Chapter 10

Approximation Algorithms III

NEW CS 473: Theory II, Fall 2015
September 24, 2015

10.1 Subset Sum

10.1.0.1 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.

10.1.0.2 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]
```

10.1.1 Subset Sum

10.1.1.1 Efficient algorithm???

(A) Algorithm solving **Subset Sum** in $O(Mn^2)$.
(B) M might be prohibitly large...
(C) if $M = 2^n \implies$ algorithm is not polynomial time.
(D) **Subset Sum** is **NPC**.
(E) Still want to solve quickly even if M huge.
(F) 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.

10.1.2 Subset Sum

10.1.2.1 2-approximation

Lemma 10.1.1. *(A) (X, t); Given instance of* **Subset Sum**. $\gamma_{\mathrm{opt}} \leq t$*: Opt.*
(B) $\implies$ Compute legal subset with sum $\geq \gamma_{\mathrm{opt}}/2$.
(C) Running time $O(n \log n)$.

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

10.1.3 On the complexity of ε-approximation algorithms

10.1.3.1 Polynomial Time Approximation Schemes

Definition 10.1.2 (PTAS). **PROB**: **Max**imization problem.
$\varepsilon > 0$: approximation parameter.
$\mathcal{A}(I, \varepsilon)$ is a ***polynomial time approximation scheme*** (**PTAS**) for **PROB**:
(A) $\forall I$: $(1-\varepsilon)\left|\mathsf{opt}(I)\right| \leq \left|\mathcal{A}(I,\varepsilon)\right| \leq \left|\mathsf{opt}(I)\right|$,
(B) $|\mathsf{opt}(I)|$: opt price,
(C) $|\mathcal{A}(I,\varepsilon)|$: price of solution of $\mathcal{A}$.
(D) $\mathcal{A}$ running time polynomial in n for fixed ε.
For minimization problem: $|\mathsf{opt}(I)| \leq |\mathcal{A}(I,\varepsilon)| \leq (1+\varepsilon)|\mathsf{opt}(I)|$.

10.1.3.2 Polynomial Time Approximation Schemes

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

Definition 10.1.3 (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 $1/\varepsilon$.

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

10.1.3.3 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 $(1 - \varepsilon)\gamma_{\text{opt}} \leq z \leq \gamma_{\text{opt}} \leq t$.

10.1.4 Approximating Subset Sum

10.1.4.1 Looking again at the exact algorithm

```
ExactSubsetSum(S, t)
    n ← |S|
    P_0 ← {0}
    for i = 1...n do
        P_i ← P_{i-1} ∪ (P_{i-1} + x_i)
        Remove from P_i all elements > t

    return largest element in P_n
```

(A) $S = \{a_1, \ldots, a_n\}$
$x + S = \{a_1 + x, a_2 + x, \ldots a_n + x\}$
(B) Lists might explode in size.

10.1.4.2 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 10.1.4. 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 10.1.5. *If $x \in L'$ then there exists a number $y \in L_{out}$ such that $y \leq x \leq y(1 + \delta)$, where $L_{out} \leftarrow$ **Trim**(L', δ).*

10.1.4.3 Trim the lists...

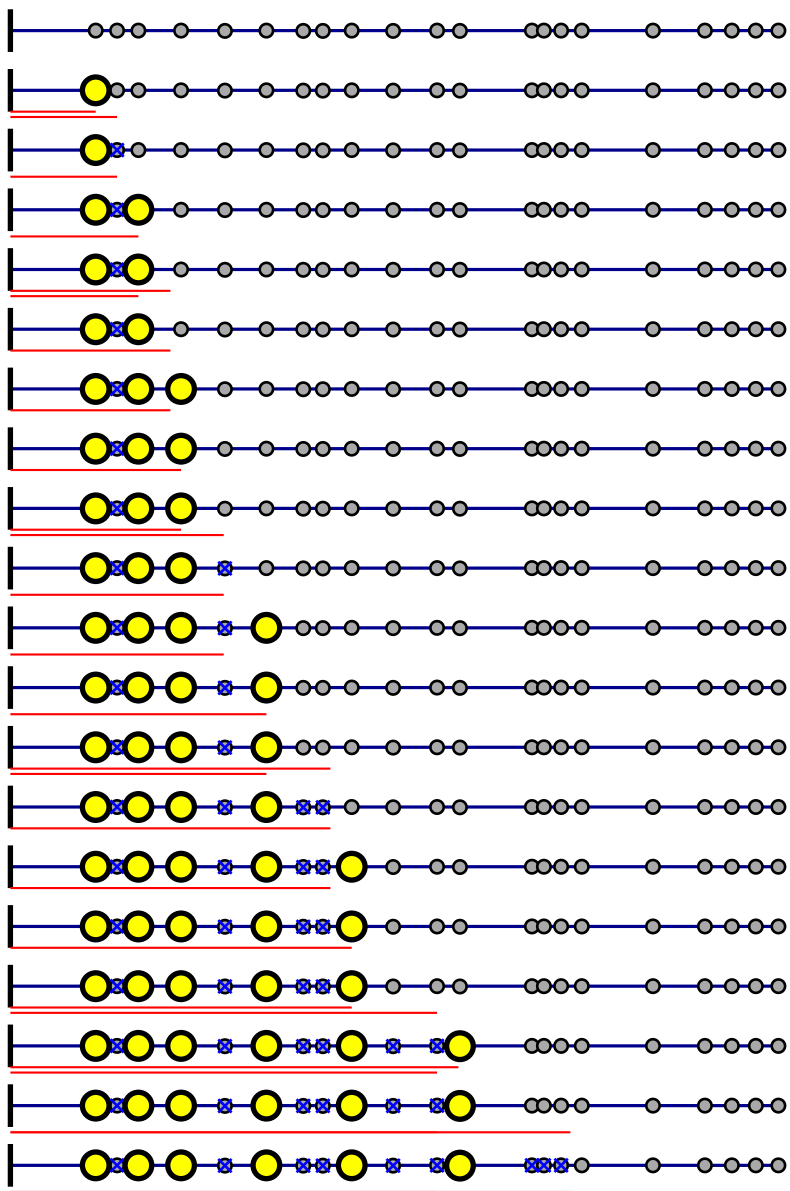
```
Trim(L', δ)
    L = ⟨y_1 ... y_m⟩
    curr ← y_1
    L_out ← {y_1}
    for i = 2 ... m do
        if y_i > curr · (1 + δ)
            Append y_i to L_out
            curr ← y_i
    return L_out
```

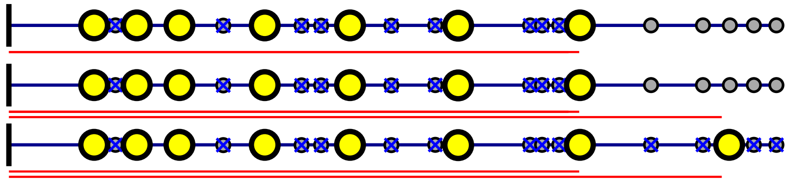
```
ApproxSubsetSum(S, t)
    // S = {x_1, ..., x_n},
    //        x_1 ≤ x_2 ≤ ... ≤ x_n
    n ← |S|, L_0 ← {0},
    δ = ε/2n
    for i = 1 ... n do
        E_i ← L_{i-1} ∪ (L_{i-1} + x_i)
        L_i ← Trim(E_i, δ)
        Remove from L_i elems > t.

    return largest element in L_n
```

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

10.1.4.4 Understanding trimming





10.1.4.5 Remark

(A) Can assume that trimmed lists L_i are sorted...
(B) Algorithm: $E_i \leftarrow L_{i-1} \cup (L_{i-1} + x_i)$

(C) So, this is just copy, shift, and merge of two sorted lists.
(D) ... resulting in a sorted lest.
(E) takes linear time in size of lists.

10.1.4.6 Analysis

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

Claim 10.1.6. *For any $x \in P_i$ there exists $y \in L_i$ such that $y \leq x \leq (1+\delta)^i y$.*

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

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

10.1.4.7 Proof continued

Proof continued
(A) If $x \in P_i \setminus P_{i-1} \implies x = \alpha + x_i$, for some $\alpha \in P_{i-1}$.
(B) By induction, $\exists \alpha' \in L_{i-1}$ s.t. $\alpha' \leq \alpha \leq (1+\delta)^{i-1} \alpha'$.
(C) Thus, $\alpha' + x_i \in E_i$.
(D) $\exists x' \in L_i$ s.t. $x' \leq \alpha' + x_i \leq (1+\delta) x'$.
(E) Thus, $x' \leq \alpha' + x_i \leq \alpha + x_i = x \leq (1+\delta)^{i-1}\alpha' + x_i \leq (1+\delta)^{i-1}(\alpha' + x_i) \leq (1+\delta)^i x'$. ∎

10.1.4.8 Running time
10.1.4.9 Running time of ApproxSubsetSum

Lemma 10.1.7. *For $x \in [0,1]$, it holds* $\exp(x/2) \leq (1+x)$.

Lemma 10.1.8. *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.

10.1.4.10 Running time of ApproxSubsetSum

Observation 10.1.9. *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 10.1.10. $|L_i| = O\Big((n/\varepsilon)\log n\Big)$*, for* $i = 1, \ldots, n$.

10.1.4.11 Running time of ApproxSubsetSum

Proof: (A) $L_{i-1} + x_i \subseteq [x_i, ix_i]$.
(B) Trimming $L_{i-1} + x_i$ results in list of size

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

(C) Now, $\delta = \varepsilon/2n$, and

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

10.1.4.12 Running time of ApproxSubsetSum

Lemma 10.1.11. *The running time of* **ApproxSubsetSum** *is* $O\Big(\frac{n^3}{\varepsilon}\log n\Big)$.

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

10.1.4.13 ApproxSubsetSum

Theorem 10.1.12. **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 $O((n^3/\varepsilon)\log n)$.

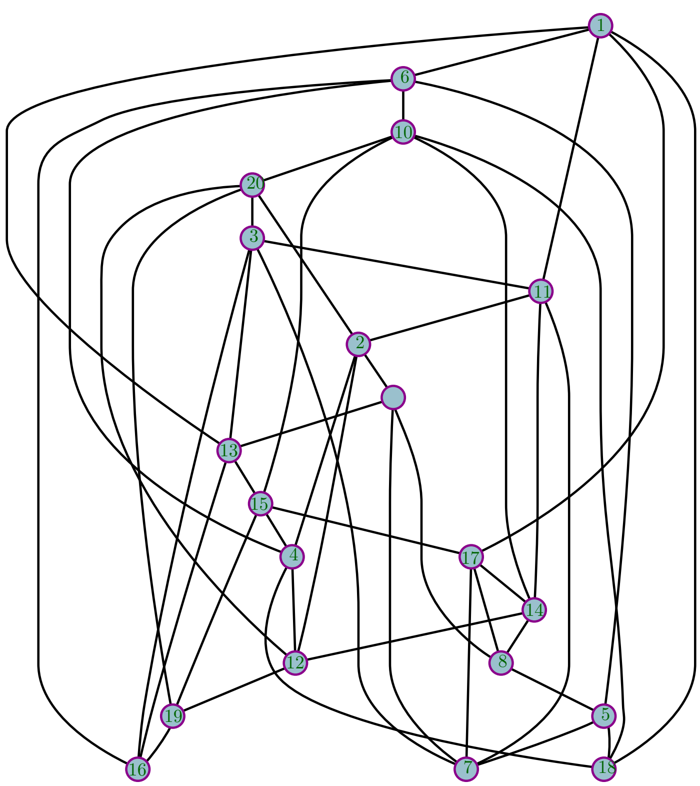
Proof: (A) Running time from above.
(B) $\gamma_{\mathrm{opt}} \in P_n$: optimal solution.
(C) $\exists z \in L_n$, such that $z \leq \mathsf{opt} \leq (1+\delta)^n z$
(D) $(1+\delta)^n = (1+\varepsilon/2n)^n \leq \exp\big(\frac{\varepsilon}{2}\big) \leq 1+\varepsilon$, since $1+x \leq e^x$ for $x \geq 0$.
(E) $\gamma_{\mathrm{opt}}/(1+\varepsilon) \leq z \leq \mathsf{opt} \leq t$.

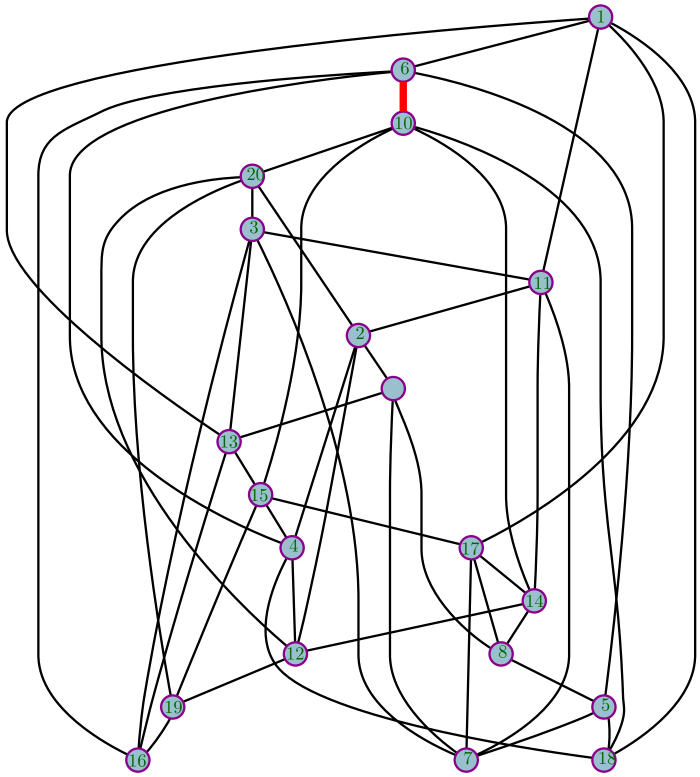
10.2 Maximal matching

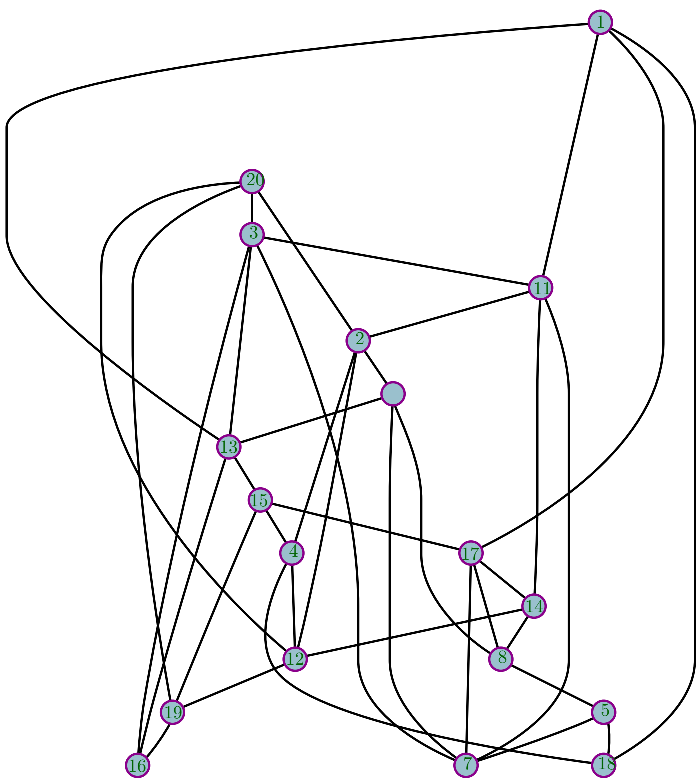
10.2.0.1 Maximal matching

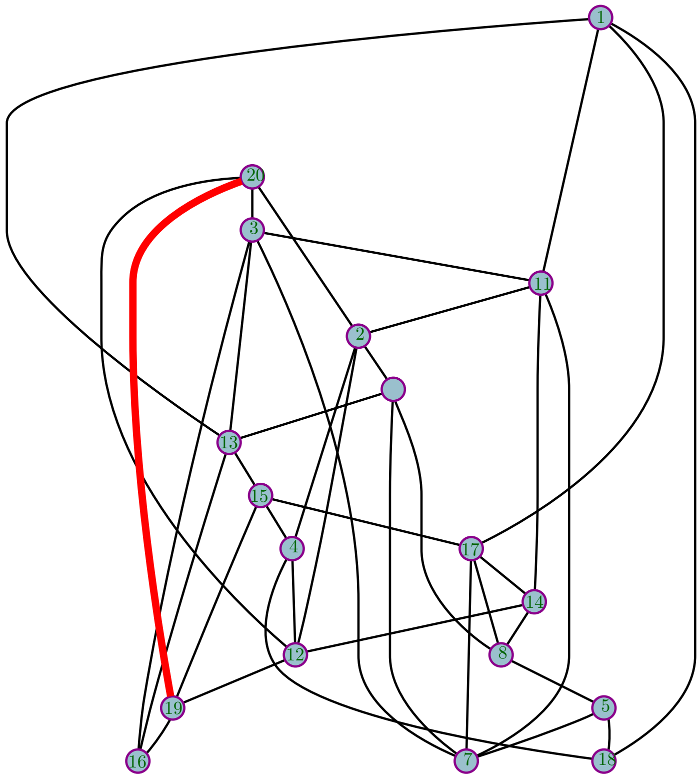
(A) $\mathsf{G} = (\mathsf{V}, \mathsf{E})$
(B) Compute maximal matching...
(C) $X \subseteq \mathsf{E}$ which is maxim**al** and independent.
(D) Maximal = can not improved by adding an edge.
(E) Maximum = largest possible set among all possible sets.
(F) Computing the maximum is hard then computing maximal solution.
(G) Q: Find maximal matching quickly and of large size...

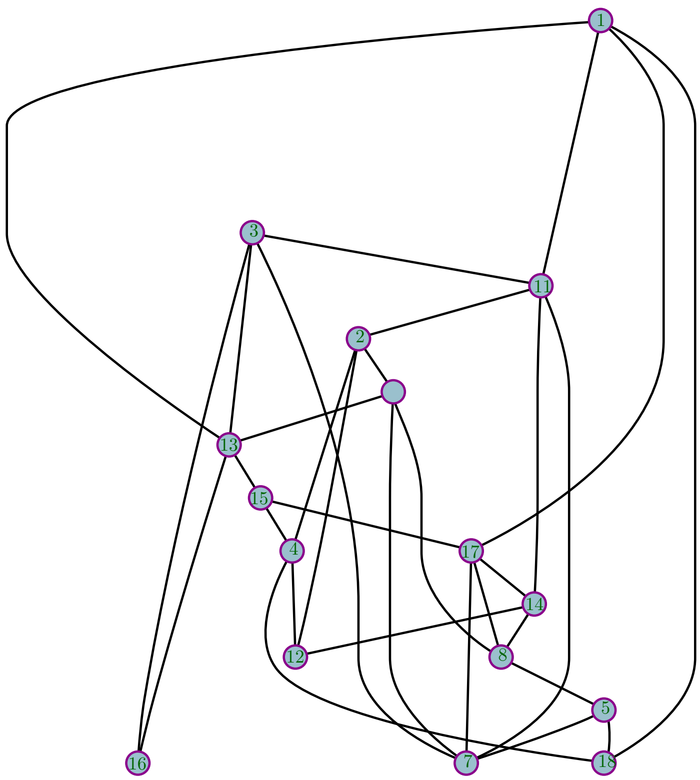
10.2.0.2 An example of the greedy algorithm...

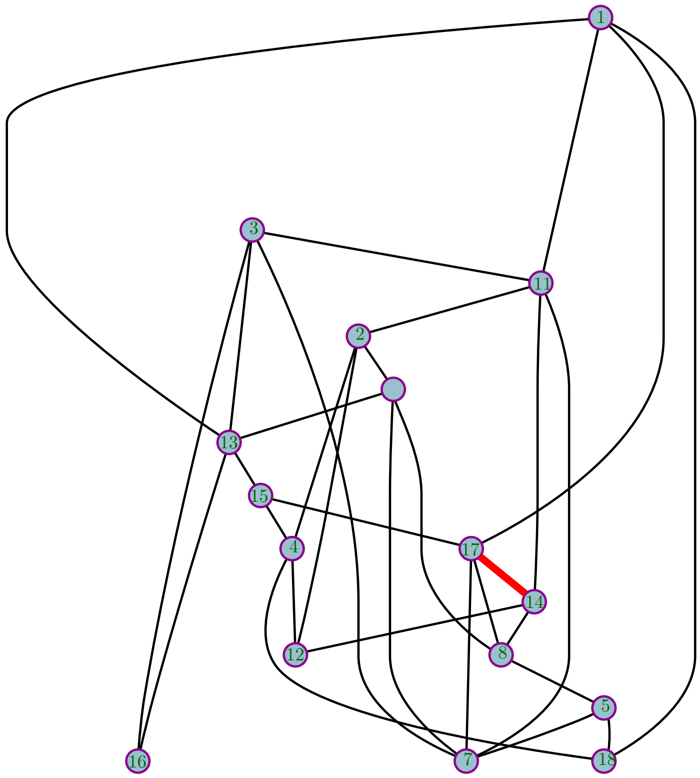


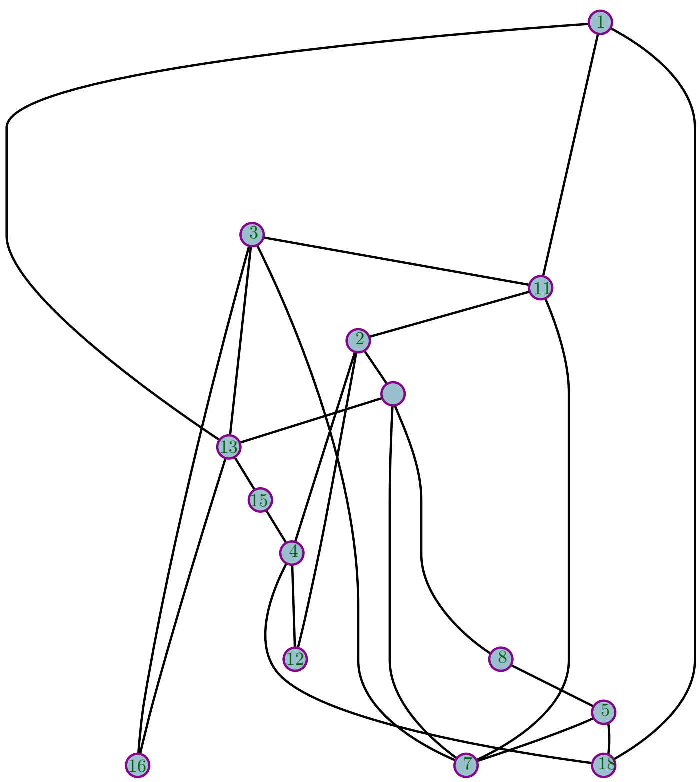

1
6
10
20
3
11
2
13
15
4
17
14
8
12
5
19
16
7
18

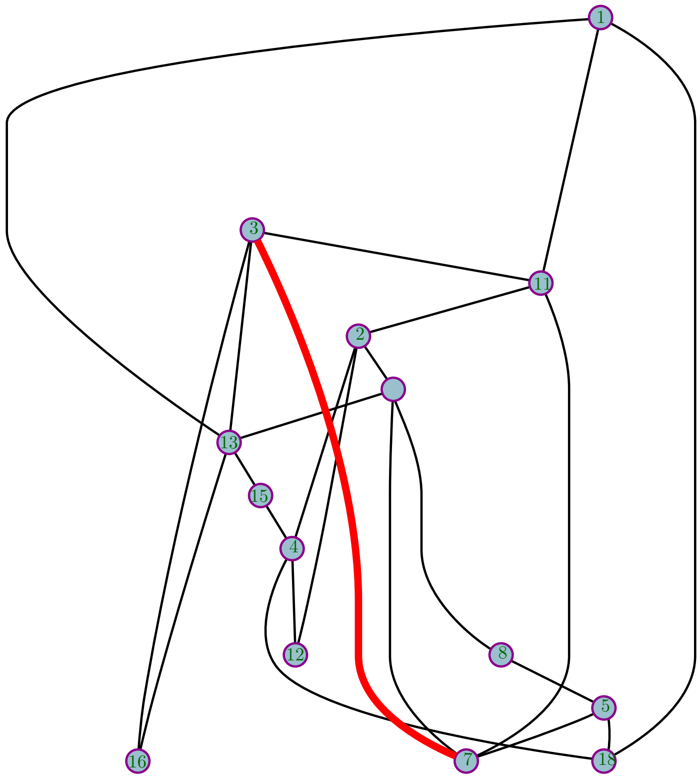

1
20
3
11
2
13
15
4
17
14
8
12
5
19
16
7
18

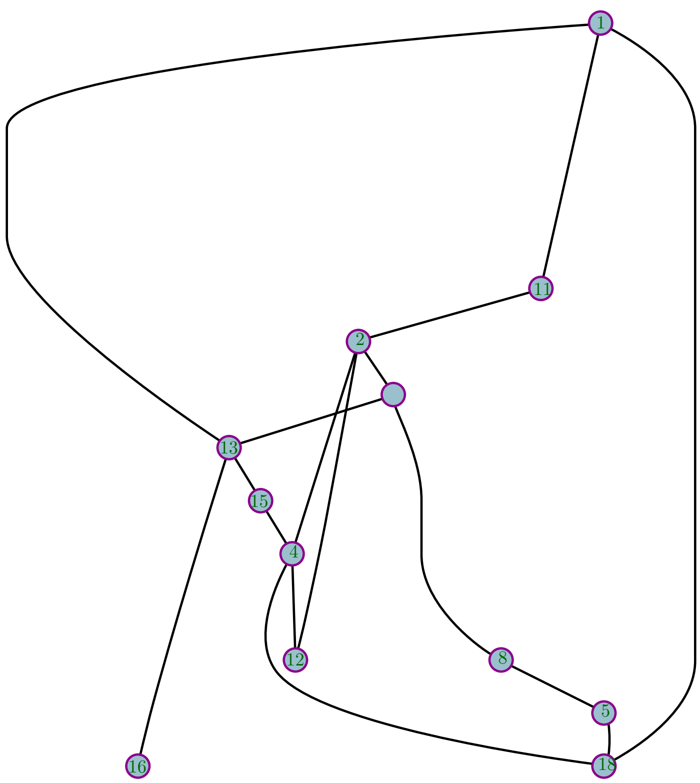

1
20
3
11
2
13
15
17
4
14
12
8
5
19
16
7
18

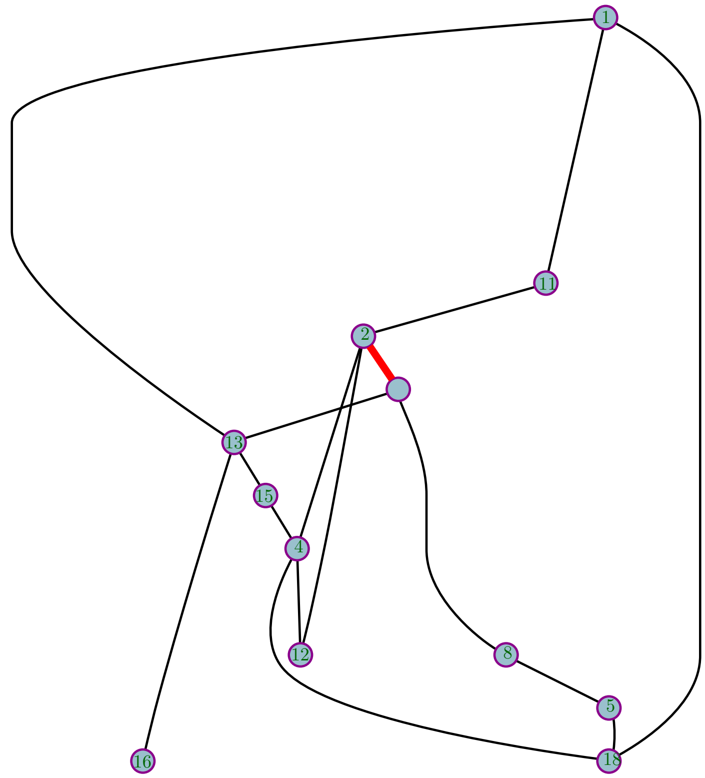

1
3
11
2
13
15
17
4
14
12
8
5
16
7
18

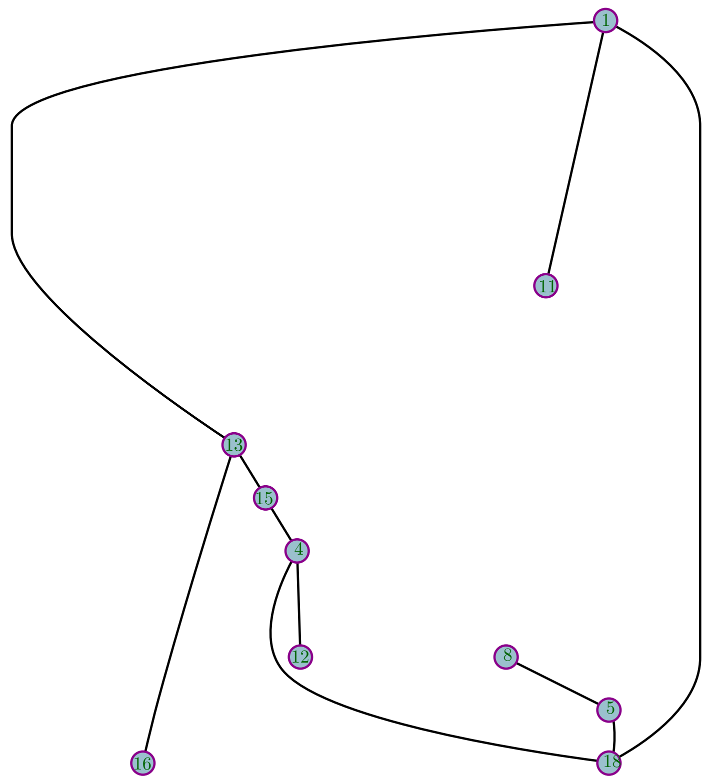

1
3
11
2
13
15
17
4
14
12
8
5
16
7
18

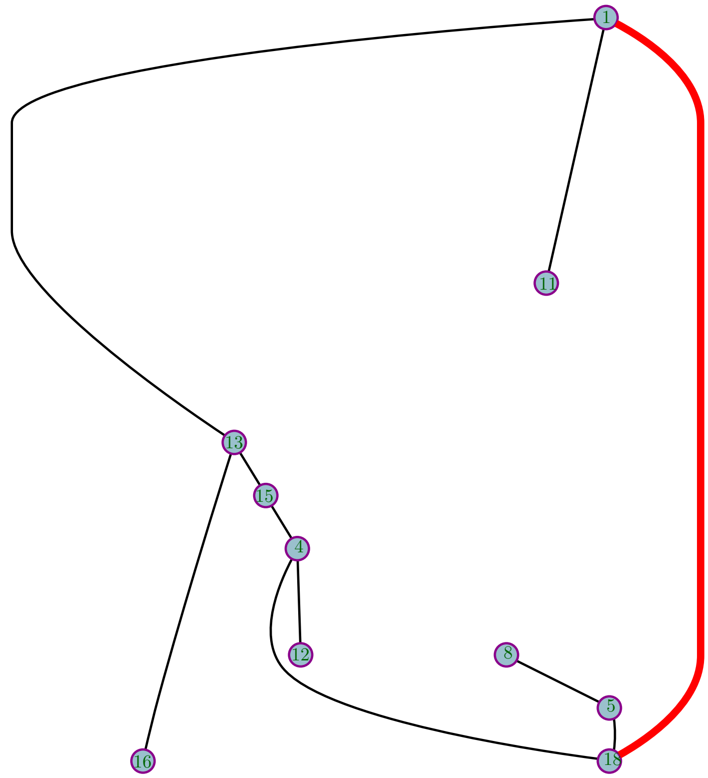

1
3
11
2
13
15
4
12
8
5
16
7
18

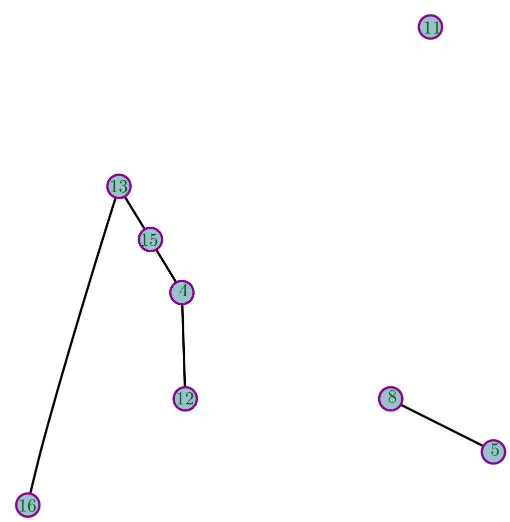

1
3
11
2
13
15
4
12
8
5
16
7
18

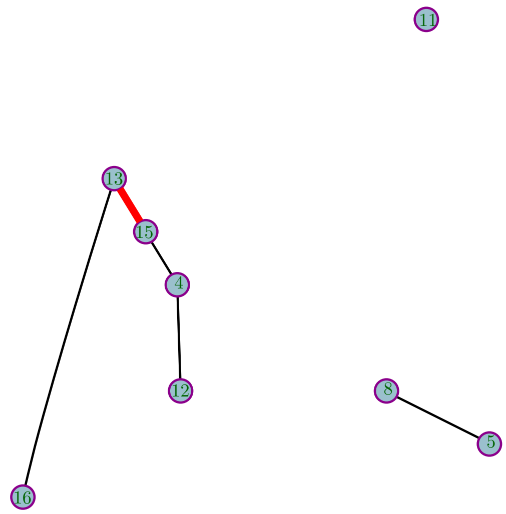

1
11
2
13
15
4
12
8
5
16
18

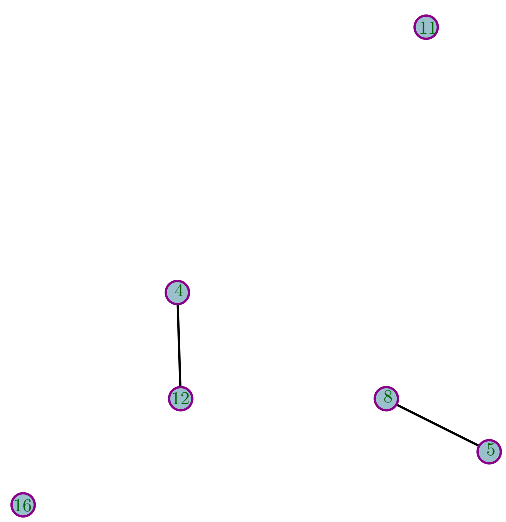

1
11
2
13
15
4
12
8
5
16
18

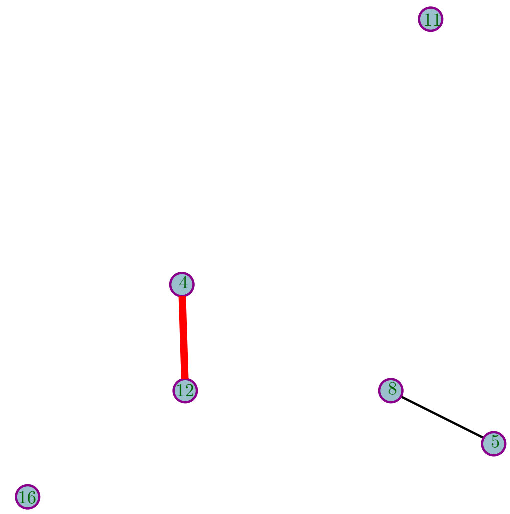

1
11
13
15
4
12
8
5
16
18

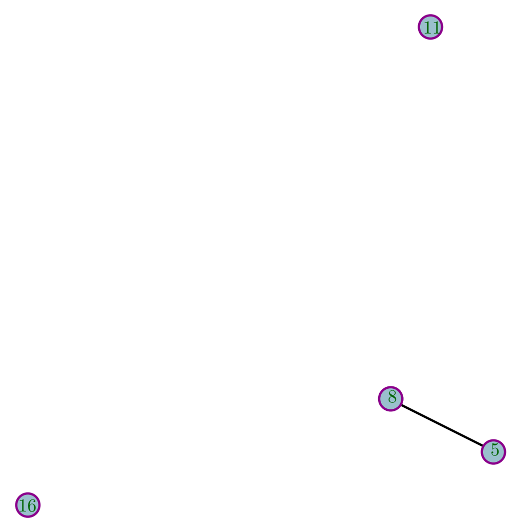

1
11
13
15
4
12
8
5
16
18

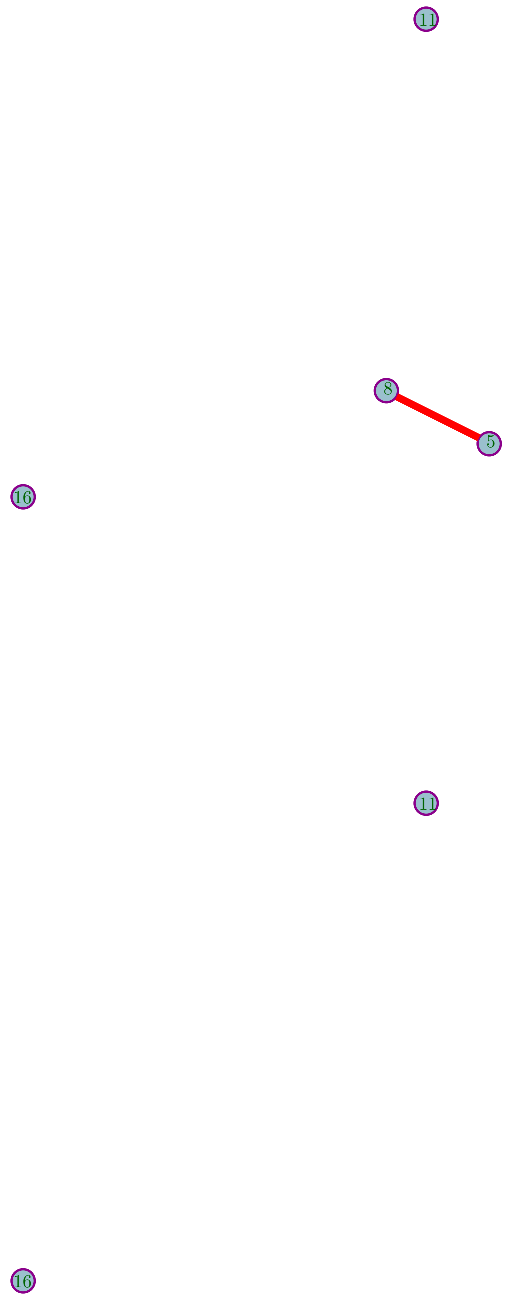

11
13
15
4
12
8
5
16

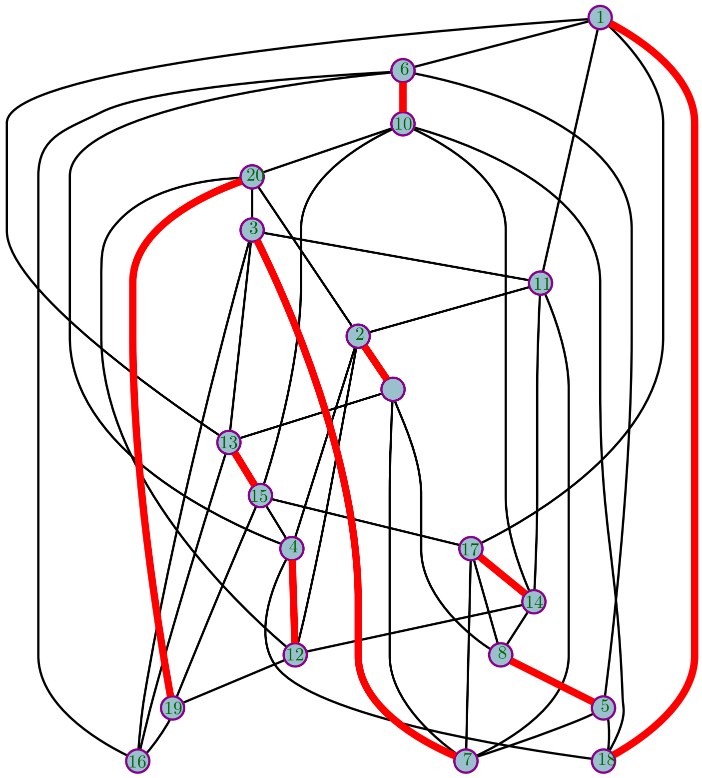

11
13
15
4
12
8
5
16


11
4
12
8
5
16


11
4
12
8
5
16


11
8
5
16


11
8
5
16
11
16



10.2.0.3 Maximal matching: Algorithm

(A) Algorithm: Repeatedly pick an arbitrary edge and remove it.
(B) M: Generated matching. X: Maximal matching.
(C) Clearly a maximal matching...
(D) This is a 2-approximation to the maximum matching.
(E) Because...
(F) Every edge in M "kills" two edges of X in the worst case.

10.2.0.4 Maximal matching: Result

Theorem 10.2.1. *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

10.2.2 Bin packing

10.2.2.1 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 $\forall j \quad \sum_{x \in S_j} \leq v$.

(A) $\cup_i S_i = S$ and $\forall i \neq j \quad S_i \cap S_j = \emptyset$.

(B) **NP-Hard** from **Partition**.
(C) **NP-Hard** to approximate within 3/2.
(D) Natural problem...
(E) How to approximate?
(F) First fit: Have a row of bins, insert items greedily into the first bin that fits them.
(G) First fit decreasing: Sort the elements first...

10.2.3 Bin packing: First fit

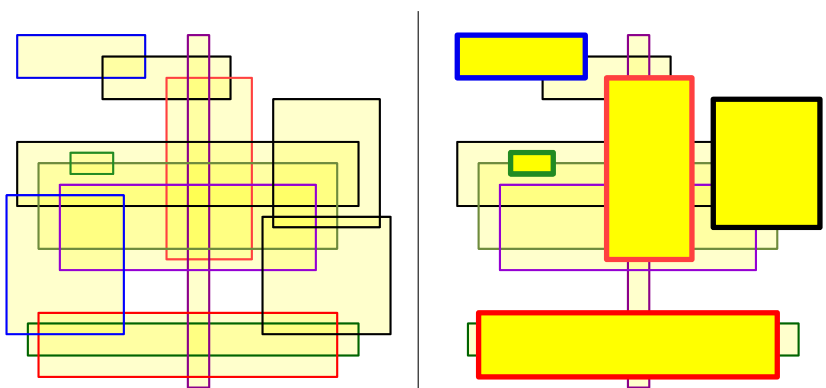
10.2.3.1 Analysis

Lemma 10.2.2. *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

10.3.0.1 An example

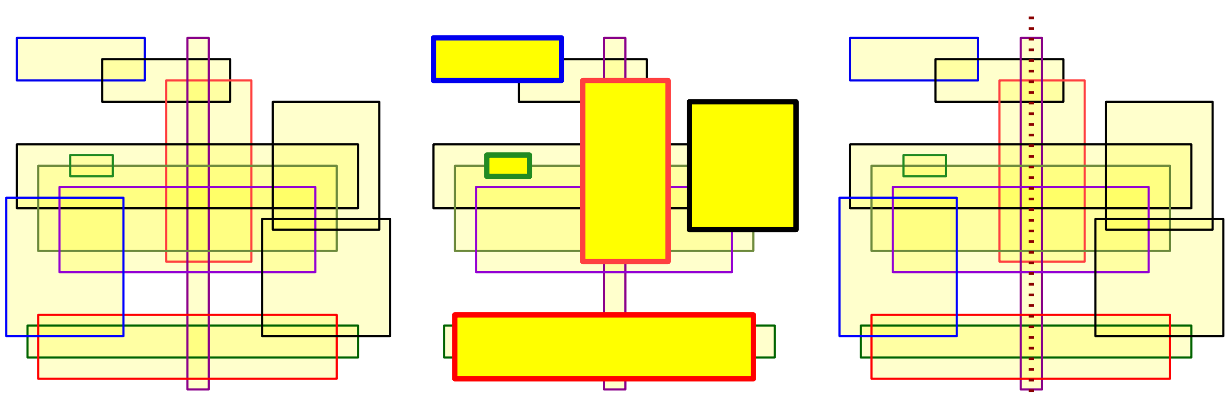


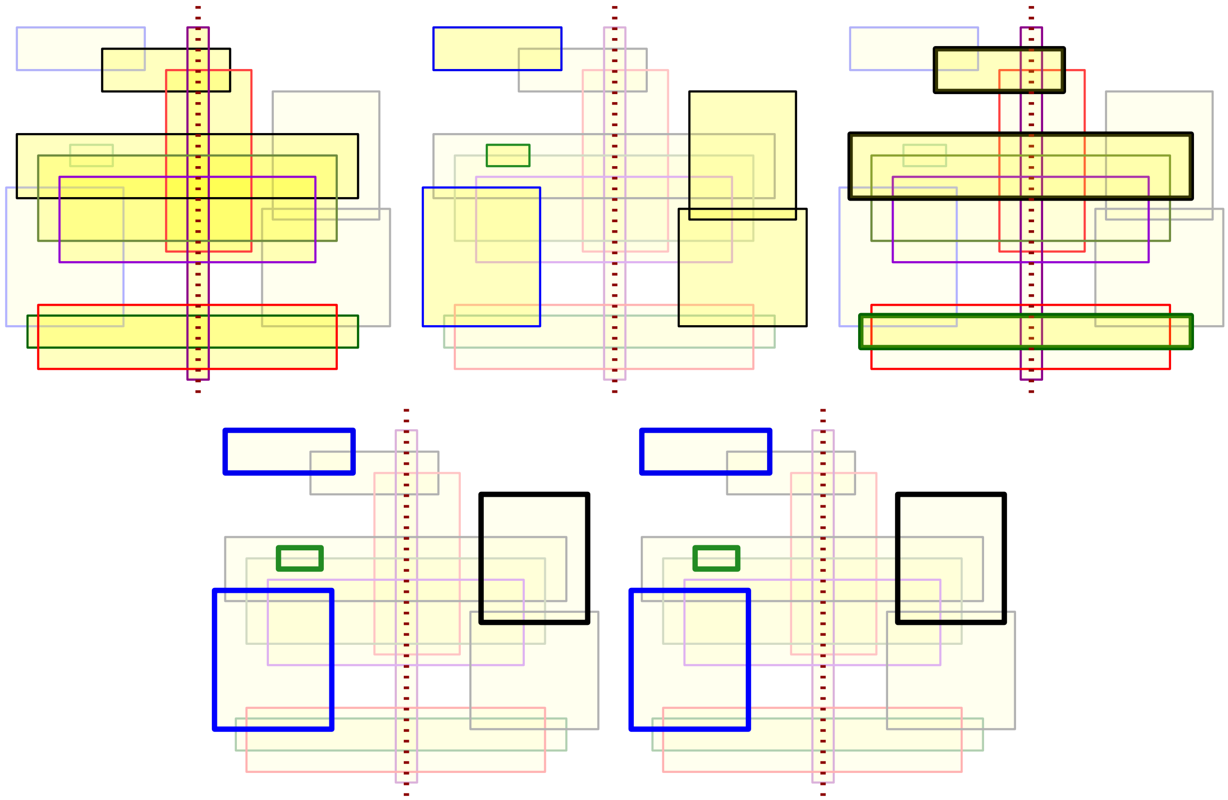
Input | Independent set of rectangles.

Assume: Open rectangles.

10.3.1 Independent set of rectangles

10.3.1.1 Algorithm: Divide & Conquer





10.3.2 Independent set of rectangles

10.3.2.1 Algorithm: Divide & Conquer

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

RectIndep($\mathcal{R}$):
    if $|\mathcal{R}| \leq 10$ **then**
        Solve by brute force
        return size of solution
    x_M: Median of right x-coordinate of rects in $\mathcal{R}$
    ℓ: Vertical line through x_M.
    $\mathcal{R}_M$: Rects of $\mathcal{R}$ intersecting ℓ
    $\mathcal{R}_L$, $\mathcal{R}_R$: Rectangles in $\mathcal{R}$ left/ right of ℓ.
    $S_L \Leftarrow$ **RectIndep**($\mathcal{R}_L$)
    $S_R \Leftarrow$ **RectIndep**($\mathcal{R}_R$)
    $S_M \Leftarrow$ compute opt solution for $\mathcal{R}_M$ (intervals!)
    return $\max(S_M, S_L + S_R)$

10.3.2.2 Analysis

(A) If $S_M \geq \mathrm{Opt}/(2\lg n)$... done.
(B) $\mathrm{Opt}_L + \mathrm{Opt}_R \geq (1 - 1/(2\lg n))\mathrm{Opt}$.
(C) By induction: $S_L \geq \mathrm{Opt}_L/(2\lg(n/2))$ and $S_R \geq \mathrm{Opt}_R/(2\lg(n/2))$.
(D) $S_L + S_R \geq \frac{(1-1/(2\lg n))\mathrm{Opt}}{2\lg(n/2)}$
(E) $\dfrac{(1 - 1/(2\lg n))}{2\lg(n/2)} = \dfrac{1}{2\lg n - 2} - \dfrac{1}{(2\lg n)(2\lg n - 2)}$

$$\geq \frac{2\lg n - 1}{(2\lg n)(2\lg n - 2)} \geq \frac{2\lg n - 2}{(2\lg n)(2\lg n - 2)} \geq \frac{1}{2\lg n}.$$

(F) Conclude: If $S_M \leq \mathrm{Opt}/(2\lg n)$, then $S_L + S_R \geq \mathrm{Opt}/(2\lg n)$.

(G) Algorithm is $2\lg n$ approximation.