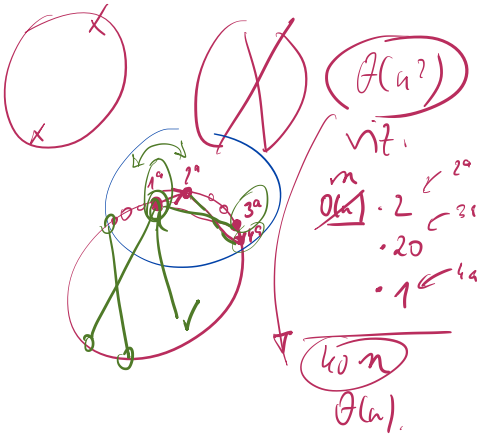

boa qualidade. Uma forma comum de reduzir a vizinhança é usar *listas de candidatos* (ingl. candidate lists).

Exemplo 2.4 (Vizinhança reduzida para o PCV)
No caso do 2-exchange para o PCV muitas das $\Theta(n^2)$ vizinhos produzem rotas inferiores, porque eles introduzem uma arestas longas, caso as duas arestas originais ficam muito distantes. Logo é possível reduzir a vizinhança heuristicamente, sem expectativa de perder soluções boas. Uma estratégia proposto por D. S. Johnson e McGeoch (2003) é: escolher uma cidade aleatória, um vizinho aleatório dessa cidade na rota, uma terceira cidade entre os 20 vizinhos mais próximos da segunda cidade, e a quarta cidade como sucessor da terceira na orientação da rota dado pelas primeiras duas cidades. Com isso uma rota tem no máximo $40n$ vizinhos. ◊

Exemplo 2.5 (Bits "don't look" para o PCV)
Considera a estratégia do exemplo anterior e supõe que para uma dada seleção da primeira cidade não tem um movimento que melhora a rota. Empiricamente, caso essa cidade continua com os mesmos vizinhos, a probabilidade de encontrar um movimente que melhora é baixa. Isso é o motivo para introduzir bits "don't look" (Bentley 1992). Cidades que não levaram a uma solução melhor recebem ficam exclusos nas próximas iterações até um vizinho mudar. ◊

A redução de vizinhanças frequentemente é uma estratégia importante para obter resultados de boa qualidade (D. S. Johnson e McGeoch 2003; Toth e Vigo 2003; Glover e Laguna 1997), motivo para

Princípio de projeto 2.2 (Redução de vizinhanças)
Considera eliminar das vizinhanças movimentos com baixa probabilidade de melhorar a solução.

2.2. Buscas 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 conhecida como *regra de pivoteamento*.

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

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

```
LocalDescent(s)=
  repeat
    seleciona s' ∈ N̂(s) de acordo com P̂_s
    s := s'
  until P_s(s) = 1
```

```
    return s
end
```

Descida aleatória (ingl. stochastic hill 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 $f = \min\{i \mid \varphi(b_i) < \varphi(s)\}$, 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 $B^*(s) = \{s' \in B(s) \mid \varphi(s') = \min_{s'' \in B(s)} \varphi(s'')\}$ 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 $B^*(s) = \{s' \in B(s) \mid \varphi(s') = \min_{s'' \in S} \varphi(s'')$ 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, então, consiste em balancear estes dois objetivos.

Exemplo 2.6 (Método Simplex)
Não conhecemos um regra de pivoteamento para o método Simplex que garante uma complexidade polinomial. Porém, a programação linear possui soluções polinomiais (que não usam busca local). Isso indica que a complexidade de encontrar ótimos locais pode ser menor que a complexidade do método iterativo. ◊

Exemplo 2.7 (Árvore geradora mínima)
Para uma árvore geradora, podemos definir vizinhos como segue: adicione uma aresta, e remove outra do (único) ciclo formado. Uma árvore geradora é mínima se e somente se não existe melhor vizinho (prova: exercício). Por isso a busca local resolve o problema de encontrar a árvore geradora mínima. A vizinhança é simétrica, fracamente otimamente conectada e exata. Porém, a busca local geralmente não é eficiente. ◊

Exemplo 2.8 (OneMax)
Para $x^* \in \{0,1\}^n$ fixo o problema OneMax consiste em encontrar o mínimo de $\varphi(x) = |x - x^*|_1$, i.e. x^*. O número de bits X corretos de

uma solução aleatória satisfaz $E[X] = n/2$ e $P(X \leq n/3) \leq e^{-n/36}$ e $P(X \geq 2n/3) \leq e^{-n/54}$ (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 sem repetição, e $O(n)$ passos, para um tempo total de $O(n^2)$. Uma análise mais detalhada do caso médio é a 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.3) 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 $\Theta(n/i)$ para encontrar um vizinho melhor, e a melhor melhora tempo $\Theta(n)$. Logo, ambas precisam tempo $\Theta(n^2)$ para encontrar x^*. ◊

Exemplo 2.9 (GSAT)
O algoritmo GSAT (Selman, Levesque et al. 1992) aplica a estratégia "melhor vizinho" na vizinhança 1-flip com função objetivo sendo o número de cláusulas satisfeitas (observe que é importante escolher entre os melhores uniformemente). Ele periodicamente recomeça a busca a partir de uma solução aleatória. ◊

Exemplo 2.10 (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 et al. (1994) 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 (Schöning (1999))
Seja φ uma fórmula em k-CNF satisfatível com n variáveis. O algoritmo WalkSAT com período $3n$ precisa esperadamente $O(n^{3/2}(2(k-1)/k)^n)$ passos até encontrar uma atribuição que satisfaz φ.

Prova. Seja a uma atribuição que satisfaz φ. 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

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

A distância Hamming para a diminui com probabilidade pelo menos $1/k$ e aumenta 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 conseguir 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

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

Para $l = \alpha j$ com $\alpha \in (0,1)$ 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.4 é podemos estimar[1]

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

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 (*)

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

Logo, o número esperado de períodos é

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

[1] Substituindo diretamente é descartando o fator $\sqrt{(1+2\alpha)/(\alpha(1+\alpha))} \geq 1$.

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 este algoritmo num algoritmo randomizado que decide se uma fórmula é satisfatível com alta probabilidade. ◊

Princípio de projeto 2.3 (Reinícios)
Considera reinícios frequentes. Eles podem ficar mais efetivos caso a probabilidade de atingir a qualidade desejada é baixo.

Exemplo 2.11 (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 (Chandra et al. (1999))
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 o 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

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

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. ■

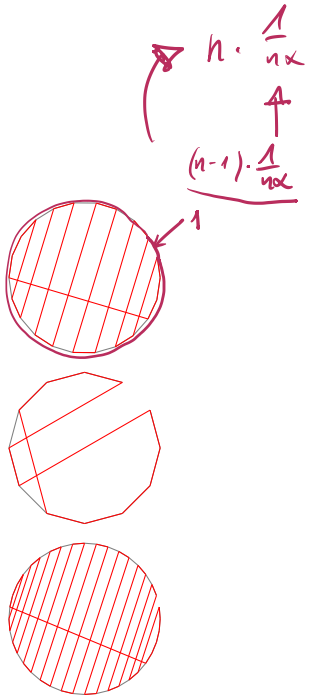

Figura 2.2.: Caminhos construídos na prova do teorema 2.1. Acima: $n = 22$, $k = 8$. Meio: $n = 12$, $k = 2$. Abaixo: $n = 40$, $k = 16$. A figura somente mostra arestas de distância $1/n\alpha$.