Situação Ideal

- Desejamos algoritmos que:
  - encontrem a solução ótima
  - em tempo polinomial
  - para qualquer instância do problema que estamos trabalhando

Situação enquanto P ≠ NP: escolha **2 opções** !

Algoritmos Aproximados

- Algoritmo com **garantia** de produzir soluções sempre em um fator da solução ótima.
- Uma heurística ***pode*** ser um algoritmo aproximado caso exista tenha sido provado que para qualquer caso possível um padrão de qualidade se mantém.
- Definição:
  - Roda em tempo polinomial
  - Produz solução dentro de um fator α da solução ótima
    - minimização: $\alpha > 1$
    - maximização: $\alpha < 1$

Algoritmos Aproximados - Motivação

- problemas NP-Difíceis precisam de soluções;
- forma matematicamente rigorosa de se estudar heurísticas;
- entender o quão difíceis são os problemas.

Questão

- Como avaliar a distância da solução ótima se ***calcular*** a solução ótima já é altamente custoso ?

Uso de limites válidos, mais facilmente calculados.

Limites

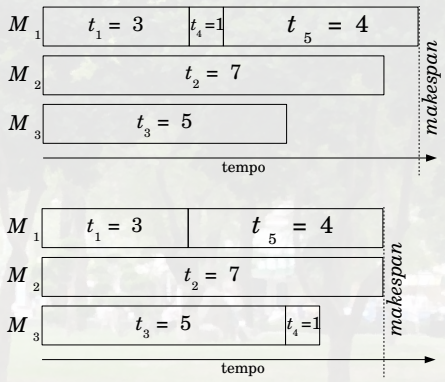
- Para um problema de **Minimização**(/**Maximização**) difícil, calcular um **Limite Inferior**(/**Superior**) pode ser fácil.
- Uma vez mostrado que nosso algoritmo sempre produz uma solução no máximo α vezes o custo do limite em questão fica provado que isso vale para a solução ótima.

Ponto fundamental: **Qualidade do limite** e **esforço computacional** gasto para obtenção do mesmo

Exemplo: Escalonamento de Tarefas

- Balanceamento de carga em máquinas paralelas, exemplo:
  - n = 5 tarefas, tempos de processamento
    - t_1=3, t_2=7, t_3=5, t_4=1, t_5=4
  - m = 3 máquinas idênticas
- Objetivo: alocar tarefas à máquinas minimizando o tempo final de processamento (*makespan*)

Soluções possíveis



Algoritmo Guloso

```
procedure GreedyScheduling( t_1, ..., t_n, m )
  T_i ← 0, A(i) ← ∅  ∀i ∈ 1 ..., m;
  for j ← 1 to n do
  begin
    Encontre k : T_k = min_{1,...,m}  T_i;
      A(k) ← A(k) ∪ {j};
    T_k ← T_k + t_j;
  end
end
```


Limites Inferiores

- Relaxação da integralidade

$$\mathcal{OPT} \geq \frac{\sum_{j=1}^{n} t_j}{m}$$

- Tarefa grande

$$\mathcal{OPT} \geq \max_{j=1}^{n} t_j$$

- Limite inferior

$$\mathcal{LB} = \max(\max_{j=1}^{n} t_j, \frac{\sum_{j=1}^{n} t_j}{m})$$

GreedyScheduling é 2-Aproximado

- Prova:
  - Seja:

M_{i^*} a máquina determinante do *makespan*;

j^* a última tarefa alocada nessa máquina

T'_{i^*} a carga que a máquina M_{i^*} apresenta antes de j^* ser inserida

(note que no momento da inserção era a menor carga)

$$m \cdot T'_{i^*} \leq \sum_{i=1}^{m} T'_i = \sum_{j=1}^{j^*} t_j \leq \sum_{j=1}^{n} t_j \leq m \cdot \mathcal{LB}$$

GreedyScheduling é 2-Aproximado

- Prova (cont):

M_{i^*} a máquina determinante do makespan;

j^* a última tarefa alocada nessa máquina

T'_{i^*} carga antes da alocação de j^*

$$m \cdot T'_{i^*} \leq \sum_{i=1}^{m} T'_i = \sum_{j=1}^{j^*} t_j \leq \sum_{j=1}^{n} t_j \leq m \cdot \mathcal{LB}$$

$$\begin{aligned} T_{i^*} &= t_{j^*} + T'_{i^*} \\ &\leq t_{j^*} + \mathcal{LB} \\ &\leq \max_{j=1}^{n} t_j + \mathcal{LB} \\ &\leq 2 \cdot \mathcal{LB} \\ &\leq 2 \cdot \mathcal{OPT} \end{aligned}$$

Greedy Scheduling errando, ex:

- quatro tarefas (n=4)
  - t_1=2, t_2=1, t_3=15, t_4=2
- duas máquinas (m=2)

Para qual ordenação das Greedy Scheduing produziria um resultado melhor ?

Greedy Scheduling Melhorado

```
procedure GreedySchedulingI(t_1,...,t_n,m)
  T_i ← 0, A(i) ← ∅  ∀i ∈ 1...,m;
  for j ← 1 to n do
  begin
    t' ← a j-sima maior tarefa;
    Encontre k : T_k = min_{1,...,m} T_i;
    A(k) ← A(k) ∪ {t'};
    T_k ← T_k + t_t';
  end
end
```


Provando a superioridade

- se $n \leq m$: resultado ótimo;
- para $n > m$, novo limite: $\mathcal{OPT} \geq t_m + t_{m+1}$
- partindo dos resultados anteriores:

$$\begin{aligned} T_{i^*} &\leq t_{j^*} + \mathcal{LB} \\ &\leq 2 \cdot \mathcal{LB} \\ &\leq 2 \cdot \mathcal{OPT} \end{aligned}$$

- e considerando que:

$$t_{j^*} \leq \frac{t_m + t_{m+1}}{2} \leq \frac{\mathcal{OPT}}{2}$$

- GreedySchedulingI é $\frac{3}{2}$ aproximado

Problema da Mochila 0/1

- Entrada:
  - capacidade c
  - n itens
    - lucros l_i
    - pesos p_i
- Saída: subconjunto de itens que respeita a capacidade e maximiza o lucro

Problema da Mochila 0/1 – Alg. Guloso

```
procedure GreedyKnapsack1(c, n, (l_1, p_1), ..., (l_n, p_n))
  Ordene os itens em ordem decrescente de l_i/p_i;
  if sum_{i=1}^{n} p_i <= c then
    return sol_g = sum_{i=1}^{n} l_i;
  else
  begin
    Encontre o maior k in {1, ..., n} tal que sum_{i=1}^{k} <= c;
    return sol_g = sum_{i=1}^{k} l_i;
  end
end
```


Falhando Miseravelmente

- Situação:
  - Considere uma mochila com **grande** capacidade, digamos B
  - Considere a existência de um item i com lucro l_i=2 e peso p_i=1, juntamente com um item de lucro B e peso B

Corrigindo o Algoritmo Guloso

```
procedure GreedyKnapsackI(c, n, (l_1, p_1), ..., (l_n, p_n))
  Ordene os itens em ordem decrescente de l_i/p_i;
  if sum_{i=1}^{n} p_i <= c then
    return sol_g = sum_{i=1}^{n} l_i;
  else
  begin
    Encontre o maior k in {1, ..., n} tal que sum_{i=1}^{k} <= c;
    sol_1 = sum_{i=1}^{k} l_i;
    sol_2 = l_{k+1};
    return sol_g = max{sol_1, sol_2}
  end
end
```


GreedyKnapsackI é 2-Aproximado

- Prova:
  - Considere novamente $k \in \{1,\ldots,n\}$ sendo o maior valor tal que $\sum_{i=1}^{k} p_i \leq c$
  - Os seguintes limites para a solução ótima são válidos

$$\sum_{i=1}^{k} l_i \leq \mathcal{OPT} \leq \sum_{i=1}^{k+1} l_i$$

  - O lado direito é um limite superior cuja distância do limite inferior é menor ou igual ao lado esquerdo (*se valesse a pena inserir o item k+1 o mesmo seria inserido*)