x_1, x_2) = (100, 300)$, with objective value 1900.

Can this answer be checked somehow?

Product Planning Revisit

Recall:

$$\max x_1 + 6x_2$$
$$x_1 \leq 200$$
$$x_2 \leq 300$$
$$x_1 + x_2 \leq 400$$
$$x_1, x_2, x_3 \geq 0$$

Simplex declares the optimum solution to be $(x_1, x_2) = (100, 300)$, with objective value 1900.

Can this answer be checked somehow?

We take the first inequality and add it to six times the second inequality:

$$x_1 + 6x_2 \leq 2000$$

Product Planning Revisit

Multiplying the three inequalities by 0, 5, and 1, respectively, and adding them up yields

$$x_1 + 6x_2 \leq 1900$$

Multipliers

Let's investigate the issue by describing what we expect of these three multipliers, call them y_1, y_2, y_3.

Multiplier	Inequality
y_1	$x_1 \leq 200$
y_2	$x_2 \leq 300$
y_3	$x_1 + x_2 \leq 400$

Multipliers

Let's investigate the issue by describing what we expect of these three multipliers, call them y_1, y_2, y_3.

$$
\begin{array}{ccccccc}
\text{Multiplier} & & & \text{Inequality} & & & \\
y_1 & & x_1 & & & \le & 200 \\
y_2 & & & & x_2 & \le & 300 \\
y_3 & & x_1 & + & x_2 & \le & 400
\end{array}
$$

These y_i's must be nonnegative,

Multipliers

Let's investigate the issue by describing what we expect of these three multipliers, call them y_1, y_2, y_3.

$$\begin{array}{ccrcrcr}
\text{Multiplier} & & & \text{Inequality} & & & \\
y_1 & & x_1 & & & \leq & 200 \\
y_2 & & & & x_2 & \leq & 300 \\
y_3 & & x_1 & + & x_2 & \leq & 400
\end{array}$$

These y_i's must be nonnegative, otherwise they are unqualified to multiply inequalities.

Multipliers

Let's investigate the issue by describing what we expect of these three multipliers, call them y_1, y_2, y_3.

Multiplier	Inequality
y_1	$x_1 \leq 200$
y_2	$x_2 \leq 300$
y_3	$x_1 + x_2 \leq 400$

These y_i's must be nonnegative, otherwise they are unqualified to multiply inequalities.

After the multiplication and addition steps, we get the bound:

$$(y_1 + y_3)x_1 + (y_2 + y_3)x_2 \leq 200y_1 + 300y_2 + 400y_3$$

Multipliers

Let's investigate the issue by describing what we expect of these three multipliers, call them y_1, y_2, y_3.

Multiplier					
		Inequality			
y_1	x_1			$\leq$	200
y_2			x_2	$\leq$	300
y_3	x_1	$+$	x_2	$\leq$	400

These y_i's must be nonnegative, otherwise they are unqualified to multiply inequalities.

After the multiplication and addition steps, we get the bound:

$$(y_1 + y_3)x_1 + (y_2 + y_3)x_2 \leq 200y_1 + 300y_2 + 400y_3$$

We want the left-hand side to look like the objective function $x_1 + 6x_2$ so that the right-hand side is an upper bound on the optimum solution.

Multipliers

$$x_1 + 6x_2 \leq 200y_1 + 300y_2 + 400y_3$$

if

$$\begin{gathered} y_1, y_2, y_3 \geq 0 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \end{gathered}$$

The Dual Program

The Dual Program

We can easily find y's that satisfy the inequalities on the right by simply making them large enough, for example $(y_1, y_2, y_3) = (5, 3, 6)$.

The Dual Program

We can easily find y's that satisfy the inequalities on the right by simply making them large enough, for example $(y_1, y_2, y_3) = (5, 3, 6)$.

These particular multipliers tell us that the optimum solution of the LP is at most

$$200 \cdot 5 + 300 \cdot 3 + 400 \cdot 6 = 4300$$

The Dual Program

We can easily find y's that satisfy the inequalities on the right by simply making them large enough, for example $(y_1, y_2, y_3) = (5, 3, 6)$.

These particular multipliers tell us that the optimum solution of the LP is at most

$$200 \cdot 5 + 300 \cdot 3 + 400 \cdot 6 = 4300$$

What we want is a bound as tight as possible, so we minimize

$$200y_1 + 300y_2 + 400y_3$$

subject to the preceding inequalities. This is a new linear program!

The Dual Program

$$\begin{aligned} \min 200y_1 + 300y_2 + 400y_3 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \\ y_1, y_2, y_3 \geq 0 \end{aligned}$$

The Dual Program

$$\begin{aligned} \min 200y_1 + 300y_2 + 400y_3 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \\ y_1, y_2, y_3 \geq 0 \end{aligned}$$

Any feasible value of this dual LP is an upper bound on the original primal LP.

The Dual Program

$$\begin{aligned} \min 200y_1 + 300y_2 + 400y_3 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \\ y_1, y_2, y_3 \geq 0 \end{aligned}$$

Any feasible value of this dual LP is an upper bound on the original primal LP.

If we find a pair of primal and dual feasible values that are equal, then they must both be optimal.

The Dual Program

$$
\begin{aligned}
\min 200y_1 + 300y_2 + 400y_3 \\
y_1 + y_3 \geq 1 \\
y_2 + y_3 \geq 6 \\
y_1, y_2, y_3 \geq 0
\end{aligned}
$$

Any feasible value of this dual LP is an upper bound on the original primal LP.

If we find a pair of primal and dual feasible values that are equal, then they must both be optimal.

Here is just such a pair:

- Primal: $(x_1, x_2) = (100, 300)$;
- Dual: $(y_1, y_2, y_3) = (0, 5, 1)$.

The Dual Program

$$\begin{aligned} \min 200y_1 + 300y_2 + 400y_3 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \\ y_1, y_2, y_3 \geq 0 \end{aligned}$$

Any feasible value of this dual LP is an upper bound on the original primal LP.

If we find a pair of primal and dual feasible values that are equal, then they must both be optimal.

Here is just such a pair:

- Primal: $(x_1, x_2) = (100, 300)$;
- Dual: $(y_1, y_2, y_3) = (0, 5, 1)$.

They both have value 1900 and certify each other's optimality.

Matrix-Vector Form and Its Dual

Primal LP	Dual LP
$\max c^T \mathbf{x}$	$\min \mathbf{y}^T b$
$A\mathbf{x} \leq b$	$\mathbf{y}^T A \geq c^T$
$\mathbf{x} \geq 0$	$\mathbf{y} \geq 0$

Primal LP:

$$\begin{aligned} \max \ c_1x_1 + \cdots + c_nx_n & \\ a_{i1}x_1 + \cdots + a_{in}x_n \leq b_i & \quad \text{for } i \in I \\ a_{i1}x_1 + \cdots + a_{in}x_n = b_i & \quad \text{for } i \in E \\ x_j \geq 0 & \quad \text{for } j \in N \end{aligned}$$

Dual LP:

$$\begin{aligned} \min \ b_1y_1 + \cdots + b_my_m & \\ a_{1j}y_1 + \cdots + a_{mj}y_m \geq c_j & \quad \text{for } j \in N \\ a_{1j}y_1 + \cdots + a_{mj}y_m = c_j & \quad \text{for } j \notin N \\ y_i \geq 0 & \quad \text{for } i \in I \end{aligned}$$

Matrix-Vector Form and Its Dual

$$\begin{aligned} \max x_1 + 6x_2 \\ x_1 \leq 200 \\ x_2 \leq 300 \\ x_1 + x_2 \leq 400 \\ x_1, x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} \min 200y_1 + 300y_2 + 400y_3 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \\ y_1, y_2, y_3 \geq 0 \end{aligned}$$

Matrix-Vector Form and Its Dual

Theorem (Duality)

If a linear program has a bounded optimum, then so does its dual, and the two optimum values coincide.

Shortest Path in LP

In the shortest path problem, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \to R$ mapping edges to real-valued weights, a source vertex s, and destination vertex t. We wish to compute the weight of a shortest path from s to t.

Shortest Path in LP

$$\min \sum_{e \in E} w_e x_e$$

$$\begin{aligned} \sum_{e \in \delta(S)} x_e &\geq 1 \quad && S \in \mathcal{S} \\ x_e &\geq 0 && e \in E \end{aligned}$$

Shortest Path in LP

$$\min \sum_{e \in E} w_e x_e$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \quad S \in \mathcal{S}$$

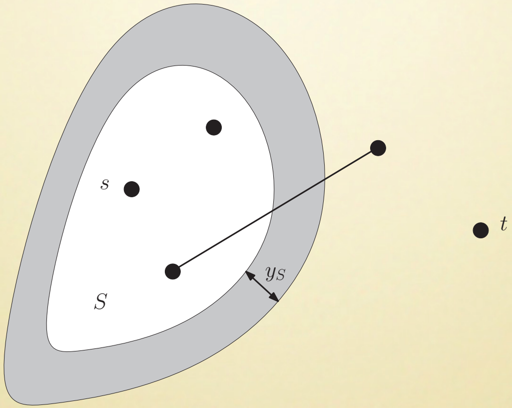
$$x_e \geq 0 \quad e \in E$$

$$\max \sum_{S \in \mathcal{S}} y_S$$

$$\sum_{S \in \mathcal{S}, e \in \delta(S)} y_S \leq w_e \quad e \in E$$

$$y_S \geq 0 \quad S \in \mathcal{S}$$

The Moat



Complementary Slackness

The number of variables in the dual is equal to that of constraints in the primal and the number of constraints in the dual is equal to that of variables in the primal.

Complementary Slackness

The number of variables in the dual is equal to that of constraints in the primal and the number of constraints in the dual is equal to that of variables in the primal.

An inequality constraint has slack if the slack variable is positive.

Complementary Slackness

The number of variables in the dual is equal to that of constraints in the primal and the number of constraints in the dual is equal to that of variables in the primal.

An inequality constraint has slack if the slack variable is positive.

The complementary slackness refers to a relationship between the slackness in a primal constraint and the associated dual variable.

LP and Its Dual

$$\begin{aligned} \max x_1 + 6x_2 \\ x_1 \le 200 \\ x_2 \le 300 \\ x_1 + x_2 \le 400 \\ x_1, x_2 \ge 0 \end{aligned}$$

$$\begin{aligned} \min 200y_1 + 300y_2 + 400y_3 \\ y_1 + y_3 \ge 1 \\ y_2 + y_3 \ge 6 \\ y_1, y_2, y_3 \ge 0 \end{aligned}$$

$$x_1 = 100, x_2 = 300$$

$$y_1 = 0, y_2 = 5, y_3 = 1$$

Complementary Slackness

Theorem

Assume LP problem (P) has a solution x^* and its dual problem (D) has a solution y^*.

1. *If $x_j^* > 0$, then the j-th constraint in (D) is binding.*
2. *If the j-th constraint in (D) is not binding, then $x_j^* = 0$.*
3. *If $y_i^* > 0$, then the i-th constraint in (P) is binding.*
4. *If the i-th constraint in (P) is not binding, then $y_i^* = 0$.*

Complementary Slackness

Theorem

Assume LP problem (P) has a solution x^* and its dual problem (D) has a solution y^*.

1. *If $x_j^* > 0$, then the j-th constraint in (D) is binding.*
2. *If the j-th constraint in (D) is not binding, then $x_j^* = 0$.*
3. *If $y_i^* > 0$, then the i-th constraint in (P) is binding.*
4. *If the i-th constraint in (P) is not binding, then $y_i^* = 0$.*

Proof.

Assignment ! $\square$

Flows in Networks

Shipping Oil

We have a network of pipelines along which oil can be sent.

Shipping Oil

We have a network of pipelines along which oil can be sent.

The goal is to ship as much oil as possible from the source s and the sink t.

Shipping Oil

We have a network of pipelines along which oil can be sent.

The goal is to ship as much oil as possible from the source s and the sink t.

Each pipeline has a maximum capacity it can handle,

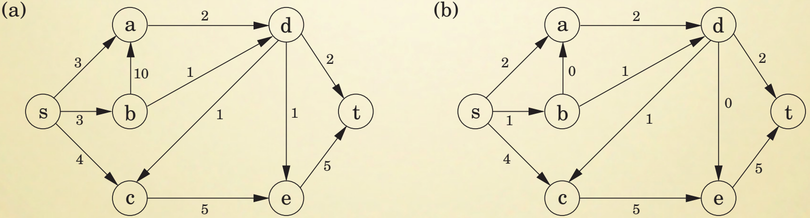
Shipping Oil

We have a network of pipelines along which oil can be sent.

The goal is to ship as much oil as possible from the source s and the sink t.

Each pipeline has a maximum capacity it can handle, and there are no opportunities for storing oil en route.

A Flow Example



Maximizing Flow

The networks consist of a directed graph $G = (V, E)$;

Maximizing Flow

The networks consist of a directed graph $G = (V, E)$; two special nodes $s, t \in V$, which are, respectively, a source and sink of G;

Maximizing Flow

The networks consist of a directed graph $G = (V, E)$; two special nodes $s, t \in V$, which are, respectively, a source and sink of G; and capacities $c_e > 0$ on the edges.

Maximizing Flow

The networks consist of a directed graph $G = (V, E)$; two special nodes $s, t \in V$, which are, respectively, a source and sink of G; and capacities $c_e > 0$ on the edges.

We would like to send as much oil as possible from s to t without exceeding the capacities of any of the edges.

Maximizing Flow

A particular shipping scheme is called a flow and consists of a variable f_e for each edge e of the network, satisfying the following two properties:

Maximizing Flow

A particular shipping scheme is called a flow and consists of a variable f_e for each edge e of the network, satisfying the following two properties:

1. It doesn't violate edge capacities: $0 \leq f_e \leq c_e$ for all $e \in E$.

Maximizing Flow

A particular shipping scheme is called a flow and consists of a variable f_e for each edge e of the network, satisfying the following two properties:

1. It doesn't violate edge capacities: $0 \leq f_e \leq c_e$ for all $e \in E$.
2. For all nodes u except s and t, the amount of flow entering u equals the amount leaving

$$\sum_{(w,v)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz}$$

In other words, flow is conserved.

Maximizing Flow

The size of a flow is the total quantity sent from s to t

Maximizing Flow

The size of a flow is the total quantity sent from s to t and, by the conservation principle, is equal to the quantity leaving s:

$$\texttt{size}(f) = \sum_{(s,u)\in E} f_{su}$$

Maximizing Flow

The size of a flow is the total quantity sent from s to t and, by the conservation principle, is equal to the quantity leaving s:

$$\texttt{size}(f) = \sum_{(s,u)\in E} f_{su}$$

Our goal is to assign values to $\{f_e | e \in E\}$ that will satisfy a set of linear constraints and maximize a linear objective function.

Maximizing Flow

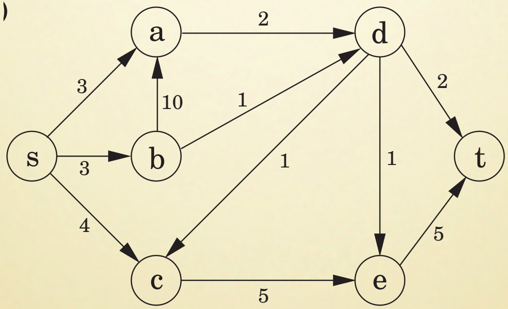
The size of a flow is the total quantity sent from s to t and, by the conservation principle, is equal to the quantity leaving s:

$$\texttt{size}(f) = \sum_{(s,u)\in E} f_{su}$$

Our goal is to assign values to $\{f_e | e \in E\}$ that will satisfy a set of linear constraints and maximize a linear objective function.

But this is a linear program! The maximum-flow problem reduces to linear programming.

The Example



LP

LP

11 variables, one per edge.

LP

11 variables, one per edge.

maximize $f_{sa} + f_{sb} + f_{sc}$

LP

11 variables, one per edge.

maximize $f_{sa} + f_{sb} + f_{sc}$

27 constraints:

LP

11 variables, one per edge.

maximize $f_{sa} + f_{sb} + f_{sc}$

27 constraints:

- 11 for nonnegativity (such as $f_{sa} \geq 0$)

LP

11 variables, one per edge.

maximize $f_{sa} + f_{sb} + f_{sc}$

27 constraints:

- 11 for nonnegativity (such as $f_{sa} \geq 0$)
- 11 for capacity (such as $f_{sa} \leq 3$)

LP

11 variables, one per edge.

maximize $f_{sa} + f_{sb} + f_{sc}$

27 constraints:

- 11 for nonnegativity (such as $f_{sa} \geq 0$)
- 11 for capacity (such as $f_{sa} \leq 3$)
- 5 for flow conservation (one for each node of the graph other than s and t, such as $f_{sc} + f_{dc} = f_{ce}$).

Another Representation

First, introduce a fictitious edge of infinite capacity from t to s thus converting the flow to a circulation;

Another Representation

First, introduce a fictitious edge of infinite capacity from t to s thus converting the flow to a circulation;

The objective now is to maximize the flow on this edge, denoted by f_{ts}.

Another Representation

First, introduce a fictitious edge of infinite capacity from t to s thus converting the flow to a circulation;

The objective now is to maximize the flow on this edge, denoted by f_{ts}.

The advantage of making this modification is that we can now require flow conservation at s and t as well.

Another Representation

$$\max f_{ts}$$

$$f_{ij} \leq c_{ij} \qquad (i,j) \in E$$

$$\sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \leq 0 \quad i \in V$$

$$f_{ij} \geq 0 \qquad (i,j) \in E$$

A Closer Look at the Algorithm

All we know so far of the simplex algorithm is the vague geometric intuition

A Closer Look at the Algorithm

All we know so far of the simplex algorithm is the vague geometric intuition that it keeps making local moves on the surface of a convex feasible region,

A Closer Look at the Algorithm

All we know so far of the simplex algorithm is the vague geometric intuition that it keeps making local moves on the surface of a convex feasible region, successively improving the objective function

A Closer Look at the Algorithm

All we know so far of the simplex algorithm is the vague geometric intuition that it keeps making local moves on the surface of a convex feasible region, successively improving the objective function until it finally reaches the optimal solution.

A Closer Look at the Algorithm

All we know so far of the simplex algorithm is the vague geometric intuition that it keeps making local moves on the surface of a convex feasible region, successively improving the objective function until it finally reaches the optimal solution.

The behavior of simplex has an elementary interpretation:

- Start with zero flow.

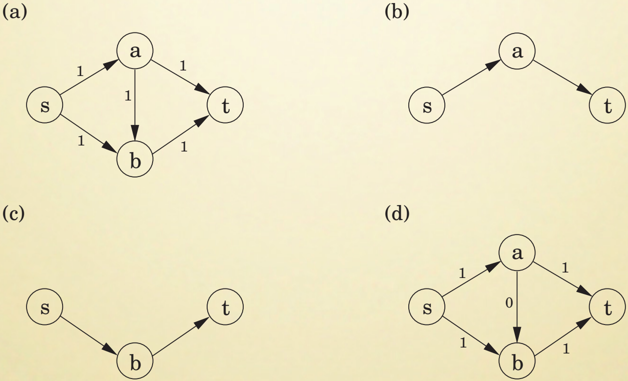
A Closer Look at the Algorithm

All we know so far of the simplex algorithm is the vague geometric intuition that it keeps making local moves on the surface of a convex feasible region, successively improving the objective function until it finally reaches the optimal solution.

The behavior of simplex has an elementary interpretation:

- Start with zero flow.
- Repeat: choose an appropriate path from s to t, and increase flow along the edges of this path as much as possible.

A Flow Example



A Closer Look at the Algorithm

There is just one complication.

A Closer Look at the Algorithm

There is just one complication.

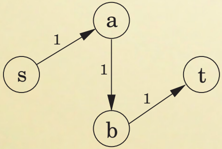
What if we choose a path that blocks all other paths?

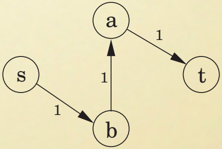
A Closer Look at the Algorithm

There is just one complication.

What if we choose a path that blocks all other paths?

Simplex gets around this problem by also allowing paths to cancel existing flow.





A Closer Look at the Algorithm

To summarize, in each iteration simplex looks for an $s - t$ path whose edges (u, v) can be of two types:

A Closer Look at the Algorithm

To summarize, in each iteration simplex looks for an $s - t$ path whose edges (u, v) can be of two types:

1. (u, v) is in the original network, and is not yet at full capacity.

A Closer Look at the Algorithm

To summarize, in each iteration simplex looks for an $s - t$ path whose edges (u, v) can be of two types:

1. (u, v) is in the original network, and is not yet at full capacity.
2. The reverse edge (v, u) is in the original network, and there is some flow along it.

A Closer Look at the Algorithm

To summarize, in each iteration simplex looks for an $s - t$ path whose edges (u, v) can be of two types:

1. (u, v) is in the original network, and is not yet at full capacity.
2. The reverse edge (v, u) is in the original network, and there is some flow along it.

If the current flow is f, then in the first case, edge (u, v) can handle up to $c_{uv} - f_{uv}$ additional units of flow;

A Closer Look at the Algorithm

To summarize, in each iteration simplex looks for an $s - t$ path whose edges (u, v) can be of two types:

1. (u, v) is in the original network, and is not yet at full capacity.
2. The reverse edge (v, u) is in the original network, and there is some flow along it.

If the current flow is f, then in the first case, edge (u, v) can handle up to $c_{uv} - f_{uv}$ additional units of flow;

in the second case, up to f_{vu} additional units (canceling all or part of the existing flow on (v, u)).

A Closer Look at the Algorithm

These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$,

A Closer Look at the Algorithm

These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

A Closer Look at the Algorithm

These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

$$\begin{cases} c_{uv} - f_{uv} & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v,u) \in E \text{ and } f_{vu} > 0 \end{cases}$$

A Closer Look at the Algorithm

These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

$$\begin{cases} c_{uv} - f_{uv} & \text{if } (u, v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v, u) \in E \text{ and } f_{vu} > 0 \end{cases}$$

Thus we can equivalently think of simplex as choosing an $s - t$ path in the residual network.

A Closer Look at the Algorithm

These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

$$\begin{cases} c_{uv} - f_{uv} & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v,u) \in E \text{ and } f_{vu} > 0 \end{cases}$$

Thus we can equivalently think of simplex as choosing an $s - t$ path in the residual network.

By simulating the behavior of simplex, we get a direct algorithm for solving max-flow.

A Closer Look at the Algorithm

These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

$$\begin{cases} c_{uv} - f_{uv} & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v,u) \in E \text{ and } f_{vu} > 0 \end{cases}$$

Thus we can equivalently think of simplex as choosing an $s - t$ path in the residual network.

By simulating the behavior of simplex, we get a direct algorithm for solving max-flow.

It proceeds in iterations, each time explicitly constructing G^f,

A Closer Look at the Algorithm

These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

$$\begin{cases} c_{uv} - f_{uv} & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v,u) \in E \text{ and } f_{vu} > 0 \end{cases}$$

Thus we can equivalently think of simplex as choosing an $s - t$ path in the residual network.

By simulating the behavior of simplex, we get a direct algorithm for solving max-flow.

It proceeds in iterations, each time explicitly constructing G^f, finding a suitable $s - t$ path in G^f by the breadth-first search,

A Closer Look at the Algorithm

These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

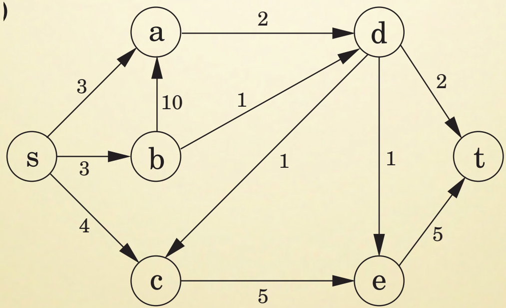
$$\begin{cases} c_{uv} - f_{uv} & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v,u) \in E \text{ and } f_{vu} > 0 \end{cases}$$

Thus we can equivalently think of simplex as choosing an $s-t$ path in the residual network.

By simulating the behavior of simplex, we get a direct algorithm for solving max-flow.

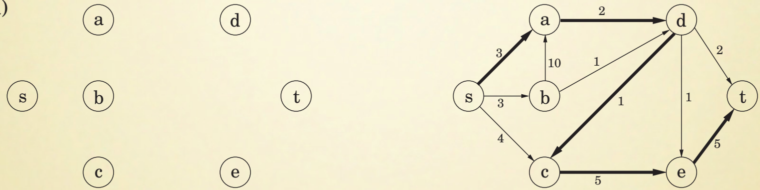
It proceeds in iterations, each time explicitly constructing G^f, finding a suitable $s-t$ path in G^f by the breadth-first search, and halting if there is no longer any such path along which flow can be increased.

The Example

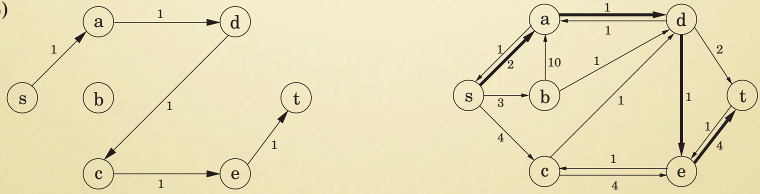


A Flow Example

(a)

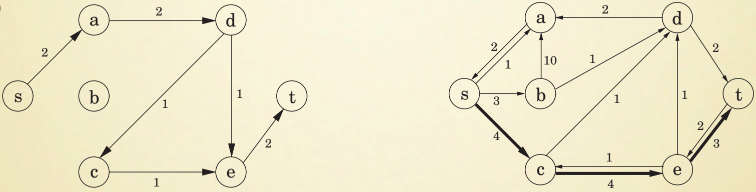


(b)

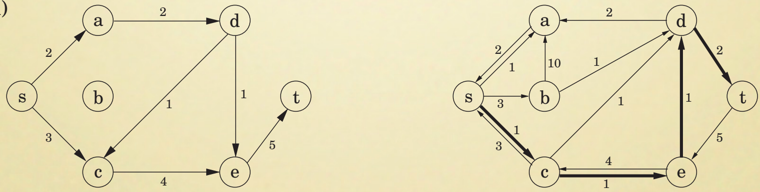


A Flow Example

(c)



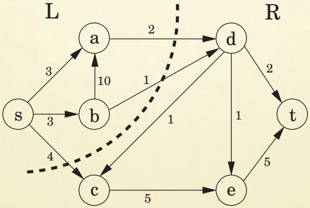
(d)



Cuts

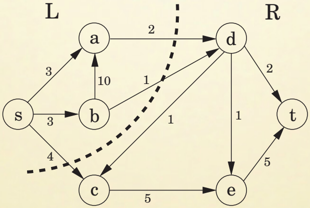
A truly remarkable fact:

Not only does simplex correctly compute a maximum flow, but it also generates a short proof of the optimality of this flow!



Cuts

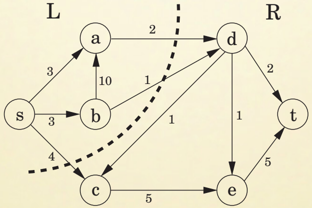
A truly remarkable fact:
Not only does simplex correctly compute a maximum flow, but it also generates a short proof of the optimality of this flow!



An (s, t)-cut partitions the vertices into two disjoint groups L and R

Cuts

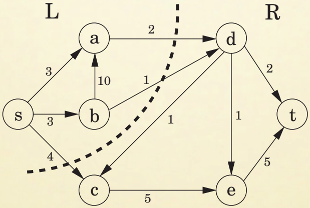
A truly remarkable fact:
Not only does simplex correctly compute a maximum flow, but it also generates a short proof of the optimality of this flow!



An (s,t)-cut partitions the vertices into two disjoint groups L and R such that $s \in L$ and $t \in R$. Its capacity is the total capacity of the edges from L to R, and as argued previously, is an upper bound on any flow:

Cuts

A truly remarkable fact:
Not only does simplex correctly compute a maximum flow, but it also generates a short proof of the optimality of this flow!



An (s,t)-cut partitions the vertices into two disjoint groups L and R such that $s \in L$ and $t \in R$. Its capacity is the total capacity of the edges from L to R, and as argued previously, is an upper bound on any flow: Pick any flow f and any (s,t)-cut (L,R). Then $\texttt{size}(f) \leq \texttt{capacity}(L,R)$.

A Certificate of Optimality

Theorem (Max-flow min-cut)
The size of the maximum flow in a network equals the capacity of the smallest (s, t)-cut.

A Certificate of Optimality

Proof

A Certificate of Optimality

Proof

Suppose f is the final flow when the algorithm terminates.

A Certificate of Optimality

Proof

Suppose f is the final flow when the algorithm terminates.

We know that node t is no longer reachable from s in the residual network G^f.

A Certificate of Optimality

Proof

Suppose f is the final flow when the algorithm terminates.

We know that node t is no longer reachable from s in the residual network G^f.

Let L be the nodes that are reachable from s in G^f, and let $R = V \backslash L$ be the rest of the nodes.

A Certificate of Optimality

Proof

Suppose f is the final flow when the algorithm terminates.

We know that node t is no longer reachable from s in the residual network G^f.

Let L be the nodes that are reachable from s in G^f, and let $R = V \setminus L$ be the rest of the nodes.

We claim that $\texttt{size}(f) = \texttt{capacity}(L, R)$.

A Certificate of Optimality

Proof

Suppose f is the final flow when the algorithm terminates.

We know that node t is no longer reachable from s in the residual network G^f.

Let L be the nodes that are reachable from s in G^f, and let $R = V \backslash L$ be the rest of the nodes.

We claim that $\texttt{size}(f) = \texttt{capacity}(L, R)$.

To see this, observe that by the way L is defined, any edge going from L to R must be at full capacity (in the current flow f), and any edge from R to L must have zero flow.

A Certificate of Optimality

Proof

Suppose f is the final flow when the algorithm terminates.

We know that node t is no longer reachable from s in the residual network G^f.

Let L be the nodes that are reachable from s in G^f, and let $R = V \setminus L$ be the rest of the nodes.

We claim that $\texttt{size}(f) = \texttt{capacity}(L, R)$.

To see this, observe that by the way L is defined, any edge going from L to R must be at full capacity (in the current flow f), and any edge from R to L must have zero flow.

Therefore the net flow across (L, R) is exactly the capacity of the cut.

Efficiency

Each iteration is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an $s-t$ path.

Efficiency

Each iteration is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an $s - t$ path.

But how many iterations are there?

Efficiency

Each iteration is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an $s - t$ path.

But how many iterations are there?

Suppose all edges in the original network have integer capacities $\leq C$.

Efficiency

Each iteration is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an $s - t$ path.

But how many iterations are there?

Suppose all edges in the original network have integer capacities $\leq C$. Then on each iteration of the algorithm, the flow is always an integer and increases by an integer amount.

Efficiency

Each iteration is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an $s-t$ path.

But how many iterations are there?

Suppose all edges in the original network have integer capacities $\leq C$. Then on each iteration of the algorithm, the flow is always an integer and increases by an integer amount. Therefore, since the maximum flow is at most $C|E|$.

Efficiency

Each iteration is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an $s-t$ path.

But how many iterations are there?

Suppose all edges in the original network have integer capacities $\leq C$. Then on each iteration of the algorithm, the flow is always an integer and increases by an integer amount. Therefore, since the maximum flow is at most $C|E|$.

If paths are chosen by using a BFS, which finds the path with the fewest edges, then the number of iterations is at most $O(|V| \cdot |E|)$.
Edmonds-Karp algorithm

Efficiency

Each iteration is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an $s-t$ path.

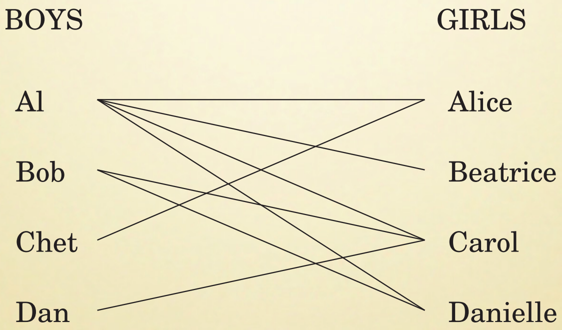
But how many iterations are there?

Suppose all edges in the original network have integer capacities $\leq C$. Then on each iteration of the algorithm, the flow is always an integer and increases by an integer amount. Therefore, since the maximum flow is at most $C|E|$.

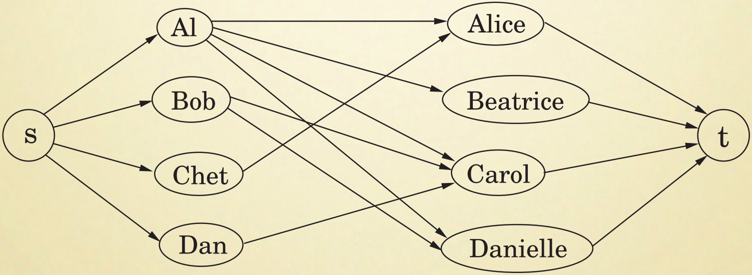
If paths are chosen by using a BFS, which finds the path with the fewest edges, then the number of iterations is at most $O(|V| \cdot |E|)$. *Edmonds-Karp algorithm*

This latter bound gives an overall running time of $O(|V| \cdot |E|^2)$ for maximum flow.

Bipartite Matching



Bipartite Matching



Homework

[DPV07] 7.6, 7.7, 7.8, 7.13, 7.21 and 7.23

** Min-Max Relations in LP **

LP for Max Flow

$$\max f_{ts}$$

$$f_{ij} \leq c_{ij} \qquad (i,j) \in E$$

$$\sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \leq 0 \quad i \in V$$

$$f_{ij} \geq 0 \qquad (i,j) \in E$$

Duality

Primal LP	Dual LP
$\max c^T \mathbf{x}$	$\min \mathbf{y}^T b$
$A\mathbf{x} \leq b$	$\mathbf{y}^T A^T \geq c^T$
$\mathbf{x} \geq 0$	$\mathbf{y} \geq 0$

Primal LP:

$$\begin{aligned} \max\ & c_1 x_1 + \cdots + c_n x_n \\ a_{i1}x_1 + \cdots + a_{in}x_n &\leq b_i \quad \text{for } i \in I \\ a_{i1}x_1 + \cdots + a_{in}x_n &= b_i \quad \text{for } i \in E \\ x_j &\geq 0 \quad \text{for } j \in N \end{aligned}$$

Dual LP:

$$\begin{aligned} \min\ & b_1 y_1 + \cdots + b_m y_m \\ a_{1j}y_1 + \cdots + a_{mj}y_m &\geq c_j \quad \text{for } j \in N \\ a_{1j}y_1 + \cdots + a_{mj}y_m &= c_j \quad \text{for } j \notin N \\ y_i &\geq 0 \quad \text{for } i \in I \end{aligned}$$

LP-Duality

$$\max f_{ts}$$

$$f_{ij} \leq c_{ij} \qquad (i,j) \in E$$

$$\sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \leq 0 \quad i \in V$$

$$f_{ij} \geq 0 \qquad (i,j) \in E$$

LP-Duality

$$\max f_{ts}$$

$$f_{ij} \leq c_{ij} \qquad (i,j) \in E$$

$$\sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \leq 0 \qquad i \in V$$

$$f_{ij} \geq 0 \qquad (i,j) \in E$$

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 & (i,j) &\in E \\ p_s - p_t &\geq 1 & & \\ d_{ij} &\geq 0 & (i,j) &\in E \\ p_i &\geq 0 & i &\in V \end{aligned}$$

Explanation of the Dual

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 && (i,j) \in E \\ p_s - p_t &\geq 1 && \\ d_{ij} &\in \{0,1\} && (i,j) \in E \\ p_i &\in \{0,1\} && i \in V \end{aligned}$$

Explanation of the Dual

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 & (i,j) &\in E \\ p_s - p_t &\geq 1 & & \\ d_{ij} &\in \{0,1\} & (i,j) &\in E \\ p_i &\in \{0,1\} & i &\in V \end{aligned}$$

- To obtain the dual program we introduce variables d_{ij} and p_i corresponding to the two types of inequalities in the primal.

Explanation of the Dual

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j \geq 0 \quad & (i,j) \in E \\ p_s - p_t \geq 1 \quad & \\ d_{ij} \in \{0,1\} \quad & (i,j) \in E \\ p_i \in \{0,1\} \quad & i \in V \end{aligned}$$

- To obtain the dual program we introduce variables d_{ij} and p_i corresponding to the two types of inequalities in the primal.
  - d_{ij}: distance labels on edges;
  - p_i: potentials on nodes.

Integer Program

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned}
d_{ij} - p_i + p_j &\geq 0 && (i,j) \in E \\
p_s - p_t &\geq 1 && \\
d_{ij} &\in \{0,1\} && (i,j) \in E \\
p_i &\in \{0,1\} && i \in V
\end{aligned}$$

Integer Program

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 & (i,j) &\in E \\ p_s - p_t &\geq 1 & & \\ d_{ij} &\in \{0,1\} & (i,j) &\in E \\ p_i &\in \{0,1\} & i &\in V \end{aligned}$$

- Let $(\mathbf{d}^*, \mathbf{p}^*)$ be an optimal solution to this integer program.

Integer Program

$$
\min \sum_{(i,j)\in E} c_{ij} d_{ij}
$$

$$
\begin{aligned}
d_{ij} - p_i + p_j \geq 0 & \quad (i,j) \in E \\
p_s - p_t \geq 1 & \\
d_{ij} \in \{0,1\} & \quad (i,j) \in E \\
p_i \in \{0,1\} & \quad i \in V
\end{aligned}
$$

- Let $(\mathbf{d}^*, \mathbf{p}^*)$ be an optimal solution to this integer program.
- The only way to satisfy the inequality $p_s^* - p_t^* \geq 1$ with a $0/1$ substitution is to set $p_s^* = 1$ and $p_t^* = 0$.

Integer Program

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 & (i,j) &\in E \\ p_s - p_t &\geq 1 & & \\ d_{ij} &\in \{0,1\} & (i,j) &\in E \\ p_i &\in \{0,1\} & i &\in V \end{aligned}$$

- Let $(\mathbf{d}^*, \mathbf{p}^*)$ be an optimal solution to this integer program.
- The only way to satisfy the inequality $p_s^* - p_t^* \geq 1$ with a $0/1$ substitution is to set $p_s^* = 1$ and $p_t^* = 0$.
- This solution naturally defines an $s - t$ cut $(X, \overline{X})$, where X is the set of potential 1 nodes, and $\overline{X}$ the set of potential 0 nodes.

Integer Program

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 & (i,j) &\in E \\ p_s - p_t &\geq 1 & & \\ d_{ij} &\in \{0,1\} & (i,j) &\in E \\ p_i &\in \{0,1\} & i &\in V \end{aligned}$$

Integer Program

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 & (i,j) \in E \\ p_s - p_t &\geq 1 & \\ d_{ij} &\in \{0,1\} & (i,j) \in E \\ p_i &\in \{0,1\} & i \in V \end{aligned}$$

- Consider an edge (i,j) with $i \in X$ and $j \in \overline{X}$, Since $p_i^* = 1$ and $p_j^* = 0$, and thus $d_{ij}^* = 1$.

Integer Program

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j \geq 0 & \quad (i,j) \in E \\ p_s - p_t \geq 1 & \\ d_{ij} \in \{0,1\} & \quad (i,j) \in E \\ p_i \in \{0,1\} & \quad i \in V \end{aligned}$$

- Consider an edge (i,j) with $i \in X$ and $j \in \overline{X}$, Since $p_i^* = 1$ and $p_j^* = 0$, and thus $d_{ij}^* = 1$.
- The distance label for each of the remaining edges can be set to either 0 or 1 without violating the first constraints; however in order to minimize the objective function value it must be set to 0.

Integer Program

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 \quad & (i,j) \in E \\ p_s - p_t &\geq 1 & \\ d_{ij} &\in \{0,1\} & (i,j) \in E \\ p_i &\in \{0,1\} & i \in V \end{aligned}$$

- Consider an edge (i,j) with $i \in X$ and $j \in \overline{X}$, Since $p_i^* = 1$ and $p_j^* = 0$, and thus $d_{ij}^* = 1$.
- The distance label for each of the remaining edges can be set to either 0 or 1 without violating the first constraints; however in order to minimize the objective function value it must be set to 0.
- The objective function value is precisely the capacity of the cut $(X, \overline{X})$, and hence $(X, \overline{X})$ must be a minimum $s-t$ cut.

Relaxation of the Integer Program

- The integer program is a formulation of the minimum $s - t$ cut problem.

Relaxation of the Integer Program

- The integer program is a formulation of the minimum $s - t$ cut problem.
- The dual program can be viewed as a relaxation of the integer program where the integrality constraint on the variables is dropped.

Relaxation of the Integer Program

- The integer program is a formulation of the minimum $s - t$ cut problem.
- The dual program can be viewed as a relaxation of the integer program where the integrality constraint on the variables is dropped.
- This leads to the constraints $1 \geq d_{ij} \geq 0$ for $(i,j) \in E$ and $1 \geq p_i \geq 0$ for $i \in V$.

Relaxation of the Integer Program

- The integer program is a formulation of the minimum $s - t$ cut problem.
- The dual program can be viewed as a relaxation of the integer program where the integrality constraint on the variables is dropped.
- This leads to the constraints $1 \geq d_{ij} \geq 0$ for $(i, j) \in E$ and $1 \geq p_i \geq 0$ for $i \in V$.
- The upper bound constraints on the variables are redundant; their omission cannot give a better solution.

Relaxation of the Integer Program

- The integer program is a formulation of the minimum $s - t$ cut problem.
- The dual program can be viewed as a relaxation of the integer program where the integrality constraint on the variables is dropped.
- This leads to the constraints $1 \geq d_{ij} \geq 0$ for $(i, j) \in E$ and $1 \geq p_i \geq 0$ for $i \in V$.
- The upper bound constraints on the variables are redundant; their omission cannot give a better solution.
- Dropping these constraints gives the dual program in the form given above. We will say that this program is the LP relaxation of the integer program.

Relaxation of the Integer Program

- In principle, the best fractional $s - t$ cut could have lower capacity than the best integral cut. Surprisingly enough, this does not happen.

Relaxation of the Integer Program

- In principle, the best fractional $s - t$ cut could have lower capacity than the best integral cut. Surprisingly enough, this does not happen.
- From linear programming theory we know that for any objective function, i.e., assignment of capacities to the edges of G, there is a vertex solution that is optimal.

Relaxation of the Integer Program

- In principle, the best fractional $s - t$ cut could have lower capacity than the best integral cut. Surprisingly enough, this does not happen.
- From linear programming theory we know that for any objective function, i.e., assignment of capacities to the edges of G, there is a vertex solution that is optimal.
- Now, it can be proven that each vertex solution is integral, with each coordinate being 0 or 1.

Relaxation of the Integer Program

- In principle, the best fractional $s-t$ cut could have lower capacity than the best integral cut. Surprisingly enough, this does not happen.
- From linear programming theory we know that for any objective function, i.e., assignment of capacities to the edges of G, there is a vertex solution that is optimal.
- Now, it can be proven that each vertex solution is integral, with each coordinate being 0 or 1.
- This follows from the fact that the constraint matrix of this program is totally unimodular, Thus, the dual program always has an integral optimal solution.