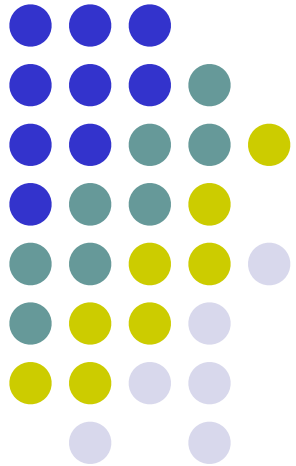
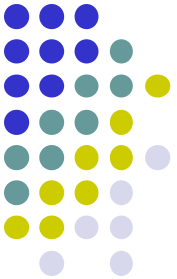
Web Algorithms

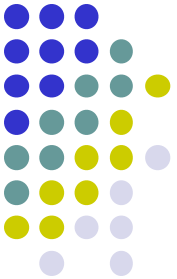
Eng. Fabio Persia, PhD





Algorithmic techniques: Greedy

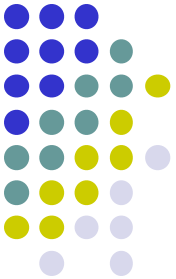
Characteristics



- The solution is determined in steps
- At every step the algorithm performs the choice that seems to be better in that step, without considering the possible consequences in the future steps

Lets see how this technique can be suitable applied for providing approximation algorithms for some optimization problems

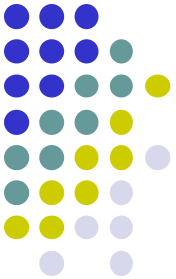
MAX 0-1 Knapsack



- INPUT: Finite set of objects O, an integral profit p_i and an integral volume a_i for every $o_i \in O$, a positive integer b
- SOLUTION: A subset of objects $Q \subseteq O$ such that $\sum_{o_i \in Q} a_i \leq b$
- MEASURE: Total profit of the chosen objects, that is $\sum_{o_i \in Q} p_i$

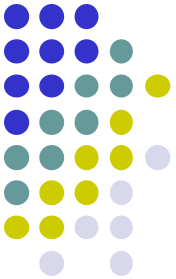
Without loss of generality in the sequel we will always assume that $a_i \leq b$ and $p_i > 0$ for every object $o_i \in O$

Greedy choice



- In the greedy choice
  - we cannot consider only the profit of the objects, as their volume could be too big
  - we cannot consider only the volumes, as their profit might be too low
- Idea: consider objects according to the profit per unit of volume, that is according to the ratio p_i/a_i
- The greedy algorithm selects the objects in non increasing order of profit per volume

Algorithm Greedy-Knapsack



Begin

$Q=\emptyset$; $v=0$. // v = *volume current subset of chosen objects*

Order the objects in non increasing order of profit per volume p_i/a_i and let $o_1,\ldots o_n$ the objects listed according to such an order.

For $i=1$ to n do

If $v+a_i \leq b$ then

Begin

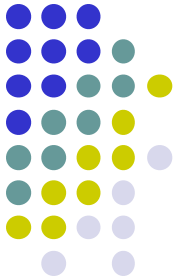
$Q=Q\cup\{o_i\}$.

$v=v+a_i$.

End

Return Q.

End



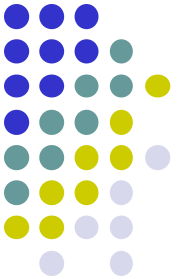
Theorem: For any given $r<1$, Greedy-Knapsack is not r-approximating.

Proof:

Given an integer $k=\lceil 1/r \rceil$, consider the following instance of MAX 0-1 Knapsack:

For any $n \geq 2$

- $b=k \cdot n$ is the knapsack volume
- $n-1$ objects with profit $p_i=1$ and volume $a_i=1$
- 1 object with profit $b-1$ and volume b



Returned solution: the set of the first *n-1* objects, that is *m=n-1*.

Optimal solution: the set containing only the n^{th} object, that is *m*=b-1=k·n-1*.

Thus

$$\frac{m}{m^*} = \frac{n-1}{k \cdot n - 1}$$

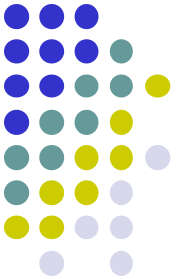
so that

Since 1/r>1

$$\frac{m}{m^*} = \frac{n-1}{k \cdot n - 1} \leq \frac{n-1}{\frac{n}{r} - 1} < \frac{n-1}{\frac{n}{r} - \frac{1}{r}} = \frac{n-1}{\frac{1}{r}(n-1)} = r.$$

□

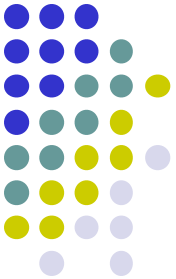
How can we improve the greedy algorithm?



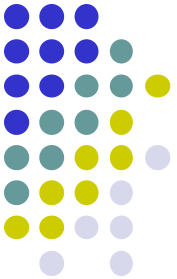
Observation:

Intuitively, Greedy-Knapsack does not return a good approximation, because it ignores the object having the maximum profit.

Modified-Greedy



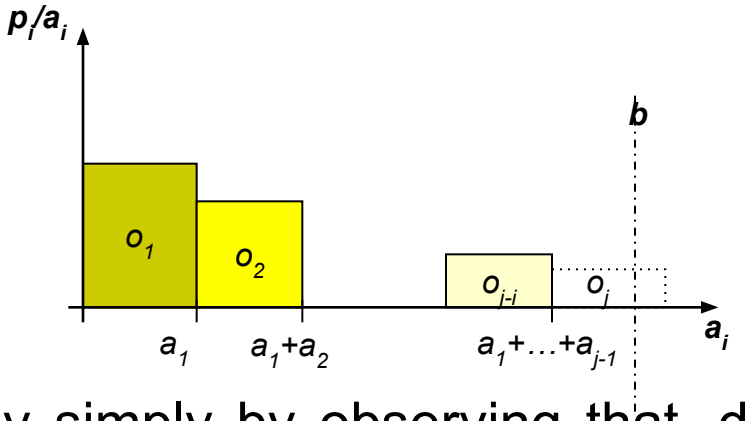
- Compute the greedy solution Q_{Gr} and let m_{Gr} be its measure
- Consider object o_{max} having maximum profit p_{max}
- If $m_{Gr} \geq p_{max}$ return Q_{Gr} otherwise return $Q=\{o_{max}\}$



Lemma 1: Let o_j be the first object that algorithm Greedy-Knapsack does not put into the knapsack and let

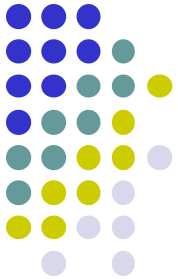
$$m_j = \sum_{i=1}^{j-1} p_i .$$

Then $m^* \leq m_j + p_j$.



Proof: $m^* \leq m_j + p_j$ comes directly simply by observing that, denoting with v the sum of the volumes of the first $j-1$ chosen objects, m_j+p_j is the value of the optimal solution of the instance in which the knapsack volume is $v+a_j>b$.

□



Lemma 2: $m^* \leq m_{GR} + p_{max}$

Proof: Direct consequence of the previous lemma by observing that $m_j \leq m_{GR}$ and $p_j \leq p_{max}$, and thus

$$m^* \leq m_j + p_j \leq m_{GR} + p_{max} .$$

□

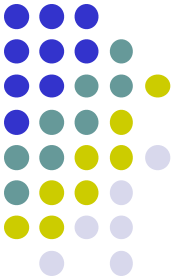
Intuition: the algorithm returns a solution of value $max(m_{GR}, p_{max})$, that is at least the half of $m_{GR}+p_{max}$, i.e., the half of an upper bound of m^*.

Theorem: Modified-Greedy is ½-approximating

Proof: $m_{Mo} \geq max(m_{Gr}, p_{max}) \geq (m_{Gr} + p_{max})/2 \geq m^*/2$

□

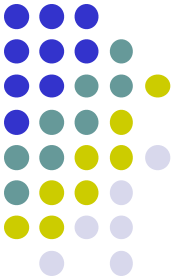
Min Multiprocessor Scheduling



- INPUT: set of n jobs P, number of processors h, running time t_j for each $p_j \in P$
- SOLUTION: A "schedule" for P, that is a function $f:P \rightarrow \{1,\ldots,h\}$
- MEASURE: makespan or completion time of f , that is

$$\max_{i \in [1\ldots h]} \sum_{p_j \in P | f(p_j)=i} t_j$$

Greedy algorithm of Graham

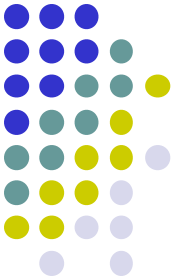


Greedy choice: at each step assign a job to the least loaded processor.

$T_i(j)$: completion time (sum running times jobs assigned to) processor i at the end of time j, that is once scheduled the first j jobs (in any order).

Algorithm:

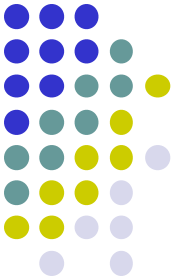
Begin
  Let $p_1,\ldots,p_n$ be the jobs listed in any order.
  For $j=1$ to n do
    Assign p_j to the processor i having minimum $T_i(j-1)$ // i.e. $f(p_j)=i$
  Return schedule f.
End



Observation: If jobs are scheduled according to the arrival time, the algorithm assigns each job without knowing future ones, that is it is ON-LINE.

Theorem: The Graham algorithm is *(2-1/h)-approximating*, where *h* is the number of processors.

Before proceeding with the proof, let us show some useful properties ….



Fact: Given $s \geq 0$ and h numbers $a_1, \ldots, a_h$ such that $a_1 + a_2 + \ldots + a_h = s$, there exists j, $1 \leq j \leq h$, such that

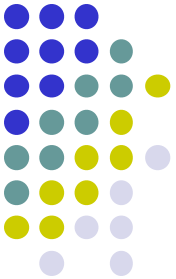
(otherwise : a contradiction)

$$a_j \geq \frac{s}{h} \qquad a_1 + a_2 + \ldots + a_h < h \cdot \frac{s}{h} = s$$

Analogously, there exists j', $1 \leq j' \leq h$, such that $a_{j'} \leq \frac{s}{h}$

In other words, one number is at most equal to the average and one greater or equal to the average

Therefore, $\min_j a_j \leq s/h$ and $\max_j a_j \geq s/h$



Proof (theorem): Let T the sum of all the job running times, i.e. $T = \sum_{j=1}^{n} t_j$.

Let $T_1^*, \ldots, T_h^*$ be the completion times of the h processors in an optimal solution. Since $T_1^* + \ldots + T_h^* = T$, by the previous fact there exists j such that $T_j^* \geq T/h$. Thus

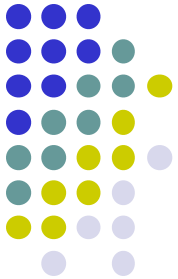
$$m^* \geq T_j^* \geq T/h.$$

Let k be one processor with maximum completion time in the schedule f returned by the algorithm, that is with maximum $T_k(n)$. Moreover let p_l be the last job assigned to processor k.

Since by the greedy choice p_l has been assigned to one of the least loaded processors at the beginning of step l, again by the previous fact we have

$$T_k(l-1) \leq (\Sigma_{j<l}\, t_j) / h \leq (T - t_l) / h,$$

since the sum of the running times of all the jobs assigned before p_l is at most $T - t_l$



Therefore

$$m = T_k(n) = T_k(l-1) + t_l \le \frac{T - t_l}{h} + t_l =$$

$$= \frac{T}{h} + \left(\frac{h-1}{h}\right) \cdot t_l \le m^* + \frac{h-1}{h} \cdot m^* = \left(2 - \frac{1}{h}\right) \cdot m^*$$

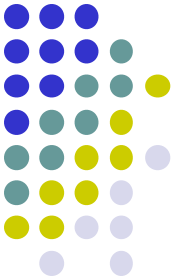
$T/h \le m^*$,
$t_l \le m^*$

And thus

$$\frac{m}{m^*} \le 2 - \frac{1}{h}.$$

□

Observations



- When *h* grows the approximation ratio *2-1/h* tends to *2*
- The analysis is tight, that is the following theorem holds

Theorem: The Graham algorithm is not *r*-approximating for *r<2-1/h*.

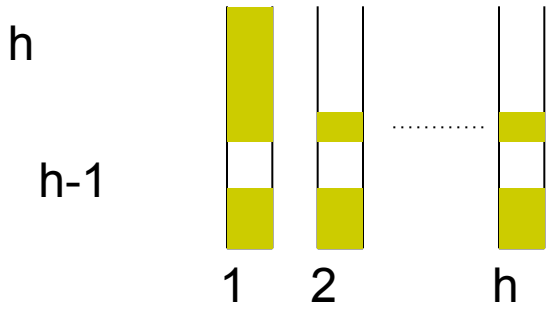
Proof: Consider the following instance:

- *h·(h-1)* jobs with running time *1*
- *1* job with running time *h*

Graham assigns the jobs in the following way:

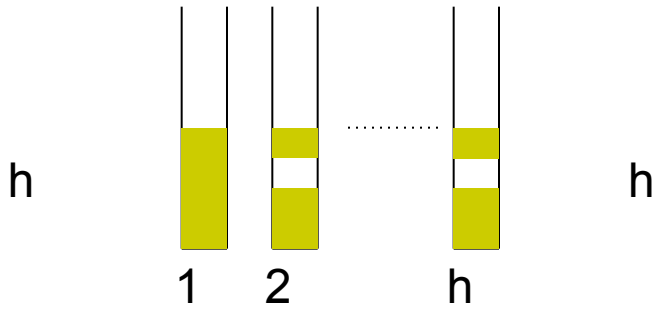
Thus:

$$m=2\cdot h-1.$$



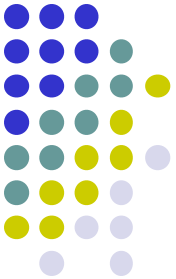
The optimal solution can be obtained assigning the long job to one processor and equally distributing the short ones among the remaining processors. Thus

$$m^*=h$$



In conclusion $$\frac{m}{m^*}=\frac{2\cdot h-1}{h}=2-\frac{1}{h}.$$

Improving the approximation ratio



Question: how can we improve the approximation ratio?

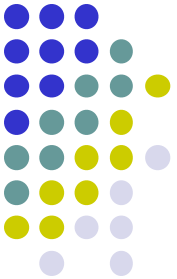
Let us fastly recall the basic steps of the proof of the Graham's approximation ratio.

We made use of the following lower bounds to the value of an optimal solution:

- $m^* \geq T/h$, as in any solution at least one processor must have completion time T/h (recall $T=\Sigma_j t_j$)
- $m^* \geq t_j$ for every job p_j, as in any solution one of the processors must run p_j

In order to upper bound the value of the returned solution, if k is one of the most loaded processors and p_l is the last job assigned to k, by the greedy choice

$$T_k(l-1) \leq (\Sigma_{j<l} t_j) / h \leq (T-t_l) / h.$$



Thus we can derive the following inequality:

$$m = T_k(n) = T_k(l-1) + t_l \le \frac{T - t_l}{h} + t_l =$$

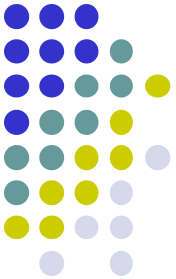
$$= \frac{T}{h} + \left(\frac{h-1}{h}\right) \cdot t_l \le m^* + \frac{h-1}{h} \cdot m^* = \left(2 - \frac{1}{h}\right) \cdot m^*$$

$T/h \le m^*$, $t_l \le m^*$

Idea for improvement: decrease t_l as much as possible and find a better ratio exploiting the inequalities:

$$m \le \frac{T}{h} + \left(\frac{h-1}{h}\right) \cdot t_l \le m^* + \left(\frac{h-1}{h}\right) \cdot t_l$$

Modifying the algorithm and/or improving the analysis you will see how to progressively upper bound t_l with $m^*/2$ (3/2-appr.), $m^*/3$ (4/3-appr.) and arbitrarily small, that is $\varepsilon \cdot m^*$ ($(1+\varepsilon)$-appr., i.e. a *PTAS*)



First improvement

- Assign jobs from the longest to shortest
- This allows to avoid the worst case of the Graham's algorithm, that is the fact that a long job arrives at the end, significantly unbalancing the loads of the processors

Algorithm Ordered-Greedy:

Begin

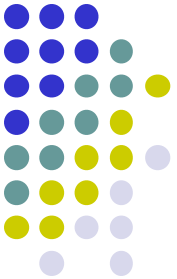
Let $p_1,\ldots,p_n$ the jobs listed in non increasing order of running time, that is such that $t_1 \geq t_2 \ldots \geq t_n$

For $j=1$ to n do

Assign p_j to a processor i with minimum $T_i(j-1)$.// i.e. $f(p_j)=i$

Return schedule f.

End



Let us see a simpler analysis leading to an approximation ratio about *3/2*

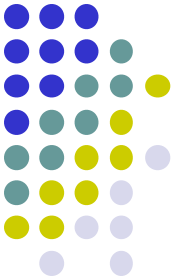
Lemma. If $n>h$, then $t_{h+1} \leq m^*/2$.
Proof. By the ordering of the jobs, the first $h+1$ ones have all running time $\geq t_{h+1}$.
But then $m^* \geq 2 \cdot t_{h+1}$, as in any schedule at least one of the h processors must receive at least two of the first $h+1$ jobs.
□

Theorem Ordered-Greedy is *(3/2 - 1/(2h))-approximating*.
Proof. Again let k be one of the most loaded processors (at the end).
If k has only one job then clearly the returned solution is optimal.
Otherwise consider the last job p_l assigned to k. Since p_l is not the first job assigned to k, $l \geq h+1$ and thus $t_l \leq t_{h+1} \leq m^*/2$, so that

□

$$m \leq \frac{T}{h} + \left(\frac{h-1}{h}\right) \cdot t_l \leq m^* + \left(\frac{h-1}{h}\right) \cdot \frac{m^*}{2} = \left(\frac{3}{2} - \frac{1}{2h}\right) m^*$$

Max Cut



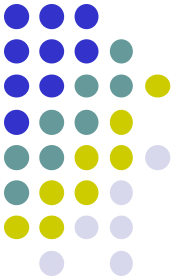
- INPUT: Graph $G=(V,E)$

- SOLUTION: Partition of V in two subsets V_1 and V_2, that is such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$

- MEASURE: Cardinality of the cut, that is number of edges with an endpoint in V_1 and the other endpoint in V_2 , that is

$$| \{ \{u,v\} \mid u \in V_1 \text{ and } v \in V_2 \} |$$

Algorithm Greedy-Max-Cut



Begin

$V_1 = V_2 = \emptyset$;

For i=1 to n do

$\Delta_i = \{ \{i,j\} \in E \mid j<i \}$; // *set of edges between i and nodes j<i*

$U_i = \{ j \mid \{i,j\} \in \Delta_i \}$; // *set of already inserted nodes adjacent to i* // *at the beginning of step i*

$\delta_i = |\Delta_i| = |U_i|$; $\delta_{1i} = |V_1 \cap U_i|$; $\delta_{2i} = |V_2 \cap U_i|$; // *clearly* $\delta_{1i} + \delta_{2i} = \delta_i$;

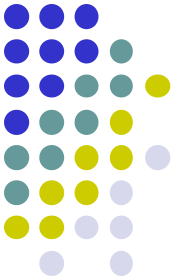
if $\delta_{1i} > \delta_{2i}$

then $V_2 = V_2 \cup \{ i \}$;

else $V_1 = V_1 \cup \{ i \}$;

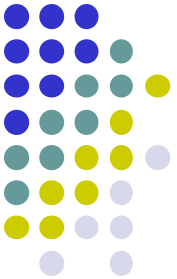
Return V_1, V_2;

End



- For the sake of simplicity let $V=\{1,2,\ldots,n\}$.
- The algorithm at each step inserts a new node in V_1 or in V_2
- Greedy choice: at step i node i is inserted so as to maximize the number of new edges in the cut, that is in V_1 if the number of edges it has to the already inserted nodes in V_2 is greater or equal to the number of edges it has towards the ones in V_1, otherwise in V_2

Example greedy choice

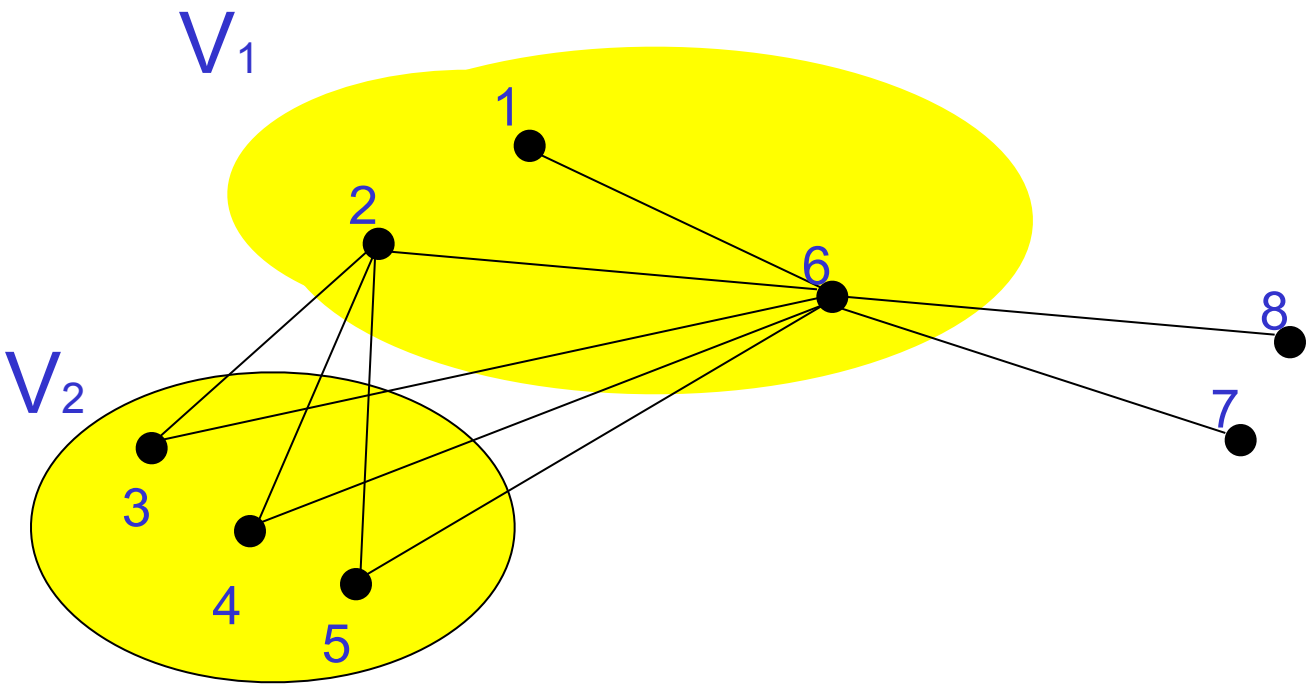


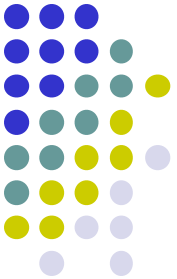
If at step *i=6*

$\Delta_6 = \{ \{1,6\}, \{2,6\}, \{3,6\}, \{4,6\}, \{5,6\} \}$ and thus

$U_6 = \{ 1, 2, 3, 4, 5 \}$, $\delta_6 = 5$, $\delta_{16} = 2$ *and* $\delta_{26} = 3$

since $\delta_{16} \leq \delta_{26}$ the algorithm inserts node *6* in V_1.





Theorem: The Greedy-MaxCut algorithm is ½-approximating

Proof.

Clearly, since that cut can only contain a subset of all the edges in E,

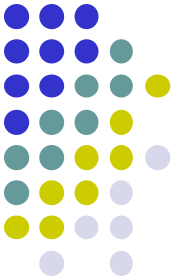
$$m^* \leq |E|$$

We now show that the measure *m* of the cut returned by the algorithm is at least the half of the total number of edges, that is

$$m \geq |E|/2$$

This clearly implies the claim, since

$$\frac{m}{m^*} \geq \frac{\frac{|E|}{2}}{|E|} = \frac{1}{2}$$



Since the sets Δ_i determined by the algorithm form a partition of E and by definition $\delta_i=|\Delta_i|$,

$$\sum_{i=1}^{n}\delta_i = \sum_{i=1}^{n}|\Delta_i| = |E|$$

Moreover, the number of edges added to the cut during step *i,* that is with an endpoint in V_1 and the other one in V_2 *(*after the execution of *i*-th iteration of the *for* statement*),* is

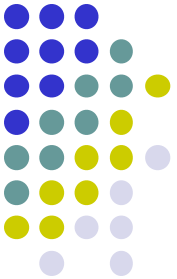
$$max(\delta_{1i},\delta_{2i}) \geq (\delta_{1i}+\delta_2)/2=\delta_i/2,$$

so that

$$m=\sum_{i=1}^{n}\max(\delta_{1i},\delta_{2i}) \geq \sum_{i=1}^{n}\frac{\delta_i}{2}=\frac{|E|}{2}$$

□

Conclusions on the greedy technique



All the algorithms seen so far have polynomial time complexity

Greedy algorithms have a good performance in practice because they can be implemented in a simple way

But as we have seen, performing the choice that seems better at each single step, without minding future consequences, in general does not allow them to find the optimal solution