
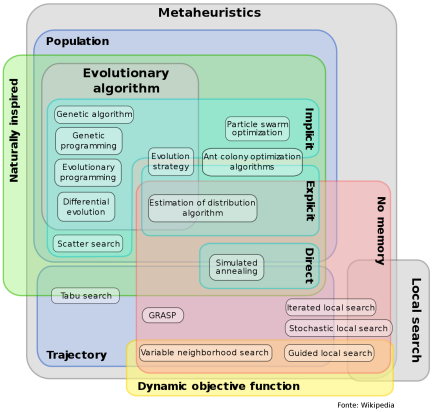


# Notas de aula

Marcus Ritt
mrpritt@inf.ufrgs.br

13 de Março de 2013

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Departamento de Informática Teórica

# Conteúdo

# 1. Introdução

Um *problema de busca* é uma relação binária $\mathcal{P} \subseteq I \times S$ com instâncias $x \in I$ e soluções $y \in S$. O par $(x, y) \in \mathcal{P}$ caso $y$ é uma solução para $x$.

**Definição 1.1**
A classe de complexidade FNP contém os problemas de busca com relações $\mathcal{P}$ polinomialmente limitadas (ver definição 1.3) tal que $(x, y) \in \mathcal{P}$ pode ser decidido em tempo polinomial.
A classe de complexidade FP contém os problemas em FNP para quais existe um algoritmo polinomial $A$ com

$$A(x) = \begin{cases} y & \text{para um } y \text{ tal que } (x, y) \in \mathcal{P} \\ \text{“insolúvel”} & \text{caso não existe } y \text{ tal que } (x, y) \in \mathcal{P} \end{cases}.$$

**Teorema 1.1**
FP=FNP se e somente se P=NP.

**Prova.** Ver por exemplo Papadimitriou [5, ch. 10.3]. ■

**Definição 1.2**
Um *problema de otimização* $\Pi = (\mathcal{P}, \varphi, \text{opt})$ é uma relação binária $\mathcal{P} \subseteq I \times S$ com instâncias $x \in I$ e soluções $y \in S$, junto com

- uma função de otimização (função de objetivo) $\varphi : \mathcal{P} \to \mathbb{N}$ (ou $\mathbb{Q}$).
- um objetivo: Encontrar mínimo ou máximo

$$\text{OPT}(x) = \text{opt}\{\varphi(x, y) \mid (x, y) \in \mathcal{P}\}$$

  junto com uma solução $y^*$ tal que $f(x, y^*) = \text{OPT}(x)$.

O par $(x, y) \in \mathcal{P}$ caso $y$ é uma solução para $x$.

Uma instância $x$ de um problema de otimização possui soluções $S(x) = \{y \mid (x, y) \in \mathcal{P}\}$.

**Convenção 1.1**
Escrevemos um problema de otimização na forma

| Nome |
| --- |
| **Instância** $x$ |
| **Solução** $y$ |
| **Objetivo** Minimiza ou maximiza $\varphi(x, y)$. |

Com um dado problema de otimização correspondem três problemas:

- Construção: Dado $x$, encontra a solução ótima $y^*$ e seu valor $\mathrm{OPT}(x)$.
- Avaliação: Dado $x$, encontra valor ótimo $\mathrm{OPT}(x)$.
- Decisão: Dado $x$ e $k$, decide se $\mathrm{OPT}(x) \geq k$ (maximização) ou $\mathrm{OPT}(x) \leq k$ (minimização).

**Definição 1.3**
Uma relação binária $R$ é *polinomialmente limitada* se

$$\exists p \in \mathsf{poly} : \forall (x, y) \in R : |y| \leq p(|x|).$$

**Definição 1.4 (Classes de complexidade)**
A classe PO consiste dos problemas de otimização tal que existe um algoritmo polinomial $A$ com $\varphi(x, A(x)) = \mathrm{OPT}(x)$ para $x \in I$.
A classe NPO consiste dos problemas de otimização tal que

(i) As instâncias $x \in I$ são reconhecíveis em tempo polinomial.

(ii) A relação $\mathcal{P}$ é polinomialmente limitada.

(iii) Para $y$ arbitrário, polinomialmente limitado: $(x, y) \in \mathcal{P}$ é decidível em tempo polinomial.

(iv) $\varphi$ é computável em tempo polinomial.

## 1.1. Não tem almoço de graça

> "Sire in eight words I will reveal to you all the wisdom that I have distilled through all these years from all the writings of all the economists who once practiced their science in your kingdom. Here is my text: 'There ain't no such thing as free lunch' " [9]

A frase "there ain't no such thing as free lunch" (TANSTAFEL) expressa que uma vantagem (p.ex. o almoço de graça em bares dos EUA no século 19) tipicamente é pago de outra forma (p.ex. comida salgada e bebidas caras). Para problemas de busca e de otimização, Wolpert e Macready [10] provaram teoremas que mostram que uma busca universal não pode ter uma vantagem em todos problemas de otimização.
Para um problema de otimização supõe que $\varphi : P \to \Phi$ é restrito para um conjunto finito $\Phi$, e seja $\mathcal{F} = \Phi^{S(x)}$ espaço de todas funções objetivos para uma instância do problema. Um algoritmo de otimização avalia pares de soluções com o seu valor $(s, v) \in S(x) \times \Phi$. Seja $D = \cup_{m \geq 0}(S(x) \times \Phi)^m$ o conjunto de todas sequencias de pares. Um algoritmo de otimização que não repete avaliações pode ser modelado como uma função $a : d \in D \to \{s \mid s \neq s_i, \text{para } d_i = (s_i, v_i), i \in [|d|]\}$. A avaliação de um algoritmo de otimização é através uma função $\Phi(d)$. Ela pode, por exemplo, atribuir a $d$ o valor mínimo encontrado durante a busca.

**Teorema 1.2 (Wolpert e Macready [10])**
Para algoritmos $a, a'$, um número de passos $m$ e uma sequencia de valores $v \in \Phi^m$

$$\sum_{f \in F} P[v \mid f, m, a] = \sum_{f \in F} P[v \mid f, m, a'].$$

O teorema mostra que uma busca genérica não vai ser melhor que uma busca aleatória em média sobre todas funções objetivos . Porém, uma grande fração das funções possíveis não ocorrem na prática (uma função aleatória é incompressível, i.e. podemos especificá-la somente por tabulação). Além disso, algoritmos de busca frequentemente aproveitam a estrutura do problema em questão.

## 1.2. Representação de soluções

A representação de soluções influencia as operações aplicáveis e a sua complexidade. Por isso a escolha de uma representação é importante para o desempenho de uma heurística. A representação também define o tamanho do espaço de busca, e uma representação compacta (e.g. 8 coordenadas versus permutações no problema das 8-rainhas) é preferível. Para problemas restritas uma representação implícita que é transformado para uma representação direta por um algoritmo pode ser vantajoso.
Para uma discussão abstrata usaremos frequentemente duas representações elementares. Na *representação por conjuntos* uma solução é um conjunto $S \subseteq U$ de um universo $U$. Os conjuntos válidos são dados por uma coleção

$\mathcal{V}$ de subconjuntos de $U$. Na *representação por variáveis* uma instância é um subconjunto $I \subseteq U$, e uma solução é uma atribuição de valores de um universo $V$ aos elementos em $I$.

**Exemplo 1.1 (Representação do PCV)**
Uma representação por conjuntos para o PCV sobre um grafo $G = (V, A)$ é o universo de arestas $U = A$, com $\mathcal{V}$ todos subconjuntos que formam ciclos. Um exemplo de uma representação por variáveis é $U = V = \mathbb{N}$ com instâncias $I = [n]$ para $n$ cidades. Uma atribuição válida define a posição de $i \in I$ na rota. ◊

### 1.2.1. Transformações entre representações

Um transformador recebe uma representação de uma solução e transforma ela numa representação diferente. Um algoritmo construtivo randomizado (ver capítulo 4) pode ser visto como um algoritmo que transforma uma sequencia de números aleatórios em uma solução explicita. Ambas são representações válidas da mesma solução. Essa ideia é aplicada também em algoritmos genéticos, onde a representação fonte se chama *fenótipo* e a representação destino *genótipo*. A ideia de representar uma solução por uma sequencia de números aleatórios é usado diretamente em algoritmo genéticos com chaves aleatórias (ver 3.4.1).
Uma transformação é tipicamente sobrejetiva ("many-to-one"), i.e. existem várias representações fonte para uma representação destino. Idealmente, existe o mesmo número de representações fontes para representações destino, para manter a mesma distribuição de soluções nos dois espaços.

**Exemplo 1.2 (Representação de permutações por chaves aleatórias)**
Uma permutação de $n$ elementos pode ser representado por $n$ números aleatórios reais em $[0, 1]$. Para números aleatórios são $a_1, \ldots, a_n$, seja $\pi$ uma permutação tal que $a_{\pi(1)} \leq \cdots a_{\pi(n)}$. Logo os números $a_i$ representam a permutação $\pi$ (ou $\pi^{-1}$). ◊

Uma transformação pode ser é útil caso o problema possui muitas restrições e o espaço de busca definido por uma representação direta contém muitas soluções inválidas.

**Exemplo 1.3 (Coloração de vértices)**
Uma representação direta da coloração de vértices pode ser uma atribuição de cores a vértices. Para um limite de no máximo $n$ cores, temos $n^n$ possíveis atribuições, mas várias são infactíveis. Uma representação indireta é uma permutação de vértices. Para uma dada permutação um algoritmo guloso processo os vértices em ordem e atribui o menor cor livre ao vértice atual. A corretude dessa abordagem mostra

**Lema 1.1**
Para uma dada k-coloração, sejam $C_1 \cup \cdots \cup C_k$ as classes de cores. Ordenando os vértices por classes de cores, o algoritmo guloso produz uma coloração com no máximo $k$ cores.

**Prova.** Mostraremos por indução que a coloração das primeiras $i$ classes não precisa mais que $i$ cores. Para a primeira classe isso é obviou. Supõe que na coloração da classe $i$ precisamos usar a cor $i+1$. Logo existe um vizinho com cor $i$. Mas pelo hipótese da indução o vizinho não pode ser de uma classe menor. Logo, temos uma aresta entre dois vértices da mesma classe, uma contradição. ■
Com essa representação, todas soluções são válidas. Observe que o tamanho do espaço da busca $n! \approx \sqrt{2\pi n}(n/e)^n$ (por A.5) é similar nas duas representações. ♢

Por fim, transformações podem ser úteis caso podemos resolver subproblemas restritas do problema eficientemente.

**Exemplo 1.4 (Sequenciamento em máquinas paralelas não relacionadas)**
Uma solução para $R \parallel \sum w_j C_j$ direta 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 sequencia em ordem de tempo de processamento ponderado não-decrescente $p_1/w_1 \leq \cdots \leq p_n w_n$.

**Prova.** Supõe uma sequencia 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 pelo 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 à máquinas. ♢

As diferentes representações compactas podem ser combinadas.

**Exemplo 1.5 (Simple assembly line balancing)**
No "simple assembly line balacing 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 definido: elas são ordenados 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 $i,\ldots,n$ em $k$ máquinas, a solução ótima é $C(1,m)$ e $C$ satisfaz

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

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 1.1 (Subproblemas)**
Identifica os subproblemas mais difíceis que podem ser resolvidos em tempo polinomial é considera uma representação que contém somente a informação necessário para definir os subproblemas.

## 1.3. Estratégia de busca: Diversificação e intensificação

No projeto de uma heurística temos que balancear dois objetivos antagonistas: a *diversificação* da busca e a *intensificação* de busca. A diversificação da busca (também chamada *exploration*) procura garantir uma boa cobertura do espaço de busca, evitando que a soluções analisadas ficam confinado a uma região pequena do espaço total. A diversificação ideal é uma algoritmo que repetidamente gera soluções aleatórias. Em contraste a *intensificação* (também chamada *exploitation*) procura melhor a solução atual o mais possível. Um exemplo de uma intensificação seria analisar todas soluções dentro uma certa distância da solução atual.

## 1.4. Notas

Mais informações sobre os teoremas NFL se encontram no artigo original de Wolpert e Macready [10] e em Burke e Kendall [2, ch. 11] e Rothlauf [6, ch.3.4.4].

# 2. Busca por modificação de soluções

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**
Uma *vizinhança* de uma instância $x$ de um problema de otimização $\Pi$ é uma função $N : S(x) \to 2^{S(x)}$. 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**
O *grafo de vizinhança* $G = (V, E)$ para uma instância $x$ de um problema de otimização $\Pi$ 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 $s'$ no grafo de vizinhança. 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 $\hat{P}_s$ 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 definido 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 $f(s)$.

```
1  LocalSearch(s)=
2    s* := s
3    repeat
4      seleciona s' ∈ N̂(s) de acordo com P̂_s
5      s := s'
6      s* = min{s*, s}
7    until critério de parada satisfeito
8    return s*
9  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

**Caminhada aleatória (ingl. random walk)** Para $N(s) \neq \emptyset$, define $P_s(s) = 1/|N(s)|$.

**Amostragem aleatória (ingl. random picking)** Uma caminhada aleatória com $N(s) = S(x)$ para todo $s \in S(x)$.

**Exemplo 2.1 (Politopos e o método Simplex)**
O método Simplex define uma vizinhança entre os vértices do politopo de um programa linear: cada par variável entrante é variável sainte admissível define um vizinho. Essa vizinhança é simétrica, conectada, fracamente otimamente conectada e exata. Logo o método resolve o problema da programação linear. ◊

**Exemplo 2.2 (k-exchange para o PCV)**
Uma vizinhança para o PCV é k-exchange Croes [3]: 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 inexata (por quê?). O tamanho da vizinhança é $O(n^k)$ para $n$ cidades.

◊

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

◊

## 2.1. Busca locais monótonas

Uma busca local monótona permite somente modificações que melhoram a solução atual, i.e. no algoritmo LocalSearch sempre temos $P_s(s') = 0$ para $s' \notin B(s)$ . Logo, o algoritmo termina num ótimo local. Pela monotonia também não é necessário guardar a melhor solução encontrada. A busca depende da estratégia de seleção da nova solução $s'$, também conhecido como *regra de pivoteamento*.

**Algoritmo 2.2 (LocalDescent)**
**Entrada** Solução inicial $s$, vizinhança $N$, distribuição $P_s$.

**Saída** Uma solução com valor no máximo $f(s)$.

```
1  LocalDescent(s):=
2    repeat
3      seleciona s' ∈ N̂(s) de acordo com P̂_s
4      s := s'
5    until s' = s
6    return s
7  end
```

**Descida aleatória (ingl. random descent)** Para $B(s) \neq \emptyset$ define $P_s(s') = 1/|B(s)|$ para $s' \in B(s)$. Esta estratégia é equivalente com a primeira melhora, mas em ordem aleatória.

**Primeira melhora (ingl. first improvement)** A primeira melhora supõe uma vizinhança ordenada $B(s) = \{b_1, \ldots, b_k\}$. Ela seleciona $f = \min\{i \mid$

$f(b_i) < f(s)\}$, i.e. $P_s(b_i) = [i = f]$. O método é conhecido pelos nomes "hill climbing" (no caso de maximização) ou "hill descent" (no caso de minimização).

**Melhor melhora (ingl. best improvement)** Para $B(s) \neq \emptyset$, define $B^*(s) = \{s' \in B(s) \mid f(s') = \min_{s'' \in B(s)} f(s'')$ e $P_s(s') = 1/|B^*(s)|$ para $s' \in B^*(s)$. O método é conhecido pelos nomes "steepest ascent" (no caso de maximização) ou "steepest descent" (no caso de minimização).

**Busca por amostragem (ingl. sample search)** Seleciona um subconjunto $S \subseteq N(x)$ aleatório de tamanho $\alpha|N(x)|$, define $B^*(s) = \{s' \in B(s) \mid f(s') = \min_{s'' \in S} f(s'')$ e $P_s(s') = 1/|B^*(s)|$ para $s' \in B^*(s)$.

As estratégias obviamente podem ser combinadas, por exemplo, aplicar uma estratégia de "primeira melhora" após uma amostragem.
A qualidade de uma busca local depende da vizinhança: para vizinhanças maiores esperamos encontrar ótimos locais melhores. Porém a complexidade da busca cresce com a vizinhança. A arte, porém, consiste em balancear estes dois objetivos.

**Exemplo 2.4 (Método Simplex)**
Não conhecemos regras de pivoteamento para o método Simplex que garantem uma complexidade polinomial. Porém, a programação linear possui soluções polinomiais (que não usam busca local). Por isso, a complexidade de encontrar ótimos locais pode ser menor que a complexidade do método iterativo. ◊

**Exemplo 2.5 (OneMax)**
Para um $x^* \in \{0,1\}^n$ fixo o problema OneMax consiste encontrar o mínimo de $f(x) = |x - x^*|_1$, i.e. $x^*$. O número de bits $X$ corretos de uma solução aleatória satisfaz $E[X] = n/2$ e $\Pr[X \leq n/3] \leq e^{-n/36}$ e $\Pr[X \geq 2n/3] \leq e^{-n/54}$ (aplicando limites de Chernoff (A.4)).
Uma descida aleatória precisa tempo $O(n)$ para selecionar um vizinho, avaliando a função objetivo em $O(1)$, e $O(n)$ passos, para um tempo total de $O(n^2)$. Uma analisa mais detalhada é o seguinte: para selecionar um vizinho melhor, podemos repetidamente selecionar um vizinho arbitrário, até encontrar um vizinho melhor. Com $i$ bits diferentes, encontramos um vizinho melhor com probabilidade $i/n$. Logo a seleção precisa esperadamente $n/i$ passos até encontrar um vizinho melhor (ver lema A.2) e logo no máximo

$$\sum_{1 \leq i \leq n} n/i = nH_n \approx n \log n$$

passos até encontrar $x^*$.

A primeira melhora precisa no pior caso (todos bits diferentes) tempo esperado $\Theta(n/w)$ para encontrar um vizinho melhor, e a melhor melhora tempo $\Theta(n)$. Logo, ambas precisam tempo $\Theta(n^2)$ para encontrar $x^*$. ◊

**Exemplo 2.6 (GSAT)**
O algoritmo GSAT [8] aplica a estratégia "melhor melhora" na vizinhança 1-flip com função objetivo sendo o número de cláusulas satisfeitas. Ele periodicamente recomeça a busca numa solução aleatória. ◊

**Exemplo 2.7 (WalkSAT)**
WalkSAT usa uma estratégia de seleção mais sofisticada: em cada passo uma cláusula não satisfeita é selecionada, e uma variável aleatória dessa cláusula é invertida. (O WalkSAT proposto por Selman, Kautz e Cohen [7] seleciona uma variável que não invalida nenhuma outra cláusula ou com probabilidade $p$ uma que invalide o menor número e com probabilidade $1-p$ uma aleatória.) Logo a vizinhança é um subconjunto da vizinhança 1-flip. WalkSAT também recomeça a busca a partir de uma solução aleatória periodicamente.

**Lema 2.1**
Seja $\varphi$ uma fórmula em $k$-CNF satisfatível com $n$ variáveis. O algoritmo WalkSAT precisa esperadamente $O(n^{3/2}(2(k-1)/k)^n)$ passos até encontrar uma atribuição que satisfaz $\varphi$.

**Prova.** Seja $a$ uma atribuição que satisfaz $\varphi$. Vamos determinar a probabilidade $q$ que um período de WalkSAT encontra $a$. Com $p_j = \binom{n}{j}2^{-n}$ a probabilidade de iniciar com distância Hamming $j$ de $a$, e $q_j$ a probabilidade de encontrar $a$ a partir da distância $j$, temos

$$q = \sum_{0 \le j \le n} p_j q_j. \qquad (*)$$

A distância Hamming para $a$ diminui com probabilidade pelo menos $1/k$ e aumento com probabilidade no máximo $1-1/k$. Podemos modelar o WalkSAT como caminhada aleatória entre classes de soluções com distância Hamming $j$, com uma probabilidade de transição de $j$ para $j-1$ ("para baixo") de $1/k$ e de $j$ para $j+1$ ("para acima") de $1-1/k$. Com isso $q_j$ é pelo menos a probabilidade de chegar na classe $0$ a partir da classe $j$ em no máximo $3n$ passos. Para isso podemos fazer $j$ passos para baixo, ou $j+1$ para baixo e um para acima, e no geral $j+l$ para baixo e $l$ para acima. Logo

$$q_j \ge \max_{0 \le l \le (3n-j)/2} \binom{j+2l}{l} \left(\frac{k-1}{k}\right)^l \left(\frac{1}{k}\right)^{j+l}.$$

Para $l = \alpha j$ com $\alpha \in (0, 1/2)$ temos

$$q_j \geq \binom{(1+2\alpha)j}{\alpha j} \left( \left( \frac{k-1}{k} \right)^{\alpha} \left( \frac{1}{k} \right)^{(1+\alpha)} \right)^j .$$

Aplicando o lema A.1 é possível estimar

$$\binom{(1+2\alpha)j}{\alpha j} \geq (8j)^{-1/2} \left( \left( \frac{1+2\alpha}{\alpha} \right)^{\alpha} \left( \frac{1+2\alpha}{1+\alpha} \right)^{1+\alpha} \right)^j$$

e logo

$$q_j \geq (8j)^{-1/2} \left( \left( \frac{1+2\alpha}{\alpha} \right)^{\alpha} \left( \frac{1+2\alpha}{1+\alpha} \right)^{1+\alpha} \left( \frac{k-1}{k} \right)^{\alpha} \left( \frac{1}{k} \right)^{(1+\alpha)} \right)^j .$$

Escolhendo $\alpha = 1/(k-2)$ e simplificando obtemos

$$q_j \geq (8j)^{-1/2} \left( \frac{1}{k-1} \right)^j .$$

Finalmente, substituindo em (*)

$$\begin{aligned} q &\geq 2^{-n} + \sum_{j \in [n]} \binom{n}{j} 2^{-n} (8j)^{-1/2} \left( \frac{1}{k-1} \right)^j \\ &\quad 2^{-n} (8n)^{-1/2} \sum_{j \in [n]} \binom{n}{j} \left( \frac{1}{k-1} \right)^j 1^{n-j} \\ &= 2^{-n} (8n)^{-1/2} \left( 1 + \frac{1}{k-1} \right)^n = \frac{1}{\sqrt{8n}} \left( \frac{k}{2(k-1)} \right)^n \end{aligned}$$

Logo, o número esperado de períodos é

$$1/q = \sqrt{8n} \left( \frac{2(k-1)}{k} \right)^n$$

e como cada período precisa tempo $O(n)$ o resultado segue. ■

Para uma fórmula satisfatível com $k = 3$, por exemplo, o algoritmo precisa $O(n^{3/2}(4/3)^n)$ passos.

É possível transformar esta algoritmo num algoritmo randomizado que decide se uma fórmula é satisfatível com alta probabilidade. ◊

**Exemplo 2.8 (2-opt para o PCV)**
A estratégia 2-opt para o PCV é uma descida aleatória na vizinhança 2-exchange. Similarmente, obtemos k-opt na vizinhança k-exchange.

**Teorema 2.1**
Para $k \geq 2$, $n \geq 2k+8$ e para $\alpha > 1/n$ existe uma instância $x$ do PCV com $n$ cidades, tal que

$$\frac{\text{k-opt}(x)}{\text{OPT}(x)} > \alpha.$$

**Prova.** Para um $k$ par, define distâncias

$$\begin{aligned}
d_{12} &= 1 && \\
d_{i,i+1} = d_{n,1} &= 1/n\alpha && i \in [2, n) \\
d_{k+3,2k+4} &= 1/n\alpha && \\
d_{j,2k+4-j} &= 1/n\alpha && j \in [k] \\
d_{i,j} &= kn && \text{caso contrário}
\end{aligned}$$

Um ciclo Hamiltoniano ótimo é dado por arestas $(i, \text{próximo}(i))$ com

$$\text{próximo}(i) = \begin{cases}
2k+4-i & \text{para } i \text{ impar e } i < k \\
i+1 & \text{para } i \text{ par e } i < k \\
i+1 & \text{para } i \in [k, k+2] \\
2k+4 & \text{para } i = k+3 \\
i-1 & \text{para } i \text{ impar e } i \in [k+3, 2k+4) \\
2k+4-i & \text{para } i \text{ par e } i \in [k+3, 2k+4) \\
i+1 & \text{para } i \in [2k+4, n] \\
1 & \text{para } i = n
\end{cases}$$

A otimalidade segue do fato que todas arestas possuem o peso mínimo $1/n\alpha$. Este ciclo é único ciclo ótimo (Exercício!). Por outro lado, o ciclo $(1, 2, \ldots, n)$ possui peso total $1 + (n-1)/n\alpha$, mas tem $k+1$ arestas diferentes. Logo este ciclo é um mínimo local para k-exchange e para a instância acima temos

$$\frac{\text{k-opt}(x)}{\text{OPT}(x)} = \alpha + 1 - 1/n > \alpha.$$

Para provar o caso para um $k$ impar, podemos observar que um mínimo local para o $k+1$-exchange, também é um mínimo local para k-exchange. ■

◊

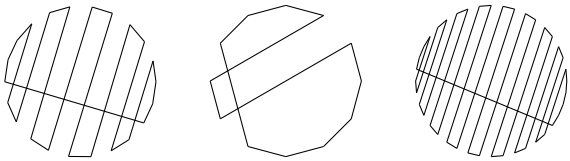

Figura 2.1.: Caminhos construídos na prova do teorema 2.1. Esquerda: $n = 22$, $k = 8$. Meio: $n = 12$, $k = 2$. Direita: $n = 40$, $k = 16$.

### 2.1.1. Segue os vencedores

Segue os vencedores (ingl. go with the winners) [1] é uma estratégia que trabalho com múltiplas soluções. Caso solução percorre uma trajetória de uma busca local monótona. Caso uma das trajetórias termina num mínimo local, ela continua no ponto atual de uma das outras trajetórias que ainda não chegaram num mínimo local. A busca termina, caso todas trajetórias terminaram num mínimo local.

**Algoritmo 2.3 (Segue os vencedores)**
**Entrada** Solução inicial $s$, vizinhança $N$, distribuição $P_s$, o número de soluções $k$.

**Saída** Uma solução com valor no máximo $f(s)$.

```
1  SV(s)=
2     s_i := s para i ∈ [k]
3     s* = s
4     repeat
5        seja L := {i ∈ [k] | B(s) = ∅}
6        atribui às soluções em L
7           uniformemente soluções em [k] \ L
8        seleciona s'_i ∈ N̂(s_i) de acordo com P̂_{s_i}
9        s_i := s'_i
10       s* = min{s_*, s_1, ..., s_k}
11    until L = [k]
12    return s*
13 end
```

Uma variante [4] distribui as soluções de acordo com o número de vizinhos melhores.

## 2.2. Buscas locais não-monótonas

Uma busca local não-monótona permite piorar a solução atual.

**Algoritmo 2.4 (S-LocalSearch)**
**Entrada** Solução inicial $s_0$.

**Saída** Solução s tal que $f(s) \leq f(s)$.

```
1  S–LocalSearch(s)=
2    repeat
3      s* := s
4      seleciona S ⊆ N(s)
5      s' = argmin_s{f(s) | s ∈ S}
6      if f(s') < f(s) ∨ acceptable(s') then
7        s := s'
8        s* := min{s*, s}
9      endif
10   until critério de parada satisfeito
11   return s*
12 end
```

# 3. Busca por recombinação de soluções

## 3.1. Religamento de caminhos

## 3.2. Scatter search

## 3.3. Probe

## 3.4. Algoritmos genéticos e meméticos

### 3.4.1. Algoritmos genéticos com chaves aleatórias

## 3.5. GRASP com religamento de caminhos

# 4. Busca por construção de soluções

## 4.1. Construção simples

### 4.1.1. Algoritmos (semi-)gulosos

### 4.1.2. Algoritmos de prioridade

### 4.1.3. Busca por raio

## 4.2. Construção repetida independente

### 4.2.1. GRASP

### 4.2.2. Bubble search randomizada

## 4.3. Construção repetida dependente

### 4.3.1. Iterated greedy algorithm

### 4.3.2. Squeaky wheel optimization

## 4.4. Otimização por colônias de formigas

# 5. Tópicos

## 5.1. Hibridização de heurísticas

## 5.2. Híper-Heurísticas

## 5.3. Heurísticas para problemas multi-objetivos

## 5.4. Heurísticas paralelas

# 6. Metodologia para o projeto de heurísticas

## 6.1. Escolha de parâmetros

## 6.2. Avaliação de meta-heurísticas

### 6.2.1. Teste de hipóteses

### 6.2.2. Comparação de meta-heurísticas

## 6.3. Demais avaliações

# A. Conceitos matemáticos

**Definição A.1**
Uma função $f$ é *convexa* se ela satisfaz a desigualdade de Jensen

$$f(\Theta x + (1-\Theta)y) \leq \Theta f(x) + (1-\Theta)f(y). \tag{A.1}$$

Similarmente uma função $f$ é *concava* caso $-f$ é convexo, i.e., ela satisfaz

$$f(\Theta x + (1-\Theta)y) \geq \Theta f(x) + (1-\Theta)f(y). \tag{A.2}$$

**Exemplo A.1**
Exemplos de funções convexas são $x^{2k}$, $1/x$. Exemplos de funções concavas são $\log x$, $\sqrt{x}$. ◊

**Proposição A.1**
Para $\sum_{i\in[n]} \Theta_i = 1$ e pontos $x_i$, $i \in [n]$ uma função convexa satisfaz

$$f\Big(\sum_{i\in[n]} \Theta_i x_i\Big) \leq \sum_{i\in[n]} \Theta_i f(x_i) \tag{A.3}$$

e uma função concava

$$f\Big(\sum_{i\in[n]} \Theta_i x_i\Big) \geq \sum_{i\in[n]} \Theta_i f(x_i) \tag{A.4}$$

**Prova.** Provaremos somente o caso convexo por indução, o caso concavo sendo similar. Para $n = 1$ a desigualdade é trivial, para $n = 2$ ela é válida por definição. Para $n > 2$ define $\bar{\Theta} = \sum_{i\in[2,n]} \Theta_i$ tal que $\Theta + \bar{\Theta} = 1$. Com isso temos

$$f\Big(\sum_{i\in[n]} \Theta_i x_i\Big) = f\big(\Theta_1 x_1 + \sum_{i\in[2,n]} \Theta_i x_i\big) = f(\Theta_1 x_1 + \bar{\Theta} y)$$

onde $y = \sum_{j\in[2,n]}(\Theta_j/\bar{\Theta})x_j$, logo

$$\begin{aligned} f\Big(\sum_{i\in[n]}\Theta_i x_i\Big) &\le \Theta_1 f(x_1) + \bar{\Theta} f(y) \\ &= \Theta_1 f(x_1) + \bar{\Theta} f\Big(\sum_{j\in[2,n]}(\Theta_j/\bar{\Theta})x_j\Big) \\ &\le \Theta_1 f(x_1) + \bar{\Theta}\sum_{j\in[2,n]}(\Theta_j/\bar{\Theta}) f(x_j) = \sum_{i\in[n]}\Theta_i x_i \end{aligned}$$

■

**Definição A.2**
O *fatorial* é a função

$$n! : \mathbb{N} \to \mathbb{N} : n \mapsto \prod_{1\le i\le n} i.$$

Temos a seguinte aproximação do fatorial (fórmula de Stirling)

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1 + O(1/n)) \tag{A.5}$$

Uma estimativa menos preciso pode ser obtido estimando

$$e^n = \sum_{i\ge 0}\frac{n^i}{i!} > \frac{n^n}{n!}$$

que implica

$$(n/e)^n \le n! \le n^n.$$

**Definição A.3 (Entropia binária)**
A entropia binária para $\alpha \in (0,1)$ é $h(\alpha) = -\alpha\log_2\alpha - (1-\alpha)\log_2 1-\alpha$.

**Lema A.1**
Para $\alpha \in (0, 1/2]$

$$(8n\alpha(1-\alpha))^{-1/2}\, 2^{h(\alpha)n} \le \sum_{1\le i\le n\alpha}\binom{n}{i} \le 2^{h(\alpha)n}.$$

**Prova.** Para a segunda desigualdade temos

$$\begin{aligned}
1 = (\alpha + (1-\alpha))^n &= \sum_{1 \le i \le n} \binom{n}{i} \alpha^i (1-\alpha)^{n-i} \\
&\ge \sum_{1 \le i \le n\alpha} \binom{n}{i} \left(\frac{\alpha}{1-\alpha}\right)^i (1-\alpha)^n \\
&\ge \sum_{1 \le i \le n\alpha} \binom{n}{i} \left(\frac{\alpha}{1-\alpha}\right)^{n\alpha} (1-\alpha)^n \\
&= \alpha^{n\alpha}(1-\alpha)^{(1-\alpha)n} \sum_{1 \le i \le n\alpha} \binom{n}{i} \\
&= 2^{-nh(\alpha)} \sum_{1 \le i \le n\alpha} \binom{n}{i}.
\end{aligned}$$

O terceiro passo é valido porque para $\alpha \in (0, 1/2]$ temos $\alpha/(1-\alpha) \le 1$ e $i \le n\alpha$. ■

## A.1. Probabilidade discreta

### Probabilidade: Noções básicas

- *Espaço amostral* finito $\Omega$ de *eventos elementares* $e \in \Omega$.
- Distribuição de probabilidade $\Pr[e]$ tal que

$$\Pr[e] \geq 0; \qquad \sum_{e \in \Omega} \Pr[e] = 1$$

- *Eventos* (compostos) $E \subseteq \Omega$ com probabilidade

$$\Pr[E] = \sum_{e \in E} \Pr[e]$$

**Exemplo A.2**
Para um dado sem bias temos $\Omega = \{1, 2, 3, 4, 5, 6\}$ e $\Pr[i] = 1/6$. O evento $\text{Par} = \{2, 4, 6\}$ tem probabilidade $\Pr[\text{Par}] = \sum_{e \in \text{Par}} \Pr[e] = 1/2$. $\Diamond$

### Probabilidade: Noções básicas

- *Variável aleatória*

$$X : \Omega \to \mathbb{N}$$

- Escrevemos $\Pr[X = i]$ para $\Pr[X^{-1}(i)]$.
- Variáveis aleatórias *independentes*

$$P[X = x \text{ e } Y = y] = P[X = x]P[Y = y]$$

- *Valor esperado*

$$E[X] = \sum_{e \in \Omega} \Pr[e]X(e) = \sum_{i \geq 0} i \Pr[X = i]$$

- Linearidade do valor esperado: Para variáveis aleatórias $X$, $Y$

$$E[X + Y] = E[X] + E[Y]$$

**Prova.** (Das formulas equivalentes para o valor esperado.)

$$\begin{aligned}\sum_{0\le i} \Pr[X = i]i &= \sum_{0\le i} \Pr[X^{-1}(i)]i \\ &= \sum_{0\le i} \sum_{e\in X^{-1}(i)} \Pr[e]X(e) = \sum_{e\in\Omega} \Pr[e]X(e)\end{aligned}$$

■

**Prova.** (Da linearidade.)

$$\begin{aligned}E[X+Y] &= \sum_{e\in\Omega} \Pr[e](X(e)+Y(e)) \\ &= \sum_{e\in\Omega} \Pr[e]X(e) \sum_{e\in\Omega} \Pr[e]Y(e)) = E[X]+E[Y]\end{aligned}$$

■

**Exemplo A.3**
(Continuando exemplo A.2.)
Seja $X$ a variável aleatório que denota o número sorteado, e $Y$ a variável aleatório tal que $Y =$ [a face em cima do dado tem um ponto no meio].

$$E[X] = \sum_{i\ge 0} \Pr[X = i]i = 1/6 \sum_{1\le i\le 6} i = 21/6 = 7/2$$

$$E[Y] = \sum_{i\ge 0} \Pr[Y = i]i = \Pr[Y = 1] = 1/2 \qquad E[X+Y] = E[X]+E[Y] = 4$$

◊

**Lema A.2**
Para tentativas repetidas com probabilidade de sucesso $p$, o número esperado de passos para o primeiro sucesso é $1/p$.

**Prova.** Seja $X$ o número de passos até o primeiro sucesso. Temos $P[X = k] = (1-p)^{k-1}p$ e logo

$$\begin{aligned}E[X] &= \sum_{k\ge 1} kP[X = k] = \sum_{k\ge 1} k(1-p)^{k-1}p \\ &= p\left(\sum_{k\ge 1} k(1-p)^k + \sum_{k\ge 0} (1-p)^k\right) \\ &= p\left((1-p)/p^2 + 1/p\right) = 1/p.\end{aligned}$$

■

**Proposição A.2**
Para $\varphi$ convexo $\varphi(E[X]) \leq E[\varphi(X)]$ e para $\varphi$ concavo $\varphi(E[X]) \geq E[\varphi(X)]$.

**Prova.** Pela proposição A.1. ■

**Proposição A.3 (Desigualdade de Markov)**
Seja $X$ uma variável aleatória com valores não-negativas. Então, para todo $a > 0$

$$Pr[X \geq a] \leq E[X]/a.$$

**Prova.** Seja $I = [X \geq a]$. Como $X \geq 0$ temos $I \leq X/a$. O valor esperado de $I$ é $E[I] = Pr[I = 1] = Pr[X \geq a]$, logo

$$Pr[X \geq a] = E[I] = E[X/a] = E[X]/a.$$

■

**Proposição A.4 (Limites de Chernoff (ingl. Chernoff bounds))**
Sejam $X_1, \ldots, X_n$ indicadores independentes com $Pr[X_i] = p_i$. Para $X = \sum_i X_i$ temos para todo $\delta > 0$

$$Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$$

para todo $\delta \in (0, 1)$

$$Pr[X \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu}$$

para todo $\delta \in (0, 1]$

$$Pr[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}$$

e para todo $\delta \in (0, 1)$

$$Pr[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}.$$

**Exemplo A.4**
Sejam $X_1, \ldots, X_k$ indicadores com $Pr[X_i = 1] = p_i$ e $X = \sum_i X_i$. Temos $\mu = E[X] = \sum_i E[X_i] = \alpha k$. Qual a probabilidade de ter menos que a metade dos $X_i = 1$?

$$Pr[X \leq \lfloor k/2 \rfloor] \leq Pr[X \leq k/2] = Pr[X \leq \mu/2\alpha] = Pr[X \leq \mu(1-(1-1/2\alpha))] \leq e^{-\mu\delta^2/2} = e^{-k/2\alpha(\alpha-1/2)^2}.$$

◊

# Bibliografia


[1] David Aldous e Umesh Vazirani. ""Go With the Winners" Algorithms". Em: *Proc. 26th STOC*. 1994.

[2] Edmund K. Burke e Graham Kendall, eds. *Search methodologies*. Springer, 2005. URL: http://www.springer.com/mathematics/applications/book/978-0-387-23460-1.

[3] G. A. Croes. "A Method for Solving Traveling-Salesman Problems". Em: *Oper. Res.* 6 (1958), pp. 791–812. DOI: 10.1287/opre.6.6.791.

[4] Tassos Dimitriou e Russell Impagliazzo. "Towards an Analysis of Local Optimization Algorithms". Em: *Proc. 28th STOC*. 1996.

[5] Christos Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993.

[6] Franz Rothlauf. *Design of Modern Heuristics: Principles and Application*. INF: Recurso eletrônico. Springer, 2011. ISBN: 978-3540729617. URL: http://link.springer.com/book/10.1007/978-3-540-72962-4/page/1.

[7] B. Selman, H. Kautz e B. Cohen. "Noise strategies for improving local search". Em: *Proc. 12th Nat. Conf. Artif. Intell.* 1994, pp. 337–343.

[8] B. Selman, H. Levesque e D. Mitchell. "A new method for solving hard satisfiability problems". Em: *Proc. 10th Nat. Conf. Artif. Intell.* 1992, pp. 440–446.

[9] The Pittsburg Press. *Economics in eight words*. 1938.

[10] David H. Wolpert e WIlliam G. Macready. "No Free Lunch Theorems for Optimization". Em: *IEEE Trans. Evol. Comp.* (1997), pp. 67–82.

# Índice