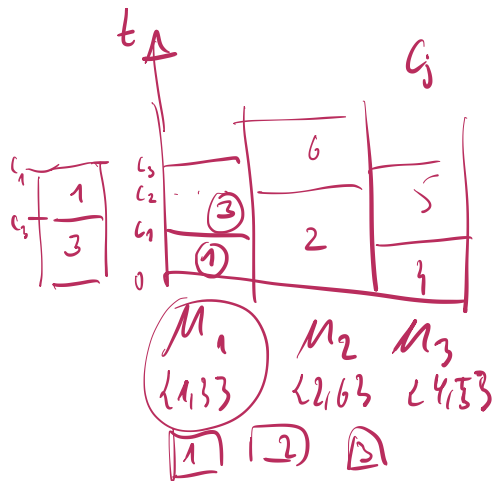


1. Introdução



- [1] Div. tarefas o máximo
- [2] Sequenciamento das tarefas em cada máquina. $\rightarrow n \log n$

Por fim, transformações podem ser úteis caso podemos resolver sub-problemas restritos do problema eficientemente.

Exemplo 1.7 (Sequenciamento em máquinas paralelas) $\in NP-C$

Uma solução direta do problema de sequenciamento em máquinas paralelas não relacionadas $R \parallel \sum w_j C_j$ é uma atribuição das tarefas às máquinas, junto com a ordem das tarefas em cada máquina.

Teorema 1.3

A solução ótima de $1 \parallel \sum w_j C_j$ é uma sequência em ordem de tempo de processamento ponderado não-decrescente $p_1/w_1 \leq \dots \leq p_n/w_n$.

Prova. Supõe uma sequência ótima com $p_i/w_i > p_{i+1}/w_{i+1}$. A contribuição das duas tarefas à função objetivo é $w = w_i C_i + w_{i+1} C_{i+1}$. Trocando as duas tarefas a contribuição das restantes tarefas não muda, e a contribuição das duas tarefas é

$$w_{i+1}(C_{i+1} - p_i) + w_i(C_i + p_{i+1}) = w + w_i p_{i+1} - w_{i+1} p_i.$$

Logo a função objetivo muda por $\Delta = w_i p_{i+1} - w_{i+1} p_i$, mas pela hipótese $\Delta < 0$. ■

Logo a ordem ótima de uma máquina pode ser computada em tempo $O(n \log n)$, e uma representação reduzida mantém somente a distribuição das tarefas às máquinas. ◇

As diferentes representações compactas podem ser combinadas.

Exemplo 1.8 (Simple assembly line balancing)

No "simple assembly line balancing problem" do tipo 2 temos que atribuir n tarefas, restritas por precedências, à m de estações de trabalho. Cada tarefa possui um tempo de execução t_i , e o tempo de estação é o tempo total das tarefas atribuídas a uma estação. O objetivo é minimizar o maior tempo de estação.

Uma representação direta é uma atribuição de tarefas a estações, mas muitas atribuições são inválidas por não satisfazer as precedências entre as tarefas. Uma representação mais compacta atribui chaves aleatórias às tarefas. Com isso, uma ordem global das tarefas é definida: elas são ordenadas topologicamente, usando as chaves aleatórias como critério de desempate, caso duas tarefas concorram para a próxima posição. Por fim, para uma dada ordem de tarefas, a solução ótima do problema pode ser obtida via programação dinâmica. Seja $C(i, k)$ o menor tempo de ciclo para tarefas $i, \dots, n$ em k máquinas, a solução ótima é $C(1, m)$ e C satisfaz

$$C(i, k) = \begin{cases} \min_{i \leq j \leq n} \max\{\sum_{i \leq j' \leq j} t_{j'}, C(j+1, k+1)\} & \text{para } i \leq n, k > 0 \\ 0 & \text{para } i > n \\ \infty & \text{para } i \leq n \text{ e } k = 0 \end{cases},$$

e logo a solução ótima pode ser obtida em tempo e espaço $O(nm)$ (pré-calculando as somas parciais). ◇

Essa observação é o motivo para o

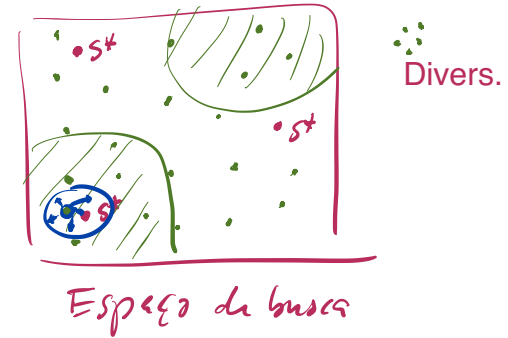
Princípio de projeto 1.3 (Subproblemas)

Identifica os subproblemas mais difíceis que podem ser resolvidos em tempo polinomial e considera uma representação que contém somente a informação necessária para definir os subproblemas.

1.3. Estratégia de busca: Diversificação e intensificação

Uma heurística tem que balancear duas estratégias antagonistas: a **diversificação** da busca e a **intensificação** de busca. A diversificação da busca (ingl. **diversification** or **exploration**) procura garantir uma boa cobertura do espaço de busca, evitando que as soluções analisadas fiquem confinadas a uma pequena região do espaço total. A diversificação ideal é um algoritmo que repetidamente gera soluções aleatórias. Em contraste a **intensificação** (ingl. **intensification** or **exploitation**) procura melhorar a solução atual o mais possível. Um exemplo de uma intensificação seria analisar todas soluções dentro uma certa distância da solução atual.

O tema de intensificação e diversificação se encontra na discussão da heurísticas individuais na seções 2 a 4; um procedimento genérico de intensificação e diversificação é apresentado na seção 4.9.



1.4. Notas

Mais informações sobre os teoremas NFL se encontram no artigo original de Wolpert e Macready (1997) e em Burke e Kendall (2005, cap. 11) e Rothlauf (2011, cap. 3.4.4). Para uma crítica ver p.ex. Hutter (2010). Talbi (2009, cap. 1.4.1) discute outras representações de soluções.

S(x): soluções instância x

$$3! = 6$$

$$n!$$

Ex. Vizinho é obtido por trocar 2 elementos adjacentes.

$$N(123) = \{213, 132, 321\}$$

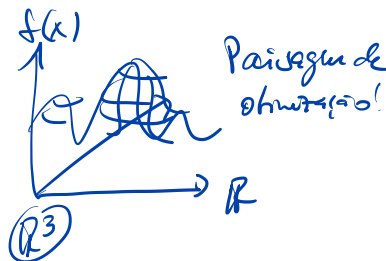
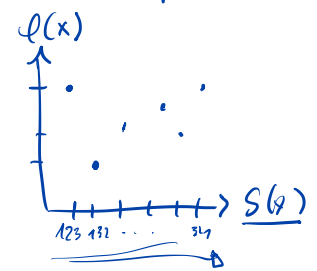
$$N(213) \ni 122$$

$$\pi: [n] \rightarrow [n]$$

$$\pi = (123); \pi(1)=1, \pi(2)=2, \pi(3)=3$$

$$\pi(i)=i$$

$$f(\pi) = \# \text{ pontos fixos}$$



2. Busca por modificação de soluções

2.1. Vizinhanças

Uma busca local procura melhorar uma solução de uma instância de um problema aplicando uma pequena modificação, chamada *movimento*. O conjunto de soluções que resultam de uma pequena modificação formam os *vizinhos* da solução.

Definição 2.1 (Vizinhança)

Uma *vizinhança* de uma instância x de um problema de otimização Π é uma função $N: S(x) \rightarrow 2^{S(x)}$. Para uma solução s , os elementos $N(s)$ são os *vizinhos* de s . Os *vizinhos melhores* de s são $B(s) = \{s' \in N(s) \mid \varphi(s') < \varphi(s)\}$. Uma vizinhança é *simétrica*, caso para $s' \in N(s)$ temos $s \in N(s')$.

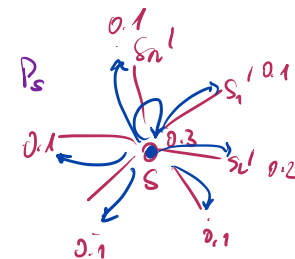
Para uma dada vizinhança um *mínimo local* é uma solução s , tal que $\varphi(s) \leq \varphi(s')$ para $s' \in N(s)$ e um *máximo local* caso $\varphi(s) \geq \varphi(s')$ para $s' \in N(s)$. Caso uma solução é estritamente menor ou maior que os seus vizinhos, o ótimo local é *estrito*. Uma vizinhança é *exata*, caso cada ótimo local também é um ótimo global.

Definição 2.2 (Grafo de vizinhança)

O *grafo de vizinhança* $G = (V, E)$ para uma instância x de um problema de otimização Π com vizinhança N possui vértices $V = \{y \mid (x, y) \in P\}$ e arcos (s, s') para $s, s' \in S(x)$, $s' \in N(s)$. Para uma vizinhança simétrica, o grafo de vizinhança é efetivamente não-direcionado. Uma solução s' é alcançável a partir da solução s , caso existe um caminho de s para s' em G . Caso todo vértice é alcançável a partir de qualquer outro, G é *conectado*. Neste caso o *diâmetro* de G é o comprimento do maior caminho mais curto entre dois vértices em G . O grafo G é *fracamente otimamente conectada* caso a partir de cada solução s uma solução ótima é alcançável.

Uma vizinhança é suficiente para definir uma busca local genérica. Ela seleciona um vizinho de acordo com uma distribuição P_s sobre a *vizinhança fechada* $\hat{N}(s) = \{s\} \cup N(s)$. Para uma distribuição P_s sobre $N(s)$, a extensão padrão para a vizinhança fechada é definida por

$$\hat{P}_s(s') = \begin{cases} 1 - \sum_{s' \in N(s)} P_s(s'), & \text{para } s' = s, \\ P_s(s'), & \text{caso contrário.} \end{cases}$$



Algoritmo 2.1 (LocalSearch)

Entrada Solução inicial s , vizinhança N , distribuição P_s .

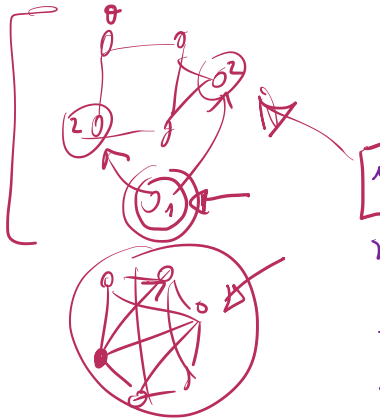
Saída Uma solução com valor no máximo $\varphi(s)$.

2. Busca por modificação de soluções

melhor sol. local *sol. inicial* *min!*

```

LocalSearch(s) =
  s* := s
  repeat
    seleciona s' ∈ N(s) de acordo com Ps
    s := s'
  until critério de parada satisfeito
  return s*
end
  
```



A complexidade de uma busca local depende da complexidade da seleção e do número de iterações. A complexidade da seleção muitas vezes é proporcional ao tamanho da vizinhança $|N(s)|$.

Duas estratégias básicas para uma busca local são

- 1) **Caminhada aleatória** (ingl. random walk) Para $N(s) \neq \emptyset$, define $P_s(s) = 1/|N(s)|$.
- 2) **Amostragem aleatória** (ingl. random picking) Uma caminhada aleatória com $N(s) = S(x)$ para todo $s \in S(x)$.
- 3) **Melhor vizinho** Para $B(s) \neq \emptyset$ define $B^*(s) = \{s' \in B(s) \mid \varphi(s') = \min_{s'' \in B(s)} \varphi(s'')\}$ e $P_s(s') = 1/|B^*(s)|$ para $s' \in B^*(s)$. Essa estratégia tipicamente não consegue sair de mínimos locais e tem que ser modificado por uma das técnicas discutidas em 2.3, mas supera plateaus.

Exemplo 2.1 (Polítopos e o método Simplex)

O método Simplex define uma vizinhança entre os vértices do polítopo de um programa linear: cada par variável entrante e saínte admissível define um vizinho. Essa vizinhança é simétrica, conectada, fracamente otimamente conectada e exata. Logo uma busca local com a estratégia "melhor vizinho" resolve o problema da programação linear. $\diamond$

Exemplo 2.2 (k-exchange para o PCV)

Uma vizinhança para o PCV é k-exchange Croes (1958): os vizinhos de um ciclo são obtidos removendo k arcos, e conectando os k caminhos resultantes de outra forma. Para qualquer k fixo, essa vizinhança é simétrica, conectada, fracamente otimamente conectada, mas inexacta (por quê?). O tamanho da vizinhança é $O = \frac{n!}{(n-k)!k!2^k} = O(n^k)$ para n cidades e k fixo. $\diamond$

Exemplo 2.3 (k-SAT)

O problema k-SAT é decidir se existe uma atribuição $x \in \{0, 1\}^n$ que satisfaz uma fórmula $\varphi(x)$ da lógica proposicional em forma normal conjuntiva com k literais por cláusula.

Seja $|x - y|_1 = \sum_{i \in [n]} [x_i \neq y_i]$ a distância Hamming entre dois vetores $x, y \in \{0, 1\}^n$. Uma vizinhança conhecida para SAT é k-flip: os vizinhos de uma solução são todas soluções de distância Hamming

MAX-2-SAT ∈ P

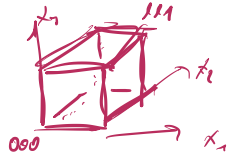
MAX-3-SAT ∈ NP-C

$x_1, x_2, \dots, x_n$

0/1 0/1 ... 0/1

0 0 0 0 0 0

0 0 1 0 0 0



1-flip ≤ 2-flip ≤ 3-flip ... ≤ 1-flip

0 → 1 → 0

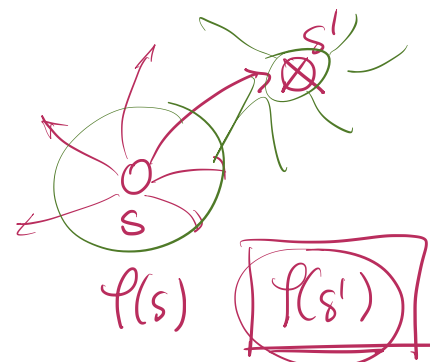
Figura 2.1.: Um movimento na vizinhança 3-exchange para o PCV.

k. A vizinhança é simétrica, fracamente otimamente conectada para $k = 1$, mas inexata. O tamanho da vizinhança é $O(n^k)$.

Observação 2.1 (Cálculo eficiente da função objetivo)

Frequentemente é mais eficiente avaliar a diferença $\Delta(s, s') = \varphi(s') - \varphi(s)$ para determinar o valor da função objetivo de um vizinho. No exemplo 2.2 avaliar $\varphi(s)$ custa $O(n)$, mas avaliar $\Delta(s, s')$ custa $O(1)$. Logo, determinar o melhor vizinho na vizinhança 2-exchange, por exemplo, custa $O(n^3)$ na abordagem ingênua, mas é possível em $O(n^2)$ avaliando as diferenças.

Em alguns casos a avaliação da diferença das diferenças é ainda mais eficiente. Um exemplo é a programação quadrática binária com função objetivo



Eficiência:

- 1) Geração eficiente de vizinhos
- 2) Cálculo eficiente do valor da f.o. de um vizinho.

$$w_{ij} = x_i x_j$$

$$\varphi(x) = x^t Q x = \sum_{(i,j) \in [n]} q_{ij} x_i x_j \quad O(n^2)$$

com $x_i \in \{0, 1\}$ e coeficientes simétricos ($Q = Q^t$). Avaliar $\varphi(s)$ custa $\Theta(n^2)$, avaliar a diferença na vizinhança 1-flip que troca $x'_k = 1 - x_k$ para um k fixo, obtemos $x' = x - (1 - 2x_k)e_k$ e logo

$$\begin{aligned} \Delta_k(x', x) &= \sum_{i,j \in [n]} q_{ij} (x'_i x'_j - x_i x_j) \\ &= \sum_{i \in [n] \setminus \{k\}} q_{ik} x_i (x'_k - x_k) + \sum_{j \in [n] \setminus \{k\}} q_{kj} (x'_k - x_k) x_j + q_{kk} (x'_k - x_k)^2 \\ &= (1 - 2x_k) \left(q_{kk} + 2 \sum_{i \in [n] \setminus \{k\}} q_{ik} x_i \right) \end{aligned} \quad O(n^2)$$

custa somente $O(n)$.

Atualizando um bit l por $x'_l = 1 - x_l$ obtemos novas diferenças

$$\Delta'_k = \begin{cases} -\Delta_k & \text{caso } l = k \\ \Delta_k + 2q_{lk}(1 - 2x_k)(1 - 2x_l) & \text{caso contrário.} \end{cases} \quad (2.1) \quad O(n)$$

Dado os valores Δ_k podemos encontrar o melhor vizinho em tempo $O(n)$. Passando para o melhor vizinho, podemos atualizar todos valores Δ_k em tempo $O(n)$ usando (2.1). Logo, o custo de encontrar o melhor vizinho é $\Theta(n^3)$ avaliando soluções completas, somente $\Theta(n^2)$ calculando as diferenças, e somente $O(n)$ atualizando diferenças.

Princípio de projeto 2.1 (Vizinhanças)

Procura o método mais eficiente de avaliar os vizinhos de uma solução e encontrar um dos melhores vizinhos.

2.1.1. Vizinhanças reduzidas

Uma técnica comum para melhorar o desempenho de buscas locais é reduzir a vizinhança heurísticamente, excluindo vizinhos com características que com baixa probabilidade se encontram em soluções de