I. Sorting Networks

Thomas Sauerwald

Easter 2015

UNIVERSITY OF CAMBRIDGE

Outline

Introduction to Sorting Networks

Batcher’s Sorting Network

Counting Networks

Load Balancing on Graphs

Overview: Sorting Networks

(Serial) Sorting Algorithms

- we already know several (comparison-based) sorting algorithms: Insertion sort, Bubble sort, Merge sort, Quick sort, Heap sort
- execute one operation at a time
- can handle arbitrarily large inputs
- sequence of comparisons is not set in advance

Sorting Networks

- only perform comparisons
- can only handle inputs of a fixed size
- sequence of comparisons is set in advance
- Comparisons can be performed in parallel

Allows to sort n numbers in sublinear time!

Simple concept, but surprisingly deep and complex theory!

Comparison Networks

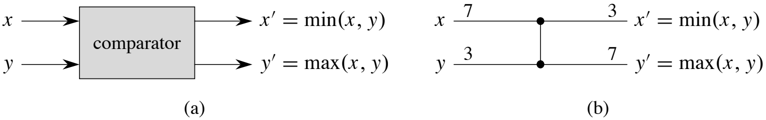
A sorting network is a comparison network which works correctly (that is, it sorts every input)

Comparison Network

- A comparison network consists solely of wires and comparators:
  - comparator is a device with, on given two inputs, x and y, returns two outputs x' and y'
  - wire connect output of one comparator to the input of another
  - special wires: n input wires $a_1, a_2, \ldots, a_n$ and n output wires $b_1, b_2, \ldots, b_n$

operates in $O(1)$

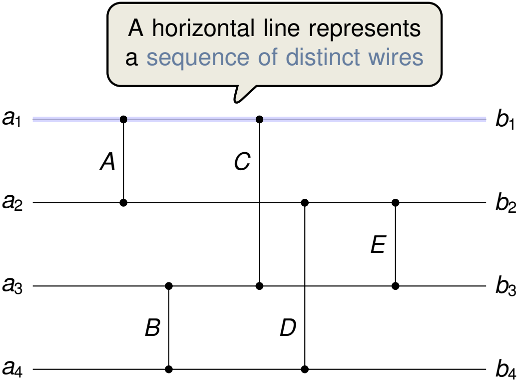
Convention: use the same name for both a wire and its value.



(a) (b)

Figure 27.1 **(a)** A comparator with inputs x and y and outputs x' and y'. **(b)** The same comparator, drawn as a single vertical line. Inputs $x = 7$, $y = 3$ and outputs $x' = 3$, $y' = 7$ are shown.

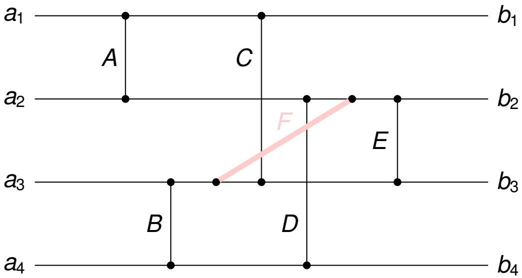
Example of a Comparison Network (Figure 27.2)



A horizontal line represents a sequence of distinct wires

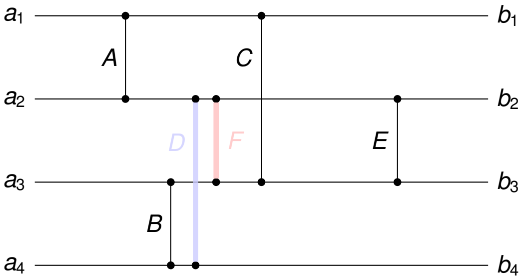
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



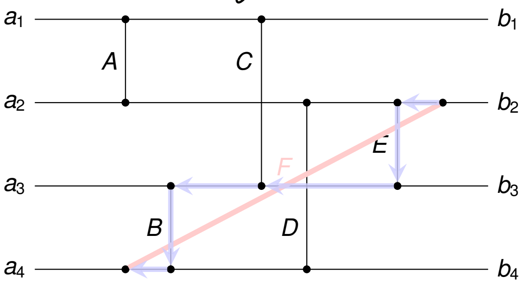
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic ✓



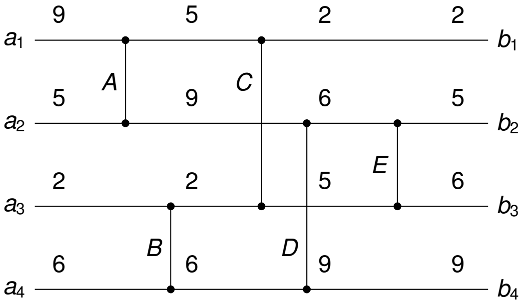
Example of a Comparison Network (Figure 27.2)

Interconnections between comparators must be acyclic



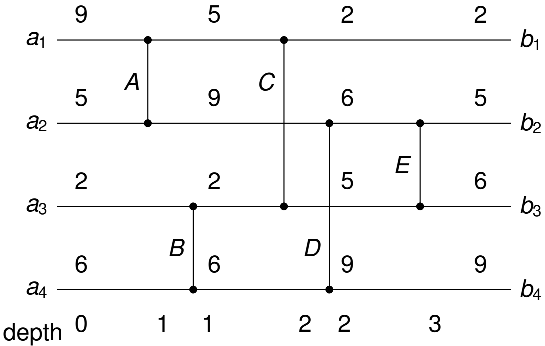
Tracing back a path must never cycle back on itself and go through the same comparator twice.

Example of a Comparison Network (Figure 27.2)



This network is in fact a sorting network!

Example of a Comparison Network (Figure 27.2)



Maximum depth of an output wire equals total running time

Depth of a wire:

- Input wire has Depth 0
- If a comparator has two inputs of depths d_x and d_y, then outputs have depth $\max\{d_x, d_y\} + 1$

Zero-One Principle

Zero-One Principle: A sorting networks works correctly on arbitrary inputs if it works correctly on binary inputs.

Lemma 27.1

If a comparison network transforms the input $a = \langle a_1, a_2, \ldots, a_n \rangle$ into the output $b = \langle b_1, b_2, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms $f(a) = \langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$ into $f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle$.

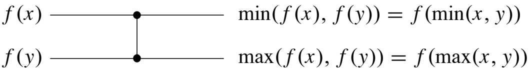


Figure 27.4 The operation of the comparator in the proof of Lemma 27.1. The function f is monotonically increasing.

Zero-One Principle

Zero-One Principle: A sorting networks works correctly on arbitrary inputs if it works correctly on binary inputs.

Lemma 27.1

If a comparison network transforms the input $a = \langle a_1, a_2, \ldots, a_n \rangle$ into the output $b = \langle b_1, b_2, \ldots, b_n \rangle$, then for any monotonically increasing function f, the network transforms $f(a) = \langle f(a_1), f(a_2), \ldots, f(a_n) \rangle$ into $f(b) = \langle f(b_1), f(b_2), \ldots, f(b_n) \rangle$.

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequence of arbitrary numbers correctly.

Proof of the Zero-One Principle

Theorem 27.2 (Zero-One Principle)

If a comparison network with n inputs sorts all 2^n possible sequences of 0's and 1's correctly, then it sorts all sequence of arbitrary numbers correctly.

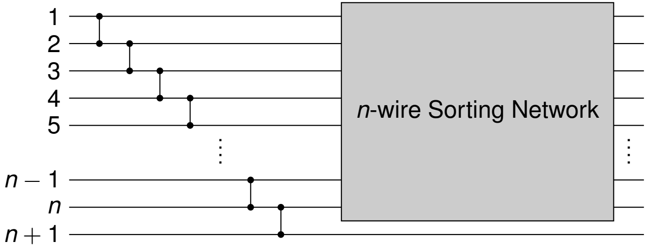
Proof:

- For the sake of contradiction, suppose the network does not correctly sort.
- Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ be the input with $a_i < a_j$, but the network places a_j before a_i in the output
- Define a monotonitcally increasing function f as:

$$f(x) = \begin{cases} 0 & \text{if } x \leq a_i, \\ 1 & \text{if } x > a_i. \end{cases}$$

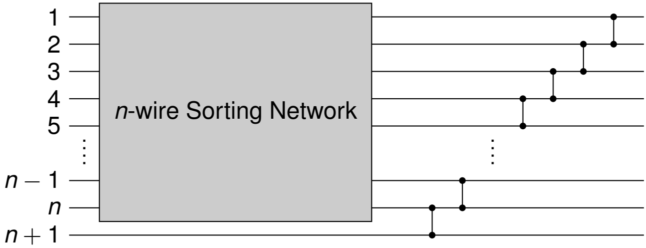
- Since the network places a_j before a_i, by the previous lemma $\Rightarrow$ $f(a_j)$ is placed before $f(a_i)$
- But $f(a_j) = 1$ and $f(a_i) = 0$, which contradicts the assumption that the network sorts all sequences of 0's and 1's correctly ☐

Some Basic (Recursive) Sorting Networks



???

These are Sorting Networks, but with depth $\Theta(n)$.



???

Outline

Introduction to Sorting Networks

Batcher's Sorting Network

Counting Networks

Load Balancing on Graphs

Bitonic Sequences

Bitonic Sequence

A sequence is bitonic if it monotonically increases and then monotonically decreases, or can be circularly shifted to become monotonically increasing and then monotonically decreasing.

Sequences of one or two numbers are defined to be bitonic.

Examples:

- $\langle 1, 4, 6, 8, 3, 2 \rangle$ ✓
- $\langle 6, 9, 4, 2, 3, 5 \rangle$ ✓
- $\langle 9, 8, 3, 2, 4, 6 \rangle$ ✓
- ~~$\langle 4, 5, 7, 1, 2, 6 \rangle$~~
- binary sequences: $0^i 1^j 0^k$, or, $1^i 0^j 1^k$, for $i, j, k \geq 0$.

Towards a Bitonic Sorting Networks

Half-Cleaner

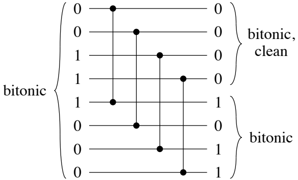
A half-cleaner is a comparison network of depth 1 in which input wire i is compared with wire $i + n/2$ for $i = 1, 2, \ldots, n/2$.

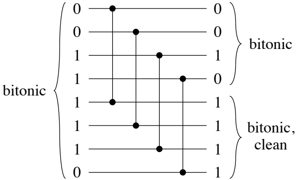
We always assume that n is even.

Lemma 27.3

If the input to a half-cleaner is a bitonic sequence of 0's and 1's, then the output satisfies the following properties:

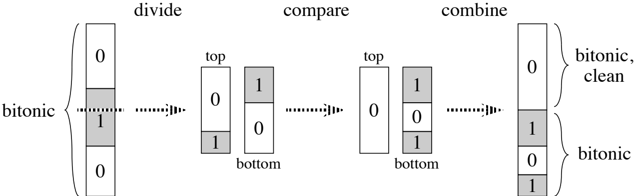
- both the top half and the bottom half are bitonic,
- every element in the top is not larger than any element in the bottom,
- at least one half is clean.



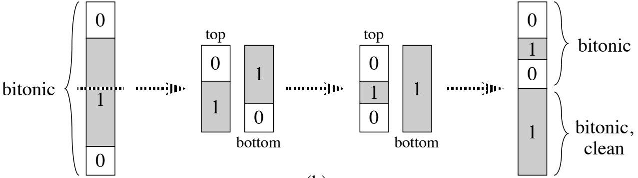


Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i 1^j 0^k$, for some $i, j, k \geq 0$.



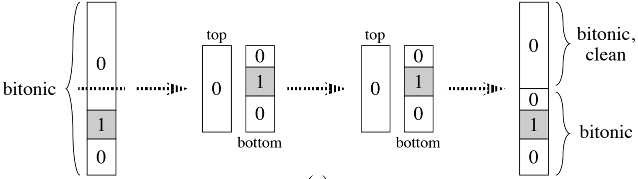
(a)



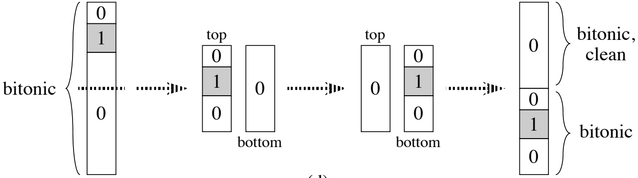
(b)

Proof of Lemma 27.3

W.l.o.g. assume that the input is of the form $0^i 1^j 0^k$, for some $i, j, k \geq 0$.



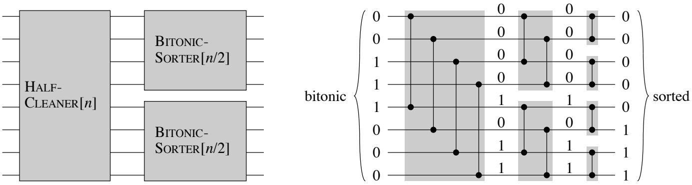
(c)



(d)

This suggests a recursive approach, since it now suffices to sort the top and bottom half separately.

The Bitonic Sorter



(a)

(b)

Figure 27.9 The comparison network BITONIC-SORTER[n], shown here for $n = 8$. **(a)** The recursive construction: HALF-CLEANER[n] followed by two copies of BITONIC-SORTER[$n/2$] that operate in parallel. **(b)** The network after unrolling the recursion. Each half-cleaner is shaded. Sample zero-one values are shown on the wires.

Henceforth we will always assume that n is a power of 2.

Recursive Formula for depth $D(n)$:

$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + 1 & \text{if } n = 2^k. \end{cases}$$

BITONIC-SORTER[n] has depth $\log n$ and sorts any zero-one bitonic sequence.

Merging Networks

Merging Networks

- can merge two sorted input sequences into one sorted output sequences
- will be based on a modification of BITONIC-SORTER[n]

Basic Idea:

- consider two given sequences $X = 00000111$, $Y = 00001111$
- concatenating X with Y^R (the reversal of Y) $\Rightarrow$ 0000011111110000

This sequence is bitonic!

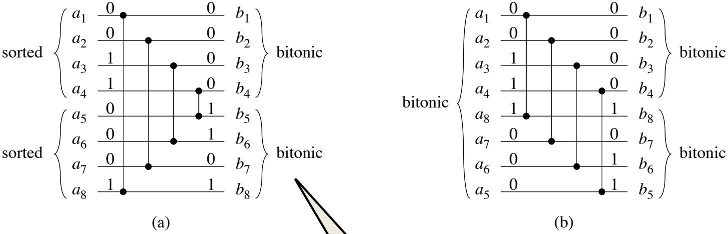
Hence in order to merge the sequences X and Y, it suffices to perform a bitonic sort on X concatenated with Y^R.

Construction of a Merging Network (1/2)

- Given two sorted sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$
- We know it suffices to bitonically sort $\langle a_1, a_2, \ldots, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+1} \rangle$
- Recall: first half-cleaner of BITONIC-SORT$[n]$ compares i and $n/2 + i$

⇒ First part of MERGER$[n]$ compares inputs i and $n - i$ for $i = 1, 2, \ldots, n/2$

- Remaining part is identical to BITONIC-SORT$[n]$



Lemma 27.3 still applies, since the reversal of a bitonic sequence is bitonic.

Figure 27.10 Comparing the first stage of MERGER$[n]$ with HALF-CLEANER$[n]$, for $n = 8$. **(a)** The first stage of MERGER$[n]$ transforms the two monotonic input sequences $\langle a_1, a_2, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, a_{n/2+2}, \ldots, a_n \rangle$ into two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ and $\langle b_{n/2+1}, b_{n/2+2}, \ldots, b_n \rangle$. **(b)** The equivalent operation for HALF-CLEANER$[n]$. The bitonic input sequence $\langle a_1, a_2, \ldots, a_{n/2-1}, a_{n/2}, a_n, a_{n-1}, \ldots, a_{n/2+2}, a_{n/2+1} \rangle$ is transformed into the two bitonic sequences $\langle b_1, b_2, \ldots, b_{n/2} \rangle$ and $\langle b_n, b_{n-1}, \ldots, b_{n/2+1} \rangle$.

Construction of a Merging Network (2/2)

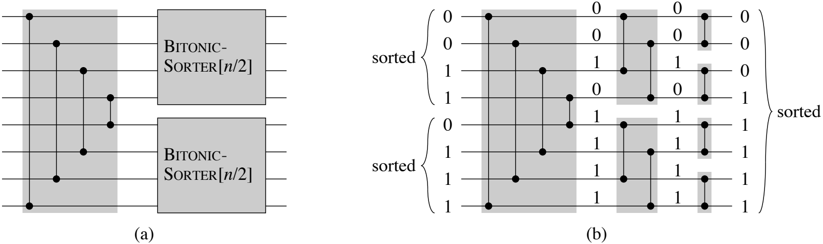


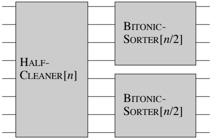
Figure 27.11 A network that merges two sorted input sequences into one sorted output sequence. The network MERGER[n] can be viewed as BITONIC-SORTER[n] with the first half-cleaner altered to compare inputs i and $n - i + 1$ for $i = 1, 2, \ldots, n/2$. Here, $n = 8$. **(a)** The network decomposed into the first stage followed by two parallel copies of BITONIC-SORTER[$n/2$]. **(b)** The same network with the recursion unrolled. Sample zero-one values are shown on the wires, and the stages are shaded.

Construction of a Sorting Network

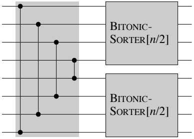
Main Components

1. BITONIC-SORT$[n]$
   - sorts any bitonic sequence
   - depth $\log n$
2. MERGER$[n]$
   - merges two sorted input sequences
   - depth $\log n$

1.

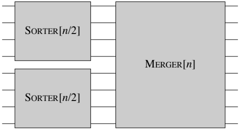


2.



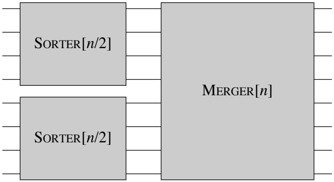
Batcher's Sorting Network

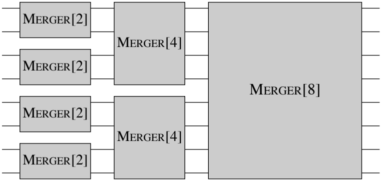
- SORTER$[n]$ is defined recursively:
  - If $n = 2^k$, use two copies of SORTER$[n/2]$ to sort two subsequences of length $n/2$ each. Then merge them using MERGER$[n]$.
  - If $n = 1$, network consists of a single wire.

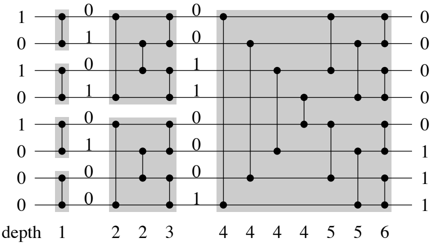


can be seen as a parallel version of merge sort

Unrolling the Recursion (Figure 27.12)







Recursion for $D(n)$:

$$D(n) = \begin{cases} 0 & \text{if } n = 1, \\ D(n/2) + \log n & \text{if } n = 2^k. \end{cases}$$

Solution: $D(n) = \Theta(\log^2 n)$.

SORTER[n] has depth $\Theta(\log^2 n)$ and sorts any input.

A Glimpse at the AKS Network

Ajtai, Komlós, Szemerédi (1983)

There exists a sorting network with depth $O(\log n)$.

Quite elaborate construction, and involves huges constants.

Perfect Halver

A perfect halver is a comparator network that, given any input, places the $n/2$ smaller keys in $b_1, \ldots, b_{n/2}$ and the $n/2$ larger keys in $b_{n/2+1}, \ldots, b_n$.

Perfect halver of depth $\log_2 n$ exist $\rightsquigarrow$ yields sorting networks of depth $\Theta((\log n)^2)$.

Approximate Halver

An (n, ϵ)-approximate halver, $\epsilon < 1$, is a comparator network that for every $k = 1, 2, \ldots, n/2$ places at most ϵk of its k smallest keys in $b_{n/2+1}, \ldots, b_n$ and at most ϵk of its k largest keys in $b_1, \ldots, b_{n/2}$.

We will prove that such networks can be constructed in constant depth!

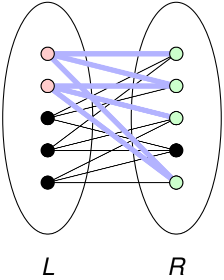
Expander Graphs

Expander Graphs

A bipartite (n, d, μ)-expander is a graph with:

- G has n vertices ($n/2$ on each side)
- the edge-set is the union of d matchings
- For every subset $S \subseteq V$ being in one part,

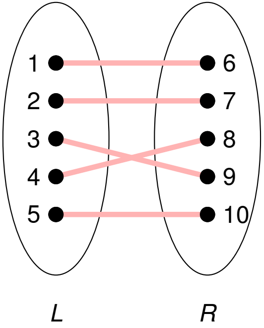
$$|N(S)| \geq \min\{\mu \cdot |S|, n/2 - |S|\}$$

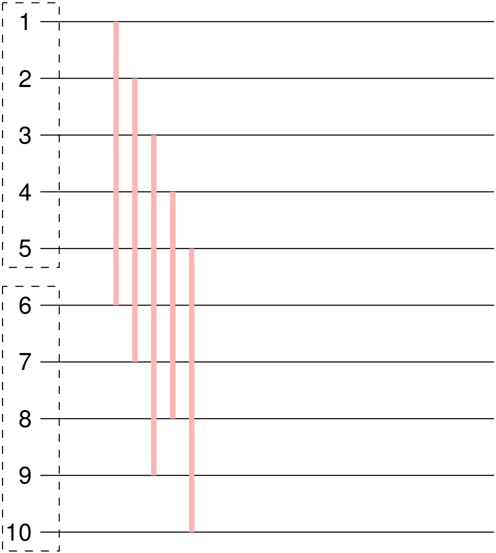


Expander Graphs:

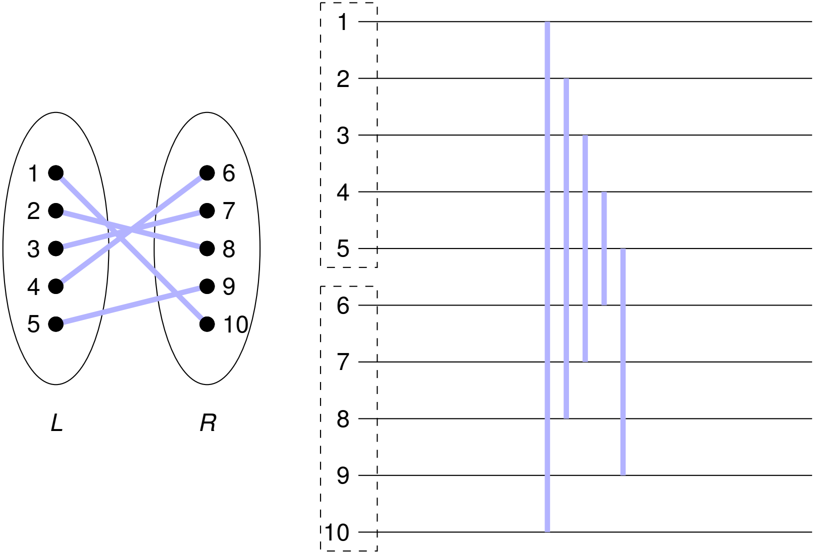
- **probabilistic construction** “easy”: take d (disjoint) random matchings
- **explicit construction** is a deep mathematical problem with ties to number theory, group theory, combinatorics etc.
- **many applications** in networking, complexity theory and coding theory

From Expanders to Approximate Halvers

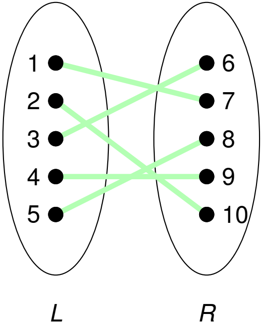


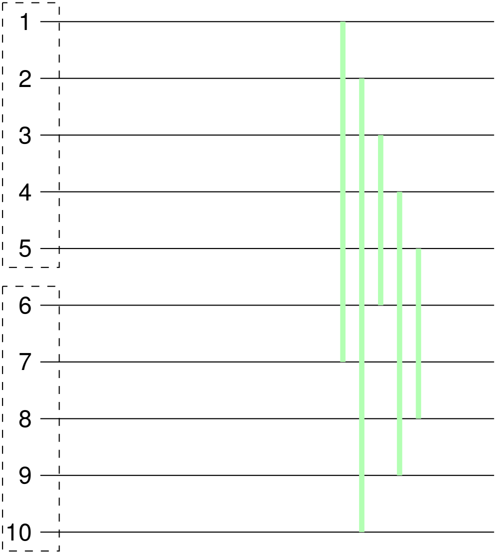


From Expanders to Approximate Halvers

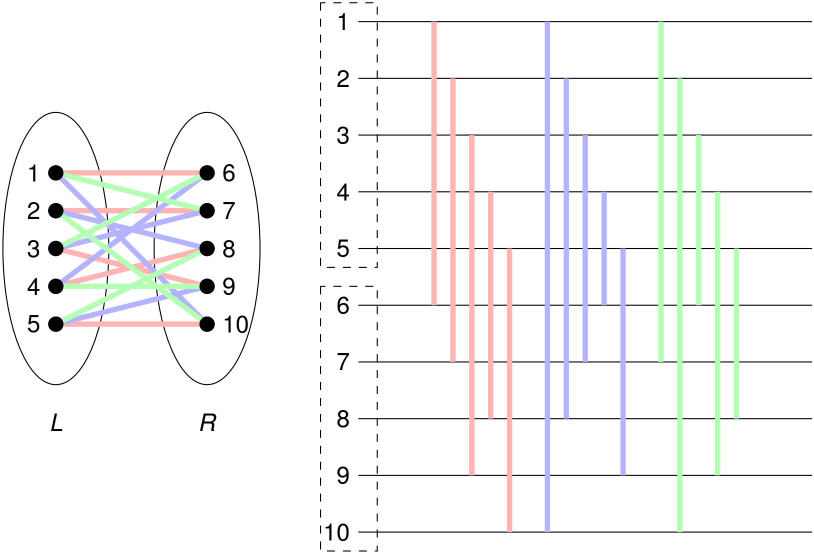


From Expanders to Approximate Halvers





From Expanders to Approximate Halvers



Existence of Approximate Halvers

Proof:

- $X :=$ wires with the k smallest inputs
- $Y :=$ wires in lower half with k smallest outputs
- For every $u \in N(Y)$: $\exists$ comparator (u, v)
- Let u_t, v_t be their keys after the comparator
  Let u_d, v_d be their keys at the output
- Note that $v_d \in Y \subseteq X$
- Further: $u_d \leq u_t \leq v_t \leq v_d \Rightarrow u_d \in X$
- Since u was arbitrary:

$$|Y| + |N(Y)| \leq k.$$

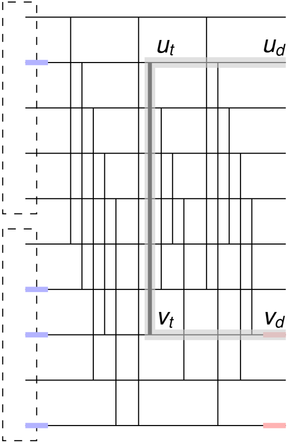
- Since G is a bipartite (n, d, μ)-expander:

$$\begin{aligned} |Y| + |N(Y)| &\geq |Y| + \min\{\mu|Y|, n/2 - |Y|\} \\ &= \min\{(1+\mu)|Y|, n/2\}. \end{aligned}$$

- Combining the two bounds above yields:

$$(1+\mu)|Y| \leq k.$$

- The same argument shows that at most $\epsilon \cdot k$, $\epsilon := 1/(\mu + 1)$, of the k largest input keys are placed in $b_1, \ldots, b_{n/2}$. ☐



AKS network vs. Batcher's network

Donald E. Knuth (Stanford)

"Batcher's method is much better, unless n exceeds the total memory capacity of all computers on earth!"

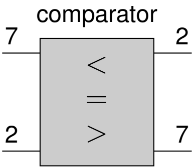
Richard J. Lipton (Georgia Tech)

"The AKS sorting network is ***galactic****: it needs that n be larger than* 2^{78} *or so to finally be smaller than Batcher's network for n items."*

Siblings of Sorting Network

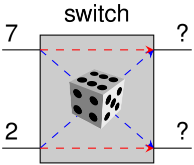
Sorting Networks

- sorts any input of size *n*
- special case of Comparison Networks



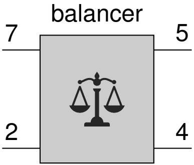
Switching (Shuffling) Networks

- creates a random permutation of *n* items
- special case of Permutation Networks



Counting Networks

- balances any stream of tokens over *n* wires
- special case of Balancing Networks



Outline

Introduction to Sorting Networks

Batcher's Sorting Network

Counting Networks

Load Balancing on Graphs

Counting Network

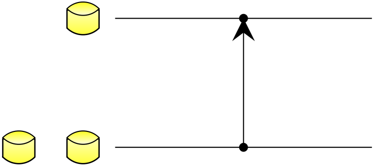
Distributed Counting

Processors collectively assign successive values from a given range.

Values could represent addresses in memories or destinations on an interconnection network

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



Counting Network

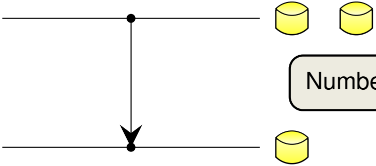
Distributed Counting

Processors collectively assign successive values from a given range.

Values could represent addresses in memories or destinations on an interconnection network

Balancing Networks

- constructed in a similar manner like sorting networks
- instead of comparators, consists of balancers
- balancers are asynchronous flip-flops that forward tokens from its inputs to one of its two outputs alternately (top, bottom, top,. . .)



Number of tokens differs by at most one

Bitonic Counting Network

Counting Network (Formal Definition)

1. Let $x_1, x_2, \ldots, x_n$ be the number of tokens (ever received) on the designated input wires
2. Let $y_1, y_2, \ldots, y_n$ be the number of tokens (ever received) on the designated output wires
3. In a quiescent state: $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$
4. A counting network is a balancing network with the **step-property:**

$$0 \leq y_i - y_j \leq 1 \text{ for any } i < j.$$

Correctness of the Bitonic Counting Network

Facts

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil$, and $\sum_{i=1}^{n/2} x_{2i} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor$
2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.
3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists!\ j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.

Key Lemma

Consider a MERGER[n]. Then if the inputs $x_1, \ldots, x_{n/2}$ and $x_{n/2+1}, \ldots, x_n$ have the step property, then so does the output $y_1, \ldots, y_n$.

Proof (by induction on n)

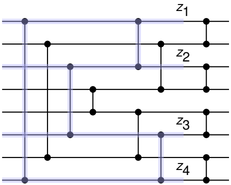
- Case $n = 2$ is clear, since MERGER[2] is a single balancer

Correctness of the Bitonic Counting Network

Facts

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil$, and $\sum_{i=1}^{n/2} x_{2i} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor$
2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.
3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists!\ j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.



Proof (by induction on n)

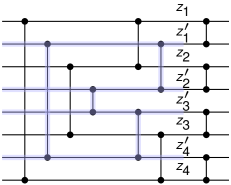
- Case $n = 2$ is clear, since MERGER[2] is a single balancer
- $n > 2$: Let $z_1, \ldots, z_{n/2}$ and $z'_{n/2+1}, \ldots, z'_n$ be the outputs of the MERGER[$n/2$] subnetworks

Correctness of the Bitonic Counting Network

Facts

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil$, and $\sum_{i=1}^{n/2} x_{2i} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor$
2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.
3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists!\ j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.



Proof (by induction on n)

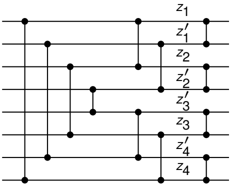
- Case $n = 2$ is clear, since MERGER[2] is a single balancer
- $n > 2$: Let $z_1, \ldots, z_{n/2}$ and $z'_{n/2+1}, \ldots, z'_n$ be the outputs of the MERGER[$n/2$] subnetworks

Correctness of the Bitonic Counting Network

Facts

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ have the step property. Then:

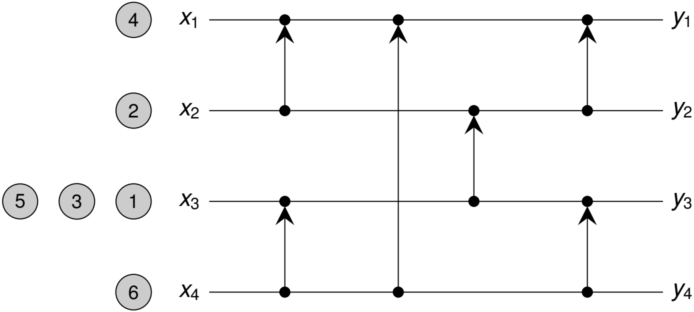
1. We have $\sum_{i=1}^{n/2} x_{2i-1} = \left\lceil \frac{1}{2} \sum_{i=1}^{n} x_i \right\rceil$, and $\sum_{i=1}^{n/2} x_{2i} = \left\lfloor \frac{1}{2} \sum_{i=1}^{n} x_i \right\rfloor$
2. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, then $x_i = y_i$ for $i = 1, \ldots, n$.
3. If $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i + 1$, then $\exists!\ j = 1, 2, \ldots, n$ with $x_j = y_j + 1$ and $x_i = y_i$ for $j \neq i$.



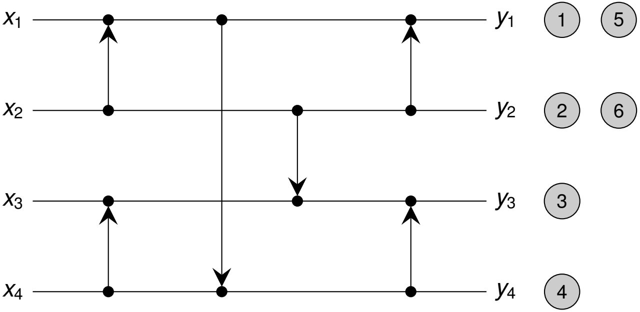
Proof (by induction on n)

- Case $n = 2$ is clear, since MERGER[2] is a single balancer
- $n > 2$: Let $z_1, \ldots, z_{n/2}$ and $z'_{n/2+1}, \ldots, z'_n$ be the outputs of the MERGER[$n/2$] subnetworks
- IH $\Rightarrow$ $z_1, \ldots, z_{n/2}$ and $z'_{n/2+1}, \ldots, z'_n$ have the step property
- Let $Z := \sum_{i=1}^{n/2} z_i$ and $Z' := \sum_{i=1}^{n/2} z'_i$
- F1 $\Rightarrow$ $Z = \left\lceil \frac{1}{2} \sum_{i=1}^{n/2} x_i \right\rceil + \left\lfloor \frac{1}{2} \sum_{i=n/2+1}^{n} x_i \right\rfloor$ and $Z' = \left\lfloor \frac{1}{2} \sum_{i=1}^{n/2} x_i \right\rfloor + \left\lceil \frac{1}{2} \sum_{i=n/2+1}^{n} x_i \right\rceil$
- Case 1: If $Z = Z'$, then F2 implies the output of MERGER[n] is $y_i = z_{1+\lfloor (i-1)/2 \rfloor}$ ✓
- Case 2: If $|Z - Z'| = 1$, F3 implies $z_i = z'_i$ for $i = 1, \ldots, n/2$ except a unique j with $z_j \neq z'_j$.
  Balancer between z_j and z'_j will ensure that the step property holds.

Bitonic Counting Network in Action

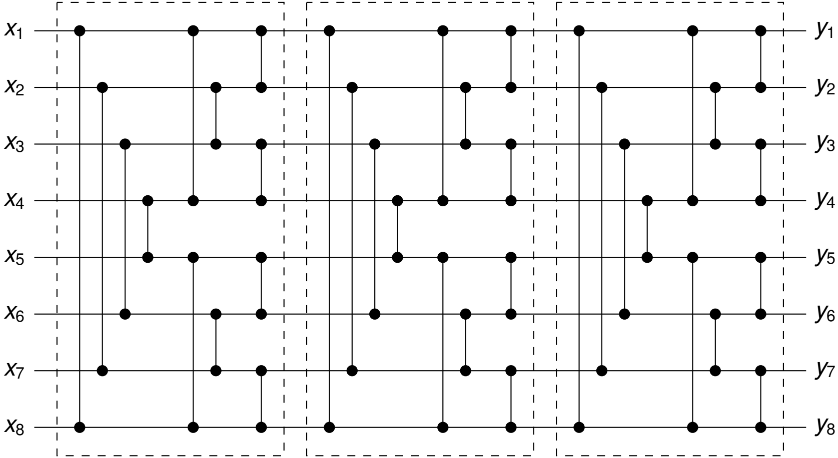


Bitonic Counting Network in Action



Counting can be done as follows:
Add **local counter** to each output wire i, to assign consecutive numbers $i, i+n, i+2\cdot n, \ldots$

A Periodic Counting Network [Aspnes, Herlihy, Shavit, JACM 1994]



Consists of log n BLOCK[n] networks each of which has depth log n

From Counting to Sorting

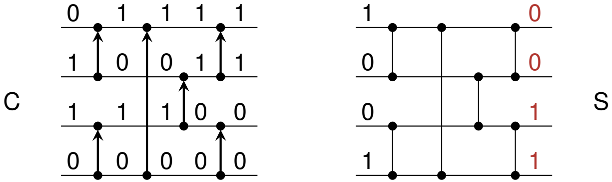
The converse is not true!

Counting vs. Sorting

If a network is a counting network, then it is also a sorting network.

Proof.

- Let C be a counting network, and S be the corresponding sorting network
- Consider an input sequence $a_1, a_2, \ldots, a_n \in \{0,1\}^n$ to S
- Define an input $x_1, x_2, \ldots, x_n \in \{0,1\}^n$ to C by $x_i = 1$ iff $a_i = 0$.
- C is a counting network $\Rightarrow$ all ones will be routed to the lower wires
- S corresponds to C $\Rightarrow$ all zeros will be routed to the lower wires
- By the Zero-One Principle, S is a sorting network. $\square$



Outline

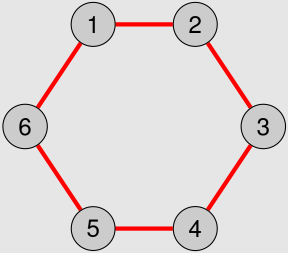
Introduction to Sorting Networks

Batcher's Sorting Network

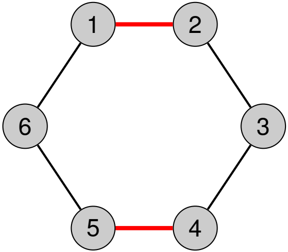
Counting Networks

Load Balancing on Graphs

Communication Models: Diffusion vs. Matching



$$\mathbf{M} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$



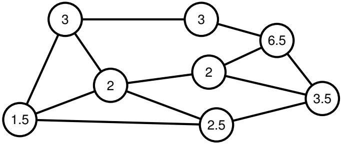
$$\mathbf{M}^{(t)} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Smoothness of the Load Distribution

- let $x \in \mathbb{R}^n$ be a load vector
- $\overline{x}$ denotes the average load

Metrics

- ℓ_2-norm: $\Phi^t = \sqrt{\sum_{i=1}^n (x_i^t - \overline{x})^2}$
- makespan: $\max_{i=1}^n x_i^t$
- discrepancy: $\max_{i=1}^n x_i^t - \min_{i=1}^n x_i$.



For this example:

- $\Phi^t = \sqrt{0^2 + 0^2 + 3.5^2 + 0.5^2 + 1^2 + 1^2 + 1.5^2 + 0.5^2} = \sqrt{17}$
- $\max_{i=1}^n x_i^t = 6.5$
- $\max_{i=1}^n x_i^t - \min_{i=1}^n x_i^t = 5$

Diffusion Matrix

How to choose α for a d-regular graph?

- $\alpha = \frac{1}{d}$ may yield to oscillation (if graph is bipartite)
- $\alpha = \frac{1}{d+1}$ ensures convergence
- $\alpha = \frac{1}{2d}$ ensures convergence (and all eigenvalues of M are non-negative)

Diffusion Matrix

Given an undirected, connected graph $G = (V, E)$ and a diffusion parameter $\alpha > 0$, the diffusion matrix M is defined as follows:

$$M_{ij} = \begin{cases} \alpha & \text{if } (i,j) \in E, \\ 1 - \alpha \underbrace{\deg(i)}_{\text{\# neighbors of } i} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Further let $\gamma(M) := \max_{\mu_i \neq 1} |\mu_i|$, where $\mu_1 = 1 \geq \mu_2 \geq \cdots \geq \mu_n \geq -1$ are the eigenvalues of M.

This can be also seen as a random walk on G!

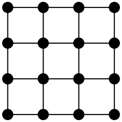
First-Order Diffusion: Load vector x^t satisfies

$$x^t = M \cdot x^{t-1}.$$

1D grid

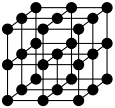
$\gamma(M) \approx 1 - \frac{1}{n^2}$

2D grid



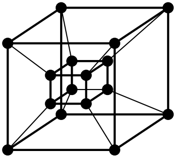
$\gamma(M) \approx 1 - \frac{1}{n}$

3D grid



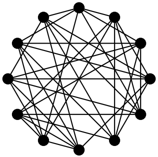
$\gamma(M) \approx 1 - \frac{1}{n^{2/3}}$

Hypercube



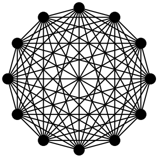
$\gamma(M) \approx 1 - \frac{1}{\log n}$

Random Graph



$\gamma(M) < 1$

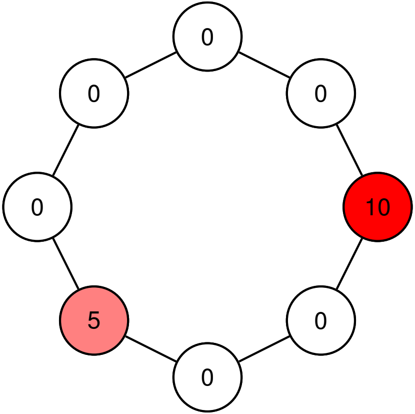
Complete Graph



$\gamma(M) \approx 0$

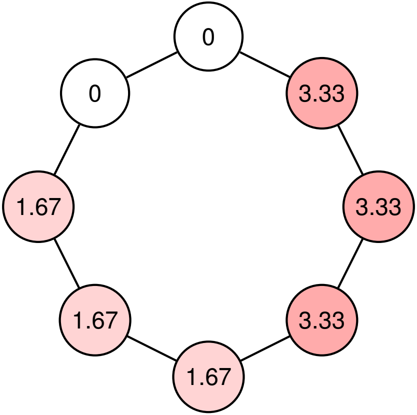
$\gamma(M) \in (0,1]$ measures connectivity of G

Diffusion on a Ring



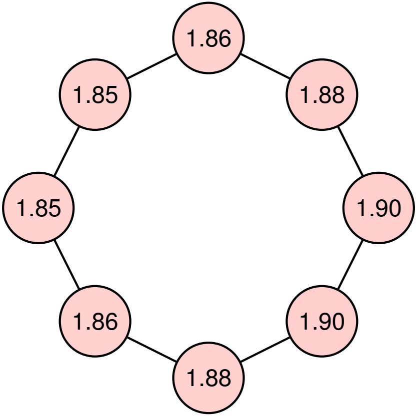
Diffusion on a Ring

after iteration 1:



Diffusion on a Ring

after iteration 20:



Convergence of the Quadratic Error (Upper Bound)

Lemma

Let $\gamma(M) := \max_{\mu_i \neq 1} |\mu_i|$, where $\mu_1 = 1 \geq \mu_2 \geq \cdots \geq \mu_n \geq -1$ are the eigenvalues of M. Then for any iteration t,

$$\Phi^t \leq \gamma(M)^{2t} \cdot \Phi^0.$$

Proof:

- Let $e^t = x^t - \overline{x}$, where $\overline{x}$ is the column vector with all entries set to $\overline{x}$
- Express e^t through the orthogonal basis given by the eigenvectors of M:

$$e^t = \alpha_1 \cdot v_1 + \alpha_2 \cdot v_2 + \cdots + \alpha_n \cdot v_n = \sum_{i=2}^{n} \alpha_i \cdot v_i.$$

e^t is orthogonal to v_1

- For the diffusion scheme,

$$e^{t+1} = Me^t = M \cdot \left(\sum_{i=2}^{n} \alpha_i v_i \right) = \sum_{i=2}^{n} \alpha_i \mu_i v_i.$$

- Taking norms and using that the v_i's are orthogonal,

$$\|e^{t+1}\|_2 = \|Me^t\|_2 = \sum_{i=2}^{n} c_i^2 \mu_i^2 \|v_i\|_2 \leq \gamma^2 \sum_{i=2}^{n} c_i^2 \|v_i\|_2 = \gamma^2 \cdot \|e^t\|_2 \qquad \square$$

Convergence of the Quadratic Error (Lower Bound)

Lemma

For any eigenvalue μ_i, $1 \leq i \leq n$, there is an initial load vector x^0 so that

$$\Phi^t = \mu_i^{2t} \cdot \Phi^0.$$

Proof:

- Let $x^0 = \overline{x} + v_i$, where v_i is the eigenvector corresponding to μ_i
- Then

$$e^t = Me^{t-1} = M^t e^0 = M^t v_i = \mu_i^t v_i,$$

and

$$\Phi^t = \|e^t\|_2 = \mu_i^{2t} \|v_i\|_2 = \mu_i^{2t} \Phi^0. \qquad \square$$

Outlook: Idealised versus Discrete Case

Here load consists of integers that cannot be divided further.

Idealised Case

$$x^t = M \cdot x^{t-1} = M^t \cdot x^0$$

Linear System

- corresponds to Markov chain
- well-understood

Given any load vector x^0, the number of iterations until x^t satisfies $\Phi^t \leq \epsilon$ is at most $\frac{\log(\Phi^0/\epsilon)}{\gamma(M)}$.

Discrete Case

Rounding Error

$$y^t = M \cdot y^{t-1} + \Delta^t = M^t \cdot y^0 + \sum_{s=1}^{t} M^{t-s} \cdot \Delta^s$$

Non-Linear System

- rounding of a Markov chain
- harder to analyze

How close can it be made to the idealised case?

II. Matrix Multiplication

Thomas Sauerwald

Easter 2015

UNIVERSITY OF CAMBRIDGE

Outline

Introduction

Serial Matrix Multiplication

Reminder: Multithreading

Multithreaded Matrix Multiplication

Matrix Multiplication

Remember: If $A = (a_{ij})$ and $B = (b_{ij})$ are square $n \times n$ matrices, then the matrix product $C = A \cdot B$ is defined by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \qquad \forall i, j = 1, 2, \ldots, n.$$

This definition suggests that $n \cdot n^2 = n^3$ arithmetic operations are necessary.

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        $c_{ij} = 0$
        for k = 1 to n
            $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return C
```

SQUARE-MATRIX-MULTIPLY(A, B) takes time $\Theta(n^3)$.

Outline

Introduction

Serial Matrix Multiplication

Reminder: Multithreading

Multithreaded Matrix Multiplication

Divide & Conquer: First Approach

Assumption: n is always an exact power of 2.

Divide & Conquer:
Partition A, B, and C into four $n/2 \times n/2$ matrices:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

Hence the equation $C = A \cdot B$ becomes:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

This corresponds to the four equations:

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{aligned}$$

Each equation specifies two multiplications of $n/2 \times n/2$ matrices and the addition of their products.

Divide & Conquer: First Approach (Pseudocode)

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE($A, B$)
1  $n = A.rows$
2  let $C$ be a new $n \times n$ matrix
3  if $n == 1$
4      $c_{11} = a_{11} \cdot b_{11}$
5  else partition $A$, $B$, and $C$ as in equations (4.9)
6      $C_{11} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{11}, B_{11}$)
              $+$ SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{12}, B_{21}$)
7      $C_{12} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{11}, B_{12}$)
              $+$ SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{12}, B_{22}$)
8      $C_{21} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{21}, B_{11}$)
              $+$ SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{22}, B_{21}$)
9      $C_{22} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{21}, B_{12}$)
              $+$ SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{22}, B_{22}$)
10 return $C$
```

Line 5: Handle submatrices implicitly through index calculations instead of creating them.

Let $T(n)$ be the runtime of this procedure. Then:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 8 \cdot T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

Solution: $T(n) = \Theta(8^{\log_2 n}) = \Theta(n^3)$

No improvement over the naive algorithm!

Divide & Conquer (Second Approach)

Idea: Make the recursion tree less bushy by performing only **7** recursive multiplications of $n/2 \times n/2$ matrices.

Strassen's Algorithm (1969)

1. Partition each of the matrices into four $n/2 \times n/2$ submatrices
2. Create 10 matrices $S_1, S_2, \dots, S_{10}$. Each is $n/2 \times n/2$ and is the sum or difference of two matrices created in the previous step.
3. Recursively compute 7 matrix products $P_1, P_2, \dots, P_7$, each $n/2 \times n/2$
4. Compute $n/2 \times n/2$ submatrices of C by adding and subtracting various combinations of the P_i.

Time for steps 1,2,4: $\Theta(n^2)$, hence $T(n) = 7 \cdot T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\log 7})$.

Details of Strassen’s Algorithm

The 10 Submatrices and 7 Products

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 = A_{11} \cdot (B_{12} - B_{22}) \\
P_2 &= S_2 \cdot B_{22} = (A_{11} + A_{12}) \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} = (A_{21} + A_{22}) \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 = A_{22} \cdot (B_{21} - B_{11}) \\
P_5 &= S_5 \cdot S_6 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
P_6 &= S_7 \cdot S_8 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) \\
P_7 &= S_9 \cdot S_{10} = (A_{11} - A_{21}) \cdot (B_{11} + B_{12})
\end{aligned}
$$

Claim

$$
\begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{21} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix} = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{pmatrix}
$$

Proof:

$$
\begin{aligned}
P_5 + P_4 - P_2 + P_6 = {} & A_{11}B_{11} + \cancel{A_{11}B_{22}} + \cancel{A_{22}B_{11}} + \cancel{A_{22}B_{22}} + \cancel{A_{22}B_{21}} - \cancel{A_{22}B_{11}} \\
& - \cancel{A_{11}B_{22}} - \cancel{A_{12}B_{22}} + A_{12}B_{21} + \cancel{A_{12}B_{22}} - \cancel{A_{22}B_{21}} - \cancel{A_{22}B_{22}} \\
= {} & A_{11}B_{11} + A_{12}B_{21} \qquad \square
\end{aligned}
$$

Current State-of-the-Art

Conjecture: Does a quadratic-time algorithm exist?

Asymptotic Complexities:

- $O(n^3)$, naive approach
- $O(n^{2.808})$, Strassen (1969)
- $O(n^{2.796})$, Pan (1978)
- $O(n^{2.522})$, Schönhage (1981)
- $O(n^{2.517})$, Romani (1982)
- $O(n^{2.496})$, Coppersmith and Winograd (1982)
- $O(n^{2.479})$, Strassen (1986)
- $O(n^{2.376})$, Coppersmith and Winograd (1989)
- $O(n^{2.374})$, Stothers (2010)
- $O(n^{2.3728642})$, V. Williams (2011)
- $O(n^{2.3728639})$, Le Gall (2014)
- ...

Outline

Introduction

Serial Matrix Multiplication

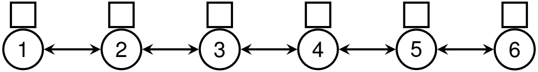
Reminder: Multithreading

Multithreaded Matrix Multiplication

Memory Models

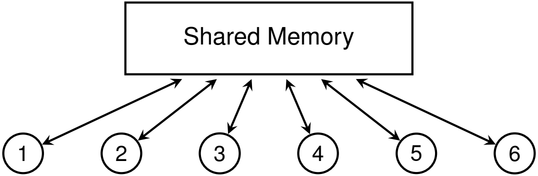
Distributed Memory

- Each processor has its private memory
- Access to memory of another processor via messages



Shared Memory

- Central location of memory
- Each processor has direct access


Shared Memory

Dynamic Multithreading

- Programming shared-memory parallel computer difficult
- Use concurrency platform which coordinates all resources

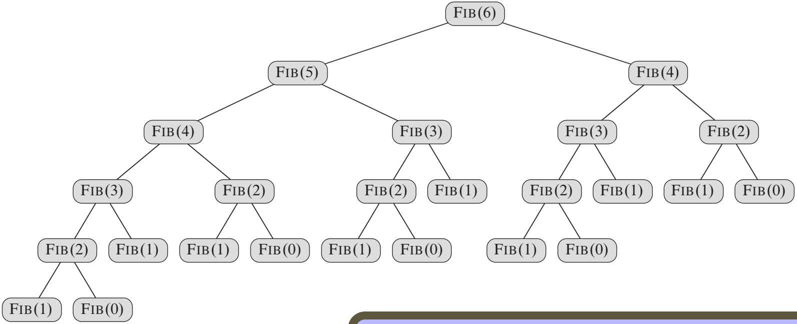
Scheduling jobs, communication protocols, load balancing etc.

Functionalities:

- **`spawn`**
  - (optional) prefix to a procedure call statement
  - procedure is executed in a separate thread
- **`sync`**
  - wait until all spawned threads are done
- **`parallel`**
  - (optinal) prefix to the standard loop **`for`**
  - each iteration is called in its own thread

Only logical parallelism, but not actual!
Need a scheduler to map threads to processors.

Computing Fibonacci Numbers Recursively (Fig. 27.1)



Very inefficient – exponential time!

```
FIB(n)
  if n<=1 return n
  else x=FIB(n-1)
       y=FIB(n-2)
       return x+y
```

Computing Fibonacci Numbers in Parallel (Fig. 27.2)

- Without **spawn** and **sync** same pseudocode as before
- **spawn** does not imply parallel execution (depends on scheduler)

```
P-FIB(n)
  if n<=1 return n
  else x=spawn P-FIB(n-1)
       y=spawn P-FIB(n-2)
       sync
       return x+y
```

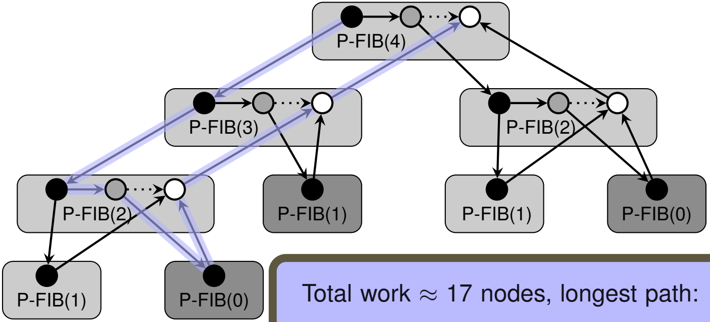
Computing Fibonacci Numbers in Parallel (Fig. 27.2)

Computation Dag $G = (V, E)$

- V set of threads (instructions/strands without parallel control)
- E set of dependencies

```
P-FIB(n)
  if n<=1 return n
  else x=spawn P-FIB(n-1)
       y=spawn P-FIB(n-2)
       sync
       return x+y
```

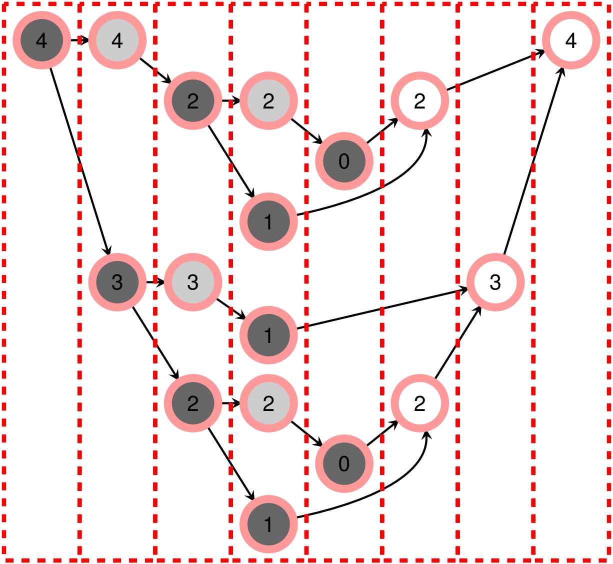
Computing Fibonacci Numbers in Parallel (Fig. 27.2)



Total work $\approx$ 17 nodes, longest path: 8 nodes

```
P-FIB(n)
   if n<=1 return n
   else x=spawn P-FIB(n-1)
        y=spawn P-FIB(n-2)
        sync
        return x+y
```

Computing Fibonacci Numbers in Parallel (DAG Perspective)

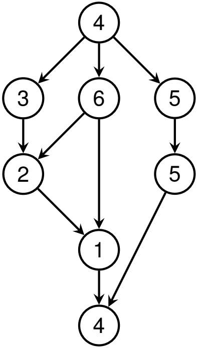


Performance Measures

Work

Total time to execute everything on single processor.

$\sum = 26$



Performance Measures

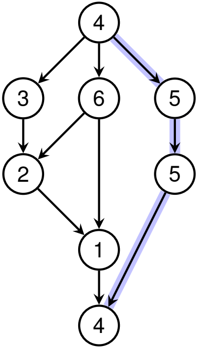
Work

Total time to execute everything on single processor.

Span

Longest time to execute the threads along any path.

$\sum = 18$



Performance Measures

Work

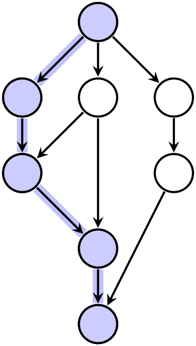
Total time to execute everything on single processor.

Span

Longest time to execute the threads along any path.

If each thread takes unit time, span is the length of the critical path.

$\#\text{nodes} = 5$



Work Law and Span Law

- T_1 = work, T_∞ = span
- P = number of (identical) processors
- T_P = running time on P processors

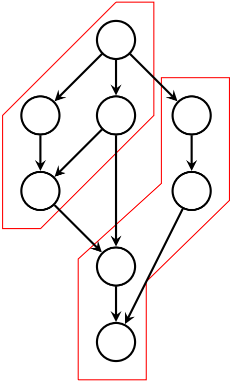
Running time actually also depends on scheduler etc.!

Work Law

$$T_P \geq \frac{T_1}{P}$$

Time on P processors can't be shorter than if all work all time

$T_1 = 8$, $P = 4$



Work Law and Span Law

- T_1 = work, T_∞ = span
- P = number of (identical) processors
- T_P = running time on P processors

Running time actually also depends on scheduler etc.!

Work Law

$$T_P \geq \frac{T_1}{P}$$

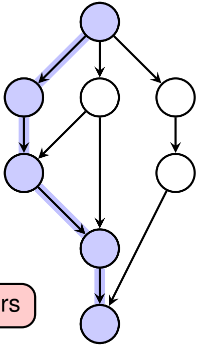
Time on P processors can't be shorter than if all work all time

Span Law

$$T_P \geq T_\infty$$

Time on P processors can't be shorter than time on ∞ processors

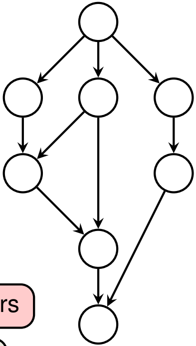
$T_\infty = 5$



Work Law and Span Law

- T_1 = work, T_∞ = span
- P = number of (identical) processors
- T_P = running time on P processors

Running time actually also depends on scheduler etc.!



Work Law

$$T_P \geq \frac{T_1}{P}$$

Time on P processors can't be shorter than if all work all time

Span Law

$$T_P \geq T_\infty$$

Time on P processors can't be shorter than time on ∞ processors

- Speed-Up: $\frac{T_1}{T_P}$
- Parallelism: $\frac{T_1}{T_\infty}$

Maximum Speed-Up bounded by P!

Maximum Speed-Up for ∞ processors!

Outline

Introduction

Serial Matrix Multiplication

Reminder: Multithreading

Multithreaded Matrix Multiplication

Warmup: Matrix Vector Multiplication

Remember: Multiplying an $n \times n$ matrix $A = (a_{ij})$ and n-vector $x = (x_j)$ yields an n-vector $y = (y_i)$ given by

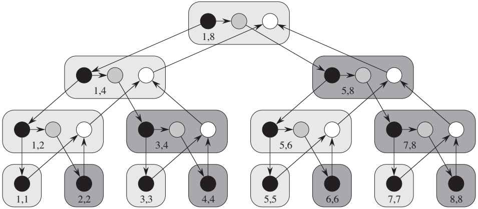
$$y_i = \sum_{j=1}^{n} a_{ij} x_j \qquad \text{for } i = 1, 2, \ldots, n.$$

```
MAT-VEC($A, x$)
$n = A.rows$
let $y$ be a new vector of length $n$
parallel for $i = 1$ to $n$
    $y_i = 0$
parallel for $i = 1$ to $n$
    for $j = 1$ to $n$
        $y_i = y_i + a_{ij} x_j$
return $y$
```

The **`parallel for`**-loops can be used since different entries of y can be computed concurrently.

How can a compiler implement the **`parallel for`**-loop?

Implementing `parallel for` based on Divide-and-Conquer



```
MAT-VEC-MAIN-LOOP($A, x, y, n, i, i'$)
if $i == i'$
    for $j = 1$ to $n$
        $y_i = y_i + a_{ij} x_j$
else $mid = \lfloor (i + i')/2 \rfloor$
    spawn MAT-VEC-MAIN-LOOP($A, x, y, n, i, mid$)
    MAT-VEC-MAIN-LOOP($A, x, y, n, mid + 1, i'$)
    sync
```

```
MAT-VEC($A, x$)
$n = A.rows$
let $y$ be a new vector of length $n$
parallel for $i = 1$ to $n$
    $y_i = 0$
parallel for $i = 1$ to $n$
    for $j = 1$ to $n$
        $y_i = y_i + a_{ij} x_j$
return $y$
```

$$T_1(n) = \Theta(n^2)$$

Work is equal to running time of its serialization; overhead of recursive spawning does not change asymptotics.

$$T_\infty(n) = \Theta(\log n) + \max_{1 \leq i \leq n} \mathrm{iter}(n)$$
$$= \Theta(n).$$

Span is the depth of recursive callings plus the maximum span of any of the n iterations.

Naive Algorithm in Parallel

```
P-SQUARE-MATRIX-MULTIPLY(A, B)
$n = A.rows$
let $C$ be a new $n \times n$ matrix
parallel for $i = 1$ to $n$
    parallel for $j = 1$ to $n$
        $c_{ij} = 0$
        for $k = 1$ to $n$
            $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return $C$
```

P-SQUARE-MATRIX-MULTIPLY(A, B) has work $T_1(n) = \Theta(n^3)$ and span $T_\infty(n) = \Theta(n)$.

The first two nested for-loops parallelise perfectly.

The Simple Divide&Conquer Approach in Parallel

```
P-MATRIX-MULTIPLY-RECURSIVE($C, A, B$)
$n = A.rows$
if $n == 1$
    $c_{11} = a_{11}b_{11}$
else let $T$ be a new $n \times n$ matrix
    partition $A$, $B$, $C$, and $T$ into $n/2 \times n/2$ submatrices
        $A_{11}, A_{12}, A_{21}, A_{22}$; $B_{11}, B_{12}, B_{21}, B_{22}$; $C_{11}, C_{12}, C_{21}, C_{22}$;
        and $T_{11}, T_{12}, T_{21}, T_{22}$; respectively
    spawn P-MATRIX-MULTIPLY-RECURSIVE($C_{11}, A_{11}, B_{11}$)
    spawn P-MATRIX-MULTIPLY-RECURSIVE($C_{12}, A_{11}, B_{12}$)
    spawn P-MATRIX-MULTIPLY-RECURSIVE($C_{21}, A_{21}, B_{11}$)
    spawn P-MATRIX-MULTIPLY-RECURSIVE($C_{22}, A_{21}, B_{12}$)
    spawn P-MATRIX-MULTIPLY-RECURSIVE($T_{11}, A_{12}, B_{21}$)
    spawn P-MATRIX-MULTIPLY-RECURSIVE($T_{12}, A_{12}, B_{22}$)
    spawn P-MATRIX-MULTIPLY-RECURSIVE($T_{21}, A_{22}, B_{21}$)
    P-MATRIX-MULTIPLY-RECURSIVE($T_{22}, A_{22}, B_{22}$)
    sync
    parallel for $i = 1$ to $n$
        parallel for $j = 1$ to $n$
            $c_{ij} = c_{ij} + t_{ij}$
```

The same as before.

P-MATRIX-MULTIPLY-RECURSIVE has work $T_1(n) = \Theta(n^3)$ and span $T_\infty(n) = \Theta(\log^2 n)$.

$T_\infty(n) = T_\infty(n/2) + \Theta(\log n)$

Strassen's Algorithm in Parallel

Strassen's Algorithm (parallelised)

1. Partition each of the matrices into four $n/2 \times n/2$ submatrices

   This step takes $\Theta(1)$ work and span by index calculations.

2. Create 10 matrices $S_1, S_2, \ldots, S_{10}$. Each is $n/2 \times n/2$ and is the sum or difference of two matrices created in the previous step.

   Can create all 10 matrices with $\Theta(n^2)$ work and $\Theta(\log n)$ span using doubly nested **`parallel for`** loops.

3. Recursively compute 7 matrix products $P_1, P_2, \ldots, P_7$, each $n/2 \times n/2$

   Recursively spawn the computation of the seven products.

4. Compute $n/2 \times n/2$ submatrices of C by adding and subtracting various combinations of the P_i.

   Using doubly nested **`parallel for`** this takes $\Theta(n^2)$ work and $\Theta(\log n)$ span.

$$T_1(n) = \Theta(n^{\log 7})$$
$$T_\infty(n) = \Theta(\log^2 n)$$

Matrix Multiplication and Matrix Inversion

Speedups for Matrix Inversion by an equivalence with Matrix Multiplication.

Theorem 28.1 (Multiplication is no harder than Inversion)

If we can invert an $n \times n$ matrix in time $I(n)$, where $I(n) = \Omega(n^2)$ and $I(n)$ satisfies the regularity condition $I(3n) = O(I(n))$, then we can multiply two $n \times n$ matrices in time $O(I(n))$.

Proof:

- Define a $3n \times 3n$ matrix D by:

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix} \quad \Rightarrow \quad D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix}.$$

- Matrix D can be constructed in $\Theta(n^2) = O(I(n))$ time,
- and we can invert D in $O(I(3n)) = O(I(n))$ time.

$\Rightarrow$ We can compute AB in $O(I(n))$ time. $\square$

The Other Direction

Theorem 28.1 (Multiplication is no harder than Inversion)

If we can invert an $n \times n$ matrix in time $I(n)$, where $I(n) = \Omega(n^2)$ and $I(n)$ satisfies the regularity condition $I(3n) = O(I(n))$, then we can multiply two $n \times n$ matrices in time $O(I(n))$.

Allows us to use Strassen's Algorithm to invert a matrix!

Theorem 28.2 (Inversion is no harder than Multiplication)

Suppose we can multiply two $n \times n$ real matrices in time $M(n)$ and $M(n)$ satisfies the two regularity conditions $M(n+k) = O(M(n))$ for any $0 \leq k \leq n$ and $M(n/2) \leq c \cdot M(n)$ for some constant $c < 1/2$. Then we can compute the inverse of any real nonsingular $n \times n$ matrix in time $O(M(n))$.

Proof of this directon much harder (CLRS) – relies on properties of SPD matrices.

III. Linear Programming

Thomas Sauerwald

Easter 2015

UNIVERSITY OF CAMBRIDGE

Outline

Introduction

Standard and Slack Forms

Formulating Problems as Linear Programs

Simplex Algorithm

Finding an Initial Solution

Introduction

Linear Programming (informal definition)

- maximize or minimize an objective, given limited resources and competing constraint
- constraints are specified as (in)equalities

Example: Political Advertising

- Imagine you are a politician trying to win an election
- Your district has three different types of areas: Urban, suburban and rural, each with, respectively, 100,000, 200,000 and 50,000 registered voters
- Aim: at least half of the registered voters in each of the three regions should vote for you
- Possible Actions: Advertise on one of the primary issues which are (i) building more roads, (ii) gun control, (iii) farm subsidies and (iv) a gasoline tax dedicated to improve public transit.

Political Advertising Continued

policy	urban	suburban	rural
build roads	-2	5	3
gun control	8	2	-5
farm subsidies	0	0	10
gasoline tax	10	0	-2

The effects of policies on voters. Each entry describes the number of thousands of voters who could be won over by spending $1,000 on advertising support of a policy on a particular issue.

- Possible Solution:
  - $20,000 on advertising to building roads
  - $0 on advertising to gun control
  - $4,000 on advertising to farm subsidies
  - $9,000 on advertising to a gasoline tax
- Total cost: $33,000

What is the best possible strategy?

Towards a Linear Program

policy	urban	suburban	rural
build roads	-2	5	3
gun control	8	2	-5
farm subsidies	0	0	10
gasoline tax	10	0	-2

The effects of policies on voters. Each entry describes the number of thousands of voters who could be won (lost) over by spending $1,000 on advertising support of a policy on a particular issue.

- $x_1 =$ number of thousands of dollars spent on advertising on building roads
- $x_2 =$ number of thousands of dollars spent on advertising on gun control
- $x_3 =$ number of thousands of dollars spent on advertising on farm subsidies
- $x_4 =$ number of thousands of dollars spent on advertising on gasoline tax

Constraints:

- $-2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50$
- $5x_1 + 2x_2 + 0x_3 + 0x_4 \geq 100$
- $3x_1 - 5x_2 + 10x_3 - 2x_4 \geq 25$

Objective: Minimize $x_1 + x_2 + x_3 + x_4$

The Linear Program

Linear Program for the Advertising Problem

$$\begin{array}{lrcrcrcrcr}
\text{minimize} & x_1 & + & x_2 & + & x_3 & + & x_4 & & \\
\text{subject to} & & & & & & & & & \\
 & -2x_1 & + & 8x_2 & + & 0x_3 & + & 10x_4 & \geq & 50 \\
 & 5x_1 & + & 2x_2 & + & 0x_3 & + & 0x_4 & \geq & 100 \\
 & 3x_1 & - & 5x_2 & + & 10x_3 & - & 2x_4 & \geq & 25 \\
 & & & x_1, x_2, x_3, x_4 & & & & & \geq & 0
\end{array}$$

The solution of this linear program yields the optimal advertising strategy.

Formal Definition of Linear Program

- Given $a_1, a_2, \ldots, a_n$ and a set of variables $x_1, x_2, \ldots, x_n$, a linear function f is defined by

$$f(x_1, x_2, \ldots, x_n) = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n.$$

- Linear Equality: $f(x_1, x_2, \ldots, x_n) = b$
- Linear Inequality: $f(x_1, x_2, \ldots, x_n) \overset{\geq}{\leq} b$

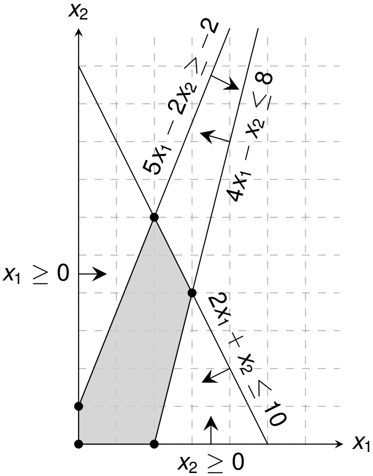
Linear Constraints

- Linear-Progamming Problem: either minimize or maximize a linear function subject to a set of linear constraints

A Small(er) Example

$$
\begin{array}{llcrcr}
\text{maximize} & x_1 & + & x_2 & & \\
\text{subject to} & & & & & \\
& 4x_1 & - & x_2 & \leq & 8 \\
& 2x_1 & + & x_2 & \leq & 10 \\
& 5x_1 & - & 2x_2 & \geq & -2 \\
& x_1, x_2 & & & \geq & 0
\end{array}
$$

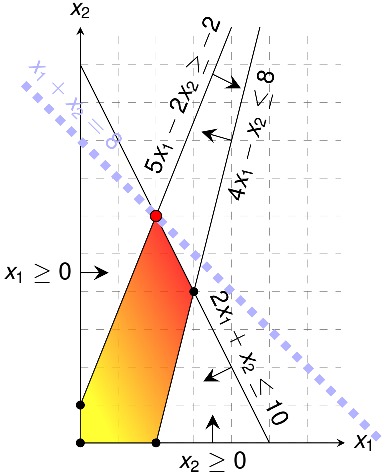
Any setting of x_1 and x_2 satisfying all constraints is a feasible solution



A Small(er) Example

$$
\begin{array}{llcrcr}
\text{maximize} & x_1 & + & x_2 & & \\
\text{subject to} & & & & & \\
 & 4x_1 & - & x_2 & \leq & 8 \\
 & 2x_1 & + & x_2 & \leq & 10 \\
 & 5x_1 & - & 2x_2 & \geq & -2 \\
 & x_1, x_2 & & & \geq & 0
\end{array}
$$

Graphical Procedure: Move the line $x_1 + x_2 = z$ as far up as possible.



While the same approach also works for higher-dimensions, we need to take a more systematic and algebraic procedure.

Outline

Introduction

Standard and Slack Forms

Formulating Problems as Linear Programs

Simplex Algorithm

Finding an Initial Solution

Standard and Slack Forms

Standard Form

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

Objective Function

subject to

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

$n + m$ Constraints

Non-Negativity Constraints

Standard Form (Matrix-Vector-Notation)

$$\text{maximize} \quad c^T x$$

Inner product of two vectors

subject to

$$Ax \leq b$$

Matrix-vector product

$$x \geq 0$$

Converting Linear Programs into Standard Form

Reasons for a LP not being in standard form:

1. The objective might be a minimization rather than maximization.
2. There might be variables without nonnegativity constraints.
3. There might be equality constraints
4. There might be inequality constraints (with $\leq$ instead of $\geq$)

Goal: Convert linear program into an equivalent program which is in standard form

Equivalence: a correspondence (not necessarily a bijection) between solutions so that their objective values are identical.

When switching from maximization to minimization, sign of objective value changes.

Converting into Standard Form (1/5)

Reasons for a LP not being in standard form:

1. The objective might be a minimization rather than maximization.

$$
\begin{array}{llrcrcl}
\boxed{\text{minimize} \quad -2x_1 + 3x_2} & & & & & & \\
\text{subject to} & & & & & & \\
 & & x_1 & + & x_2 & = & 7 \\
 & & x_1 & - & 2x_2 & \leq & 4 \\
 & & x_1 & & & \geq & 0
\end{array}
$$

$\downarrow$ Negate objective function

$$
\begin{array}{llrcrcl}
\boxed{\text{maximize} \quad 2x_1 - 3x_2} & & & & & & \\
\text{subject to} & & & & & & \\
 & & x_1 & + & x_2 & = & 7 \\
 & & x_1 & - & 2x_2 & \leq & 4 \\
 & & x_1 & & & \geq & 0
\end{array}
$$

Converting into Standard Form (2/5)

Reasons for a LP not being in standard form:

2. There might be variables without nonnegativity constraints.

$$
\begin{array}{llcrcl}
\text{maximize} & 2x_1 & - & 3x_2 & & \\
\text{subject to} & & & & & \\
 & x_1 & + & x_2 & = & 7 \\
 & x_1 & - & 2x_2 & \leq & 4 \\
 & \boxed{x_1 \qquad\qquad\qquad \geq \quad 0} & & & &
\end{array}
$$

$\big\downarrow$ Replace x_2 by two non-negative variables x_2' and x_2''

$$
\begin{array}{llcccl}
\text{maximize} & 2x_1 & - & \boxed{3x_2' + 3x_2''} & & \\
\text{subject to} & & & & & \\
 & x_1 & + & \boxed{x_2' - x_2''} & = & 7 \\
 & x_1 & - & \boxed{2x_2' + 2x_2''} & \leq & 4 \\
 & \boxed{x_1, x_2', x_2'' \qquad\qquad\qquad\qquad \geq \quad 0} & & & &
\end{array}
$$

Converting into Standard Form (3/5)

Reasons for a LP not being in standard form:

3. There might be equality constraints

$$\begin{array}{llllllll} \text{maximize} & 2x_1 & - & 3x_2' & + & 3x_2'' & & \\ \text{subject to} & & & & & & & \\ & \boxed{x_1} & + & x_2' & - & x_2'' & = & 7 \\ & x_1 & - & 2x_2' & + & 2x_2'' & \leq & 4 \\ & x_1, x_2', x_2'' & & & & & \geq & 0 \end{array}$$

Replace each equality by two inequalities.

$$\begin{array}{llllllll} \text{maximize} & 2x_1 & - & 3x_2' & + & 3x_2'' & & \\ \text{subject to} & & & & & & & \\ & x_1 & + & x_2' & - & x_2'' & \geq & 7 \\ & x_1 & + & x_2' & - & x_2'' & \leq & 7 \\ & x_1 & - & 2x_2' & + & 2x_2'' & \leq & 4 \\ & x_1, x_2', x_2'' & & & & & \geq & 0 \end{array}$$

Converting into Standard Form (4/5)

Reasons for a LP not being in standard form:

4. There might be inequality constraints (with $\leq$ instead of $\geq$)

$$\begin{array}{ll}
\text{maximize} & 2x_1 - 3x_2' + 3x_2'' \\
\text{subject to} & \\
& x_1 + x_2' - x_2'' \geq 7 \\
& \boxed{x_1 + x_2' - x_2'' \leq 7} \\
& x_1 - 2x_2' + 2x_2'' \leq 4 \\
& x_1, x_2', x_2'' \geq 0
\end{array}$$

$$\downarrow$$

Negate respective inequalities.

$$\begin{array}{ll}
\text{maximize} & 2x_1 - 3x_2' + 3x_2'' \\
\text{subject to} & \\
& x_1 + x_2' - x_2'' \geq 7 \\
& \boxed{-x_1 - x_2' + x_2'' \geq -7} \\
& x_1 - 2x_2' + 2x_2'' \leq 4 \\
& x_1, x_2', x_2'' \geq 0
\end{array}$$

Converting into Standard Form (5/5)

Rename variable names (for consistency).

$$\begin{array}{lrcrcrcr}
\text{maximize} & 2x_1 & - & 3x_2 & + & 3x_3 & & \\
\text{subject to} & & & & & & & \\
& x_1 & + & x_2 & - & x_3 & \geq & 7 \\
& -x_1 & - & x_2 & + & x_3 & \geq & -7 \\
& x_1 & - & 2x_2 & + & 2x_3 & \leq & 4 \\
& \multicolumn{5}{l}{x_1, x_2, x_3} & \geq & 0
\end{array}$$

It is always possible to convert a linear program into standard form.

Converting Standard Form into Slack Form (1/3)

Goal: Convert standard form into slack form, where all constraints except for the non-negativity constraints are equalities.

For the simplex algorithm, it is more convenient to work with equality constraints.

Introducing Slack Variables

- Let $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ be an inequality constraint
- Introduce a slack variable s by

$$s = b_i - \sum_{j=1}^{n} a_{ij} x_j$$

$$s \geq 0.$$

s measures the slack between the two sides of the inequality.

- Denote slack variable of the ith inequality by x_{n+i}

Converting Standard Form into Slack Form (2/3)

$$
\begin{array}{lrcrcrcr}
\text{maximize} & 2x_1 & - & 3x_2 & + & 3x_3 & & \\
\text{subject to} & & & & & & & \\
& x_1 & + & x_2 & - & x_3 & \geq & 7 \\
& -x_1 & - & x_2 & + & x_3 & \geq & -7 \\
& x_1 & - & 2x_2 & + & 2x_3 & \leq & 4 \\
& \multicolumn{5}{l}{x_1, x_2, x_3} & \geq & 0
\end{array}
$$

Introduce slack variables

$$
\begin{array}{lrcrcrcrcr}
\text{maximize} & & & & & 2x_1 & - & 3x_2 & + & 3x_3 \\
\text{subject to} & & & & & & & & & \\
& x_4 & = & 7 & - & x_1 & - & x_2 & + & x_3 \\
& x_5 & = & -7 & + & x_1 & + & x_2 & - & x_3 \\
& x_6 & = & 4 & - & x_1 & + & 2x_2 & - & 2x_3 \\
& \multicolumn{5}{l}{x_1, x_2, x_3, x_4, x_5, x_6} & \geq & 0 & &
\end{array}
$$

Converting Standard Form into Slack Form (3/3)

$$
\begin{array}{lrcrcrcrcr}
\text{maximize} & & & & & 2x_1 & - & 3x_2 & + & 3x_3 \\
\text{subject to} & & & & & & & & & \\
 & x_4 & = & 7 & - & x_1 & - & x_2 & + & x_3 \\
 & x_5 & = & -7 & + & x_1 & + & x_2 & - & x_3 \\
 & x_6 & = & 4 & - & x_1 & + & 2x_2 & - & 2x_3 \\
 & \multicolumn{5}{r}{x_1, x_2, x_3, x_4, x_5, x_6} & \geq & 0 & &
\end{array}
$$

Use variable z to denote objective function and omit the nonnegativity constraints.

$$
\begin{array}{rcrcrcrcr}
z & = & & & 2x_1 & - & 3x_2 & + & 3x_3 \\
x_4 & = & 7 & - & x_1 & - & x_2 & + & x_3 \\
x_5 & = & -7 & + & x_1 & + & x_2 & - & x_3 \\
x_6 & = & 4 & - & x_1 & + & 2x_2 & - & 2x_3
\end{array}
$$

This is called slack form.

Basic and Non-Basic Variables

$$\begin{array}{rcrcrcrcr}
z & = & & & 2x_1 & - & 3x_2 & + & 3x_3 \\
x_4 & = & 7 & - & x_1 & - & x_2 & + & x_3 \\
x_5 & = & -7 & + & x_1 & + & x_2 & - & x_3 \\
x_6 & = & 4 & - & x_1 & + & 2x_2 & - & 2x_3
\end{array}$$

Basic Variables: $B = \{4, 5, 6\}$

Non-Basic Variables: $N = \{1, 2, 3\}$

Slack Form (Formal Definition)

Slack form is given by a tuple (N, B, A, b, c, v) so that

$$z = v + \sum_{j \in N} c_j x_j$$

$$x_i = b_i - \sum_{j \in N} a_{ij} x_j \qquad \text{for } i \in B,$$

and all variables are non-negative.

Variables on the right hand side are indexed by the entries of N.

Slack Form (Example)

$$\begin{array}{rclcccccc} z & = & 28 & - & \frac{x_3}{6} & - & \frac{x_5}{6} & - & \frac{2x_6}{3} \\ x_1 & = & 8 & + & \frac{x_3}{6} & + & \frac{x_5}{6} & - & \frac{x_6}{3} \\ x_2 & = & 4 & - & \frac{8x_3}{3} & - & \frac{2x_5}{3} & + & \frac{x_6}{3} \\ x_4 & = & 18 & - & \frac{x_3}{2} & + & \frac{x_5}{2} & & \end{array}$$

Slack Form Notation

- $B = \{1, 2, 4\}$, $N = \{3, 5, 6\}$
- $$A = \begin{pmatrix} a_{13} & a_{15} & a_{16} \\ a_{23} & a_{25} & a_{26} \\ a_{43} & a_{45} & a_{46} \end{pmatrix} = \begin{pmatrix} -1/6 & -1/6 & 1/3 \\ 8/3 & 2/3 & -1/3 \\ 1/2 & -1/2 & 0 \end{pmatrix}$$
- $$b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 8 & 4 & 18 \end{pmatrix}, \quad c = \begin{pmatrix} c_3 & c_5 & c_6 \end{pmatrix}^T = \begin{pmatrix} -1/6 & -1/6 & -2/3 \end{pmatrix}^T$$
- $v = 28$

The Structure of Optimal Solutions

Definition

A point x is a vertex if it cannot be represented as a strict convex combination of two other points in the feasible set.

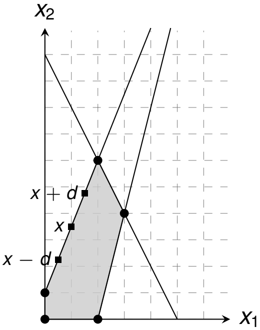
The set of feasible solutions is a convex set.

Theorem

If there exists an optimal solution, it occurs at a vertex of the polygon.

Proof:

- Let x be an optimal solution which is not a vertex $\Rightarrow \exists$ vector d so that $x - d$ and $x + d$ are feasible
- Since $A(x + d) = b$ and $Ax = b \Rightarrow Ad = 0$
- W.l.o.g. assume $c^T d \leq 0$ (otherwise replace d by $-d$)
- Consider $x + \lambda d$ as a function of $\lambda \geq 0$



- Case 1: There exists j with $d_j < 0$
  - Increase λ from 0 to λ' until a new entry of $x + \lambda d$ becomes zero
  - $x + \lambda' d$ feasible, since $A(x + \lambda' d) = Ax = b$ and $x + \lambda' d \geq 0$
  - $c^T (x + \lambda' d) = c^T x + c^T \lambda' d \leq c^T x$

The Structure of Optimal Solutions

Definition

A point x is a vertex if it cannot be represented as a strict convex combination of two other points in the feasible set.

The set of feasible solutions is a convex set.

Theorem

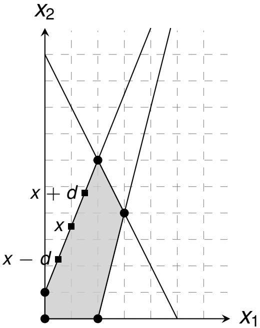
If there exists an optimal solution, it occurs at a vertex of the polygon.

Proof:

- Let x be an optimal solution which is not a vertex $\Rightarrow \exists$ vector d so that $x - d$ and $x + d$ are feasible
- Since $A(x + d) = b$ and $Ax = b \Rightarrow Ad = 0$
- W.l.o.g. assume $c^T d \leq 0$ (otherwise replace d by $-d$)
- Consider $x + \lambda d$ as a function of $\lambda \geq 0$

- Case 2: For all j, $d_j \geq 0$
  - $x + \lambda d$ is feasible for all $\lambda \geq 0$: $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
  - If $\lambda \to \infty$, then $c^T(x + \lambda d) \to \infty$

  $\Rightarrow$ This contradicts the assumption that there exists an optimal solution. $\square$



Outline

Introduction

Standard and Slack Forms

Formulating Problems as Linear Programs

Simplex Algorithm

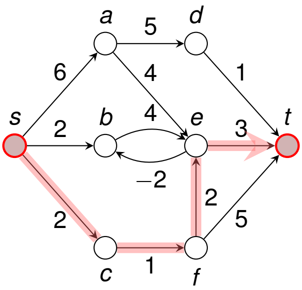
Finding an Initial Solution

Shortest Paths

Single-Pair Shortest Path Problem

- Given: directed graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}^+$, pair of vertices $s, t \in V$
- Goal: Find a path of minimum weight from s to t in G

$p = (v_0 = s, v_1, \dots, v_k = t)$ such that $w(p) = \sum_{i=1}^{k} w(v_{k-1}, v_k)$ is minimized.



Shortest Paths as LP

$$\begin{array}{llll} \text{maximize} & d_t & & \\ \text{subject to} & & & \\ & d_v \leq d_u + w(u,v) & & \text{for each edge } (u,v) \in E, \\ & d_s = 0. & & \end{array}$$

Recall: When BELLMAN-FORD terminates, all these inequalities are satisfied.

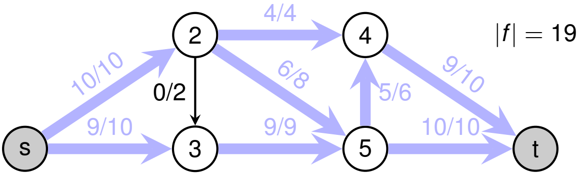
this is a maximization problem!

Solution $\overline{d}$ satisfies $\overline{d}_v = \min_{u:\ (u,v)\in E} \left\{ \overline{d}_u + w(u,v) \right\}$

Maximum Flow

Maximum Flow Problem

- Given: directed graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{R}^+$, pair of vertices $s, t \in V$
- Goal: Find a maximum flow $f : V \times V \to \mathbb{R}$ from s to t which satisfies the capacity constraints and flow conservation



Maximum Flow as LP

$$
\begin{array}{lrcll}
\text{maximize} & \sum_{v \in V} f_{sv} & - & \sum_{v \in V} f_{vs} & \\
\text{subject to} & & & & \\
 & f_{uv} & \leq & c(u,v) & \text{for each } u, v \in V, \\
 & \sum_{v \in V} f_{vu} & = & \sum_{v \in V} f_{uv} & \text{for each } u \in V \setminus \{s, t\}, \\
 & f_{uv} & \geq & 0 & \text{for each } u, v \in V.
\end{array}
$$

Minimum-Cost Flow

Generalization of the Maximum Flow Problem

Minimum-Cost-Flow Problem

- Given: directed graph $G = (V, E)$ with capacities $c : E \to \mathbb{R}^+$, pair of vertices $s, t \in V$, cost function $a : E \to \mathbb{R}^+$, flow demand of d units
- Goal: Find a flow $f : V \times V \to \mathbb{R}$ from s to t with $|f| = d$ while minimising the total cost $\sum_{(u,v)\in E} a(u,v) f_{uv}$ incurrred by the flow.

Optimal Solution with total cost:

$$\sum_{(u,v)\in E} a(u,v) f_{uv} = (2\cdot2)+(5\cdot2)+(3\cdot1)+(7\cdot1)+(1\cdot3) = 27$$

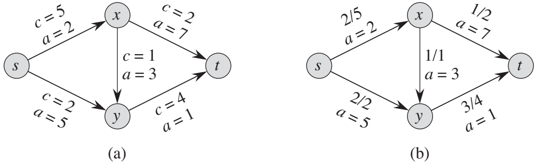


Figure 29.3 **(a)** An example of a minimum-cost-flow problem. We denote the capacities by c and the costs by a. Vertex s is the source and vertex t is the sink, and we wish to send 4 units of flow from s to t. **(b)** A solution to the minimum-cost flow problem in which 4 units of flow are sent from s to t. For each edge, the flow and capacity are written as flow/capacity.

Minimum-Cost Flow as a LP

Minimum Cost Flow as LP

minimize $\sum_{(u,v)\in E} a(u,v) f_{uv}$

subject to

$$\begin{aligned} f_{uv} &\leq c(u,v) && \text{for each } u,v \in V, \\ \sum_{v\in V} f_{vu} - \sum_{v\in V} f_{uv} &= 0 && \text{for each } u \in V \setminus \{s,t\}, \\ \sum_{v\in V} f_{sv} - \sum_{v\in V} f_{vs} &= d\,, \\ f_{uv} &\geq 0 && \text{for each } u,v \in V. \end{aligned}$$

Real power of Linear Programming comes from the ability to solve **new problems**!

Outline

Introduction

Standard and Slack Forms

Formulating Problems as Linear Programs

Simplex Algorithm

Finding an Initial Solution

Simplex Algorithm: Introduction

Simplex Algorithm

- classical method for solving linear programs (Dantzig, 1947)
- usually fast in practice although worst-case runtime not polynomial
- iterative procedure somewhat similar to Gaussian elimination

Basic Idea:

- Each iteration corresponds to a “basic solution” of the slack form
- All non-basic variables are 0, and the basic variables are determined from the equality constraints
- Each iteration converts one slack form into an equivalent one while the objective value will not decrease
- Conversion (“pivoting”) is achieved by switching the roles of one basic and one non-basic variable

In that sense, it is a greedy algorithm.

Extended Example: Conversion into Slack Form

$$\begin{array}{llllllll}
\text{maximize} & 3x_1 & + & x_2 & + & 2x_3 & & \\
\text{subject to} & & & & & & & \\
 & x_1 & + & x_2 & + & 3x_3 & \leq & 30 \\
 & 2x_1 & + & 2x_2 & + & 5x_3 & \leq & 24 \\
 & 4x_1 & + & x_2 & + & 2x_3 & \leq & 36 \\
 & & & x_1, x_2, x_3 & & & \geq & 0
\end{array}$$

Conversion into slack form

$$\begin{array}{rcrcrcrcr}
z & = & & & 3x_1 & + & x_2 & + & 2x_3 \\
x_4 & = & 30 & - & x_1 & - & x_2 & - & 3x_3 \\
x_5 & = & 24 & - & 2x_1 & - & 2x_2 & - & 5x_3 \\
x_6 & = & 36 & - & 4x_1 & - & x_2 & - & 2x_3
\end{array}$$

Extended Example: Iteration 1

$$\begin{aligned} z &= & & 3x_1 &+& x_2 &+& 2x_3 \\ x_4 &= 30 &-& x_1 &-& x_2 &-& 3x_3 \\ x_5 &= 24 &-& 2x_1 &-& 2x_2 &-& 5x_3 \\ x_6 &= 36 &-& 4x_1 &-& x_2 &-& 2x_3 \end{aligned}$$

Basic solution: $(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_6}) = (0, 0, 0, 30, 24, 36)$

This basic solution is **feasible**

Objective value is 0.

Extended Example: Iteration 1

Increasing the value of x_1 would increase the objective value.

$$\begin{array}{rcrcrcrcr} z &=& & & 3x_1 &+& x_2 &+& 2x_3 \\ x_4 &=& 30 &-& x_1 &-& x_2 &-& 3x_3 \\ x_5 &=& 24 &-& 2x_1 &-& 2x_2 &-& 5x_3 \end{array}$$

$$\boxed{x_6 = 36 - 4x_1 - x_2 - 2x_3}$$

The third constraint is the tightest and limits how much we can increase x_1.

Switch roles of x_1 and x_6:

- Solving for x_1 yields:

$$x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}.$$

- Substitute this into x_1 in the other three equations

Extended Example: Iteration 2

Increasing the value of x_3 would increase the objective value.

$$\begin{array}{lclclclcl} z & = & 27 & + & \frac{x_2}{4} & + & \frac{x_3}{2} & - & \frac{3x_6}{4} \\ x_1 & = & 9 & - & \frac{x_2}{4} & - & \frac{x_3}{2} & - & \frac{x_6}{4} \\ x_4 & = & 21 & - & \frac{3x_2}{4} & - & \frac{5x_3}{2} & + & \frac{x_6}{4} \\ x_5 & = & 6 & - & \frac{3x_2}{2} & - & 4x_3 & + & \frac{x_6}{2} \end{array}$$

Basic solution: $(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_6}) = (9, 0, 0, 21, 6, 0)$ with objective value 27

Extended Example: Iteration 2

$$\begin{array}{rcrcrcrcr} z &=& 27 &+& \frac{x_2}{4} &+& \frac{x_3}{2} &-& \frac{3x_6}{4} \\ x_1 &=& 9 &-& \frac{x_2}{4} &-& \frac{x_3}{2} &-& \frac{x_6}{4} \\ x_4 &=& 21 &-& \frac{3x_2}{4} &-& \frac{5x_3}{2} &+& \frac{x_6}{4} \\ x_5 &=& 6 &-& \frac{3x_2}{2} &-& 4x_3 &+& \frac{x_6}{2} \end{array}$$

The third constraint is the tightest and limits how much we can increase x_3.

Switch roles of x_3 and x_5:

- Solving for x_3 yields:

$$x_3 = \frac{3}{2} - \frac{3x_2}{8} - \frac{x_5}{4} - \frac{x_6}{8}.$$

- Substitute this into x_3 in the other three equations

Extended Example: Iteration 3

Increasing the value of x_2 would increase the objective value.

$$\begin{array}{rcrcrcrcr}
z & = & \frac{111}{4} & + & \frac{x_2}{16} & - & \frac{x_5}{8} & - & \frac{11x_6}{16} \\
x_1 & = & \frac{33}{4} & - & \frac{x_2}{16} & + & \frac{x_5}{8} & - & \frac{5x_6}{16} \\
x_3 & = & \frac{3}{2} & - & \frac{3x_2}{8} & - & \frac{x_5}{4} & + & \frac{x_6}{8} \\
x_4 & = & \frac{69}{4} & + & \frac{3x_2}{16} & + & \frac{5x_5}{8} & - & \frac{x_6}{16}
\end{array}$$

Basic solution: $(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_6}) = (\frac{33}{4}, 0, \frac{3}{2}, \frac{69}{4}, 0, 0)$ with objective value $\frac{111}{4} = 27.75$

Extended Example: Iteration 3

$$\begin{array}{lcccccccc}
z & = & \frac{111}{4} & + & \frac{x_2}{16} & - & \frac{x_5}{8} & - & \frac{11x_6}{16} \\
x_1 & = & \frac{33}{4} & - & \frac{x_2}{16} & + & \frac{x_5}{8} & - & \frac{5x_6}{16} \\
x_3 & = & \frac{3}{2} & - & \frac{3x_2}{8} & - & \frac{x_5}{4} & + & \frac{x_6}{8} \\
x_4 & = & \frac{69}{4} & + & \frac{3x_2}{16} & + & \frac{5x_5}{8} & - & \frac{x_6}{16}
\end{array}$$

The second constraint is the tightest and limits how much we can increase x_2.

Switch roles of x_2 and x_3:

- Solving for x_2 yields:

$$x_2 = 4 - \frac{8x_2}{3} - \frac{2x_5}{3} + \frac{x_6}{3}.$$

- Substitute this into x_2 in the other three equations

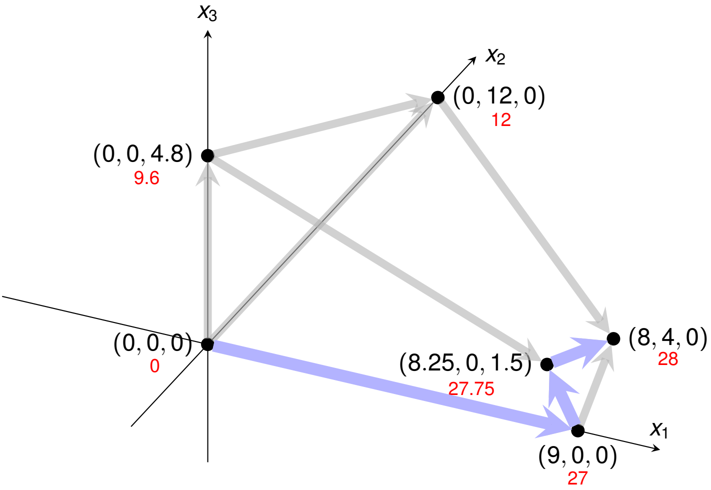
Extended Example: Iteration 4

All coefficients are negative, and hence this basic solution is **optimal**!

$$\begin{aligned}
z &= 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3} \\
x_1 &= 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3} \\
x_2 &= 4 - \frac{8x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3} \\
x_4 &= 18 - \frac{x_3}{2} + \frac{x_5}{2}
\end{aligned}$$

Basic solution: $(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_6}) = (8, 4, 0, 18, 0, 0)$ with objective value 28

Extended Example: Visualization of SIMPLEX



Extended Example: Alternative Runs (1/2)

$$\begin{array}{rcrcrcrcr}
z & = & & & 3x_1 & + & x_2 & + & 2x_3 \\
x_4 & = & 30 & - & x_1 & - & x_2 & - & 3x_3 \\
x_5 & = & 24 & - & 2x_1 & - & 2x_2 & - & 5x_3 \\
x_6 & = & 36 & - & 4x_1 & - & x_2 & - & 2x_3
\end{array}$$

$\Downarrow$ Switch roles of x_2 and x_5

$$\begin{array}{rcrcrcrcr}
z & = & 12 & + & 2x_1 & - & \frac{x_3}{2} & - & \frac{x_5}{2} \\
x_2 & = & 12 & - & x_1 & - & \frac{5x_3}{2} & - & \frac{x_5}{2} \\
x_4 & = & 18 & - & x_2 & - & \frac{x_3}{2} & + & \frac{x_5}{2} \\
x_6 & = & 24 & - & 3x_1 & + & \frac{x_3}{2} & + & \frac{x_5}{2}
\end{array}$$

$\Downarrow$ Switch roles of x_1 and x_6

$$\begin{array}{rcrcrcrcr}
z & = & 28 & - & \frac{x_3}{6} & - & \frac{x_5}{6} & - & \frac{2x_6}{3} \\
x_1 & = & 8 & + & \frac{x_3}{6} & + & \frac{x_5}{6} & - & \frac{x_6}{3} \\
x_2 & = & 4 & - & \frac{8x_3}{3} & - & \frac{2x_5}{3} & + & \frac{x_6}{3} \\
x_4 & = & 18 & - & x_2 & - & \frac{x_3}{2} & + & \frac{x_5}{2}
\end{array}$$

Extended Example: Alternative Runs (2/2)

$$\begin{array}{rcrcrcrcr}
z & = & & & 3x_1 & + & x_2 & + & 2x_3 \\
x_4 & = & 30 & - & x_1 & - & x_2 & - & 3x_3 \\
x_5 & = & 24 & - & 2x_1 & - & 2x_2 & - & 5x_3 \\
x_6 & = & 36 & - & 4x_1 & - & x_2 & - & 2x_3
\end{array}$$

Switch roles of x_3 and x_5

$$\begin{array}{rcrcrcrcr}
z & = & \frac{48}{5} & + & \frac{11x_1}{5} & + & \frac{x_2}{5} & - & \frac{2x_5}{5} \\
x_4 & = & \frac{78}{5} & + & \frac{x_1}{5} & + & \frac{x_2}{5} & + & \frac{3x_5}{5} \\
x_3 & = & \frac{24}{5} & - & \frac{2x_1}{5} & - & \frac{2x_2}{5} & - & \frac{x_5}{5} \\
x_6 & = & \frac{132}{5} & - & \frac{16x_1}{5} & - & \frac{x_2}{5} & + & \frac{2x_3}{5}
\end{array}$$

Switch roles of x_1 and x_6

$$\begin{array}{rcrcrcrcr}
z & = & \frac{111}{4} & + & \frac{x_2}{16} & - & \frac{x_5}{8} & - & \frac{11x_6}{16} \\
x_1 & = & \frac{33}{4} & - & \frac{x_2}{16} & + & \frac{x_5}{8} & - & \frac{5x_6}{16} \\
x_3 & = & \frac{3}{2} & - & \frac{3x_2}{8} & - & \frac{x_5}{4} & + & \frac{x_6}{8} \\
x_4 & = & \frac{69}{4} & + & \frac{3x_2}{16} & + & \frac{5x_5}{8} & - & \frac{x_6}{16}
\end{array}$$

Switch roles of x_2 and x_3

$$\begin{array}{rcrcrcrcr}
z & = & 28 & - & \frac{x_3}{6} & - & \frac{x_5}{6} & - & \frac{2x_6}{3} \\
x_1 & = & 8 & + & \frac{x_3}{6} & + & \frac{x_5}{6} & - & \frac{x_6}{3} \\
x_2 & = & 4 & - & \frac{8x_3}{3} & - & \frac{2x_5}{3} & + & \frac{x_6}{3} \\
x_4 & = & 18 & - & \frac{x_3}{2} & + & \frac{x_5}{2} & &
\end{array}$$

The Pivot Step Formally

```
PIVOT($N, B, A, b, c, v, l, e$)
// Compute the coefficients of the equation for new basic variable $x_e$.
let $\widehat{A}$ be a new $m \times n$ matrix
$\widehat{b}_e = b_l / a_{le}$
for each $j \in N - \{e\}$
    $\widehat{a}_{ej} = a_{lj} / a_{le}$
$\widehat{a}_{el} = 1/a_{le}$
// Compute the coefficients of the remaining constraints.
for each $i \in B - \{l\}$
    $\widehat{b}_i = b_i - a_{ie}\widehat{b}_e$
    for each $j \in N - \{e\}$
        $\widehat{a}_{ij} = a_{ij} - a_{ie}\widehat{a}_{ej}$
    $\widehat{a}_{il} = -a_{ie}\widehat{a}_{el}$
// Compute the objective function.
$\widehat{v} = v + c_e\widehat{b}_e$
for each $j \in N - \{e\}$
    $\widehat{c}_j = c_j - c_e\widehat{a}_{ej}$
$\widehat{c}_l = -c_e\widehat{a}_{el}$
// Compute new sets of basic and nonbasic variables.
$\widehat{N} = N - \{e\} \cup \{l\}$
$\widehat{B} = B - \{l\} \cup \{e\}$
return $(\widehat{N}, \widehat{B}, \widehat{A}, \widehat{b}, \widehat{c}, \widehat{v})$
```

Need that $a_{le} \neq 0$!

Rewrite “tight” equation for enterring variable x_e.

Substituting x_e into other equations.

Substituting x_e into objective function.

Update non-basic and basic variables

Effect of the Pivot Step

Lemma 29.1

Consider a call to PIVOT(N, B, A, b, c, v, l, e) in which $a_{le} \neq 0$. Let the values returned from the call be $(\widehat{N}, \widehat{B}, \widehat{A}, \widehat{b}, \widehat{c}, \widehat{v})$, and let $\overline{x}$ denote the basic solution after the call. Then

1. $\overline{x}_j = 0$ for each $j \in \widehat{N}$.
2. $\overline{x}_e = b_l / a_{le}$.
3. $\overline{x}_i = b_i - a_{ie}\widehat{b}_e$ for each $i \in \widehat{B} \setminus \{e\}$.

Proof:

1. holds since the basic solution always sets all non-basic variables to zero.
2. When we set each non-basic variable to 0 in a constraint

$$x_i = \widehat{b}_i - \sum_{j \in \widehat{N}} \widehat{a}_{ij} x_j,$$

   we have $\overline{x}_i = \widehat{b}_i$ for each $i \in \widehat{B}$. Hence $\overline{x}_e = \widehat{b}_e = b_l / a_{le}$.
3. After the substituting in the other constraints, we have

$$\overline{x}_i = \widehat{b}_i = b_i - a_{ie}\widehat{b}_e. \qquad \square$$

Formalizing the Simplex Algorithm: Questions

Questions:

- How do we determine whether a linear program is feasible?
- What do we do if the linear program is feasible, but the initial basic solution is not feasible?
- How do we determine whether a linear program is unbounded?
- How do we choose the entering and leaving variables?

Example before was a particularly nice one!

The formal procedure SIMPLEX

```
SIMPLEX$(A, b, c)$
 1  $(N, B, A, b, c, v) = $ INITIALIZE-SIMPLEX$(A, b, c)$
 2  let $\Delta$ be a new vector of length $n$
 3  while some index $j \in N$ has $c_j > 0$
 4      choose an index $e \in N$ for which $c_e > 0$
 5      for each index $i \in B$
 6          if $a_{ie} > 0$
 7              $\Delta_i = b_i / a_{ie}$
 8          else $\Delta_i = \infty$
 9      choose an index $l \in B$ that minimizes $\Delta_i$
10      if $\Delta_l == \infty$
11          return "unbounded"
12      else $(N, B, A, b, c, v) = $ PIVOT$(N, B, A, b, c, v, l, e)$
13  for $i = 1$ to $n$
14      if $i \in B$
15          $\bar{x}_i = b_i$
16      else $\bar{x}_i = 0$
17  return $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$
```

Returns a slack form with a feasible basic solution (if it exists)

Main Loop:

- terminates if all coefficients in objective function are negative
- Line 4 picks enterring variable x_e with negative coefficient
- Lines $6 - 9$ pick the tightest constraint, associated with x_l
- Line 11 returns “unbounded” if there are no constraints
- Line 12 calls PIVOT, switching roles of x_l and x_e

Return corresponding solution.

The formal procedure SIMPLEX

```
SIMPLEX($A, b, c$)
 1  $(N, B, A, b, c, \nu) = $ INITIALIZE-SIMPLEX($A, b, c$)
 2  let $\Delta$ be a new vector of length $n$
 3  while some index $j \in N$ has $c_j > 0$
 4      choose an index $e \in N$ for which $c_e > 0$
 5      for each index $i \in B$
 6          if $a_{ie} > 0$
 7              $\Delta_i = b_i / a_{ie}$
 8          else $\Delta_i = \infty$
 9      choose an index $l \in B$ that minimizes $\Delta_i$
10      if $\Delta_l == \infty$
11          return "unbounded"
```

Proof is based on the following three-part loop invariant:

1. the slack form is always equivalent to the one returned by INITIALIZE-SIMPLEX,
2. for each $i \in B$, we have $b_i \geq 0$,
3. the basic solution associated with the (current) slack form is feasible.

Lemma 29.2

Suppose the call to INITIALIZE-SIMPLEX in line 1 returns a slack form for which the basic solution is feasible. Then if SIMPLEX returns a solution, it is a feasible solution. If SIMPLEX returns "unbounded", the linear program is unbounded.

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

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

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

Pivot with x_3 entering and x_5 leaving

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

Termination and Running Time

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

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

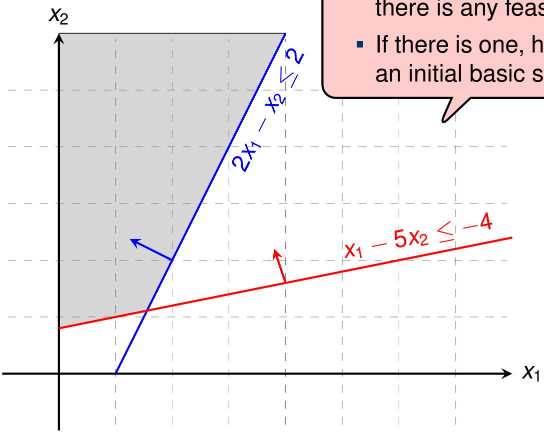
Conversion into slack form

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

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

Geometric Illustration

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



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

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

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

$$\Big\downarrow$$

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}
- "$\Leftarrow$": 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. □

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 $L_{\text{aux}}$ by adding $-x_0$ to the left-hand side of each constraint
    and setting the objective function to $-x_0$
let $(N, B, A, b, c, \nu)$ be the resulting slack form for $L_{\text{aux}}$
$l = n + k$
// $L_{\text{aux}}$ has $n+1$ nonbasic variables and $m$ basic variables.
$(N, B, A, b, c, \nu) = $ PIVOT$(N, B, A, b, c, \nu, l, 0)$
// The basic solution is now feasible for $L_{\text{aux}}$.
iterate the while loop of lines 3–12 of SIMPLEX until an optimal solution
    to $L_{\text{aux}}$ is found
if the optimal solution to $L_{\text{aux}}$ 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 $L_{\text{aux}}$, 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.

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

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

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)

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

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)

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

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

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

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

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.

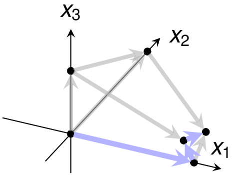
Linear Programming and Simplex: Summary

Linear Programming

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

Simplex Algorithm

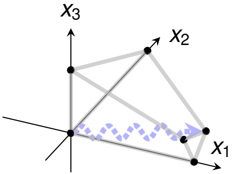
- In practice: usually terminates in polynomial time, i.e., $O(m+n)$
- In theory: even with anti-cycling may need exponential time



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!)



IV. Approximation Algorithms: Covering Problems

Thomas Sauerwald

Easter 2015

Outline

Introduction

Vertex Cover

The Set-Covering Problem

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:

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

- 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

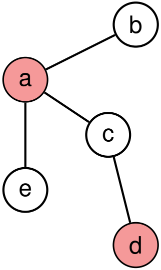
The Vertex-Cover Problem

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.

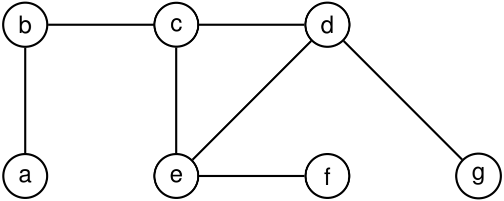


Applications:

- 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 or hypergraphs

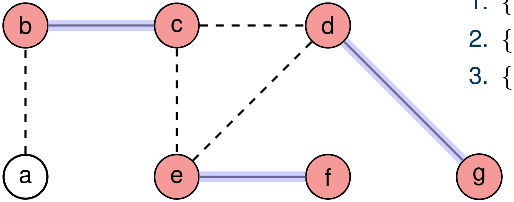
An Approximation Algorithm based on Greedy

```
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$(G)$
$C = \emptyset$
$E' = G.E$
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 $C$
```



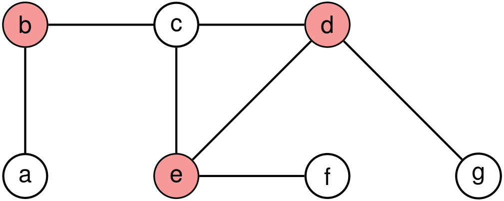
Edges removed from E':

1. $\{b, c\}$
2. $\{e, f\}$
3. $\{d, g\}$

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)
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 $C$
```

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.

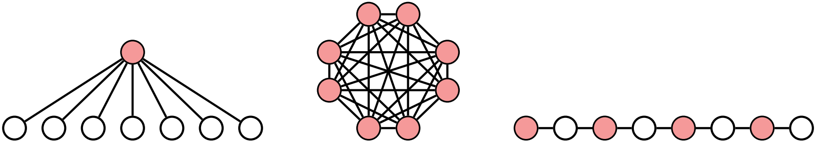
Proof:

- 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 $|C|$: $\boxed{|C| = 2|A| \leq 2|C^*|.}$ □

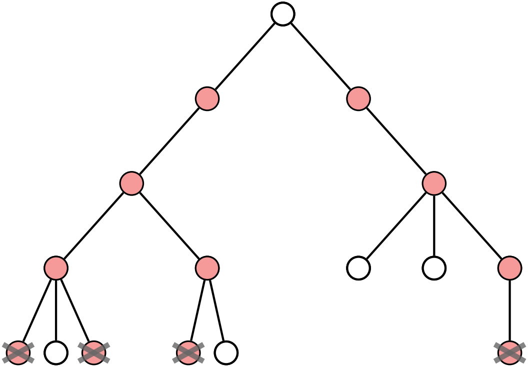
Solving Special Cases

Strategies to cope with NP-complete problems

1. If inputs 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.



Vertex Cover on Trees



There exists an optimal vertex cover which does not include any leaves.

Exchange-Argument: Replace any leaf in the cover by its parent.

Solving Vertex Cover on Trees

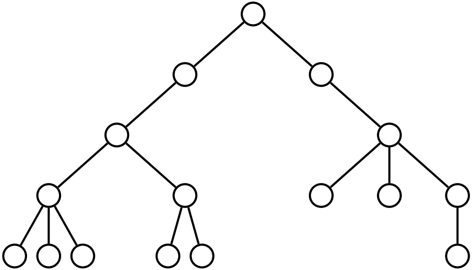
There exists an optimal vertex cover which does not include any leaves.

```
VERTEX-COVER-TREES(G)
$C = \emptyset$
while $\exists$ leaves in $G$
    Add all parents to $C$
    Remove all leafs and their parents from $G$
return $C$
```

Clear: Running time is $O(V)$, and the returned solution is a vertex cover.

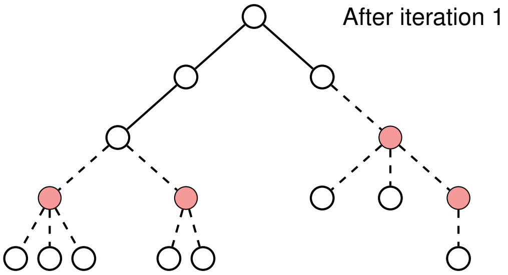
Solution is also optimal. (Use inductively the existence of an optimal vertex cover without leaves)

Execution on a Small Example



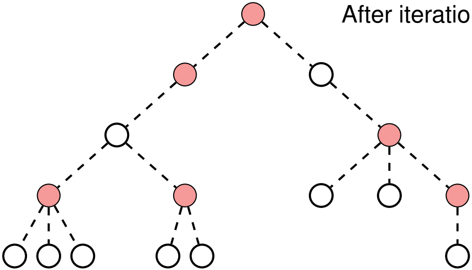
```
VERTEX-COVER-TREES(G)
$C = \emptyset$
while $\exists$ leaves in $G$
    Add all parents to $C$
    Remove all leafs and their parents from $G$
return $C$
```

Execution on a Small Example



```
VERTEX-COVER-TREES(G)
C = $\emptyset$
while $\exists$ leaves in $G$
    Add all parents to $C$
    Remove all leafs and their parents from $G$
return $C$
```

Execution on a Small Example



```
VERTEX-COVER-TREES(G)
$C = \emptyset$
while $\exists$ leaves in $G$
    Add all parents to $C$
    Remove all leafs and their parents from $G$
return $C$
```

Problem can be also solved on bipartite graphs, using Max-Flows and Min-Cuts.

Exact Algorithms

Such algorithms are called exact algorithms.

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.

Focus on instances of where the minimum vertex cover is small, that is, smaller than some given integer k.

Simple Brute-Force Search would take $\approx \binom{n}{k} = \Theta(n^k)$ time.

Towards a more efficient Search

Substructure Lemma

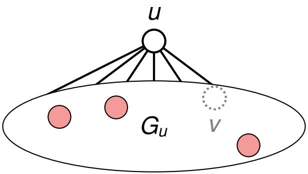
Consider a graph $G = (V, E)$, edge $(u, v) \in E(G)$ and integer $k \geq 1$. Let G_u be the graph obtained by deleting u and its incident edges (G_v is defined similarly). Then G has a vertex cover of size k if and only if G_u or G_v (or both) have a vertex cover of size $k - 1$.

Reminiscent of Dynamic Programming.

Proof:

$\Leftarrow$ Assume G_u has a vertex cover C_u of size $k - 1$.
Adding u yields a vertex cover of G which is of size k

$\Rightarrow$ Assume G has a vertex cover C of size k, which contains, say u.
Removing u from C yields a vertex cover of G_u which is of size $k - 1$. $\square$



A More Efficient Search Algorithm

```
$\textsc{Vertex-Cover-Search}(G, k)$
Pick an arbitrary edge $(u, v) \in E$
$S_1 = \textsc{Vertex-Cover-Search}(G_u, k-1)$
$S_2 = \textsc{Vertex-Cover-Search}(G_v, k-1)$
if $S_1 \neq \emptyset$ return $S_1 \cup \{u\}$
if $S_2 \neq \emptyset$ return $S_2 \cup \{v\}$
return $\emptyset$
```

Correctness follows by the Substructure Lemma and induction.

Running time:

- Depth $\log_2 k$, branching factor 2 $\Rightarrow$ total number of calls is $O(2^k)$
- $O(E)$ work per recursive call
- Total runtime: $O(2^k \cdot E)$.

exponential in k, but much better than $\Theta(n^k)$ (i.e., still polynomial for $k = O(\log n)$)

Outline

Introduction

Vertex Cover

The Set-Covering Problem

The Set-Covering Problem

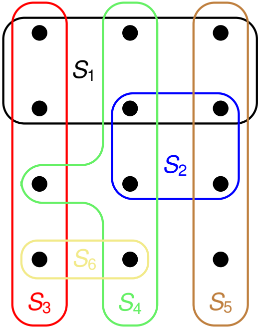
Set Cover Problem

- Given: set X of size n and family of subsets $\mathcal{F}$
- Goal: Find a minimum-size subset $\mathcal{C} \subseteq \mathcal{F}$

s.t. $$X = \bigcup_{S \in \mathcal{C}} S.$$

Number of sets (and not elements)

Only solvable if $\bigcup_{S \in \mathcal{F}} S = X$!



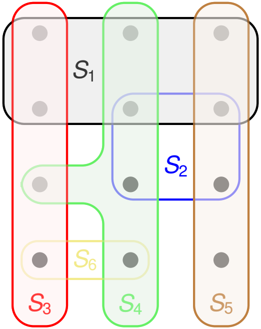
Remarks:

- generalisation of the vertex-cover problem and hence also NP-hard.
- models resource allocation problems

Greedy

Strategy: Pick the set S that covers the largest number of uncovered elements.

```
GREEDY-SET-COVER(X, 𝓕)
U = X
𝒞 = ∅
while U ≠ ∅
    select an S ∈ 𝓕 that maximizes |S ∩ U|
    U = U − S
    𝒞 = 𝒞 ∪ {S}
return 𝒞
```



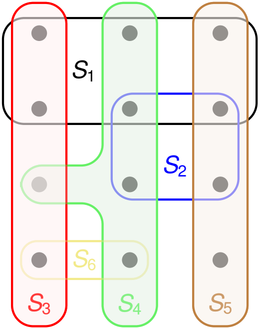
Greedy chooses S_1, S_4, S_5 and S_3 (or S_6), which is a cover of size 4.

Greedy

Strategy: Pick the set S that covers the largest number of uncovered elements.

```
GREEDY-SET-COVER$(X, \mathcal{F})$
U = X
$\mathcal{C} = \emptyset$
while $U \neq \emptyset$
    select an $S \in \mathcal{F}$ that maximizes $|S \cap U|$
    $U = U - S$
    $\mathcal{C} = \mathcal{C} \cup \{S\}$
return $\mathcal{C}$
```

Can be easily implemented to run in time polynomial in $|X|$ and $|\mathcal{F}|$



Optimal cover is $\mathcal{C} = \{S_3, S_4, S_5\}$

How good is the approximation ratio?

Approximation Ratio of Greedy

Theorem 35.4

GREEDY-SET-COVER is a polynomial-time $\rho(n)$-algorithm, where

$$\rho(n) = H(\max\{|S| \colon |S| \in \mathcal{F}\}) \leq \ln(n) + 1.$$

$H(k) := \sum_{i=1}^{k} \frac{1}{k} \leq \ln(k) + 1$

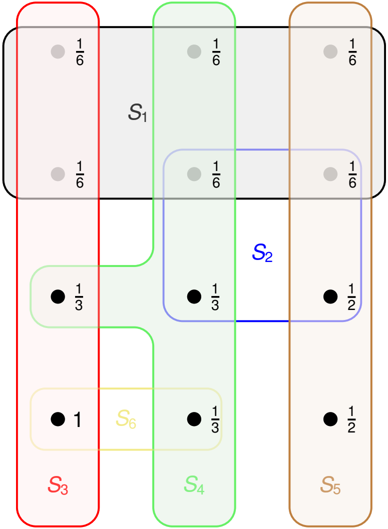
Idea: Distribute cost of 1 for each added set over the newly covered elements.

Definition of cost

If an element x is covered for the first time by set S_i in iteration i, then

$$c_x := \frac{1}{|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}.$$

Illustration of Costs



$$\frac{1}{6}+\frac{1}{6}+\frac{1}{6}+\frac{1}{6}+\frac{1}{6}+\frac{1}{6}+\frac{1}{3}+\frac{1}{3}+\frac{1}{3}+\frac{1}{2}+\frac{1}{2}+1=4$$

Proof of Theorem 35.4 (1/2)

Definition of cost

If x is covered for the first time by a set S_i, then $c_x := \frac{1}{\left|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})\right|}$.

Proof.

- Each step of the algorithm assigns 1 unit of cost, so

$$|\mathcal{C}| = \sum_{x \in X} c_x \tag{1}$$

- Each element $x \in X$ is in at least one set in the optimal cover $\mathcal{C}^*$, so

$$\sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x \tag{2}$$

- Combining 1 and 2 gives

$$|\mathcal{C}| \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x \leq \sum_{S \in \mathcal{C}*} H(|S|) \leq |\mathcal{C}^*| \cdot H(\max\{|S| : S \in \mathcal{F}\}) \qquad \square$$

Key Inequality: $\sum_{x \in S} c_x \leq H(|S|)$.

Proof of Theorem 35.4 (2/2)

Proof of the Key Inequality $\boxed{\sum_{x \in S} c_x \leq H(|S|)}$

Remaining uncovered elements in S

Sets chosen by the algorithm

- For any $S \in \mathcal{F}$ and $i = 1, 2, \ldots, |\mathcal{C}| = k$ let $u_i := |S \setminus (S_1 \cup S_2 \cup \cdots \cup S_i)|$

$\Rightarrow$ $u_0 > u_1 > \cdots > u_{|\mathcal{C}|} = 0$ and $u_{i-1} - u_i$ counts the items covered first time by S_i.

$\Rightarrow$

Each factor equals one.

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$$

- Further, by definition of the Greedy-Set-Cover:

$$|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| \geq |S \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| = u_{i-1}.$$

- Combining the last inequalities gives:

$$\begin{aligned}\sum_{x \in S} c_x \leq \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}} &= \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}} \\ &\leq \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \\ &\leq \sum_{i=1}^{k} (H(u_{i-1}) - H(u_i)) = H(u_0) - H(u_k) = H(|S|). \quad \square\end{aligned}$$

Set-Covering Problem (Summary)

The same approach also gives an approximation ratio of $O(\ln(n))$ if there exists a cost function $c : S \to \mathbb{Z}^+$

Theorem 35.4

GREEDY-SET-COVER is a polynomial-time $\rho(n)$-algorithm, where

$$\rho(n) = H(\max\{|S| : |S| \in \mathcal{F}\}) \leq \ln(n) + 1.$$

- Is the bound on the approximation ratio tight?
- Is there a better algorithm?

Lower Bound

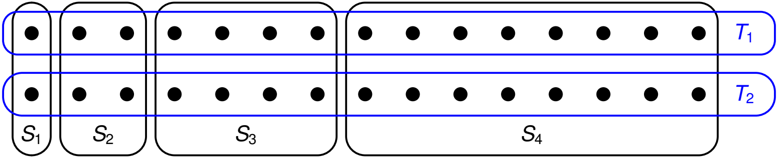
Unless P=NP, there is no $c \cdot \ln(n)$ approximation algorithm for set cover for some constant $0 < c < 1$.

Example where Greedy is a $(1/2) \cdot \log_2 n$ factor off

Instance

- Given any integer $k \geq 3$
- There are $n = 2^{k+1} - 2$ elements overall
- Sets $S_1, S_2, \ldots, S_k$ are pairwise disjoint and each set contains $2, 4, \ldots, 2^k$ elements
- Sets T_1, T_2 are disjoint and each set contains half of the elements of each set $S_1, S_2, \ldots, S_k$

$k = 4$:

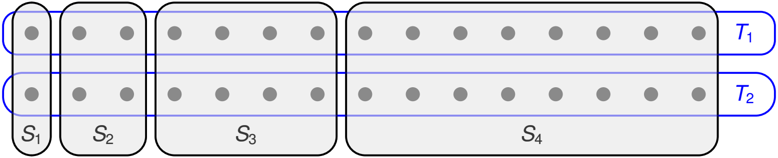


Example where Greedy is a $(1/2) \cdot \log_2 n$ factor off

Instance

- Given any integer $k \geq 3$
- There are $n = 2^{k+1} - 2$ elements overall
- Sets $S_1, S_2, \ldots, S_k$ are pairwise disjoint and each set contains $2, 4, \ldots, 2^k$ elements
- Sets T_1, T_2 are disjoint and each set contains half of the elements of each set $S_1, S_2, \ldots, S_k$

$k = 4$:



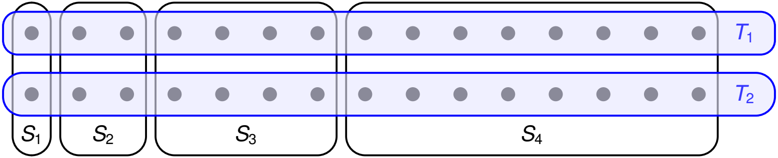
Solution of Greedy consists of k sets.

Example where Greedy is a $(1/2) \cdot \log_2 n$ factor off

Instance

- Given any integer $k \geq 3$
- There are $n = 2^{k+1} - 2$ elements overall
- Sets $S_1, S_2, \ldots, S_k$ are pairwise disjoint and each set contains $2, 4, \ldots, 2^k$ elements
- Sets T_1, T_2 are disjoint and each set contains half of the elements of each set $S_1, S_2, \ldots, S_k$

$k = 4$:



Optimum consists of 2 sets.

V. Approximation Algorithms via Exact Algorithms

Thomas Sauerwald

Easter 2015

UNIVERSITY OF
CAMBRIDGE

Outline

The Subset-Sum Problem

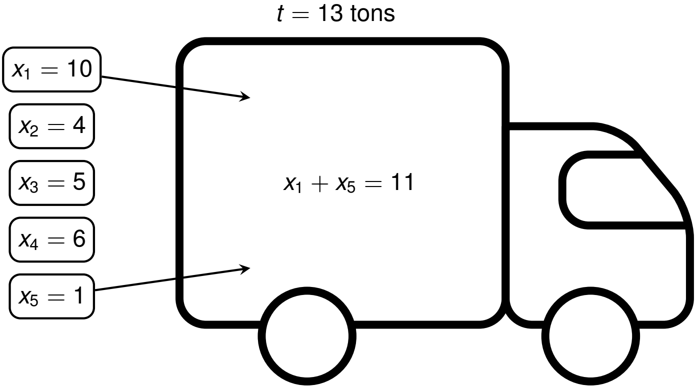
Parallel Machine Scheduling

The Subset-Sum Problem

The Subset-Sum Problem

- Given: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer t
- Goal: Find a subset $S' \subseteq S$ which maximizes $\sum_{i:\ x_i \in S'} x_i \leq t$.

This problem is NP-hard

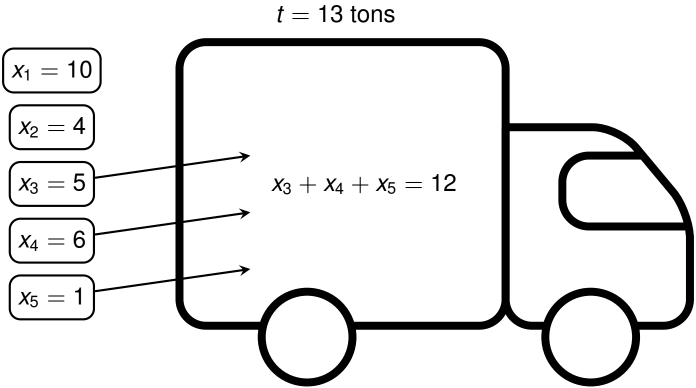


The Subset-Sum Problem

The Subset-Sum Problem

- Given: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer t
- Goal: Find a subset $S' \subseteq S$ which maximizes $\sum_{i:\ x_i \in S'} x_i \leq t$.

This problem is NP-hard


$t = 13$ tons

$x_1 = 10$

$x_2 = 4$

$x_3 = 5$

$x_4 = 6$

$x_5 = 1$

$x_3 + x_4 + x_5 = 12$

An Exact (Exponential-Time) Algorithm

Dynamic Progamming: Compute bottom-up all possible sums $\leq t$

implementable in time $O(|L_{i-1}|)$ (like Merge-Sort)

Returns the merged list (in sorted order and without duplicates)

$S + x := \{s + x : s \in S\}$

```
EXACT-SUBSET-SUM$(S, t)$
$n = |S|$
$L_0 = \langle 0 \rangle$
for $i = 1$ to $n$
    $L_i$ = MERGE-LISTS$(L_{i-1}, L_{i-1} + x_i)$
    remove from $L_i$ every element that is greater than $t$
return the largest element in $L_n$
```

Example:

- $S = \{1, 4, 5\}$
- $L_0 = \langle 0 \rangle$
- $L_1 = \langle 0, 1 \rangle$
- $L_2 = \langle 0, 1, 4, 5 \rangle$
- $L_3 = \langle 0, 1, 4, 5, 6, 9, 10 \rangle$

An Exact (Exponential-Time) Algorithm

Dynamic Progamming: Compute bottom-up all possible sums $\leq t$

```
EXACT-SUBSET-SUM$(S, t)$
$n = |S|$
$L_0 = \langle 0 \rangle$
for $i = 1$ to $n$
    $L_i = $ MERGE-LISTS$(L_{i-1}, L_{i-1} + x_i)$
    remove from $L_i$ every element that is greater than $t$
return the largest element in $L_n$
```

can be shown by induction on n

- **Correctness:** L_n contains all sums of $\{x_1, x_2, \ldots, x_n\}$
- **Runtime:** $O(2^1 + 2^2 + \cdots + 2^n) = O(2^n)$

There are 2^i subsets of $\{x_1, x_2, \ldots, x_i\}$.

Better runtime if t and/or $|L_i|$ are small.

Towards a FPTAS

Idea: Don't need to maintain two values in L which are close to each other.

Trimming a List

- Given a trimming parameter $0 < \delta < 1$
- Trimming L yields minimal sublist L' so that for every $y \in L$: $\exists z \in L$:

$$\frac{y}{1+\delta} \leq z \leq y.$$

- $L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$
- $\delta = 0.1$
- $L' = \langle 10, 12, 15, 20, 23, 29 \rangle$

```
TRIM($L, \delta$)
let $m$ be the length of $L$
$L' = \langle y_1 \rangle$
$last = y_1$
for $i = 2$ to $m$
    if $y_i > last \cdot (1+\delta)$        // $y_i \geq last$ because $L$ is sorted
        append $y_i$ onto the end of $L'$
        $last = y_i$
return $L'$
```

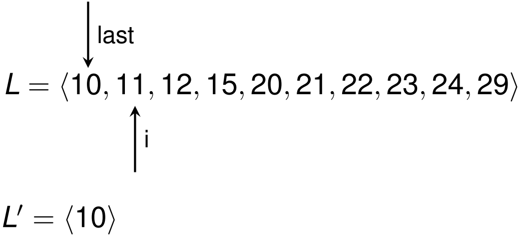
Trims list in time $\Theta(m)$, if L is given in sorted order.

Illustration of the Trim Operation

```
TRIM($L, \delta$)
1  let $m$ be the length of $L$
2  $L' = \langle y_1 \rangle$
3  $last = y_1$
4  for $i = 2$ to $m$
5      if $y_i > last \cdot (1 + \delta)$        // $y_i \geq last$ because $L$ is sorted
6          append $y_i$ onto the end of $L'$
7          $last = y_i$
8  return $L'$
```

$\delta = 0.1$

After the initialization (lines 1-3)



$L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$

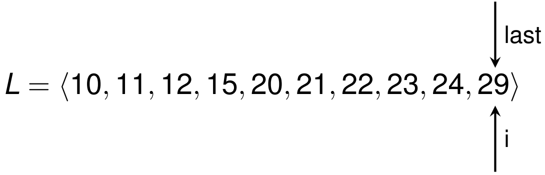
$L' = \langle 10 \rangle$

Illustration of the Trim Operation

```
TRIM($L, \delta$)
let $m$ be the length of $L$
$L' = \langle y_1 \rangle$
$last = y_1$
for $i = 2$ to $m$
    if $y_i > last \cdot (1 + \delta)$        // $y_i \geq last$ because $L$ is sorted
        append $y_i$ onto the end of $L'$
        $last = y_i$
return $L'$
```

$\delta = 0.1$

The returned list L'



$$L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$$

$$L' = \langle 10, 12, 15, 20, 23, 29 \rangle$$

The FPTAS

```
APPROX-SUBSET-SUM$(S, t, \epsilon)$
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    $L_i$ = MERGE-LISTS$(L_{i-1}, L_{i-1} + x_i)$
    $L_i$ = TRIM$(L_i, \epsilon/2n)$
    remove from $L_i$ every element that is greater than $t$
let $z^*$ be the largest value in $L_n$
return $z^*$
```

Repeated application of TRIM to make sure L_i's remain short.

```
EXACT-SUBSET-SUM$(S, t)$
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    $L_i$ = MERGE-LISTS$(L_{i-1}, L_{i-1} + x_i)$
    remove from $L_i$ every element that is greater than $t$
return the largest element in $L_n$
```

- We must bound the inaccuracy introduced by repeated trimming
- We must show that the algorithm is polynomial time

Running through an Example

```
APPROX-SUBSET-SUM($S, t, \epsilon$)
1 $n = |S|$
2 $L_0 = \langle 0 \rangle$
3 for $i = 1$ to $n$
4     $L_i$ = MERGE-LISTS($L_{i-1}, L_{i-1} + x_i$)
5     $L_i$ = TRIM($L_i, \epsilon/2n$)
6     remove from $L_i$ every element that is greater than $t$
7 let $z^*$ be the largest value in $L_n$
8 return $z^*$
```

- Input: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

⇒ Trimming parameter: $\delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05$

- line 2: $L_0 = \langle 0 \rangle$

- line 4: $L_1 = \langle 0, 104 \rangle$
- line 5: $L_1 = \langle 0, 104 \rangle$
- line 6: $L_1 = \langle 0, 104 \rangle$

- line 4: $L_2 = \langle 0, 102, 104, 206 \rangle$
- line 5: $L_2 = \langle 0, 102, 206 \rangle$
- line 6: $L_2 = \langle 0, 102, 206 \rangle$

- line 4: $L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle$
- line 5: $L_3 = \langle 0, 102, 201, 303, 407 \rangle$
- line 6: $L_3 = \langle 0, 102, 201, 303 \rangle$

- line 4: $L_4 = \langle 0, 101, 102, 201, 203, 302, 303, 404 \rangle$
- line 5: $L_4 = \langle 0, 101, 201, 302, 404 \rangle$
- line 6: $L_4 = \langle 0, 101, 201, 302 \rangle$

Returned solution $z^* = 302$, which is 2% of the optimum $307 = 104 + 102 + 101$

Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Approximation Ratio):

- Returned solution z^* is a valid solution ✓
- Let y^* denote an optimal solution
- For every possible sum $y \leq t$, there exists an element $z \in L_i$ such that:

$$\frac{y}{(1+\epsilon/(2n))^i} \leq z \leq y \quad \overset{y=y^*}{\Rightarrow} \quad \frac{y^*}{(1+\epsilon/(2n))^i} \leq z \leq y^*$$

Can be shown by induction on i

$$\frac{y^*}{z} \leq \left(1+\frac{\epsilon}{2n}\right)^n,$$

and now using the fact that $\left(1+\frac{\epsilon/2}{n}\right)^n \xrightarrow{n\to\infty} e^{\epsilon/2}$ yields

$$\frac{y^*}{z} \leq e^{\epsilon/2}$$

Taylor approximation of e

$$\leq 1+\epsilon/2+(\epsilon/2)^2 \leq 1+\epsilon$$

Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Running Time):

- Strategy: Derive a bound on $|L_i|$ (running time is polynomial in $|L_i|$)
- After trimming, two successive elements z and z' satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible Values after trimming are 0, 1, and up to $\lceil \log_{1+\epsilon/(2n)} t \rceil$ additional values. Hence,

$$\begin{aligned} \log_{1+\epsilon/(2n)} t + 2 &= \frac{\ln t}{\ln(1+\epsilon/(2n))} + 2 \\ &\leq \frac{2n(1+\epsilon/(2n)) \ln t}{\epsilon} + 2 \\ &< \frac{3n \ln t}{\epsilon} + 2. \end{aligned}$$

For $x > -1$, $\ln(1+x) \geq \frac{x}{1+x}$

- This bound on $|L_i|$ is polynomial in the size of the input and in $1/\epsilon$. $\square$

Need $\log(t)$ bits to represent t and n bits to represent S.

Concluding Remarks

The Subset-Sum Problem

- Given: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer t
- Goal: Find a subset $S' \subseteq S$ which maximizes $\sum_{i:\ x_i \in S'} x_i \leq t$.

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

A more general problem than Subset-Sum.

The Knapsack Problem

- Given: Items $i = 1, 2, \ldots, n$ with weights w_i and values v_i, and integer t
- Goal: Find a subset $S' \subseteq S$ which
  1. maximizes $\sum_{i \in S'} v_i$
  2. satisfies $\sum_{i \in S'} w_i \leq t$

Algorithm very similar to APPROX-SUBSET-SUM.

Theorem

There is a FPTAS for the Knapsack problem.

Outline

The Subset-Sum Problem

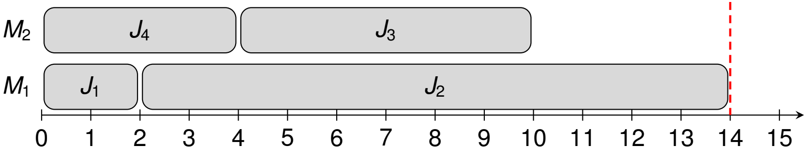
Parallel Machine Scheduling

Parallel Machine Scheduling

Machine Scheduling Problem

- Given: n jobs $J_1, J_2, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$, and m identical machines $M_1, M_2, \ldots, M_m$
- Goal: Schedule the jobs on the machines minimizing the makespan $C_{\max} = \max_{1 \leq j \leq n} C_j$, where C_k is the completion time of job J_k.

- J_1: $p_1 = 2$
- J_2: $p_2 = 12$
- J_3: $p_3 = 6$
- J_4: $p_4 = 4$

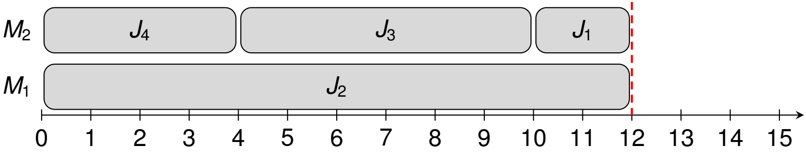


Parallel Machine Scheduling

Machine Scheduling Problem

- Given: n jobs $J_1, J_2, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$, and m identical machines $M_1, M_2, \ldots, M_m$
- Goal: Schedule the jobs on the machines minimizing the makespan $C_{\max} = \max_{1 \leq j \leq n} C_j$, where C_k is the completion time of job J_k.

- J_1: $p_1 = 2$
- J_2: $p_2 = 12$
- J_3: $p_3 = 6$
- J_4: $p_4 = 4$

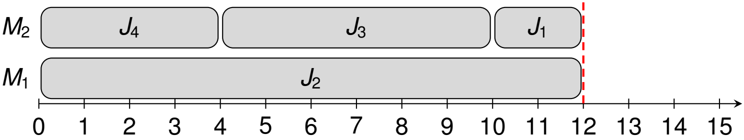


NP-Completeness of Parallel Machine Scheduling

Lemma

Parallel Machine Scheduling is NP-complete even if there are only two machines.

Proof Idea: Polynomial time reduction from NUMBER-PARTITIONING.



Equivalent to the following Online Algorithm [CLRS]:
Whenever a machine is idle, schedule any job that has not yet been scheduled.

```
LIST SCHEDULING($J_1, J_2, \ldots, J_n, m$)
while there exists an unassigned job
    Schedule job on the machine with the least load
```

How good is this most basic Greedy Approach?

List Scheduling Analysis (Observations)

Ex 35-5 a.&b.

a. The optimal makespan is at least as large as the greatest processing time, that is,

$$C^*_{\max} \geq \max_{1 \leq k \leq n} p_k.$$

b. The optimal makespan is at least as large as the average machine load, that is,

$$C^*_{\max} \geq \frac{1}{m} \sum_{k=1}^{n} p_k.$$

Proof:

a. The total processing times of all n jobs equals $\sum_{k=1}^{n} p_k$

b. One machine must have a load of at least $\frac{1}{m} \cdot \sum_{k=1}^{n} p_k$ □

List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

$$C_{\max} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k .$$

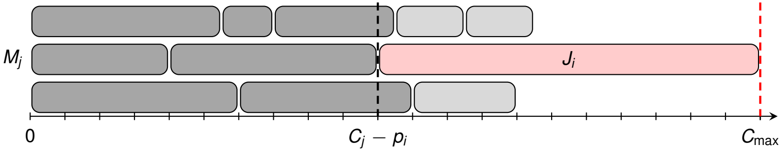
Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let J_i be the last job scheduled on machine M_j with $C_{\max} = C_j$
- When J_i was scheduled to machine M_j, $C_j - p_i \leq C_k$ for all $1 \leq k \leq m$
- Averaging over k yields:

Using Ex 35-5 a. & b.

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{m} C_k = \frac{1}{m} \sum_{k=1}^{n} p_k \quad \Rightarrow \quad C_{\max} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k \leq 2 \cdot C^*_{\max}$$



Improving Greedy

The problem of the List-Scheduling Approach were the large jobs

Analysis can be shown to be almost tight. Is there a better algorithm?

```
LEAST PROCESSING TIME($J_1, J_2, \ldots, J_n, m$)
Sort jobs decreasingly in their processing times
for $i = 1$ to $n$
    $C_i = 0$
    $S_i = \emptyset$
end for
for $j = 1$ to $n$
    $i = \text{argmin}_{1 \leq k \leq m} L_k$
    $S_i = S_i \cup \{j\}$, $C_i = C_i + p_j$
end for
return $S_1, \ldots, S_m$
```

Runtime:

- $O(n \log n)$ for sorting
- $O(n \log n)$ for extracting the minimum (use priority queue).

Analysis of Improved Greedy

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

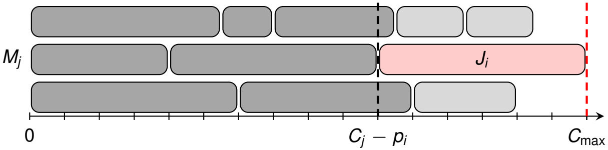
This can be shown to be tight (see next slide).

Proof (of approximation ratio $3/2$).

- Observation 1: If there are at most m jobs, then the solution is optimal.
- Observation 2: If there are more than m jobs, then $C^*_{\max} \geq 2 \cdot p_{m+1}$.
- As in the analysis for list scheduling, we have

$$C_j = (C_j - p_i) + p_i \leq C^*_{\max} + \frac{1}{2} C^*_{\max} = \frac{3}{2} C_{\max}. \qquad \square$$

This is for the case $j \geq m + 1$ (otherwise, an even stronger inequality holds)



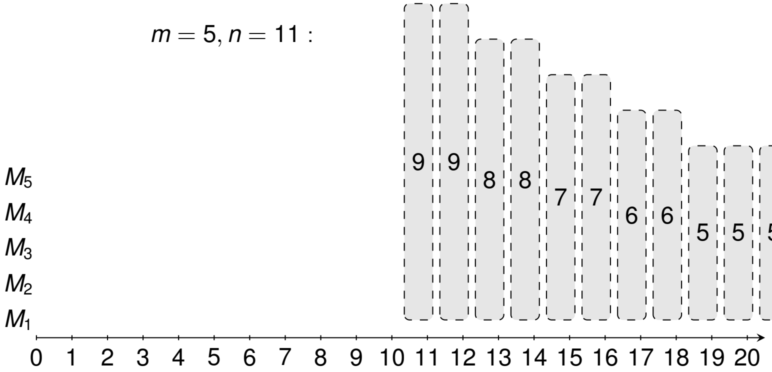
Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

- m machines
- $n = 2m + 1$ jobs of length $2m - 1, 2m - 2, \ldots, m$ and one job of length m



Tightness of the Bound for LPT

Graham 1966

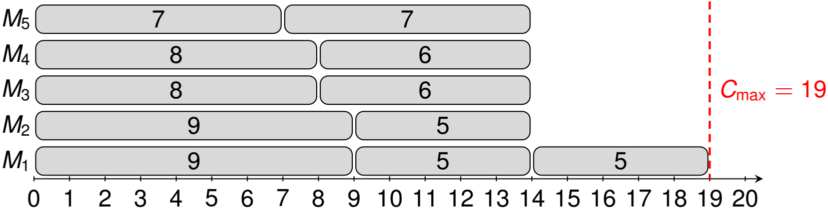
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

- m machines
- $n = 2m + 1$ jobs of length $2m - 1, 2m - 2, \ldots, m$ and one job of length m

$m = 5, n = 11$:

LPT gives $C_{\max} = 19$



Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

$\frac{19}{15} = \frac{20}{15} - \frac{1}{15}$

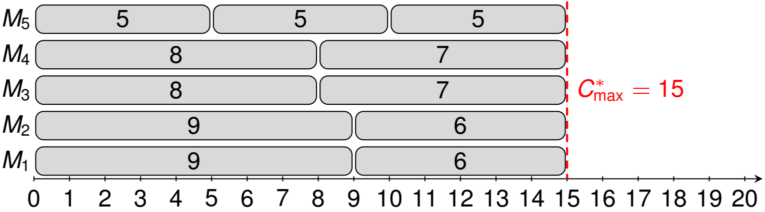
Proof of an instance which shows tightness:

- m machines
- $n = 2m + 1$ jobs of length $2m - 1, 2m - 2, \ldots, m$ and one job of length m

$m = 5, n = 11$:

LPT gives $C_{\max} = 19$

Optimum is $C^*_{\max} = 15$



A PTAS for Parallel Machine Scheduling

Basic Idea: For $(1+\epsilon)$-approximation, don't have to work with exact p_k's.

```
SUBROUTINE($J_1, J_2, \ldots, J_n, m, T$)
1: Either: Return a solution with $C_{\max} \leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\}$
2: Or: Return there is no solution with makespan $< T$
```

We will prove this on the next slides.

Key Lemma

SUBROUTINE can be implemented in time $n^{O(1/\epsilon^2)}$.

Theorem (Hochbaum, Shmoys'87)

There exists a PTAS for Parallel Machine Scheduling which runs in time $O(n^{O(1/\epsilon^2)} \cdot \log P)$, where $P := \sum_{k=1}^{n} p_k$.

polynomial in the size of the input

Since $0 \leq C^*_{\max} \leq P$ and $C^*_{\max}$ is integral, binary search terminates after $O(\log P)$ steps.

Proof (using Key Lemma):

```
PTAS($J_1, J_2, \ldots, J_n, m$)
1: Do binary search to find smallest $T$ s.t. $C_{\max} \leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\}$.
2: Return solution computed by SUBROUTINE($J_1, J_2, \ldots, J_n, m, T$)
```

Implementation of Subroutine

```
SUBROUTINE($J_1, J_2, \ldots, J_n, m, T$)
Either: **Return** a solution with $C_{\max} \leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\}$
Or: **Return** there is no solution with makespan $< T$
```

Observation

Divide jobs into two groups: $J_{\text{small}} = \{J_i : p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = J \setminus J_{\text{small}}$. Given a solution for J_{large} only with makespan $(1+\epsilon) \cdot T$, then greedily placing J_{small} yields a solution with makespan $(1+\epsilon) \cdot \max\{T, C^*_{\max}\}$.

Proof:

- Let M_j be the machine with largest load
- If there are no jobs from J_{small}, then makespan is at most $(1+\epsilon) \cdot T$.
- Otherwise, let $i \in J_{\text{small}}$ be the last job added to M_j.

$$C_j - p_i \leq \frac{1}{m}\sum_{k=1}^{n} p_k \qquad \Rightarrow \qquad C_j \leq p_i + \frac{1}{m}\sum_{k=1}^{n} p_k$$

the "well-known" formula

$$\leq \epsilon \cdot T + C^*_{\max}$$

$$\leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\} \qquad \square$$

Proof of Key Lemma

Use Dynamic Programming to schedule J_{large} with makespan $(1+\epsilon)\cdot T$.

- Let b be the smallest integer with $1/b \leq \epsilon$. Define processing times $p_i' = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ Every $p_i' = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b+1, \ldots, b^2$

Can assume there are no jobs with $p_j \geq T$!

- Let $\mathcal{C}$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

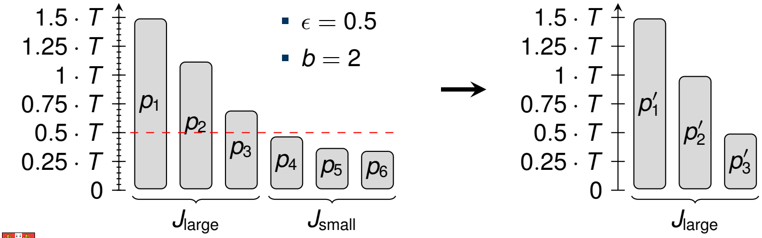
Assignments to one machine with makespan $\leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

Assign some jobs to one machine, and then use as few machines as possible for the rest.

$$f(0,0,\ldots,0) = 0$$

$$f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in \mathcal{C}} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).$$



- $\epsilon = 0.5$
- $b = 2$

Proof of Key Lemma

Use Dynamic Programming to schedule J_{large} with makespan $(1+\epsilon)\cdot T$.

- Let b be the smallest integer with $1/b \leq \epsilon$. Define processing times $p_i' = \lceil \frac{p_i b^2}{T} \rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ Every $p_i' = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b+1, \ldots, b^2$

- Let $\mathcal{C}$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.
- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

$$f(0,0,\ldots,0) = 0$$

$$f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in \mathcal{C}} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).$$

- Number of table entries is at most n^{b^2}, hence filling all entries takes $n^{O(b^2)}$
- If $f(n_b, n_{b+1}, \ldots, n_{b^2}) \leq m$ (for the jobs with p'), then return yes, otherwise no.
- As every machine is assigned at most b jobs ($p_i' \geq \frac{T}{b}$) and the makespan is $\leq T$,

$$\begin{aligned} C_{\max} &\leq T + b \cdot \max_{i \in J_{\text{large}}} (p_i - p_i') \\ &\leq T + b \cdot \frac{T}{b^2} \leq (1+\epsilon)\cdot T. \qquad \square \end{aligned}$$

Final Remarks

Graham 1966

List scheduling has an approximation ratio of 2.

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Theorem (Hochbaum, Shmoys'87)

There exists a PTAS for Parallel Machine Scheduling which runs in time $O(n^{O(1/\epsilon^2)} \cdot \log P)$, where $P := \sum_{k=1}^{n} p_k$.

Can we find a FPTAS (for polynomially bounded processing times)? **No!**

Because for sufficiently small approximation ratio $1 + \epsilon$, the computed solution has to be optimal.

VI. Approximation Algorithms: Travelling Salesman Problem

Thomas Sauerwald

Easter 2015

Outline

Introduction

General TSP

Metric TSP

The Traveling Salesman Problem (TSP)

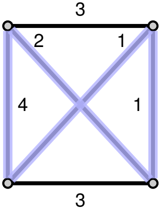
Given a set of cities along with the cost of travel between them, find the cheapest route visiting all cities and returning to your starting point.

Formal Definition

- Given: A complete undirected graph $G = (V, E)$ with nonnegative integer cost $c(u, v)$ for each edge $(u, v) \in E$
- Goal: Find a hamiltonian cycle of G with minimum cost.

Solution space consists of $n!$ possible tours!

Actually the right number is $(n-1)!/2$



$2 + 4 + 1 + 1 = 8$

Special Instances

- Metric TSP: costs satisfy triangle inequality:

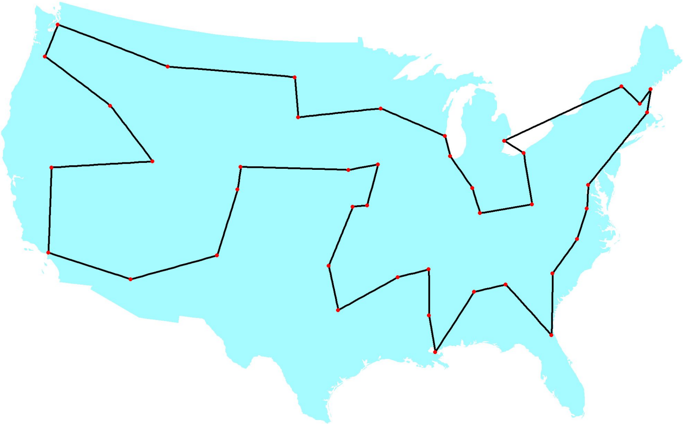
Even this version is NP hard (Ex. 35.2-2)

$$\forall u, v, w \in V: \qquad c(u, w) \leq c(u, v) + c(v, w).$$

- Euclidean TSP: cities are points in the Euclidean space, costs are equal to their Euclidean distance

History of the TSP problem (1954)

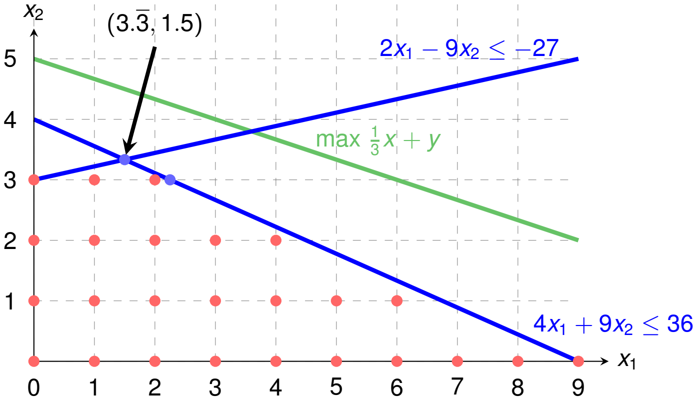
Dantzig, Fulkerson and Johnson found an optimal tour through 42 cities.



`http://www.math.uwaterloo.ca/tsp/history/img/dantzig_big.html`

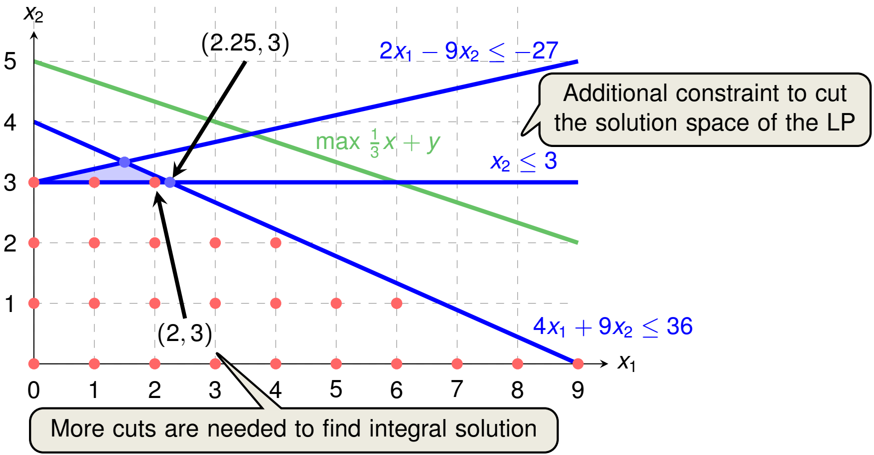
The Dantzig-Fulkerson-Johnson Method

1. Create a linear program (variable $x(u,v) = 1$ iff tour goes between u and v)
2. Solve the linear program. If the solution is integral and form a tour, stop. Otherwise find a new constraint to add (cutting plane)



The Dantzig-Fulkerson-Johnson Method

1. Create a linear program (variable $x(u,v) = 1$ iff tour goes between u and v)
2. Solve the linear program. If the solution is integral and form a tour, stop. Otherwise find a new constraint to add (cutting plane)



Outline

Introduction

General TSP

Metric TSP

Hardness of Approximation

Theorem 35.3

If P $\neq$ NP, then for any constant $\rho \geq 1$, there is no polynomial-time approximation algorithm with approximation ratio ρ for the general TSP.

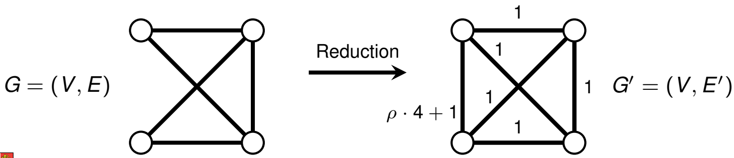
Proof: Idea: Reduction from the hamiltonian-cycle problem.

- Let $G = (V, E)$ be an instance of the hamiltonian-cycle problem
- Let $G' = (V, E')$ be a complete graph with costs for each $(u, v) \in E'$:

Can create representations of G' and c in time polynomial in $|V|$ and $|E|$!

$$c(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E, \\ \rho|V| + 1 & \text{otherwise.} \end{cases}$$

Large weight will render this edge useless!



Hardness of Approximation

Theorem 35.3

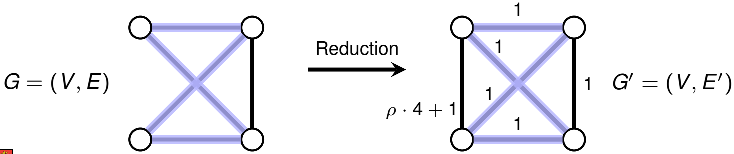
If P $\neq$ NP, then for any constant $\rho \geq 1$, there is no polynomial-time approximation algorithm with approximation ratio ρ for the general TSP.

Proof: Idea: Reduction from the hamiltonian-cycle problem.

- Let $G = (V, E)$ be an instance of the hamiltonian-cycle problem
- Let $G' = (V, E')$ be a complete graph with costs for each $(u, v) \in E'$:

$$c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E, \\ \rho|V| + 1 & \text{otherwise.} \end{cases}$$

- If G has a hamiltonian cycle H, then (G', c) contains a tour of cost $|V|$



Hardness of Approximation

Theorem 35.3

If P $\neq$ NP, then for any constant $\rho \geq 1$, there is no polynomial-time approximation algorithm with approximation ratio ρ for the general TSP.

Proof: **Idea:** Reduction from the hamiltonian-cycle problem.

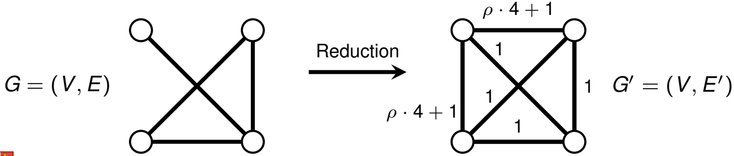
- Let $G = (V, E)$ be an instance of the hamiltonian-cycle problem
- Let $G' = (V, E')$ be a complete graph with costs for each $(u, v) \in E'$:

$$c(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E, \\ \rho|V| + 1 & \text{otherwise.} \end{cases}$$

- If G has a hamiltonian cycle H, then (G', c) contains a tour of cost $|V|$
- If G does not have a hamiltonian cycle, then any tour T must use some edge $\notin E$,

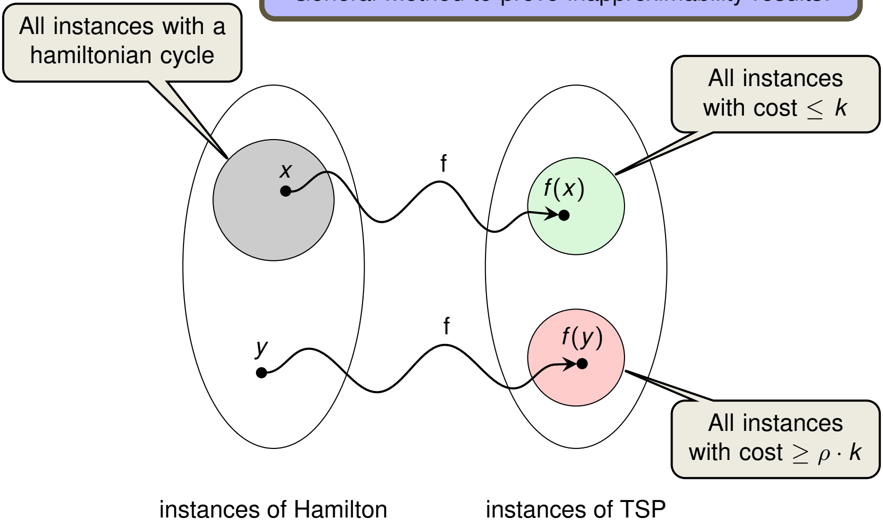
$$\Rightarrow \qquad c(T) \geq (\rho|V| + 1) + (|V| - 1) = (\rho + 1)|V|.$$

- Gap of $\rho + 1$ between tours which are using only edges in G and those which don't
- ρ-Approximation of TSP in G' computes hamiltonian cycle in G (if one exists) $\square$



Proof of Theorem 35.3 from a higher perspective

General Method to prove inapproximability results!



All instances with a hamiltonian cycle

All instances with cost $\leq k$

All instances with cost $\geq \rho \cdot k$

Outline

Introduction

General TSP

Metric TSP

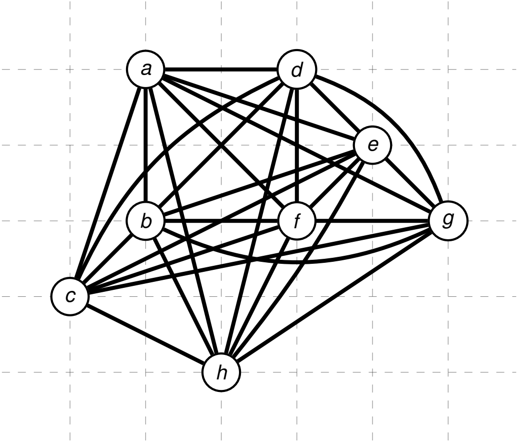
The TSP Problem with the Triangle Inequality

Idea: First compute an MST, and then create a tour based on the tree.

```
APPROX-TSP-TOUR$(G, c)$
select a vertex $r \in G.V$ to be a “root” vertex
compute a minimum spanning tree $T$ for $G$ from root $r$ using MST-PRIM$(G, c, r)$
let $H$ be a list of vertices, ordered according to when they are first visited in a preorder tree walk of $T$
return the hamiltonian cycle $H$
```

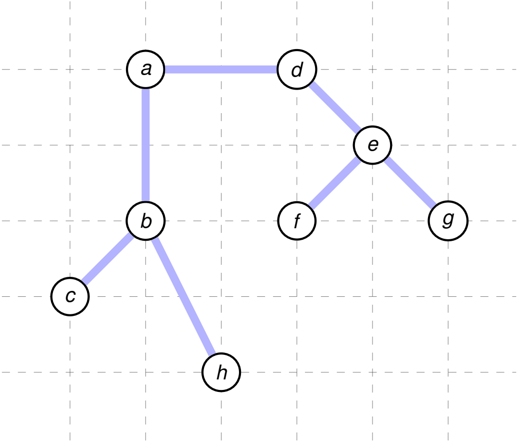
Runtime is dominated by MST-PRIM, which is $\Theta(V^2)$.

Run of APPROX-TSP-TOUR



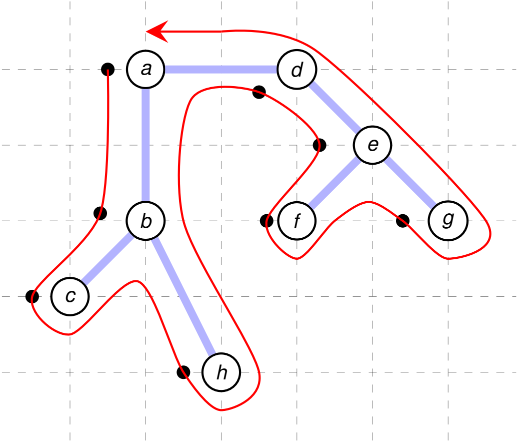
1. Compute MST

Run of APPROX-TSP-TOUR



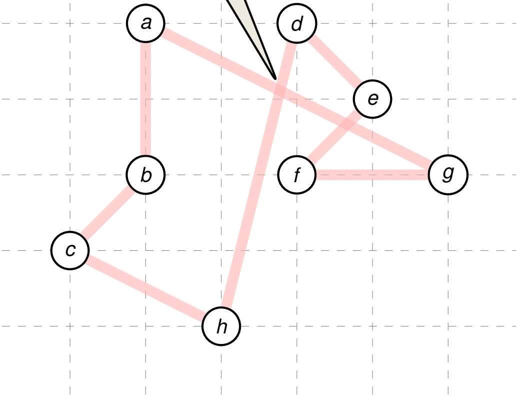
1. Compute MST ✓
2. Perform preorder walk on MST

Run of APPROX-TSP-TOUR



1. Compute MST ✓
2. Perform preorder walk on MST ✓
3. Return list of vertices according to the preorder tree walk

Run of APPROX-TSP-TOUR

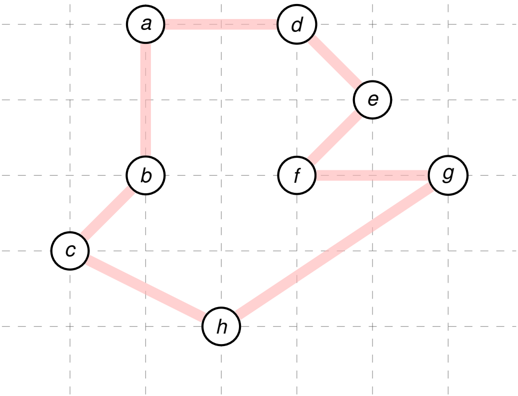


Solution has cost ≈ 19.704 - not optimal!

1. Compute MST ✓
2. Perform preorder walk on MST ✓
3. Return list of vertices according to the preorder tree walk ✓

Run of APPROX-TSP-TOUR

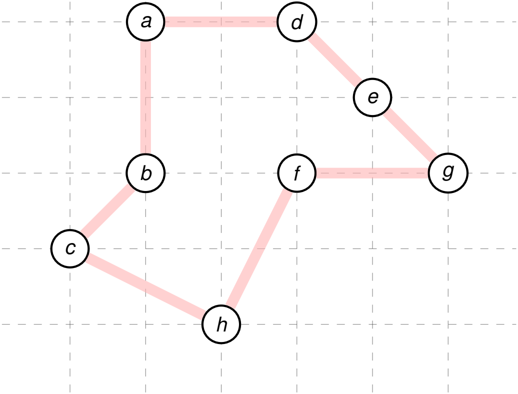
Better solution, yet still not optimal!



1. Compute MST ✓
2. Perform preorder walk on MST ✓
3. Return list of vertices according to the preorder tree walk ✓

Run of APPROX-TSP-TOUR

This is the optimal solution (cost ≈ 14.715).



1. Compute MST ✓
2. Perform preorder walk on MST ✓
3. Return list of vertices according to the preorder tree walk ✓

Proof of the Approximation Ratio

Theorem 35.2

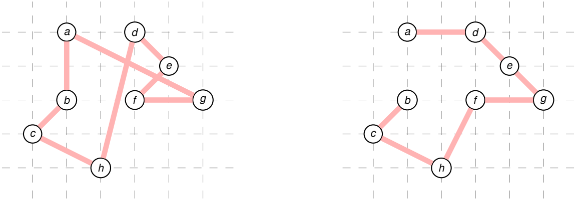
APPROX-TSP-TOUR is a polynomial-time 2-approximation for the traveling-salesman problem with the triangle inequality.

Proof:

- Consider the optimal tour H^* and remove one edge

$\Rightarrow$ yields a spanning tree and therefore $c(T) \leq c(H^*)$

exploiting that all edge costs are non-negative!



solution H of APPROX-TSP spanning tree as a subset of H^*

Proof of the Approximation Ratio

Theorem 35.2

APPROX-TSP-TOUR is a polynomial-time 2-approximation for the traveling-salesman problem with the triangle inequality.

Proof:

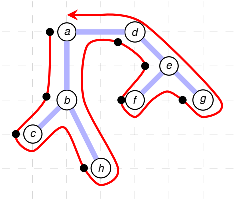
- Consider the optimal tour H^* and remove one edge

$\Rightarrow$ yields a spanning tree and therefore $c(T) \leq c(H^*)$

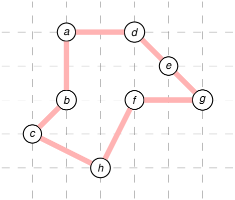
- Let W be the full walk of the spanning tree T (including repeated visits)

$\Rightarrow$ Full walk traverses every edge exactly twice, so

$$c(W) = 2(T) \leq 2c(H^*)$$



Walk $W = (a, b, c, b, h, b, a, d, e, f, e, g, e, d, a)$



optimal solution H^*

Proof of the Approximation Ratio

Theorem 35.2

APPROX-TSP-TOUR is a polynomial-time 2-approximation for the traveling-salesman problem with the triangle inequality.

Proof:

- Consider the optimal tour H^* and remove one edge

$\Rightarrow$ yields a spanning tree and therefore $c(T) \leq c(H^*)$

- Let W be the full walk of the spanning tree T (including repeated visits)

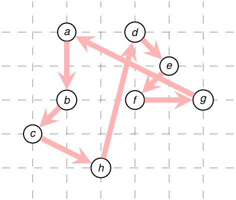
$\Rightarrow$ Full walk traverses every edge exactly twice, so

$$c(W) = 2(T) \leq 2c(H^*)$$

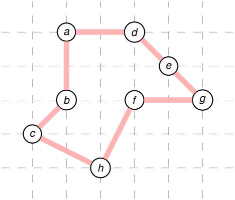
exploiting triangle inequality!

- Deleting duplicate vertices from W yields a tour T with smaller cost:

$$c(H) \leq c(W) \leq 2c(H^*) \qquad \square$$



Walk $W = (a, b, c, \cancel{b}, h, \cancel{b}, \cancel{a}, d, e, f, \cancel{e}, g, \cancel{e}, \cancel{d}, a)$



optimal solution H^*

Christofides Algorithm

Theorem 35.2

APPROX-TSP-TOUR is a polynomial-time 2-approximation for the traveling-salesman problem with the triangle inequality.

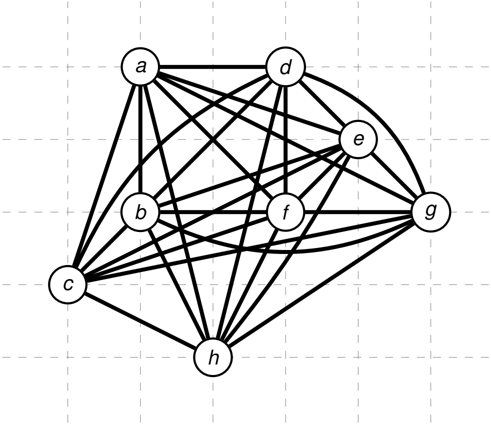
Can we get a better approximation ratio?

```
CHRISTOFIDES($G, c$)
select a vertex $r \in G.V$ to be a "root" vertex
compute a minimum spanning tree $T$ for $G$ from root $r$
        using MST-PRIM($G, c, r$)
compute a perfect matching $M$ with minimum weight in the complete graph
        over the odd-degree vertices in $T$
let $H$ be a list of vertices, ordered according to when they are first visited
        in a Eulearian circuit of $T \cup M$
return $H$
```

Theorem (Christofides'76)

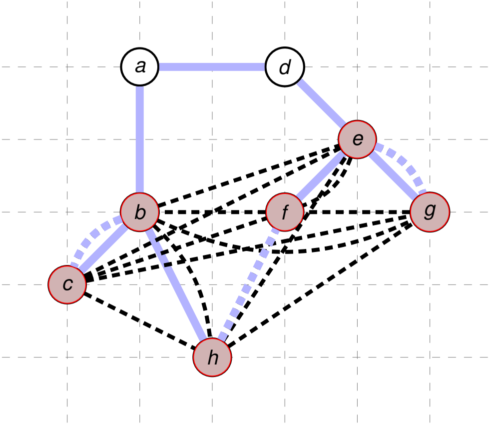
There is a polynomial-time $\frac{3}{2}$-approximation algorithm for the travelling salesman problem with the triangle inequality.

Run of CHRISTOFIDES



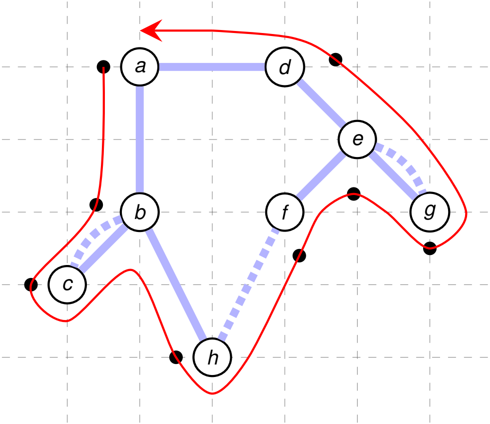
1. Compute MST

Run of CHRISTOFIDES



1. Compute MST ✓
2. Add a minimum-weight perfect matching M of the odd vertices in T✓

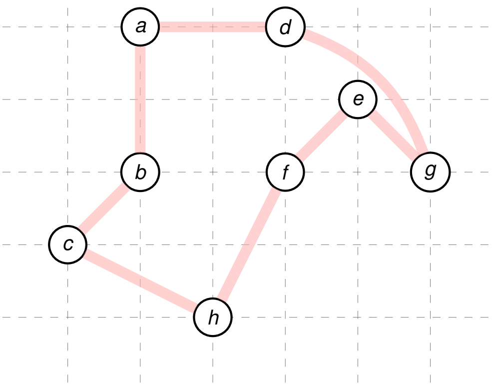
Run of CHRISTOFIDES



1. Compute MST ✓
2. Add a minimum-weight perfect matching M of the odd vertices in T ✓
3. Find an Eulerian Circuit ✓

Run of CHRISTOFIDES

Solution has cost ≈ 15.54 - within 10% of the optimum!



1. Compute MST ✓
2. Add a minimum-weight perfect matching M of the odd vertices in T✓
3. Find an Eulerian Circuit ✓
4. Transform the Circuit into a Hamiltonian Cycle✓

Concluding Remarks

Theorem (Christofides'76)

There is a polynomial-time $\frac{3}{2}$-approximation algorithm for the travelling salesman problem with the triangle inequality.

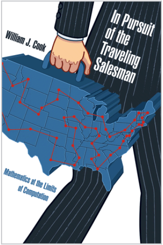
Both received the Gödel Award 2010

Theorem (Arora'96, Mitchell'96)

There is a PTAS for the Euclidean TSP Problem.

"Christos Papadimitriou told me that the traveling salesman problem is not a problem. It's an addiction."

Jon Bentley 1991



VII. Approximation Algorithms: Randomisation and Rounding

Thomas Sauerwald

Easter 2015

UNIVERSITY OF CAMBRIDGE

Outline

Randomised Approximation

MAX-3-CNF

Weighted Vertex Cover

Weighted Set Cover

Performance Ratios for Randomised Approximation Algorithms

Approximation Ratio

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

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

Call such an algorithm randomized $\rho(n)$-approximation algorithm.

extends in the natural way to randomized algorithms

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

Randomised Approximation

MAX-3-CNF

Weighted Vertex Cover

Weighted Set Cover

MAX-3-CNF Satisfiability

Assume that no literal (including its negation) appears more than once in the same clause.

MAX-3-CNF Satisfiability

- Given: 3-CNF formula, e.g.: $(x_1 \vee x_3 \vee \overline{x_4}) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_5}) \wedge \cdots$
- Goal: Find an assignment of the variables that satisfies as many clauses as possible.

Relaxation of the satisfiability problem. Want to compute how “close” the formula to being satisfiable is.

Example:

$$(x_1 \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_3} \vee \overline{x_5}) \wedge (x_2 \vee \overline{x_4} \vee x_5) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$$

$x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$ and $x_5 = 1$ satisfies 3 (out of 4 clauses)

Idea: What about assigning each variable independently at random?

Analysis

Theorem 35.6

Given an instance of MAX-3-CNF with n variables $x_1, x_2, \ldots, x_n$ and m clauses, the randomised algorithm that sets each variable independently at random is a randomized $8/7$-approximation algorithm.

Proof:

- For every clause $i = 1, 2, \ldots, m$, define a random variable:

$$Y_i = \mathbf{1}\{\text{clause } i \text{ is satisfied}\}$$

- Since each literal (including its negation) appears at most once in clause i,

$$\begin{aligned} &\mathbf{Pr}[\,\text{clause } i \text{ is not satisfied}\,] = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8} \\ \Rightarrow \quad &\mathbf{Pr}[\,\text{clause } i \text{ is satisfied}\,] = 1 - \frac{1}{8} = \frac{7}{8} \\ \Rightarrow \quad &\mathbf{E}[\,Y_i\,] = \mathbf{Pr}[\,Y_i = 1\,] \cdot 1 = \frac{7}{8}. \end{aligned}$$

- Let $Y := \sum_{i=1}^{m} Y_i$ be the number of satisfied clauses. Then,

$$\mathbf{E}[\,Y\,] = \mathbf{E}\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m} \mathbf{E}[\,Y_i\,] = \sum_{i=1}^{m} \frac{7}{8} = \frac{7}{8} \cdot m. \qquad \square$$

Linearity of Expectations

maximum number of satisfiable clauses is m!

Interesting Implications

Theorem 35.6

Given an instance of MAX-3-CNF with n variables $x_1, x_2, \ldots, x_n$ and m clauses, the randomised algorithm that sets each variable independently at random is a polynomial-time randomized $8/7$-approximation algorithm.

Corollary

For any instance of MAX-3-CNF, there exists an assigment which satisfies at least $\frac{7}{8}$ of all clauses.

There is $\omega \in \Omega$ such that $Y(\omega) \geq \mathbf{E}[\,Y\,]$!

Probabilistic Method: powerful tool to show existence of a non-obvious property.

Corollary

Any instance of MAX-3-CNF with at most 7 clauses is satisfiable.

Follows from the previous Corollary.

Expected Approximation Ratio

Theorem 35.6

Given an instance of MAX-3-CNF with n variables $x_1, x_2, \ldots, x_n$ and m clauses, the randomised algorithm that sets each variable independently at random is a polynomial-time randomized $8/7$-approximation algorithm.

One could prove that the probability to satisfy $(7/8) \cdot m$ clauses is at least $1/(8m)$

$$\mathbf{E}[\,Y\,] = \frac{1}{2} \cdot \mathbf{E}[\,Y \mid x_1 = 1\,] + \frac{1}{2} \cdot \mathbf{E}[\,Y \mid x_1 = 0\,].$$

Y is defined as in the previous proof.

One of the two conditional expectations is greater than $\mathbf{E}[\,Y\,]$!

```
GREEDY-3-CNF($\phi, n, m$)
for $j = 1, 2, \ldots, n$
    Compute $\mathbf{E}[\,Y \mid x_1 = v_1 \ldots, x_{j-1} = v_{j-1}, x_j = 1\,]$
    Compute $\mathbf{E}[\,Y \mid x_1 = v_1, \ldots, x_{j-1} = v_{j-1}, x_j = 0\,]$
    Let $x_j = v_j$ so that the conditional expectation is maximized
return the assignment $v_1, v_2, \ldots, v_n$
```

Analysis of GREEDY-3-CNF(ϕ, n, m)

This algorithm is deterministic.

Theorem

GREEDY-3-CNF(ϕ, n, m) is a polynomial-time $7/8$-approximation.

Proof:

- **Step 1:** polynomial-time algorithm
  - In iteration $j = 1, 2, \ldots, n$, $Y = Y(\phi)$ averages over 2^{n-j+1} assignments
  - A smarter way is to use linearity of (conditional) expectations:

$$\mathbf{E}\left[\, Y \mid x_1 = v_1, \ldots, x_{j-1} = v_{j-1}, x_j = 1 \,\right] = \sum_{i=1}^{m} \mathbf{E}\left[\, Y_i \mid x_1 = v_1, \ldots, x_{j-1} = v_{j-1}, x_j = 1 \,\right]$$

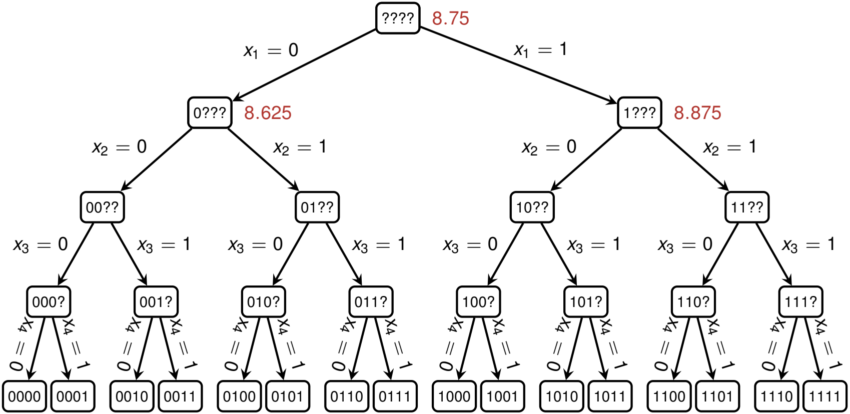
computable in $O(1)$

- **Step 2:** satisfies at least $7/8 \cdot m$ clauses
  - Due to the greedy choice in each iteration $j = 1, 2, \ldots, n$,

$$\begin{aligned}\mathbf{E}\left[\, Y \mid x_1 = v_1, \ldots, x_{j-1} = v_{j-1}, x_j = v_j \,\right] &\geq \mathbf{E}\left[\, Y \mid x_1 = v_1, \ldots, x_{j-1} = v_{j-1} \,\right] \\ &\geq \mathbf{E}\left[\, Y \mid x_1 = v_1, \ldots, x_{j-2} = v_{j-2} \,\right] \\ &\vdots \\ &\geq \mathbf{E}\left[\, Y \,\right] = \frac{7}{8} \cdot m. \qquad \square\end{aligned}$$

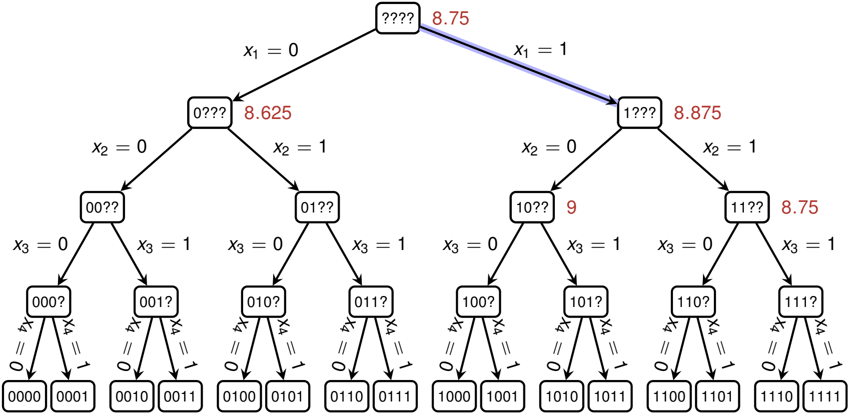
Run of GREEDY-3-CNF(φ, n, m)

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$



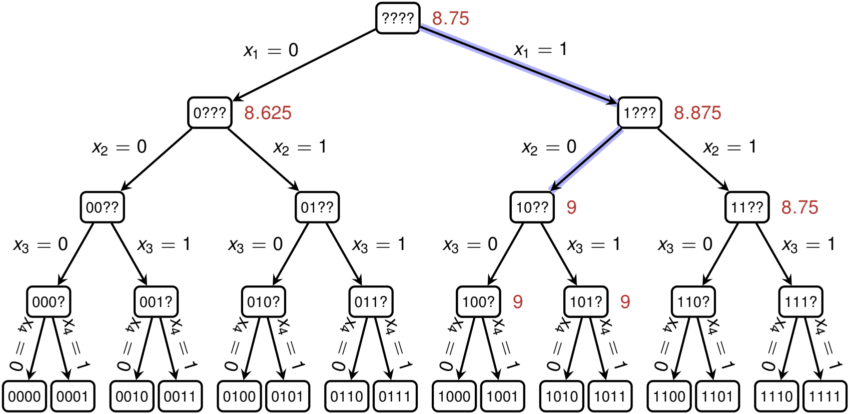
Run of GREEDY-3-CNF(φ, n, m)

$$1 \wedge 1 \wedge 1 \wedge (\overline{x_3} \vee x_4) \wedge 1 \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3) \wedge 1 \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$



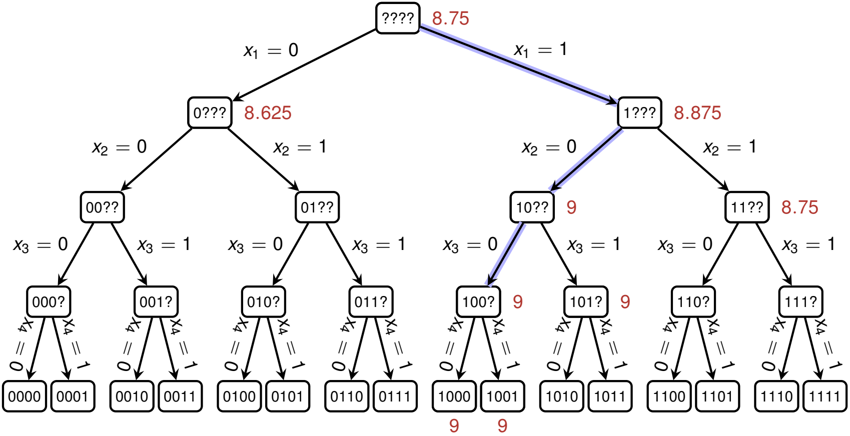
Run of GREEDY-3-CNF(φ, n, m)

$$1 \wedge 1 \wedge 1 \wedge (\overline{x_3} \vee x_4) \wedge 1 \wedge 1 \wedge (x_3) \wedge 1 \wedge 1 \wedge (\overline{x_3} \vee \overline{x_4})$$



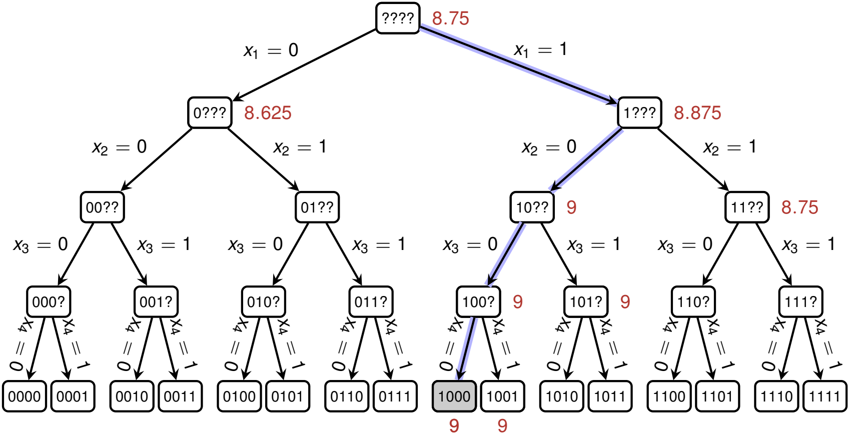
Run of GREEDY-3-CNF(φ, n, m)

$1 \wedge 1 \wedge 1 \wedge 1 \wedge 1 \wedge 1 \wedge 0 \wedge 1 \wedge 1 \wedge 1$



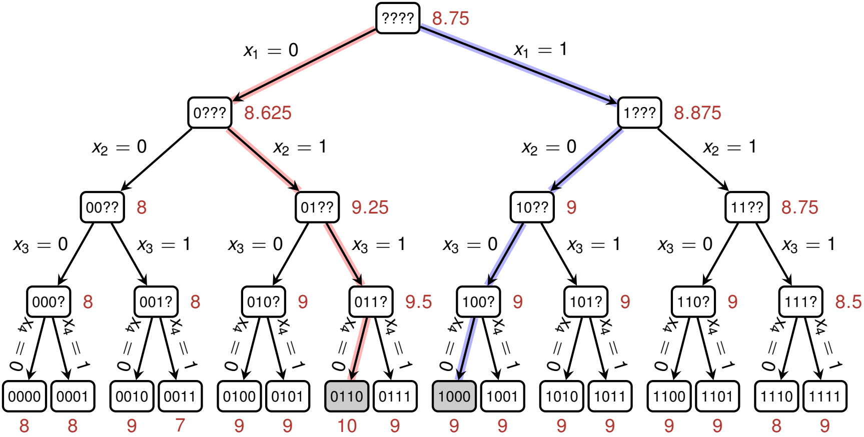
Run of GREEDY-3-CNF(φ, n, m)

$1 \wedge 1 \wedge 1 \wedge 1 \wedge 1 \wedge 1 \wedge 0 \wedge 1 \wedge 1 \wedge 1$



Run of GREEDY-3-CNF(φ, n, m)

$$(x_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor \overline{x_4}) \land (x_1 \lor x_2 \lor \overline{x_4}) \land (\overline{x_1} \lor \overline{x_3} \lor x_4) \land (x_1 \lor x_2 \lor \overline{x_4}) \land$$
$$(\overline{x_1} \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_1} \lor x_2 \lor x_3) \land (\overline{x_1} \lor \overline{x_2} \lor x_3) \land (x_1 \lor x_3 \lor x_4) \land (x_2 \lor \overline{x_3} \lor \overline{x_4})$$



Returned solution satisfies 9 out of 10 clauses, but the formula is satisfiable.

MAX-3-CNF: Concluding Remarks

Theorem 35.6

Given an instance of MAX-3-CNF with n variables $x_1, x_2, \ldots, x_n$ and m clauses, the randomised algorithm that sets each variable independently at random is a randomized $8/7$-approximation algorithm.

Theorem

GREEDY-3-CNF(ϕ, n, m) is a polynomial-time $8/7$-approximation.

Theorem (Hastad'97)

For any $\epsilon > 0$, there is no polynomial time $8/7 - \epsilon$ approximation algorithm of MAX3-SAT unless P=NP.

Roughly speaking, there is nothing smarter than just guessing.

Outline

Randomised Approximation

MAX-3-CNF

Weighted Vertex Cover

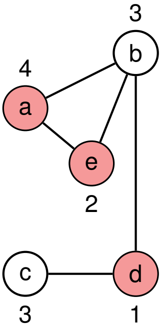
Weighted Set Cover

The Weighted Vertex-Cover Problem

Vertex Cover Problem

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

This is (still) an NP-hard problem.

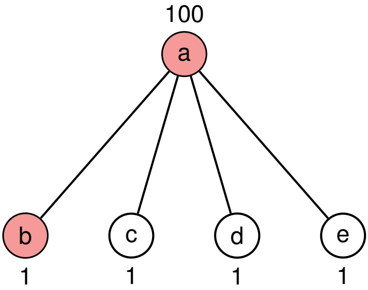


Applications:

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

The Greedy Approach from (Unweighted) Vertex Cover

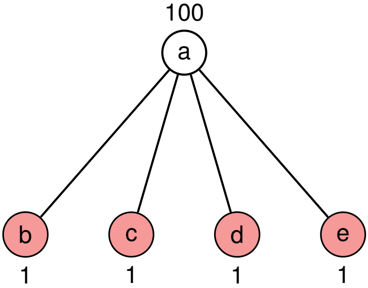
```
APPROX-VERTEX-COVER($G$)
$C = \emptyset$
$E' = G.E$
while $E' \neq \emptyset$
    let $(u, v)$ be an arbitrary edge of $E'$
    $C = C \cup \{u, v\}$
    remove from $E'$ every edge incident on either $u$ or $v$
return $C$
```



Computed solution has weight 101.

The Greedy Approach from (Unweighted) 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
```



Optimal solution has weight 4.

Invoking an (Integer) Linear Program

Idea: Round the solution of an associated linear program.

0-1 Integer Program

$$\begin{array}{llcll} \text{minimize} & \sum_{v \in V} w(v)x(v) & & & \\ \text{subject to} & x(u) + x(v) & \geq & 1 & \text{for each } (u,v) \in E \\ & x(v) & \in & \{0,1\} & \text{for each } v \in V \end{array}$$

optimum is a lower bound on the optimal weight of a minimum weight-cover.

Linear Program

$$\begin{array}{llcll} \text{minimize} & \sum_{v \in V} w(v)x(v) & & & \\ \text{subject to} & x(u) + x(v) & \geq & 1 & \text{for each } (u,v) \in E \\ & x(v) & \in & [0,1] & \text{for each } v \in V \end{array}$$

Rounding Rule: if $x(v) \geq 1/2$ then round up, otherwise round down.

The Algorithm

```
APPROX-MIN-WEIGHT-VC$(G, w)$
$C = \emptyset$
compute $\bar{x}$, an optimal solution to the linear program
for each $v \in V$
    if $\bar{x}(v) \geq 1/2$
        $C = C \cup \{v\}$
return $C$
```

Theorem 35.7

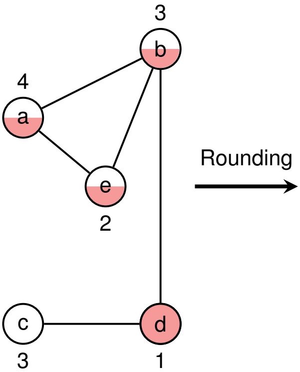
APPROX-MIN-WEIGHT-VC is a polynomial-time 2-approximation algorithm for the minimum-weight vertex-cover problem.

is polynomial-time because we can solve the linear program in polynomial time

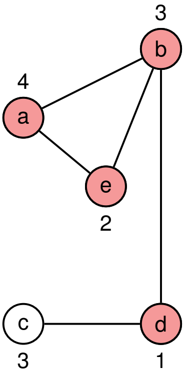
Example of APPROX-MIN-WEIGHT-VC

$\overline{x}(a) = \overline{x}(b) = \overline{x}(c) = \frac{1}{2}, \overline{x}(d) = 1, \overline{x}(e) = 0$

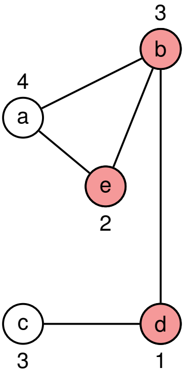
$x(a) = x(b) = x(c) = 1, x(d) = 1, x(e) = 0$



fractional solution of LP
with weight $= 5.5$



rounded solution of LP
with weight $= 10$



optimal solution
with weight $= 6$

Approximation Ratio

Proof (Approximation Ratio is 2):

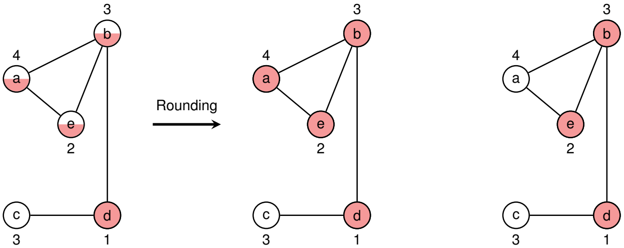
- Let C^* be an optimal solution to the minimum-weight vertex cover problem
- Let z^* be the value of an optimal solution to the linear program, so

$$z^* \leq w(C^*)$$

- **Step 1:** The computed set C covers all vertices:
  - Consider any edge $(u, v) \in E$ which imposes the constraint $x(u) + x(v) \geq 1$

  $\Rightarrow$ at least one of $\overline{x}(u)$ and $\overline{x}(v)$ is at least $1/2 \Rightarrow C$ covers edge (u, v)
- **Step 2:** The computed set C satisfies $w(C) \leq 2z^*$:

$$w(C^*) \geq z^* = \sum_{v \in V} w(v)\overline{x}(v) \;\geq \sum_{v \in V:\ \overline{x}(v) \geq 1/2} w(v) \cdot \frac{1}{2} = \frac{1}{2} w(C). \quad \square$$



Outline

Randomised Approximation

MAX-3-CNF

Weighted Vertex Cover

Weighted Set Cover

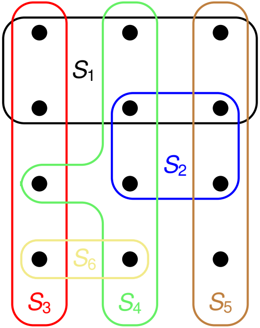
The Weighted Set-Covering Problem

Set Cover Problem

- Given: set X and a family of subsets $\mathcal{F}$, and a cost function $c : \mathcal{F} \to \mathbb{R}^+$
- Goal: Find a minimum-cost subset $\mathcal{C} \subseteq \mathcal{F}$

s.t. $$X = \bigcup_{S \in \mathcal{C}} S.$$

Sum over the costs of all sets in $\mathcal{C}$



	S_1	S_2	S_3	S_4	S_5	S_6
c :	2	3	3	5	1	2

Remarks:

- generalisation of the weighted vertex-cover problem.
- models resource allocation problems

Setting up an Integer Program

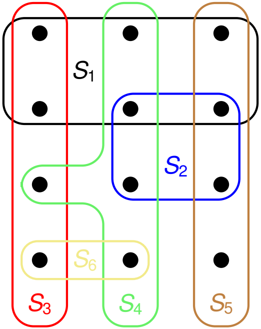
0-1 Integer Program

$$\begin{array}{llcll} \text{minimize} & \displaystyle\sum_{S \in \mathcal{F}} c(S) y(S) & & & \\ \text{subject to} & \displaystyle\sum_{S \in \mathcal{F}\colon x \in S} y(S) & \geq & 1 & \text{for each } x \in X \\ & y(S) & \in & \{0,1\} & \text{for each } S \in \mathcal{F} \end{array}$$

Linear Program

$$\begin{array}{llcll} \text{minimize} & \displaystyle\sum_{S \in \mathcal{F}} c(S) y(S) & & & \\ \text{subject to} & \displaystyle\sum_{S \in \mathcal{F}\colon x \in S} y(S) & \geq & 1 & \text{for each } x \in X \\ & y(S) & \in & [0,1] & \text{for each } S \in \mathcal{F} \end{array}$$

Back to the Example



	S_1	S_2	S_3	S_4	S_5	S_6
c :	2	3	3	5	1	2
$y(.)$:	$1/2$	$1/2$	$1/2$	$1/2$	1	$1/2$

Cost equals 8.5

The strategy employed for Vertex-Cover would take all 6 sets!

Even worse: If all y's were below $1/2$, we would not even return a valid cover!

Randomised Rounding

	S_1	S_2	S_3	S_4	S_5	S_6
c :	2	3	3	5	1	2
$y(.)$:	$1/2$	$1/2$	$1/2$	$1/2$	1	$1/2$

Idea: Interpret the y-values as probabilities for picking the respective set.

Lemma

Let $\mathcal{C} \subseteq \mathcal{F}$ be a random subset with each set S being included independently with probability $y(S)$.

- The expected cost satisfies

$$\mathbf{E}\left[\, c(\mathcal{C}) \,\right] = \sum_{S \in \mathcal{F}} c(S) \cdot y(S)$$

- The probability that an element $x \in X$ is covered satisfies

$$\mathbf{Pr}\left[x \in \bigcup_{S \in \mathcal{C}} S \right] \geq 1 - \frac{1}{e}.$$

Proof of Lemma

Lemma

Let $\mathcal{C} \subseteq \mathcal{F}$ be a random subset with each set S being included independently with probability $y(S)$.

- The expected cost satisfies $\mathbf{E}\left[\, c(\mathcal{C}) \,\right] = \sum_{S \in \mathcal{F}} c(S) \cdot y(S)$.
- The probability that x is covered satisfies $\mathbf{Pr}\left[\, x \in \cup_{S \in \mathcal{C}} S \,\right] \geq 1 - \frac{1}{e}$.

Proof:

- **Step 1**: The expected cost of the random set S

$$\mathbf{E}\left[\, c(\mathcal{C}) \,\right] = \mathbf{E}\left[\sum_{S \in \mathcal{C}} c(S) \right] = \sum_{S \in \mathcal{C}} \mathbf{E}\left[\, c(S) \,\right]$$

$$= \sum_{S \in \mathcal{F}} \mathbf{Pr}\left[\, S \in \mathcal{C} \,\right] \cdot c(S) = \sum_{S \in \mathcal{F}} y(S) \cdot c(S).$$

- **Step 2**: The probability for an element to be (not) covered

$$\mathbf{Pr}\left[\, x \notin \cup_{S \in \mathcal{C}} S \,\right] = \prod_{S \in \mathcal{C}\colon\, x \in \mathcal{F}} \mathbf{Pr}\left[\, S \notin \mathcal{C} \,\right] = \prod_{S \in \mathcal{F}\colon\, x \in S} (1 - y(S))$$

$$\leq \prod_{S \in \mathcal{F}\colon\, x \in S} e^{-y(S)}$$

$$= e^{-\sum_{S \in \mathcal{F}\colon\, x \in S} y(S)} \leq e^{-1} \qquad \square$$

$1 + x \leq e^x$ for any $x \in \mathbb{R}$

y solves the LP!

The Final Step

Lemma

Let $\mathcal{C} \subseteq \mathcal{F}$ be a random subset with each set S being included independently with probability $y(S)$.

- The expected cost satisfies $\mathbf{E}\left[\, c(\mathcal{C}) \,\right] = \sum_{S \in \mathcal{F}} c(S) \cdot y(S)$.
- The probability that x is covered satisfies $\mathbf{Pr}\left[\, x \in \cup_{S \in \mathcal{C}} S \,\right] \geq 1 - \frac{1}{e}$.

Problem: Need to make sure that every element is covered!

Idea: Amplify this probability by taking the union of $\Omega(\log n)$ random sets $\mathcal{C}$.

```
WEIGHTED SET COVER-LP$(X, \mathcal{F}, c)$
compute $y$, an optimal solution to the linear program
$\mathcal{C} = \emptyset$
repeat $2 \ln n$ times
    for each $S \in \mathcal{F}$
        let $\mathcal{C} = \mathcal{C} \cup \{S\}$ with probability $y(S)$
return $\mathcal{C}$
```

clearly runs in polynomial-time!

Analysis of WEIGHTED SET COVER-LP

Theorem

- With probability at least $1 - \frac{1}{n}$, the returned set $\mathcal{C}$ is a valid cover of X.
- The expected approximation ratio is $2\ln(n)$.

Proof:

- **Step 1**: The probability that $\mathcal{C}$ is a cover
  - By previous Lemma, an element $x \in X$ is covered in one of the $2\ln n$ iterations with probability at least $1 - \frac{1}{e}$, so that

$$\mathbf{Pr}\left[\, x \notin \cup_{S \in \mathcal{C}} S \,\right] \leq \left(\frac{1}{e}\right)^{2\ln n} = \frac{1}{n^2}.$$

  - This implies for the event that all elements are covered:

$$\mathbf{Pr}\left[\, X = \cup_{S \in \mathcal{C}} S \,\right] = 1 - \mathbf{Pr}\left[\bigcup_{x \in X} \{x \notin \cup_{S \in \mathcal{C}} S\}\right]$$

$\mathbf{Pr}[\, A \cup B \,] \geq \mathbf{Pr}[\, A \,] + \mathbf{Pr}[\, B \,]$

$$\geq 1 - \sum_{x \in X} \mathbf{Pr}\left[\, x \notin \cup_{S \in \mathcal{C}} S \,\right] \geq 1 - n \cdot \frac{1}{n^2} = 1 - \frac{1}{n}.$$

- **Step 2**: The expected approximation ratio
  - By previous lemma, the expected cost of one iteration is $\sum_{S \in \mathcal{F}} c(S) \cdot y(S)$.
  - Linearity $\Rightarrow \mathbf{E}\left[\, c(\mathcal{C}) \,\right] \leq 2\ln(n) \cdot \sum_{S \in \mathcal{F}} c(S) \cdot y(S) \leq 2\ln(n) \cdot c(\mathcal{C}^*)$ □

Analysis of WEIGHTED SET COVER-LP

Theorem

- With probability at least $1 - \frac{1}{n}$, the returned set $\mathcal{C}$ is a valid cover of X.
- The expected approximation ratio is $2\ln(n)$.

By Markov's inequality, $\mathbf{Pr}\left[\, c(\mathcal{C}) \leq 4\ln(n) \cdot c(\mathcal{C}^*) \,\right] \geq 1/2$.

Hence with probability at least $1 - \frac{1}{n} - \frac{1}{2} > \frac{1}{3}$, solution is within a factor of $4\ln(n)$ of the optimum.

probability could be further increased by repeating

Typical Approach for Designing Approximation Algorithms based on LPs

VIII. Approximation Algorithms: MAX-CUT Problem

Thomas Sauerwald

Easter 2015

UNIVERSITY OF CAMBRIDGE

Outline

Simple Algorithms for MAX-CUT

A Solution based on Semidefinite Programming

Summary

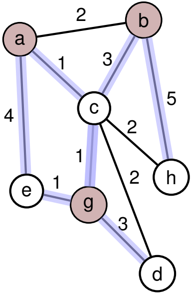
Max-Cut

Weighted MAX-CUT: Every edge $e \in E$ has a non-negative weight $w(e)$

MAX-CUT Problem

- Given: Undirected graph $G = (V, E)$
- Goal: Find a subset $S \subseteq V$ such that $|E(S, V \setminus S)|$ is maximized.

Weighted MAX-CUT: Maximize the weights of edges crossing the cut, i.e., maximize $w(S) := \sum_{\{u,v\} \in E(S, V \setminus S)} w(\{u, v\})$



$S = \{a, b, g\}$

$w(S) = 18$

Applications:

- cluster analysis
- VLSI design

Random Sampling

Ex 35.4-3

Suppose that for each vertex v, we randomly and independently place v in S with probability $1/2$ and in $V \setminus S$ with probability $1/2$. Then this algorithm is a randomized 2-approximation algorithm.

We could employ the same derandomisation used for MAX-3-CNF.

Proof: We express the expected weight of the random cut $(S, V \setminus S)$ as:

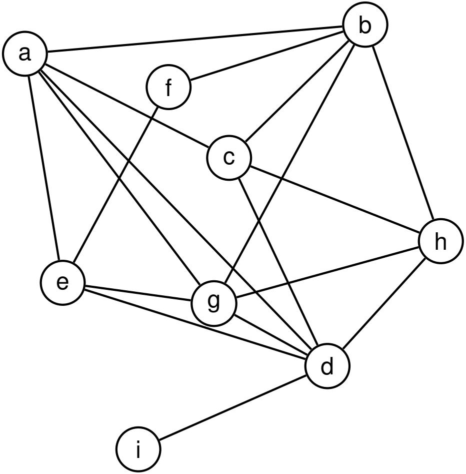
$$
\begin{aligned}
&\mathbf{E}\left[\, w(S, V \setminus S) \,\right] \\
&= \mathbf{E}\left[\sum_{\{u,v\} \in E(S, V \setminus S)} w(\{u,v\}) \right] \\
&= \sum_{\{u,v\} \in E} \mathbf{Pr}\left[\, \{u \in S \cap v \in (V \setminus S)\} \cup \{u \in (V \setminus S) \cap v \in S\} \,\right] \cdot w(\{u,v\}) \\
&= \sum_{\{u,v\} \in E} \left(\frac{1}{4} + \frac{1}{4} \right) \cdot w(\{u,v\}) \\
&= \frac{1}{2} \sum_{\{u,v\} \in E} w(\{u,v\}) \geq \frac{1}{2} w^{*}. \qquad \square
\end{aligned}
$$

Local Search

Local Search: Switch side of a vertex if it increases the cut.

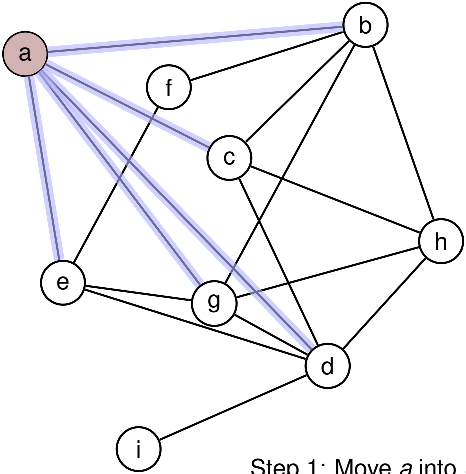
```
LOCAL SEARCH($G, w$)
Let $S$ be an arbitrary subset of $V$
do
    $flag = 0$
    if $\exists u \in S$ with $w(S \setminus \{u\}, (V \setminus S) \cup \{u\}) \geq w(S, V \setminus S)$ then
        $S = S \setminus \{u\}$
        $flag = 1$
    end if
    if $\exists u \in V \setminus S$ with $w(S \cup \{u\}, (V \setminus S) \setminus \{u\}) \geq w(S, V \setminus S)$ then
        $S = S \cup \{u\}$
        $flag = 1$
    end if
while $flag = 1$
return $S$
```

Illustration of Local Search



Cut $= 0$

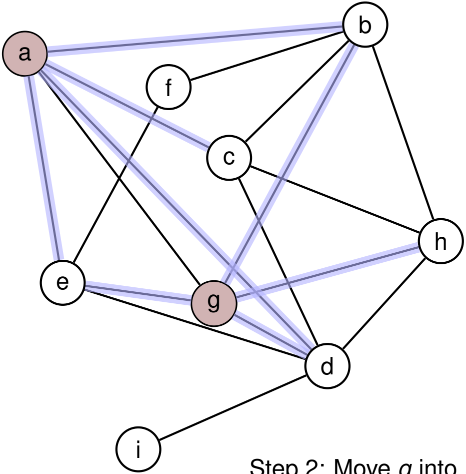
Illustration of Local Search



Step 1: Move *a* into S

Cut $= 5$

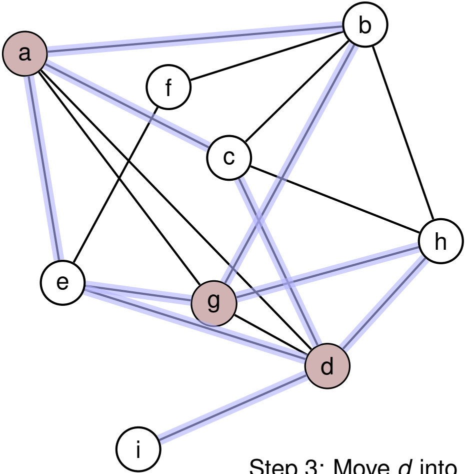
Illustration of Local Search



Step 2: Move *g* into *S*

Cut $= 8$

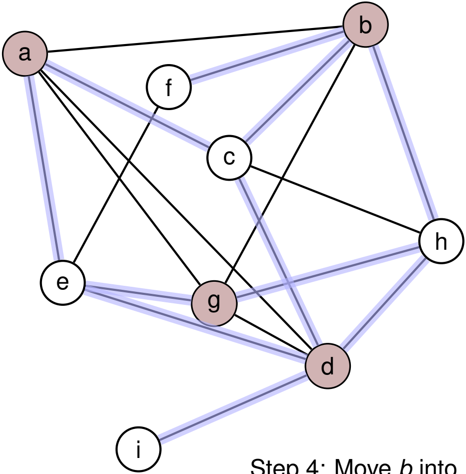
Illustration of Local Search



Step 3: Move *d* into *S*

Cut $= 10$

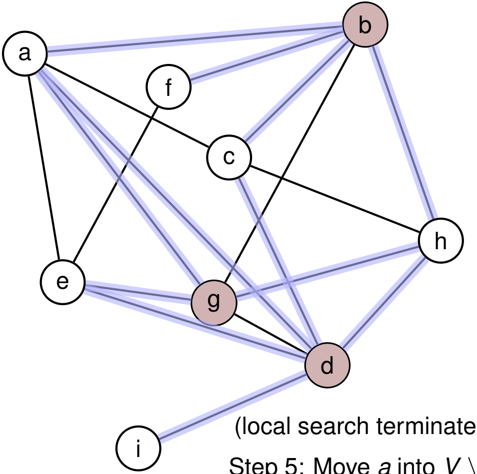
Illustration of Local Search



Step 4: Move *b* into *S*

Cut $= 11$

Illustration of Local Search

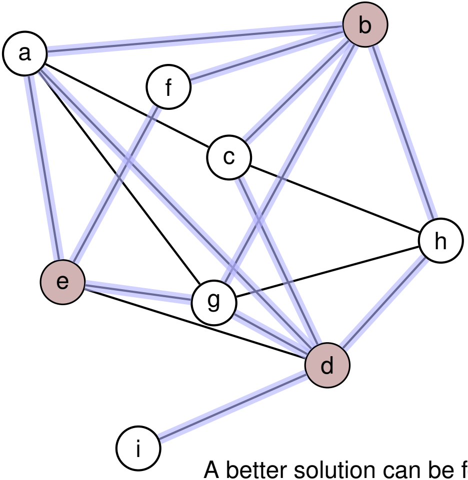


(local search terminates)

Step 5: Move *a* into $V \setminus S$

Cut $= 12$

Illustration of Local Search



A better solution can be found:

$$\text{Cut} = 13$$

Analysis of Local Search (1/2)

Theorem

The cut returned by LOCAL-SEARCH satisfies $W \geq (1/2)W^*$.

Proof:

- At the time of termination, for every vertex $u \in S$:

$$\sum_{v \in V \setminus S, v \sim u} w(\{u,v\}) \geq \sum_{v \in S, v \sim u} w(\{u,v\}), \qquad (1)$$

- Similarly, for any vertex $u \in V \setminus S$:

$$\sum_{v \in S, v \sim u} w(\{u,v\}) \geq \sum_{v \in V \setminus S, v \sim u} w(\{u,v\}). \qquad (2)$$

- Adding up equation 1 for all vertices in S and equation 2 for all vertices in $V \setminus S$,

$$w(S) \geq 2 \cdot \sum_{v \in S, u \in S, u \sim v} w(\{u,v\}) \quad \text{and} \quad w(S) \geq 2 \cdot \sum_{v \in V \setminus S, u \in V \setminus S, u \sim v} w(\{u,v\}).$$

- Adding up these two inequalities, and diving by 2 yields

$$w(S) \geq \sum_{v \in S, u \in S, u \sim v} w(\{u,v\}) + \sum_{v \in V \setminus S, u \in V \setminus S, u \sim v} w(\{u,v\}). \qquad \square$$

Every edge appears on one of the two sides.

Analysis of Local Search (2/2)

Theorem

The cut returned by LOCAL-SEARCH satisfies $W \geq (1/2)W^*$.

What is the running time of LOCAL-SEARCH?

- **Unweighted Graphs:** Cut increases by at least one in each iteration $\Rightarrow$ at most n^2 iterations
- **Weighted Graphs:** could take exponential time in n (not obvious...)

Outline

Simple Algorithms for MAX-CUT

A Solution based on Semidefinite Programming

Summary

Max-Cut Problem

High-Level-Approach:

1. Describe the Max-Cut Problem as a quadratic optimisation problem
2. Solve a corresponding semidefinite program that is a relaxation of the original problem
3. Recover an approximation for the original problem from the approximation for the semidefinite program

Label vertices by $1, 2, \ldots, n$ and express weight function etc. as a $n \times n$-matrix.

Quadratic program

maximize $$\frac{1}{2} \sum_{(i,j) \in E} w_{i,j} \cdot (1 - y_i y_j)$$

subject to $$y_i \in \{-1, +1\}, \qquad i = 1, \ldots, n.$$

This models the MAX-CUT problem

$$S = \{i \in V : y_i = +1\},$$
$$V \setminus S = \{i \in V : y_i = -1\}$$

Relaxation

Quadratic program

$$\begin{array}{lll} \text{maximize} & \displaystyle\frac{1}{2}\sum_{(i,j)\in E} w_{i,j}\cdot(1-y_iy_j) & \\ \text{subject to} & y_i\in\{-1,+1\}, & i=1,\dots,n. \end{array}$$

Any solution of the original program can be recovered by setting $v_i = (y_i, 0, 0, \dots, 0)$!

Vector Programming Relaxation

$$\begin{array}{lll} \text{maximize} & \displaystyle\frac{1}{2}\sum_{(i,j)\in E} w_{i,j}\cdot(1-v_iv_j) & \\ \text{subject to} & v_i\cdot v_i = 1 & i=1,\dots,n. \\ & v_i\in\mathbb{R}^n & \end{array}$$

Positive Definite Matrices

Definition

A matrix $A \in \mathbb{R}^{n \times n}$ is positive semidefinite iff for all $y \in \mathbb{R}^n$,

$$y^T \cdot A \cdot y \geq 0.$$

Remark

1. A is symmetric and positive definite iff there exists a $n \times n$ matrix B with $B^T \cdot B = A$.
2. If A is symmetric and positive definite, then the matrix B above can be computed in polynomial time.

using Cholesky-decomposition

Examples:

$$A = \begin{pmatrix} 18 & 2 \\ 2 & 6 \end{pmatrix} = \begin{pmatrix} 4 & -1 \\ 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 4 & 1 \\ -1 & 2 \end{pmatrix}, \quad \text{so } A \text{ is SPD.}$$

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad \text{since } \begin{pmatrix} 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} = -2, \quad A \text{ is not SPD.}$$

Reformulating the Quadratic Program as a Semidefinite Program

Vector Programming Relaxation

$$\begin{array}{lll} \text{maximize} & \displaystyle\frac{1}{2}\sum_{(i,j)\in E} w_{i,j}\cdot(1-v_i v_j) & \\ \text{subject to} & v_i\cdot v_i = 1 & i = 1,\ldots,n. \\ & v_i \in \mathbb{R}^n & \end{array}$$

Reformulation:

- Introduce n^2 variables $a_{i,j} = v_i \cdot v_j$, which give rise to a matrix A
- If V is the matrix given by the vectors $(v_1, v_2, \ldots, v_n)$, then $A = V^T \cdot V$ is symmetric and positive definite

Solve this (which can be done in polynomial time), and recover V using Cholesky Decomposition.

Semidefinite Program

$$\begin{array}{ll} \text{maximize} & \displaystyle\frac{1}{2}\sum_{(i,j)\in E} w_{i,j}\cdot(1-a_{i,j}) \\ \text{subject to} & A = (a_{i,j}) \text{ is symmetric and positive definite,} \\ & \text{and } a_{i,i} = 1 \text{ for all } i = 1,\ldots,n \end{array}$$

Rounding the Vector Program

Vector Programming Relaxation

$$\begin{array}{lll} \text{maximize} & \frac{1}{2}\sum_{(i,j)\in E} w_{i,j}\cdot(1-v_iv_j) & \\ \text{subject to} & v_i\cdot v_i = 1 & i=1,\ldots,n. \\ & v_i\in\mathbb{R}^n & \end{array}$$

Rounding by a random hyperplane :

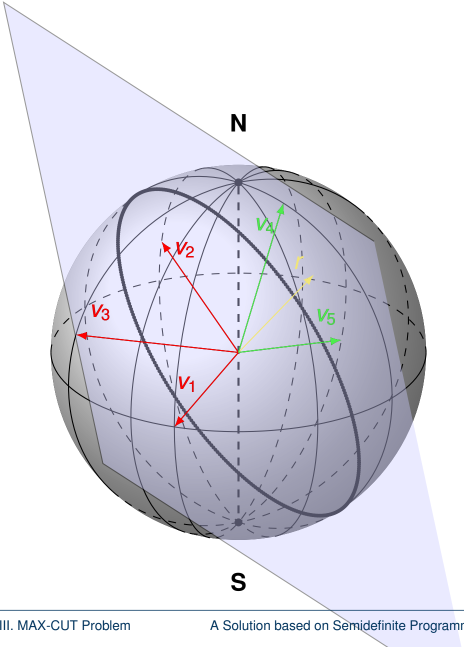
1. Pick a random vector $r = (r_1, r_2, \ldots, r_n)$ by drawing each component from $\mathcal{N}(0,1)$
2. Put $i \in V$ if $v_i \cdot r \geq 0$ and $i \in V \setminus S$ otherwise

Lemma 1

The probability that two vectors $v_i, v_j \in \mathbb{R}^n$ are separated by the (random) hyperplane given by r equals $\frac{\arccos(v_i \cdot v_j)}{\pi}$.

Follows by projecting on the plane given by v_i and v_j.

Illustration of the Hyperplane

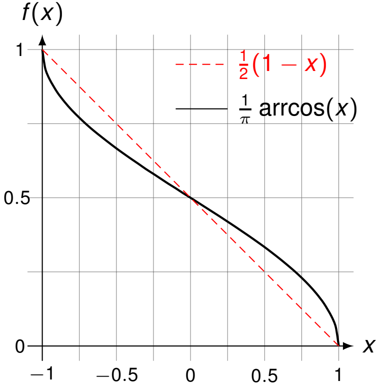


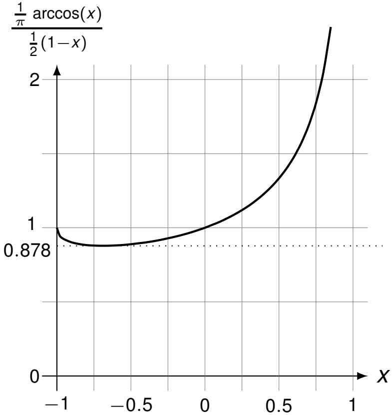
A second (technical) Lemma

Lemma 2

For any $x \in [-1, 1]$,

$$\frac{1}{\pi}\arccos(x) \geq 0.878 \cdot \frac{1}{2}(1-x).$$





Putting Everything Together

Theorem (Goemans, Willamson'96)

The algorithm has an approximation ratio of $\frac{1}{0.878} \approx 1.139$.

Proof: Define an indicator variable

$$X_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \text{ are on different sides of the hyperplane} \\ 0 & \text{otherwise.} \end{cases}$$

Hence for the (random) weight of the computed cut,

$$\begin{aligned} \mathbf{E}\left[\, w(S) \,\right] &= \mathbf{E}\left[\sum_{\{i,j\} \in E} X_{i,j} \right] \\ &= \sum_{\{i,j\} \in E} \mathbf{E}\left[\, X_{i,j} \,\right] \\ &= \sum_{\{i,j\} \in E} w_{i,j} \cdot \mathbf{Pr}\left[\, \{i,j\} \in E \text{ is in the cut} \,\right] \\ &= \sum_{\{i,j\} \in E} w_{i,j} \cdot \frac{1}{\pi} \arccos(v_i \cdot v_j) \\ &\geq 0.878 \cdot \frac{1}{2} \sum_{\{i,j\} \in E} w_{i,j} \cdot (1 - v_i \cdot v_j) = 0.878 \cdot z^* \geq 0.878 \cdot W^*. \qquad \square \end{aligned}$$

By Lemma 1

By Lemma 2

MAX-CUT: Concluding Remarks

Theorem (Goemans, Willamson'96)

There is a randomised polynomial-time 1.139-approximation algorithm for MAX-CUT.

can be derandomized (with some effort)

Similar approach can be applied to MAX-3-CNF and yields an approximation ratio of 1.345

Theorem (Håstad'97)

Unless P$=$NP, there is no ρ-approximation algorithm for MAX-CUT with $\rho \leq \frac{17}{16} = 1.0625$.

Theorem (Khot, Kindler, Mossel, O'Donnell'04)

Assuming the so-called Unique Games Conjecture holds, unless P$=$NP there is no ρ-approximation algorithm for MAX-CUT with

$$\rho \leq \max_{-1 \leq x \leq 1} \frac{\frac{1}{2}(1-x)}{\frac{1}{\pi}\arccos(x)} \leq 1.139$$

Other Approximation Algorithms for MAX-CUT

Theorem (Mathieu, Schudy'08)

For any $\epsilon > 0$, there is a randomised algorithm with running time $O(n^2)2^{O(1/\epsilon^2)}$ so that the expected value of the output deviates from the maximum cut value by at most $O(\epsilon \cdot n^2)$.

This is an additive approximation!

Algorithm (1):

1. Take a sample S of $x = O(1/\epsilon^2)$ vertices chosen uniformly at random
2. For each of the 2^x possible cuts, go through vertices in $V \setminus S$ in random order and place them on the side of the cut which maximizes the crossing edges
3. Output the best cut found

Theorem (Trevisan'08)

There is a randomised 1.833-approximation algorithm for MAX-CUT which runs in $O(n^2 \cdot \text{polylog}(n))$ time.

Exploits relation between the smallest eigenvalue and the structure of the graph.

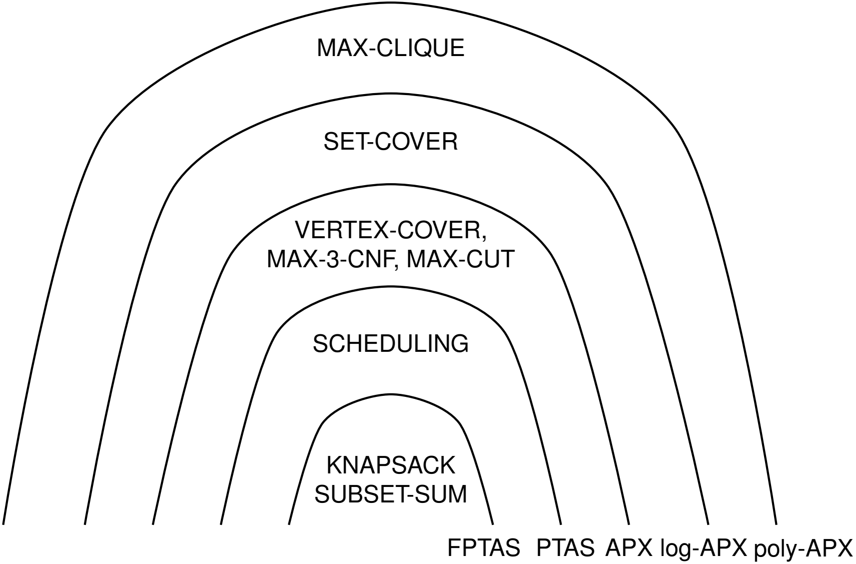
Outline

Simple Algorithms for MAX-CUT

A Solution based on Semidefinite Programming

Summary

Spectrum of Approximations



MAX-CLIQUE

SET-COVER

VERTEX-COVER,
MAX-3-CNF, MAX-CUT

SCHEDULING

KNAPSACK
SUBSET-SUM

FPTAS PTAS APX log-APX poly-APX