Approximation Algorithms III

Lecture 10

September 24, 2015

10.1: Subset Sum

Subset Sum

Instance: $X = \{x_1, \ldots, x_n\}$ – n integer positive numbers, t - target number
Question: $\exists$ subset of X s.t. sum of its elements is t?

Assume $x_1, \ldots, x_n$ are all $\leq n$. Then this problem can be solved in

(A) The problem is still **NP-Hard**, so probably exponential time.
(B) $O(n^3)$.
(C) $2^{O(\log^2 n)}$.
(D) $O(n \log n)$.
(E) None of the above.

Subset Sum

Instance: $X = \{x_1, \ldots, x_n\}$ – n integer positive numbers, t - target number
Question: $\exists$ subset of X s.t. sum of its elements is t?

M: Max value input numbers.

SolveSubsetSum (X, t, M)
    $b[0 \ldots Mn] \Leftarrow$ **false**
        // $b[x]$ is **true** if x can be
            // realized by subset of X.
    $b[0] \leftarrow$ **true**.
    for $i = 1, \ldots, n$ **do**
        for $j = Mn$ down to x_i **do**
            $b[j] \leftarrow B[j - x_i] \vee B[j]$
    return $B[t]$

Subset Sum

Instance: $X = \{x_1, \ldots, x_n\}$ – n integer positive numbers, t - target number
Question: $\exists$ subset of X s.t. sum of its elements is t?

M: Max value input numbers.

```
SolveSubsetSum (X, t, M)
  b[0 ... Mn] ⇐ false
    // b[x] is true if x can be
          // realized by subset of X.
  b[0] ← true.
  for i = 1, ..., n do
    for j = Mn down to x_i do
      b[j] ← B[j − x_i] ∨ B[j]
  return B[t]
```

Subset Sum

Instance: $X = \{x_1, \ldots, x_n\}$ – n integer positive numbers, t - target number
Question: $\exists$ subset of X s.t. sum of its elements is t?

M: Max value input numbers.

SolveSubsetSum (X, t, M)
 $b[0 \ldots Mn] \Leftarrow$ **false**
  // $b[x]$ is **true** if x can be
   // realized by subset of X.
 $b[0] \leftarrow$ **true**.
 for $i = 1, \ldots, n$ **do**
  for $j = Mn$ down to x_i **do**
   $b[j] \leftarrow B[j - x_i] \vee B[j]$
 return $B[t]$

Subset Sum

Subset Sum

Instance: $X = \{x_1, \ldots, x_n\}$ – n integer positive numbers, t - target number
Question: $\exists$ subset of X s.t. sum of its elements is t?

M: Max value input numbers.

R.T. $O(Mn^2)$.

```
SolveSubsetSum (X, t, M)
    b[0 ... Mn] ⇐ false
      // b[x] is true if x can be
            // realized by subset of X.
    b[0] ← true.
    for i = 1, ..., n do
      for j = Mn down to x_i do
        b[j] ← B[j − x_i] ∨ B[j]
    return B[t]
```

Subset Sum

Efficient algorithm???

1. Algorithm solving **Subset Sum** in $O(Mn^2)$.
2. M might be prohibitly large...
3. if $M = 2^n \implies$ algorithm is not polynomial time.
4. **Subset Sum** is **NPC**.
5. Still want to solve quickly even if M huge.
6. Optimization version:

Subset Sum Optimization

Instance: (X, t): A set X of n positive integers, and a target number t.
Question: The largest number γ_{opt} one can represent as a subset sum of X which is smaller or equal to t.

Subset Sum

Efficient algorithm???

1. Algorithm solving **Subset Sum** in $O(Mn^2)$.
2. M might be prohibitly large...
3. if $M = 2^n \implies$ algorithm is not polynomial time.
4. **Subset Sum** is **NPC**.
5. Still want to solve quickly even if M huge.
6. Optimization version:

Subset Sum Optimization

Instance: (X, t): A set X of n positive integers, and a target number t.
Question: The largest number γ_{opt} one can represent as a subset sum of X which is smaller or equal to t.

Subset Sum

Efficient algorithm???

1. Algorithm solving **Subset Sum** in $O(Mn^2)$.
2. M might be prohibitly large...
3. if $M = 2^n \implies$ algorithm is not polynomial time.
4. **Subset Sum** is **NPC**.
5. Still want to solve quickly even if M huge.
6. Optimization version:

Subset Sum Optimization

Instance: (X, t): A set X of n positive integers, and a target number t.
Question: The largest number γ_{opt} one can represent as a subset sum of X which is smaller or equal to t.

Subset Sum

2-approximation

Lemma

1. (X, t); *Given instance of* **Subset Sum**. $\gamma_{\mathrm{opt}} \leq t$: *Opt.*
2. $\implies$ *Compute legal subset with sum* $\geq \gamma_{\mathrm{opt}}/2$.
3. *Running time* $O(n \log n)$.

Proof.

1. Sort numbers in X in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. s: Generates sum.
4. u: First rejected number. s': sum before rejection.
5. $s' > u > 0$, $s' < t$, and $s' + u > t \implies$
   $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$.

Subset Sum

2-approximation

Lemma

1. (X, t); *Given instance of* **Subset Sum**. $\gamma_{\text{opt}} \leq t$: *Opt.*
2. $\implies$ *Compute legal subset with sum* $\geq \gamma_{\text{opt}}/2$.
3. *Running time* $O(n \log n)$.

Proof.

1. Sort numbers in X in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. s: Generates sum.
4. u: First rejected number. s': sum before rejection.
5. $s' > u > 0$, $s' < t$, and $s' + u > t \implies$
   $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$.

Lemma

1. (X, t); *Given instance of* **Subset Sum**. $\gamma_{\text{opt}} \leq t$: *Opt.*
2. $\implies$ *Compute legal subset with sum* $\geq \gamma_{\text{opt}}/2$.
3. *Running time* $O(n \log n)$.

Proof.

1. Sort numbers in X in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. s: Generates sum.
4. u: First rejected number. s': sum before rejection.
5. $s' > u > 0$, $s' < t$, and $s' + u > t \implies$
   $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$.

Subset Sum

2-approximation

Lemma

1. (X, t); *Given instance of* **Subset Sum**. $\gamma_{\text{opt}} \leq t$: *Opt.*
2. $\implies$ *Compute legal subset with sum* $\geq \gamma_{\text{opt}}/2$.
3. *Running time* $O(n \log n)$.

Proof.

1. Sort numbers in X in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. s: Generates sum.
4. u: First rejected number. s': sum before rejection.
5. $s' > u > 0$, $s' < t$, and $s' + u > t \implies$
   $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$.

Lemma

1. (X, t); *Given instance of* **Subset Sum**. $\gamma_{opt} \leq t$: *Opt.*
2. $\implies$ *Compute legal subset with sum* $\geq \gamma_{opt}/2$.
3. *Running time* $O(n \log n)$.

Proof.

1. Sort numbers in X in decreasing order.
2. Greedily - add numbers from largest to smallest (if possible).
3. s: Generates sum.
4. u: First rejected number. s': sum before rejection.
5. $s' > u > 0$, $s' < t$, and $s' + u > t \implies$
   $t < s' + u < s' + s' = 2s' \implies s' \geq t/2$.

10.1.1: On the complexity of ε-approximation algorithms

Polynomial Time Approximation Schemes

Definition (PTAS)

PROB: **Max**imization problem.

$\varepsilon > 0$: approximation parameter.

$\mathcal{A}(I, \varepsilon)$ is a **polynomial time approximation scheme** (**PTAS**) for **PROB**:

1. $\forall I$: $(1 - \varepsilon) \left|\mathbf{opt}(I)\right| \leq \left|\mathcal{A}(I, \varepsilon)\right| \leq \left|\mathbf{opt}(I)\right|$,
2. $|\mathbf{opt}(I)|$: opt price,
3. $|\mathcal{A}(I, \varepsilon)|$: price of solution of $\mathcal{A}$.
4. $\mathcal{A}$ running time polynomial in n for fixed ε.

For minimization problem:

$|\mathbf{opt}(I)| \leq |\mathcal{A}(I, \varepsilon)| \leq (1 + \varepsilon)|\mathbf{opt}(I)|$.

Polynomial Time Approximation Schemes

Definition (PTAS)

PROB: **Max**imization problem.

$\varepsilon > 0$: approximation parameter.

$\mathcal{A}(I, \varepsilon)$ is a **polynomial time approximation scheme** (**PTAS**) for **PROB**:

1. $\forall I$: $(1-\varepsilon)\left|\mathbf{opt}(I)\right| \leq \left|\mathcal{A}(I,\varepsilon)\right| \leq \left|\mathbf{opt}(I)\right|$,
2. $|\mathbf{opt}(I)|$: opt price,
3. $|\mathcal{A}(I,\varepsilon)|$: price of solution of $\mathcal{A}$.
4. $\mathcal{A}$ running time polynomial in n for fixed ε.

For minimization problem:
$|\mathbf{opt}(I)| \leq |\mathcal{A}(I,\varepsilon)| \leq (1+\varepsilon)|\mathbf{opt}(I)|$.

Polynomial Time Approximation Schemes

1. Example: Approximation algorithm with running time $O(n^{1/\varepsilon})$ is a PTAS.
   Algorithm with running time $O(1/\varepsilon^n)$ is not.
2. Fully polynomial...

Definition (FPTAS)

An approximation algorithm is **fully polynomial time approximation scheme** (**FPTAS**) if it is a PTAS, and its running time is polynomial both in n and $1/\varepsilon$.

3. Example: PTAS with running time $O(n^{1/\varepsilon})$ is not a FPTAS.
4. Example: PTAS with running time $O(n^2/\varepsilon^3)$ is a FPTAS.

Polynomial Time Approximation Schemes

1. Example: Approximation algorithm with running time $O(n^{1/\varepsilon})$ is a PTAS.
   Algorithm with running time $O(1/\varepsilon^n)$ is not.
2. Fully polynomial...

Definition (FPTAS)

An approximation algorithm is **fully polynomial time approximation scheme** (**FPTAS**) if it is a PTAS, and its running time is polynomial both in n and $1/\varepsilon$.

3. Example: PTAS with running time $O(n^{1/\varepsilon})$ is not a FPTAS.
4. Example: PTAS with running time $O(n^2/\varepsilon^3)$ is a FPTAS.

Polynomial Time Approximation Schemes

1. Example: Approximation algorithm with running time $O(n^{1/\varepsilon})$ is a PTAS.
   Algorithm with running time $O(1/\varepsilon^{n})$ is not.
2. Fully polynomial...

Definition (FPTAS)

An approximation algorithm is **fully polynomial time approximation scheme** (**FPTAS**) if it is a PTAS, and its running time is polynomial both in n and $1/\varepsilon$.

3. Example: PTAS with running time $O(n^{1/\varepsilon})$ is not a FPTAS.
4. Example: PTAS with running time $O(n^{2}/\varepsilon^{3})$ is a FPTAS.

Polynomial Time Approximation Schemes

1. Example: Approximation algorithm with running time $O(n^{1/\varepsilon})$ is a PTAS.
   Algorithm with running time $O(1/\varepsilon^n)$ is not.
2. Fully polynomial...

Definition (FPTAS)

An approximation algorithm is **fully polynomial time approximation scheme** (**FPTAS**) if it is a PTAS, and its running time is polynomial both in n and $1/\varepsilon$.

3. Example: PTAS with running time $O(n^{1/\varepsilon})$ is not a FPTAS.
4. Example: PTAS with running time $O(n^2/\varepsilon^3)$ is a FPTAS.

Approximating Subset Sum

Subset Sum Approx

Instance: (X, t, ε): A set X of n positive integers, a target number t, and parameter $\varepsilon > 0$.
Question: A number z that one can represent as a subset sum of X, such that $(1 - \varepsilon)\gamma_{\text{opt}} \le z \le \gamma_{\text{opt}} \le t$.

Approximating Subset Sum

Looking again at the exact algorithm

ExactSubsetSum(**S**, t)
  $n \leftarrow |S|$
  $P_0 \leftarrow \{0\}$
  for $i = 1 \dots n$ **do**
    $P_i \leftarrow P_{i-1} \cup (P_{i-1} + x_i)$
    Remove from P_i all elements $> t$

  return largest element in P_n

1. $S = \{a_1, \dots, a_n\}$
   $x + S = \{a_1 + x, a_2 + x, \dots a_n + x\}$
2. Lists might explode in size.

Trim the lists...

L': Inc. sorted list of numbers

```
Trim(L', δ)
   L = ⟨y_1 ... y_m⟩
   curr ← y_1
   L_out ← {y_1}
   for i = 2...m do
      if y_i > curr · (1 + δ)
         Append y_i to L_out
         curr ← y_i
   return L_out
```

Definition

For two positive real numbers $z \leq y$, the number y is a δ-approximation to z if

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

Observation

If $x \in L'$ then there exists a number $y \in L_{out}$ such that $y \leq x \leq y(1+\delta)$, where $L_{out} \leftarrow \textbf{Trim}(L', \delta)$.

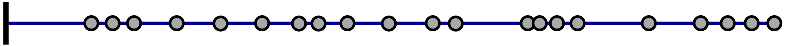
Trim the lists...

```
Trim(L', δ)
    L = ⟨y_1 ... y_m⟩
    curr ← y_1
    L_out ← {y_1}
    for i = 2 ... m do
        if y_i > curr · (1 + δ)
            Append y_i to L_out
            curr ← y_i
    return L_out
```

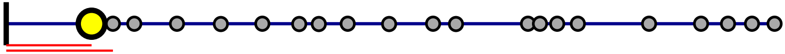
```
ApproxSubsetSum(S, t)
    // S = {x_1, ..., x_n},
    //        x_1 ≤ x_2 ≤ ... ≤ x_n
    n ← |S|, L_0 ← {0},
    δ = ε/2n
    for i = 1 ... n do
        E_i ← L_{i-1} ∪ (L_{i-1} + x_i)
        L_i ← Trim(E_i, δ)
        Remove from L_i elems > t.

    return largest element in L_n
```

E_i: Computed by merging two sorted lists in linear time.

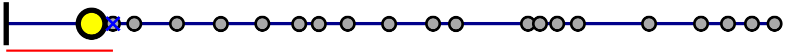
Understanding trimming



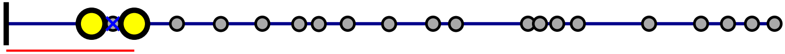
Understanding trimming



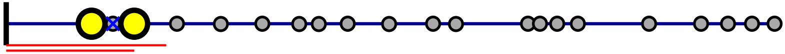
Understanding trimming



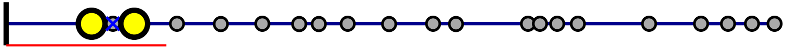
Understanding trimming



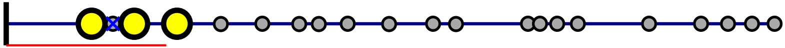
Understanding trimming



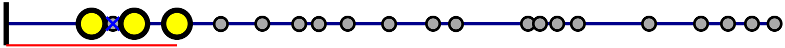
Understanding trimming



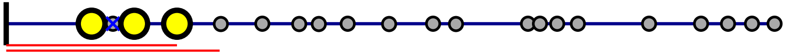
Understanding trimming



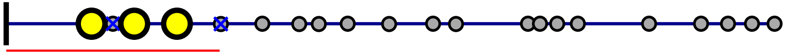
Understanding trimming



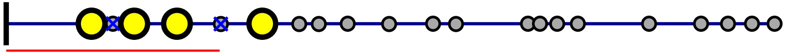
Understanding trimming



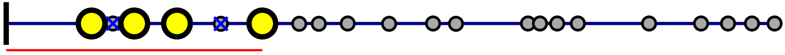
Understanding trimming



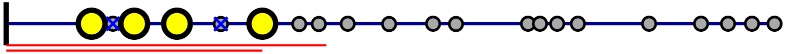
Understanding trimming



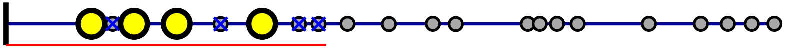
Understanding trimming



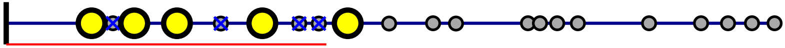
Understanding trimming



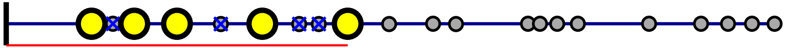
Understanding trimming



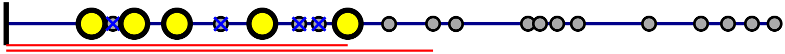
Understanding trimming



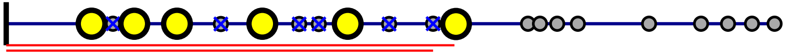
Understanding trimming



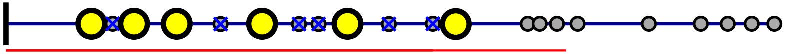
Understanding trimming



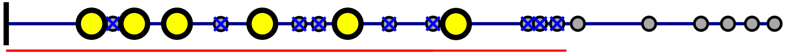
Understanding trimming



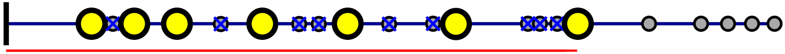
Understanding trimming



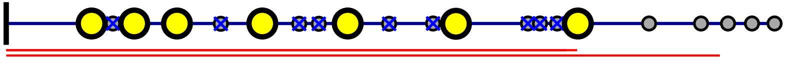
Understanding trimming



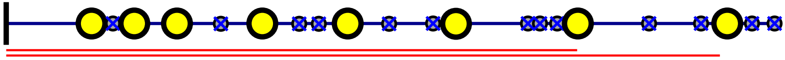
Understanding trimming



Understanding trimming



Understanding trimming



Remark

1. Can assume that trimmed lists L_i are sorted...
2. Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted lest.
5. takes linear time in size of lists.

Remark

1. Can assume that trimmed lists L_i are sorted...
2. Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted lest.
5. takes linear time in size of lists.

Remark

1. Can assume that trimmed lists L_i are sorted...
2. Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted lest.
5. takes linear time in size of lists.

Remark

1. Can assume that trimmed lists L_i are sorted...
2. Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted lest.
5. takes linear time in size of lists.

Remark

1. Can assume that trimmed lists L_i are sorted...
2. Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$
3. So, this is just copy, shift, and merge of two sorted lists.
4. ... resulting in a sorted lest.
5. takes linear time in size of lists.

Analysis

1. E_i list generated by algorithm in ith iteration.
2. P_i: list of numbers (no trimming).

Claim

For any $x \in P_i$ there exists $y \in L_i$ such that $y \leq x \leq (1+\delta)^i y$.

Proof

1. If $x \in P_1$ then follows by observation above.
2. If $x \in P_{i-1} \implies$ (induction) $\exists y' \in L_{i-1}$ s.t. $y' \leq x \leq (1+\delta)^{i-1} y'$.
3. By observation $\exists y \in L_i$ s.t. $y \leq y' \leq (1+\delta)y$, As such,

$$y \leq y' \leq x \leq (1+\delta)^{i-1} y' \leq (1+\delta)^i y.$$

Analysis

1. E_i list generated by algorithm in ith iteration.
2. P_i: list of numbers (no trimming).

Claim

For any $x \in P_i$ there exists $y \in L_i$ such that $y \le x \le (1+\delta)^i y$.

Proof

1. If $x \in P_1$ then follows by observation above.
2. If $x \in P_{i-1} \implies$ (induction) $\exists y' \in L_{i-1}$ s.t. $y' \le x \le (1+\delta)^{i-1} y'$.
3. By observation $\exists y \in L_i$ s.t. $y \le y' \le (1+\delta) y$, As such,

$$y \le y' \le x \le (1+\delta)^{i-1} y' \le (1+\delta)^i y.$$

Analysis

1. E_i list generated by algorithm in ith iteration.
2. P_i: list of numbers (no trimming).

Claim

For any $x \in P_i$ there exists $y \in L_i$ such that $y \le x \le (1+\delta)^i y$.

Proof

1. If $x \in P_1$ then follows by observation above.
2. If $x \in P_{i-1} \implies$ (induction) $\exists y' \in L_{i-1}$ s.t. $y' \le x \le (1+\delta)^{i-1} y'$.
3. By observation $\exists y \in L_i$ s.t. $y \le y' \le (1+\delta) y$, As such,

$$y \le y' \le x \le (1+\delta)^{i-1} y' \le (1+\delta)^i y.$$

Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1+\delta)^{i-1}\alpha'$.
3. Thus, $\alpha' + x_i \in E_i$.
4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1+\delta)x'$.
5. Thus, $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1}\alpha' + x_i \leq (1+\delta)^{i-1}(\alpha' + x_i) \leq (1+\delta)^i x'$. ■

Proof continued

Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1+\delta)^{i-1}\alpha'$.
3. Thus, $\alpha' + x_i \in E_i$.
4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1+\delta)x'$.
5. Thus, $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1}\alpha' + x_i \leq (1+\delta)^{i-1}(\alpha' + x_i) \leq (1+\delta)^i x'$.

Proof continued

Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \le \alpha \le (1+\delta)^{i-1}\alpha'$.
3. Thus, $\alpha' + x_i \in E_i$.
4. $\exists x' \in L_i$ s.t. $x' \le \alpha' + x_i \le (1+\delta)x'$.
5. Thus, $x' \le \alpha' + x_i \le \alpha + x_i = x \le (1+\delta)^{i-1}\alpha' + x_i \le (1+\delta)^{i-1}(\alpha' + x_i) \le (1+\delta)^i x'$. ∎

Proof continued

Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1+\delta)^{i-1}\alpha'$.
3. Thus, $\alpha' + x_i \in E_i$.
4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1+\delta)x'$.
5. Thus, $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1}\alpha' + x_i \leq (1+\delta)^{i-1}(\alpha' + x_i) \leq (1+\delta)^i x'$.

Proof continued

Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1+\delta)^{i-1}\alpha'$.
3. Thus, $\alpha' + x_i \in E_i$.
4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1+\delta)x'$.
5. Thus, $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1}\alpha' + x_i \leq (1+\delta)^{i-1}(\alpha' + x_i) \leq (1+\delta)^i x'$. ■

Proof continued

Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \le \alpha \le (1+\delta)^{i-1}\alpha'$.
3. Thus, $\alpha' + x_i \in E_i$.
4. $\exists x' \in L_i$ s.t. $x' \le \alpha' + x_i \le (1+\delta)x'$.
5. Thus, $x' \le \alpha' + x_i \le \alpha + x_i = x \le (1+\delta)^{i-1}\alpha' + x_i \le (1+\delta)^{i-1}(\alpha' + x_i) \le (1+\delta)^i x'$.

Proof continued

Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1+\delta)^{i-1}\alpha'$.
3. Thus, $\alpha' + x_i \in E_i$.
4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1+\delta)x'$.
5. Thus, $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1}\alpha' + x_i \leq (1+\delta)^{i-1}(\alpha' + x_i) \leq (1+\delta)^i x'$.

Proof continued

Proof continued

1. If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
2. By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1+\delta)^{i-1}\alpha'$.
3. Thus, $\alpha' + x_i \in E_i$.
4. $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1+\delta)x'$.
5. Thus, $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1}\alpha' + x_i \leq (1+\delta)^{i-1}(\alpha' + x_i) \leq (1+\delta)^i x'$. ■

10.1.1.1: Running time

Running time of ApproxSubsetSum

Lemma

For $x \in [0, 1]$*, it holds* $\exp(x/2) \leq (1 + x)$.

Lemma

For $0 < \delta < 1$*, and* $x \geq 1$*, we have*

$$\log_{1+\delta} x \leq \frac{2 \ln x}{\delta} = O\left(\frac{\ln x}{\delta}\right).$$

See notes for a proof of lemmas.

Running time of ApproxSubsetSum

Observation

In a list generated by **Trim**, *for any number* x, *there are no two numbers in the trimmed list between* x *and* $(1+\delta)x$.

Lemma

$|L_i| = O\Big((n/\varepsilon)\log n\Big)$, *for* $i = 1, \ldots, n$.

Running time of ApproxSubsetSum

Proof.

1. $L_{i-1} + x_i \subseteq [x_i, ix_i]$.
2. Trimming $L_{i-1} + x_i$ results in list of size

$$\log_{1+\delta} \frac{ix_i}{x_i} = O\left(\frac{\ln i}{\delta}\right) = O\left(\frac{\ln n}{\delta}\right),$$

3. Now, $\delta = \varepsilon / 2n$, and

$$|L_i| \leq |L_{i-1}| + O\left(\frac{\ln n}{\delta}\right) \leq |L_{i-1}| + O\left(\frac{n \ln n}{\varepsilon}\right)$$
$$= O\left(\frac{n^2 \log n}{\varepsilon}\right).$$

□

Running time of ApproxSubsetSum

Proof.

1. $L_{i-1} + x_i \subseteq [x_i, ix_i]$.
2. Trimming $L_{i-1} + x_i$ results in list of size

$$\log_{1+\delta} \frac{ix_i}{x_i} = O\left(\frac{\ln i}{\delta}\right) = O\left(\frac{\ln n}{\delta}\right),$$

3. Now, $\delta = \varepsilon / 2n$, and

$$|L_i| \leq |L_{i-1}| + O\left(\frac{\ln n}{\delta}\right) \leq |L_{i-1}| + O\left(\frac{n \ln n}{\varepsilon}\right)$$
$$= O\left(\frac{n^2 \log n}{\varepsilon}\right).$$

□

Running time of ApproxSubsetSum

Lemma

The running time of **ApproxSubsetSum** *is* $O\left(\frac{n^3}{\varepsilon}\log n\right)$.

Proof.

1. Running time of **ApproxSubsetSum** dominated by total length of $L_1, \ldots, L_n$.
2. Above lemma implies
$$\sum_i |L_i| = O\left(n \times \frac{n^2}{\varepsilon}\log n\right) = O\left(\frac{n^3}{\varepsilon}\log n\right).$$
3. **Trim** runs in time proportional to size of lists.
4. Overall, R.T. $O\left(\frac{n^3}{\varepsilon}\log n\right)$.

□

ApproxSubsetSum

Theorem

ApproxSubsetSum *returns* $u \leq t$*, s.t.* $\frac{\gamma_{\mathrm{opt}}}{1+\varepsilon} \leq u \leq \gamma_{\mathrm{opt}} \leq t$,
γ_{opt}*: opt solution.*
Running time is $O((n^3/\varepsilon)\log n)$.

Proof.

1. Running time from above.
2. $\gamma_{\mathrm{opt}} \in P_n$: optimal solution.
3. $\exists z \in L_n$, such that $z \leq \mathbf{opt} \leq (1+\delta)^n z$
4. $(1+\delta)^n = (1+\varepsilon/2n)^n \leq \exp\left(\frac{\varepsilon}{2}\right) \leq 1+\varepsilon$, since $1+x \leq e^x$ for $x \geq 0$.
5. $\gamma_{\mathrm{opt}}/(1+\varepsilon) \leq z \leq \mathbf{opt} \leq t$.

ApproxSubsetSum

Theorem

ApproxSubsetSum *returns* $u \leq t$, *s.t.* $\frac{\gamma_{\mathrm{opt}}}{1+\varepsilon} \leq u \leq \gamma_{\mathrm{opt}} \leq t$,
γ_{opt}: *opt solution.*
Running time is $O((n^3/\varepsilon)\log n)$.

Proof.

1. Running time from above.
2. $\gamma_{\mathrm{opt}} \in P_n$: optimal solution.
3. $\exists z \in L_n$, such that $z \leq \mathbf{opt} \leq (1+\delta)^n z$
4. $(1+\delta)^n = (1+\varepsilon/2n)^n \leq \exp\left(\frac{\varepsilon}{2}\right) \leq 1+\varepsilon$, since $1+x \leq e^x$ for $x \geq 0$.
5. $\gamma_{\mathrm{opt}}/(1+\varepsilon) \leq z \leq \mathbf{opt} \leq t$.

□

ApproxSubsetSum

Theorem

ApproxSubsetSum *returns* $u \leq t$, *s.t.* $\frac{\gamma_{\mathrm{opt}}}{1+\varepsilon} \leq u \leq \gamma_{\mathrm{opt}} \leq t$,
γ_{opt}*: opt solution.*
Running time is $O((n^3/\varepsilon)\log n)$.

Proof.

1. Running time from above.
2. $\gamma_{\mathrm{opt}} \in P_n$: optimal solution.
3. $\exists z \in L_n$, such that $z \leq \mathbf{opt} \leq (1+\delta)^n z$
4. $(1+\delta)^n = (1+\varepsilon/2n)^n \leq \exp\left(\frac{\varepsilon}{2}\right) \leq 1+\varepsilon$, since $1+x \leq e^x$ for $x \geq 0$.
5. $\gamma_{\mathrm{opt}}/(1+\varepsilon) \leq z \leq \mathbf{opt} \leq t$.

ApproxSubsetSum

Theorem

ApproxSubsetSum *returns* $u \leq t$*, s.t.* $\frac{\gamma_{\text{opt}}}{1+\varepsilon} \leq u \leq \gamma_{\text{opt}} \leq t$,
γ_{opt}*: opt solution.*
Running time is $O((n^3/\varepsilon)\log n)$*.*

Proof.

1. Running time from above.
2. $\gamma_{\text{opt}} \in P_n$: optimal solution.
3. $\exists z \in L_n$, such that $z \leq \mathbf{opt} \leq (1+\delta)^n z$
4. $(1+\delta)^n = (1+\varepsilon/2n)^n \leq \exp\left(\frac{\varepsilon}{2}\right) \leq 1+\varepsilon$, since $1+x \leq e^x$ for $x \geq 0$.
5. $\gamma_{\text{opt}}/(1+\varepsilon) \leq z \leq \mathbf{opt} \leq t$.

ApproxSubsetSum

Theorem

ApproxSubsetSum *returns* $u \le t$, *s.t.* $\frac{\gamma_{\text{opt}}}{1+\varepsilon} \le u \le \gamma_{\text{opt}} \le t$,
γ_{opt}: *opt solution.*
Running time is $O((n^3/\varepsilon)\log n)$.

Proof.

1. Running time from above.
2. $\gamma_{\text{opt}} \in P_n$: optimal solution.
3. $\exists z \in L_n$, such that $z \le \textbf{opt} \le (1+\delta)^n z$
4. $(1+\delta)^n = (1+\varepsilon/2n)^n \le \exp\left(\frac{\varepsilon}{2}\right) \le 1+\varepsilon$, since $1+x \le e^x$ for $x \ge 0$.
5. $\gamma_{\text{opt}}/(1+\varepsilon) \le z \le \textbf{opt} \le t$.

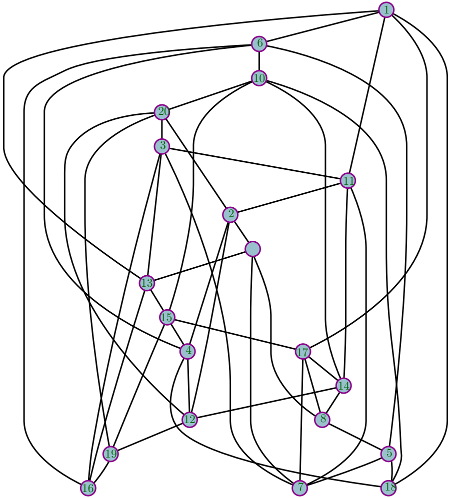
□

10.2: Maximal matching

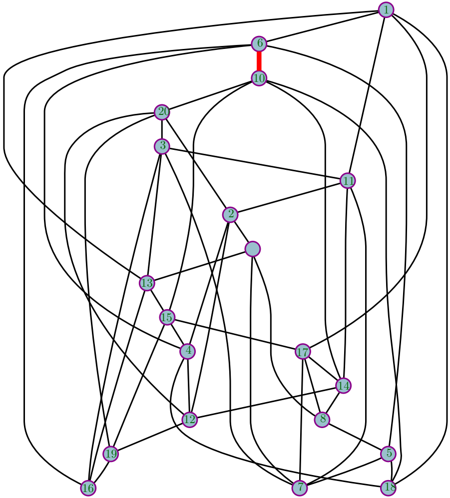
Maximal matching

1. $\mathbf{G} = (\mathbf{V}, \mathbf{E})$
2. Compute maximal matching...
3. $X \subseteq \mathbf{E}$ which is maxim**al** and independent.
4. Maximal $=$ can not improved by adding an edge.
5. Maximum $=$ largest possible set among all possible sets.
6. Computing the maximum is hard then computing maximal solution.
7. Q: Find maximal matching quickly and of large size...

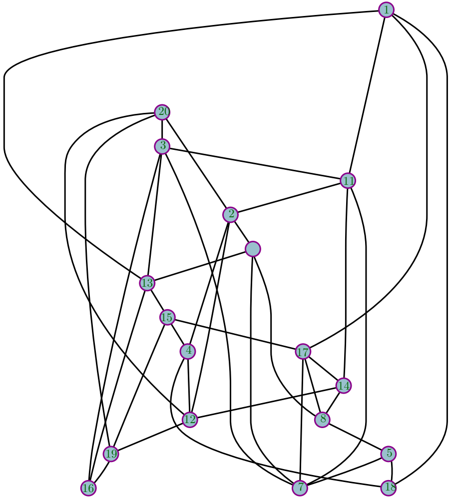
An example of the greedy algorithm...



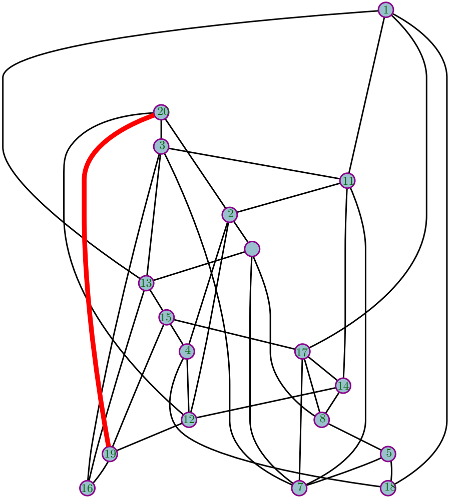
An example of the greedy algorithm...



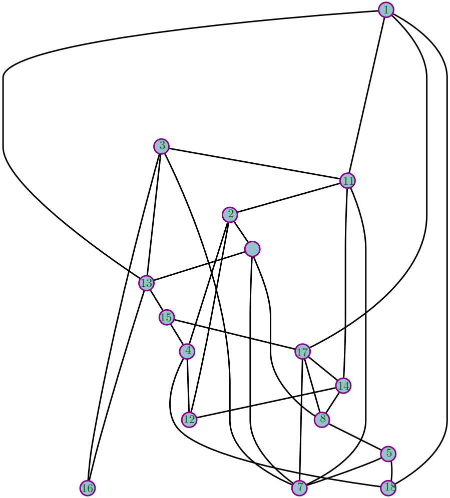
An example of the greedy algorithm...



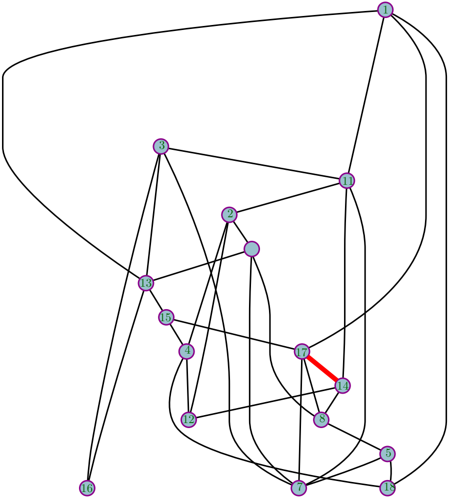
An example of the greedy algorithm...



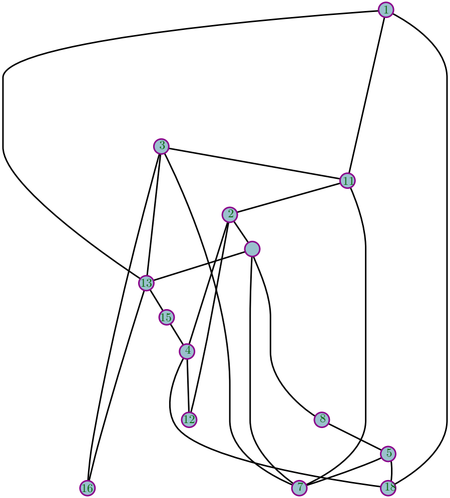
An example of the greedy algorithm...



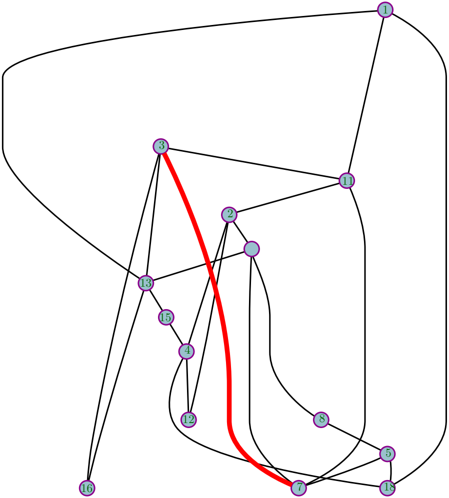
An example of the greedy algorithm...



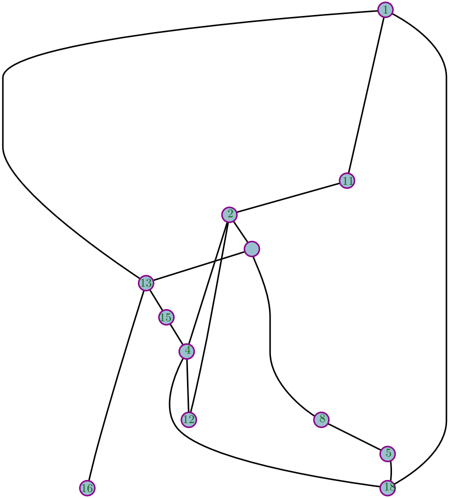
An example of the greedy algorithm...



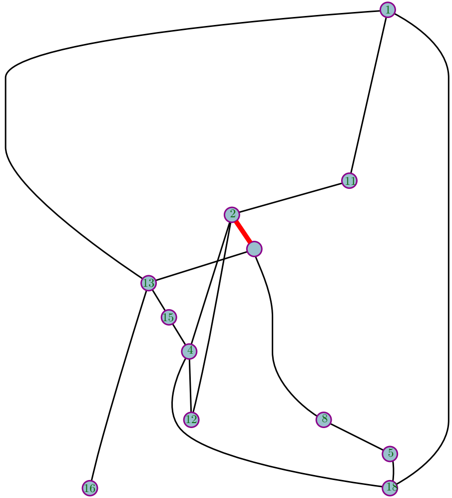
An example of the greedy algorithm...



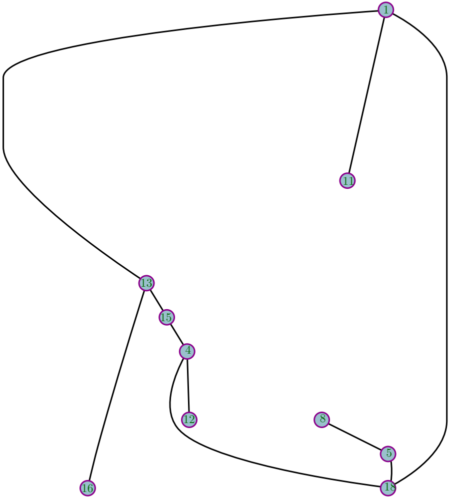
An example of the greedy algorithm...



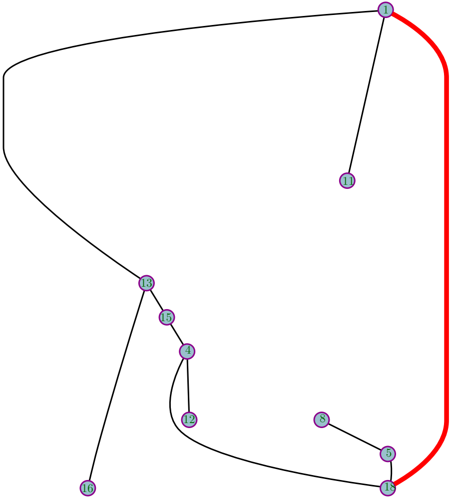
An example of the greedy algorithm...



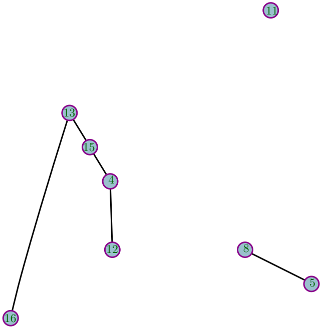
An example of the greedy algorithm...



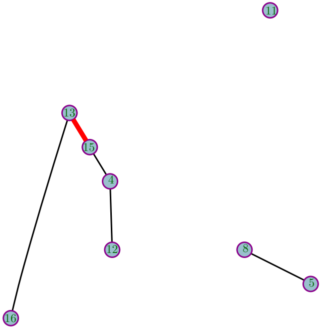
An example of the greedy algorithm...



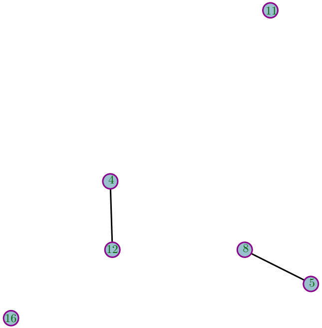
An example of the greedy algorithm...



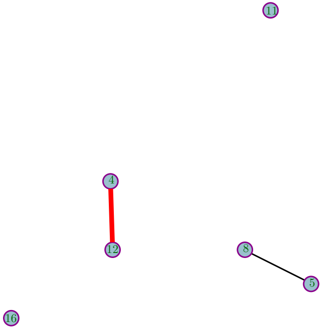
An example of the greedy algorithm...



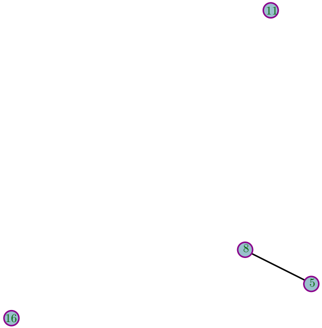
An example of the greedy algorithm...



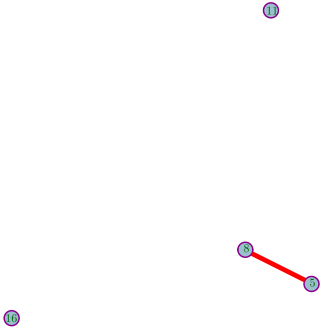
An example of the greedy algorithm...



An example of the greedy algorithm...

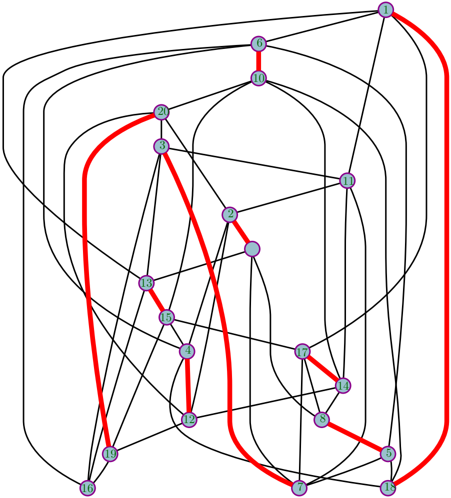


An example of the greedy algorithm...



An example of the greedy algorithm...

An example of the greedy algorithm...



Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
2. M: Generated matching. X: Maximal matching.
3. Clearly a maximal matching...
4. This is a 2-approximation to the maximum matching.
5. Because...
6. Every edge in M "kills" two edges of X in the worst case.

Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
2. M: Generated matching. X: Maximal matching.
3. Clearly a maximal matching...
4. This is a 2-approximation to the maximum matching.
5. Because...
6. Every edge in M "kills" two edges of X in the worst case.

Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
2. M: Generated matching. X: Maximal matching.
3. Clearly a maximal matching...
4. This is a 2-approximation to the maximum matching.
5. Because...
6. Every edge in M "kills" two edges of X in the worst case.

Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
2. M: Generated matching. X: Maximal matching.
3. Clearly a maximal matching...
4. This is a 2-approximation to the maximum matching.
5. Because...
6. Every edge in M "kills" two edges of X in the worst case.

Maximal matching: Algorithm

1. Algorithm: Repeatedly pick an arbitrary edge and remove it.
2. M: Generated matching. X: Maximal matching.
3. Clearly a maximal matching...
4. This is a 2-approximation to the maximum matching.
5. Because...
6. Every edge in M "kills" two edges of X in the worst case.

Maximal matching: Result

Theorem

Given a graph **G** *one can compute in* $O(n+m)$ *time, a maximal matching with at least* $|X|/2$ *edges, where* X *is the size of the maximum (optimal) matching.*

10.2.1: Bin packing

Bin Packing

Instance: v: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: n items
α_i: size of ith item.
Target: Find min # B, and a decomposition $S_1, \ldots, S_B$ of S, such that $\forall j \quad \sum_{x \in S_j} \leq v$.

1. $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...

Bin packing

Problem definition

Bin Packing

Instance: v: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: n items
α_i: size of ith item.
Target: Find min $\#$ B, and a decomposition $S_1, \ldots, S_B$ of S, such that $\forall j \quad \sum_{x \in S_j} \leq v$.

1. $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...

Bin Packing

Instance: v: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: n items
α_i: size of ith item.
Target: Find min # B, and a decomposition $S_1, \ldots, S_B$ of S, such that $\forall j \quad \sum_{x \in S_j} \leq v$.

1. $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...

Bin packing

Problem definition

Bin Packing

Instance: v: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: n items
α_i: size of ith item.
Target: Find min # B, and a decomposition $S_1, \ldots, S_B$ of S, such that $\forall j \quad \sum_{x \in S_j} \leq v$.

1. $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...

Bin Packing

Instance: v: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: n items
α_i: size of ith item.
Target: Find min # B, and a decomposition $S_1, \ldots, S_B$ of S, such that $\forall j \quad \sum_{x \in S_j} \leq v$.

1. $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...

Bin Packing

Instance: v: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: n items
α_i: size of ith item.
Target: Find min # B, and a decomposition $S_1, \ldots, S_B$ of S, such that $\forall j \quad \sum_{x \in S_j} \leq v$.

1. $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...

Bin Packing

Instance: v: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: n items
α_i: size of ith item.
Target: Find min # B, and a decomposition $S_1, \ldots, S_B$ of S, such that $\forall j \quad \sum_{x \in S_j} \leq v$.

1. $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...

Bin Packing

Instance: v: Bin size. $S = \{\alpha_1, \ldots, \alpha_n\}$: n items
α_i: size of ith item.
Target: Find min # B, and a decomposition $S_1, \ldots, S_B$ of S, such that $\forall j \quad \sum_{x \in S_j} \leq v$.

1. $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$.
2. **NP-Hard** from **Partition**.
3. **NP-Hard** to approximate within $3/2$.
4. Natural problem...
5. How to approximate?
6. First fit: Have a row of bins, insert items greedily into the first bin that fits them.
7. First fit decreasing: Sort the elements first...

Bin packing: First fit

Analysis

Lemma

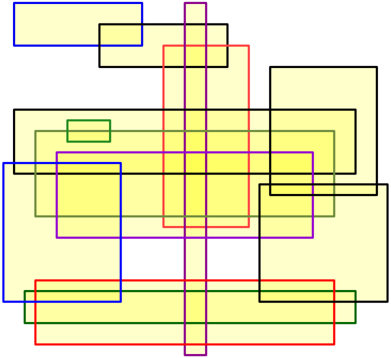
First fit is a 2-approximation.

Proof.

Observe that only one bin can have less than $v/2$ content in it... □

10.3: Independent set of axis-parallel rectangles

An example

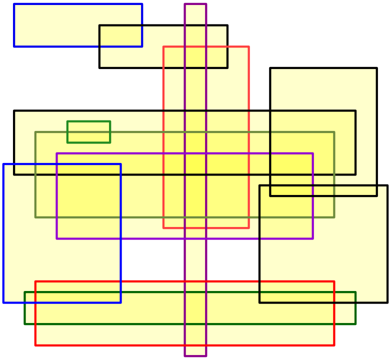


Input

Assume: Open rectangles.

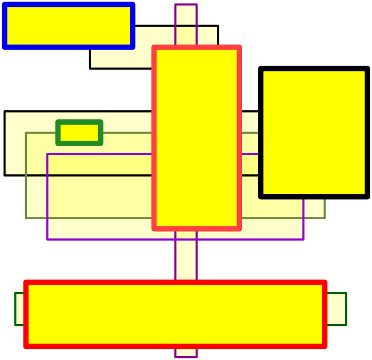
Independent set of rectangles.

An example



Input

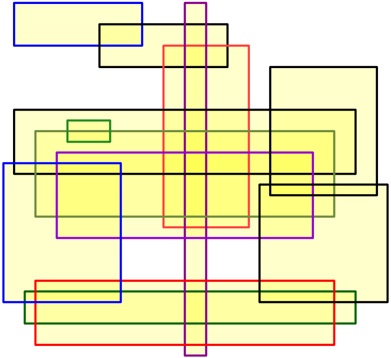
Assume: Open rectangles.



Independent set of rectangles.

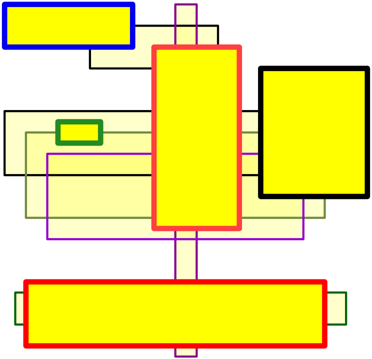
Independent set of rectangles

Algorithm: Divide & Conquer



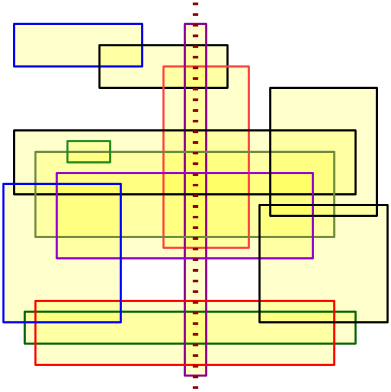
Independent set of rectangles

Algorithm: Divide & Conquer



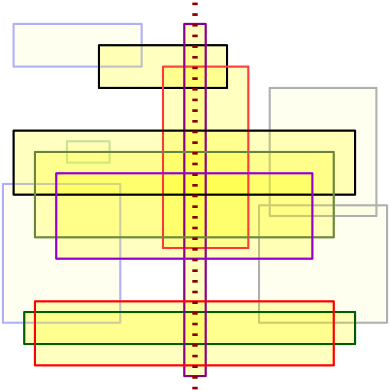
Independent set of rectangles

Algorithm: Divide & Conquer



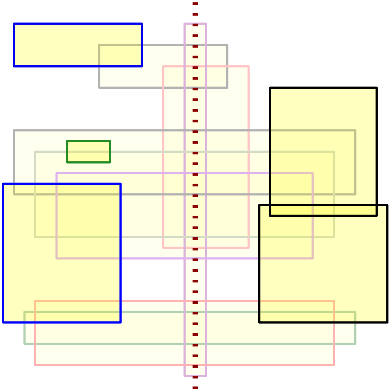
Independent set of rectangles

Algorithm: Divide & Conquer



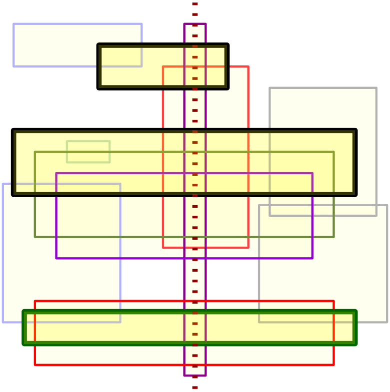
Independent set of rectangles

Algorithm: Divide & Conquer



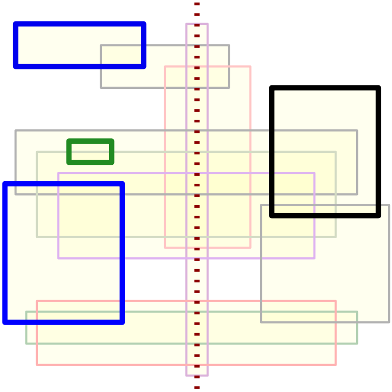
Independent set of rectangles

Algorithm: Divide & Conquer



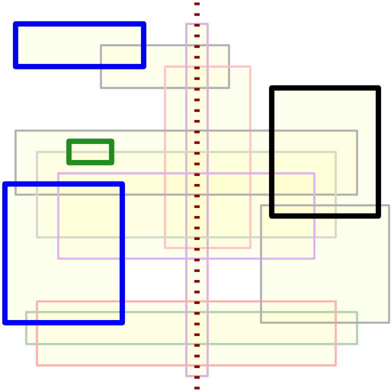
Independent set of rectangles

Algorithm: Divide & Conquer



Independent set of rectangles

Algorithm: Divide & Conquer



Independent set of rectangles

Algorithm: Divide & Conquer

$\mathcal{R}$: A set of axis parallel rectangles.

```
RectIndep(R):
  if |R| ≤ 10 then
    Solve by brute force
    return size of solution
  x_M:  Median of right x-coordinate of rects in R
  ℓ:  Vertical line through x_M.
  R_M:  Rects of R intersecting ℓ
  R_L, R_R:  Rectangles in R left/ right of ℓ.
  S_L ⇐ RectIndep(R_L)
  S_R ⇐ RectIndep(R_R)
  S_M ⇐ compute opt solution for R_M (intervals!)
  return max(S_M, S_L + S_R)
```

Analysis

1. If $S_M \geq \mathrm{Opt}/(2\lg n)$... done.
2. $\mathrm{Opt}_L + \mathrm{Opt}_R \geq (1 - 1/(2\lg n))\mathrm{Opt}$.
3. By induction: $S_L \geq \mathrm{Opt}_L/(2\lg(n/2))$ and $S_R \geq \mathrm{Opt}_R/(2\lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2\lg n))\mathrm{Opt}}{2\lg(n/2)}$
5. $$\frac{(1-1/(2\lg n))}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)}$$ $$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$
6. Conclude: If $S_M \leq \mathrm{Opt}/(2\lg n)$, then $S_L + S_R \geq \mathrm{Opt}/(2\lg n)$.
7. Algorithm is $2\lg n$ approximation.

Analysis

1. If $S_M \geq \mathrm{Opt}/(2\lg n)$... done.
2. $\mathrm{Opt}_L + \mathrm{Opt}_R \geq (1 - 1/(2\lg n))\mathrm{Opt}$.
3. By induction: $S_L \geq \mathrm{Opt}_L/(2\lg(n/2))$ and $S_R \geq \mathrm{Opt}_R/(2\lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2\lg n))\mathrm{Opt}}{2\lg(n/2)}$
5. $$\frac{(1-1/(2\lg n))}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)}$$ $$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$
6. Conclude: If $S_M \leq \mathrm{Opt}/(2\lg n)$, then $S_L + S_R \geq \mathrm{Opt}/(2\lg n)$.
7. Algorithm is $2\lg n$ approximation.

Analysis

1. If $S_M \geq \mathrm{Opt}/(2\lg n)$... done.
2. $\mathrm{Opt}_L + \mathrm{Opt}_R \geq (1 - 1/(2\lg n))\mathrm{Opt}$.
3. By induction: $S_L \geq \mathrm{Opt}_L/(2\lg(n/2))$ and $S_R \geq \mathrm{Opt}_R/(2\lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2\lg n))\mathrm{Opt}}{2\lg(n/2)}$
5. $$\frac{(1 - 1/(2\lg n))}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)}$$ $$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$
6. Conclude: If $S_M \leq \mathrm{Opt}/(2\lg n)$, then $S_L + S_R \geq \mathrm{Opt}/(2\lg n)$.
7. Algorithm is $2\lg n$ approximation.

Analysis

1. If $S_M \geq \mathrm{Opt}/(2\lg n)$... done.
2. $\mathrm{Opt}_L + \mathrm{Opt}_R \geq (1 - 1/(2\lg n))\mathrm{Opt}$.
3. By induction: $S_L \geq \mathrm{Opt}_L/(2\lg(n/2))$ and $S_R \geq \mathrm{Opt}_R/(2\lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2\lg n))\mathrm{Opt}}{2\lg(n/2)}$
5. $$\frac{(1-1/(2\lg n))}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)}$$ $$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$
6. Conclude: If $S_M \leq \mathrm{Opt}/(2\lg n)$, then $S_L + S_R \geq \mathrm{Opt}/(2\lg n)$.
7. Algorithm is $2\lg n$ approximation.

Analysis

1. If $S_M \geq \text{Opt}/(2\lg n)$... done.
2. $\text{Opt}_L + \text{Opt}_R \geq (1 - 1/(2\lg n))\text{Opt}$.
3. By induction: $S_L \geq \text{Opt}_L/(2\lg(n/2))$ and $S_R \geq \text{Opt}_R/(2\lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2\lg n))\text{Opt}}{2\lg(n/2)}$
5. $$\frac{(1 - 1/(2\lg n))}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)}$$
   $$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$
6. Conclude: If $S_M \leq \text{Opt}/(2\lg n)$, then $S_L + S_R \geq \text{Opt}/(2\lg n)$.
7. Algorithm is $2\lg n$ approximation.

Analysis

1. If $S_M \geq \mathrm{Opt}/(2\lg n)$... done.
2. $\mathrm{Opt}_L + \mathrm{Opt}_R \geq (1 - 1/(2\lg n))\mathrm{Opt}$.
3. By induction: $S_L \geq \mathrm{Opt}_L/(2\lg(n/2))$ and $S_R \geq \mathrm{Opt}_R/(2\lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2\lg n))\mathrm{Opt}}{2\lg(n/2)}$
5. $$\frac{(1-1/(2\lg n))}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)}$$ $$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$
6. Conclude: If $S_M \leq \mathrm{Opt}/(2\lg n)$, then $S_L + S_R \geq \mathrm{Opt}/(2\lg n)$.
7. Algorithm is $2\lg n$ approximation.

Analysis

1. If $S_M \geq \mathrm{Opt}/(2\lg n)$... done.
2. $\mathrm{Opt}_L + \mathrm{Opt}_R \geq (1 - 1/(2\lg n))\mathrm{Opt}$.
3. By induction: $S_L \geq \mathrm{Opt}_L/(2\lg(n/2))$ and $S_R \geq \mathrm{Opt}_R/(2\lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2\lg n))\mathrm{Opt}}{2\lg(n/2)}$
5. $$\frac{(1 - 1/(2\lg n))}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)}$$
$$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$
6. Conclude: If $S_M \leq \mathrm{Opt}/(2\lg n)$, then $S_L + S_R \geq \mathrm{Opt}/(2\lg n)$.
7. Algorithm is $2\lg n$ approximation.

Analysis

1. If $S_M \geq \mathrm{Opt}/(2\lg n)$... done.
2. $\mathrm{Opt}_L + \mathrm{Opt}_R \geq (1 - 1/(2\lg n))\mathrm{Opt}$.
3. By induction: $S_L \geq \mathrm{Opt}_L/(2\lg(n/2))$ and $S_R \geq \mathrm{Opt}_R/(2\lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2\lg n))\mathrm{Opt}}{2\lg(n/2)}$
5. $$\frac{(1 - 1/(2\lg n))}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)}$$ $$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$
6. Conclude: If $S_M \leq \mathrm{Opt}/(2\lg n)$, then $S_L + S_R \geq \mathrm{Opt}/(2\lg n)$.
7. Algorithm is $2\lg n$ approximation.

Analysis

1. If $S_M \geq \mathrm{Opt}/(2\lg n)$... done.
2. $\mathrm{Opt}_L + \mathrm{Opt}_R \geq (1 - 1/(2\lg n))\mathrm{Opt}$.
3. By induction: $S_L \geq \mathrm{Opt}_L/(2\lg(n/2))$ and $S_R \geq \mathrm{Opt}_R/(2\lg(n/2))$.
4. $S_L + S_R \geq \frac{(1-1/(2\lg n))\mathrm{Opt}}{2\lg(n/2)}$
5. $$\frac{(1 - 1/(2\lg n))}{2\lg(n/2)} = \frac{1}{2\lg n - 2} - \frac{1}{(2\lg n)(2\lg n - 2)}$$ $$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$
6. Conclude: If $S_M \leq \mathrm{Opt}/(2\lg n)$, then $S_L + S_R \geq \mathrm{Opt}/(2\lg n)$.
7. Algorithm is $2\lg n$ approximation.

Notes

Notes

Notes

Notes