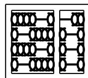


UNIVERSIDADE DE CAMPINAS - UNICAMP
INSTITUTO DE COMPUTAÇÃO - IC

Heuristicas e Metaheurísticas

Flávio Keidi Miyazawa

Campinas, 2008

Suponha $P \neq NP$.

Para tratar um problema NP-difícil, devemos sacrificar uma das características.

1. Resolver o problema na otimalidade
2. Resolver o problema em tempo polinomial

Para isto, podemos desenvolver:

Algoritmos Aproximados que sacrificaram 1.

Algoritmos Exatos que sacrificaram 2.

Heurísticas : sacrificam 1 e possivelmente 2

- Tentam encontrar soluções boas, guiadas por uma boa idéia
- Podem futuramente vir a ter análise mais formal, por análise de pior caso, aproximação, análise probabilística, etc.

Veremos:

- Heurísticas Construtivas
- Heurísticas de Busca Local
- Equilibrio de Nash e Busca Local
- Algoritmo Metropolis
- Simulated Annealing
- Busca Tabu
- Algoritmos Genéticos
- GRASP e Path Relinking
- entre outras

Heurísticas Construtivas

- Tentam construir solução iterativamente

Exemplos para TSP:

- Heurística do Vizinho mais próximo
- Heurística da Inserção do mais próximo
- Heurística da Inserção do mais distante

Já vimos algoritmos gulosos ótimos para

- Encontrar Árvore Geradora de Custo Mínimo
- Construir Árvore de Huffman (compressão)
- Outros.

Heurísticas Gulosas

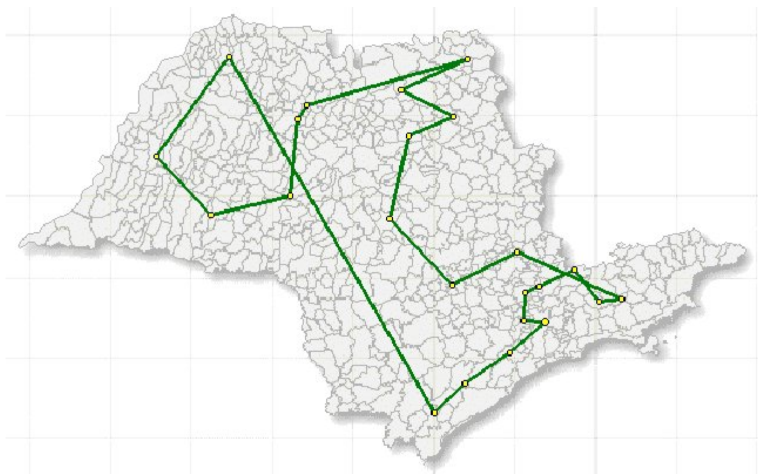
- Heuristicas Construtivas
- A cada passo procuram estender a solução pelo menor custo (se minimização)

Vizinho Mais Próximo - Bellmore e Nemhauser

TSP-VIZINHO-MAIS-PRÓXIMO (G, V, E, c), G é completo

1 Escolha vértice inicial $v \in V$. Faça $C \leftarrow (v)$.
2 Repita $n - 1$ vezes
3   Seja $C = (v_1, \ldots, v_i)$
4   Escolha um vértice u mais próximo de um dos extremos de C.
5   Acrescente u na respectiva extremidade.
6 devolva C

Exemplo de solução obtida pelo TSP-Vizinho-Mais-Próximo



Heurística de Inserção Mais Barata

TSP-INSERÇÃO-MAIS-BARATA (G, V, E, c), G é completo

1 Seja C um circuito qualquer (e.g. fecho convexo).

2 Enquanto C não é hamiltoniano

3 Seja $(u, v) \in C$ e $x \notin C$ tal que $c(u, x) + c(x, v) - c(u, v)$ é mínimo

4 $C \leftarrow C - (u, v) + (u, x) + (x, v)$.

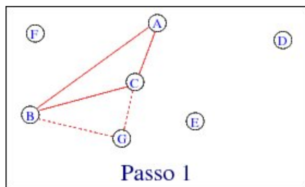
5 devolva C

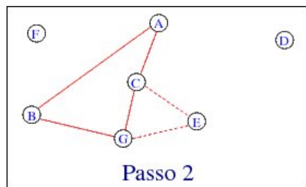
Por ser uma heurística útil e fácil de implementar, foi analisada com mais detalhes.

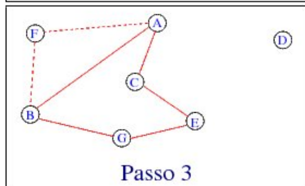
Teorema: *Rosenkrantz, Stearns, Lewis: TSP-Vizinho-Mais-Próximo é uma 2-aproximação para grafos métricos.*

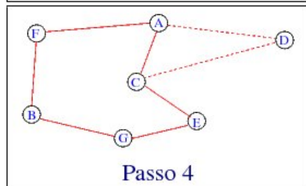
Prova. Exercício. □

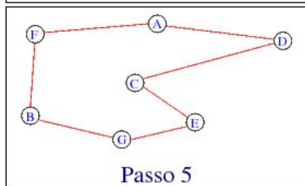
Simulação do TSP-Inserção-mais-barata



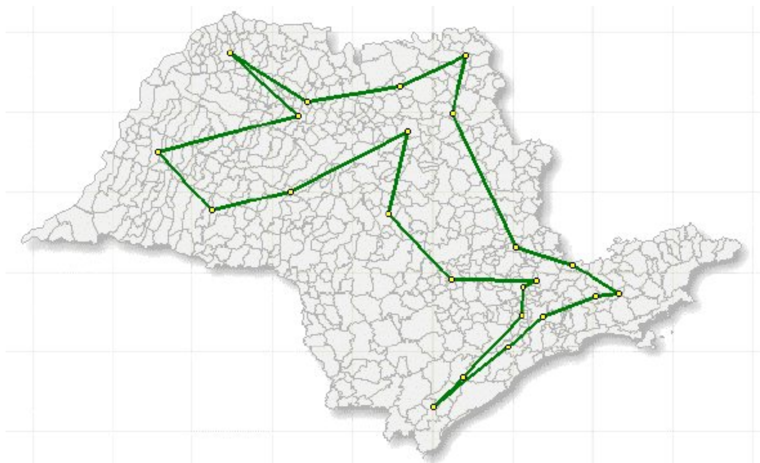








Exemplo de solução obtida pelo TSP-Inserção-mais-barata



Heurística de Inserção do Mais Distante

TSP-INSERÇÃO-MAIS-DISTANTE (G, V, E, c), G é completo

1 Seja C um circuito qualquer (e.g. fecho convexo).
2 Enquanto C não é hamiltoniano
3 Denote por $c(v, C)$ a menor distância de v a um vértice de C.
4 Seja $x \notin C$ um vértice tal que $c(x, C)$ é máxima.
5 Insira x em C na posição onde há menor aumento de custo
6 devolva C

Esta heurística dá resultados melhores na prática que o TSP-Inserção-mais-barata. Mas não há prova de sua aproximação. O pior caso são fatores de 6.5 em uma métrica e de 2.43 no plano euclidiano [Hurkens'92].

Problema MOCHILA: Dados itens $S = \{1, \ldots, n\}$ com valor v_i e tamanho s_i inteiros, $i = 1, \ldots, n$, e inteiro B, encontrar $S' \subseteq S$ que maximiza $\sum_{i \in S'} v_i$ tal que $\sum_{i \in S'} s_i \leq B$.

MOCHILA-GULOSO

1 Ordenar itens de S tal que $\frac{v_1}{s_1} \geq \frac{v_2}{s_2} \geq \cdots \geq \frac{v_n}{s_n}$

2 $S \leftarrow \emptyset$

3 Para $i \leftarrow 1$ até n faça

4   Se $s_i + \sum_{j \in S} s_j \leq B$

5   então $S \leftarrow S \cup \{i\}$

6 devolva S

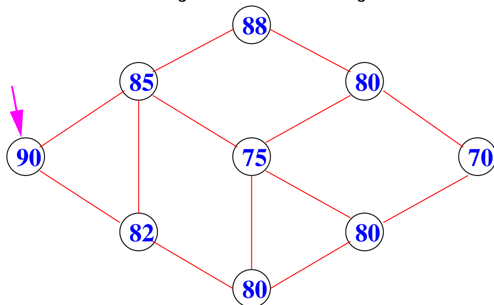
Exercício: Construa uma instância arbitrariamente ruim para MOCHILA-GULOSO

Busca Local

- Começam com uma atribuição ou solução
- Iterativamente fazem operações locais melhorando a solução anterior
- Quando não puder melhorar, devolvem a solução obtida

Grafo de Vizinhanças

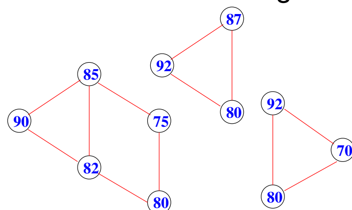
- Vértices são “soluções” viáveis: em geral é um conjunto muito grande
- Arestas indicam transformações de uma solução para outra
- Exemplo de grafo de vizinhança e uma solução inicial (e seu valor)



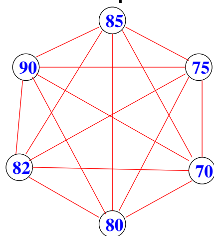
- Objetivo: Chegar na solução de melhor custo pelo grafo de vizinhanças

Grafos de vizinhança ruins

- Grafo desconexo:Podemos nunca chegar na solução ótima



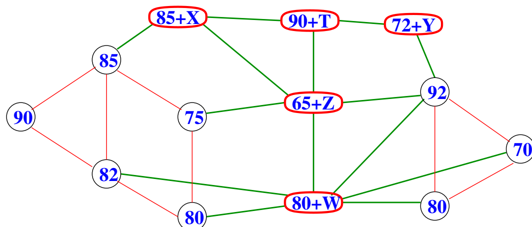
- Grafo denso: Percorrer os vizinhos
  de uma “solução” tem a mesma complexidade do problema original



Aumentando a conectividade do grafo de vizinhança

Idéia: Inserir nós inviáveis e considerar um custo adicional conforme o grau de inviabilidade

- Aumenta possibilidades de sair de uma solução para outras
- Possibilidades de sair de mínimos locais por soluções inviávies
- Soluções iniciais podem ser inviáveis



- Seja $\mathcal{S}$ conjunto das soluções viáveis do problema
- $c(S)$ é o custo de $S + p(S)$, onde $p(S)$ é uma penalidade de acordo com o 'grau de inviabilidade' de S

Heurísticas de Busca Local e Hill Climbing

- Uso de vizinhança entre soluções viáveis
- Uso de solução inicial
- Melhorias sucessivas a partir da solução atual

Notação:

I = Instância do problema

$\mathcal{N}$ = Conjunto de soluções viáveis para I

$\mathcal{N}(S)$ = Conjunto de soluções vizinhas a S (no grafo de vizinhanças)

$c(S)$ = valor da solução S

Hill Climbing

VIZINHO-MELHOR (S, I)

1 se existe vizinho $S' \in \mathcal{N}(S)$ com valor melhor que S
2   então retorne S'
3   senão retorne $\emptyset$

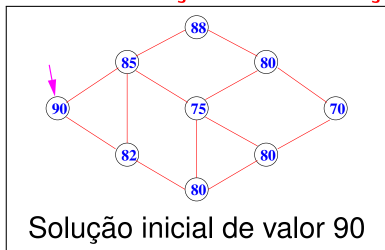
BUSCA-LOCAL-GERAL (I)

1 encontre “solução” inicial $S \in \mathcal{N}$ para I
2 $S' \leftarrow$ VIZINHO-MELHOR(S, I)
3 enquanto $S' \neq \emptyset$ faça
4     $S \leftarrow S'$
5     $S' \leftarrow$ VIZINHO-MELHOR(S, I)
6 devolva S

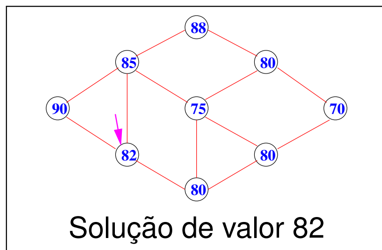
Busca Local Geral (minimização):

Escolher o melhor dentre todos os vizinhos que tem valor melhor

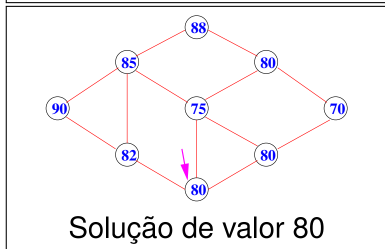
Grafo de vizinhança entre "soluções" viáveis



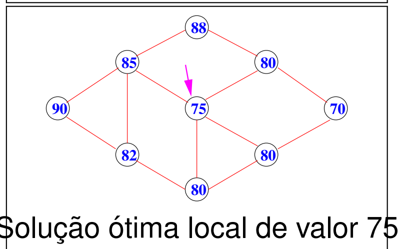
Solução inicial de valor 90



Solução de valor 82

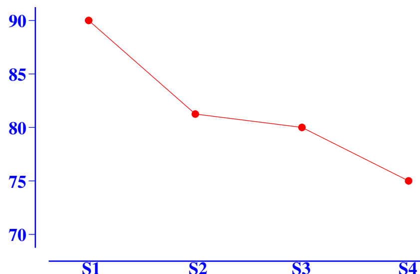


Solução de valor 80

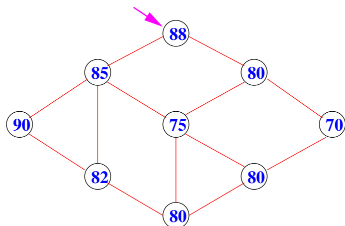


Solução ótima local de valor 75

Comportamento do algoritmo de busca local

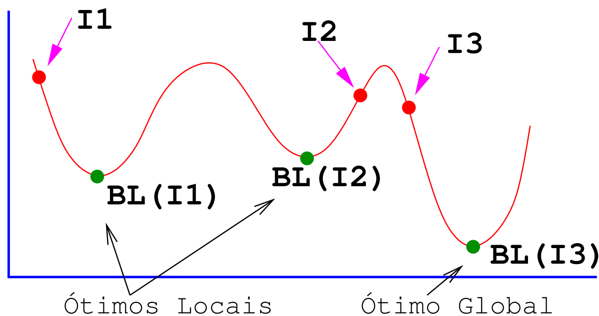


Se a primeira solução fosse a de valor 88, teríamos chegado na solução ótima

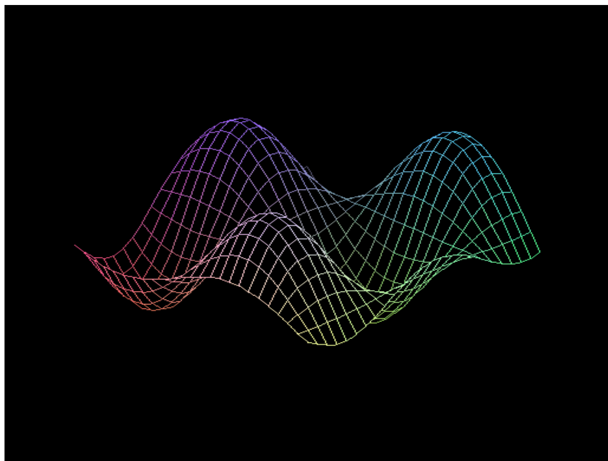


Saindo de mínimos locais: Multi-Start Local Search

- **Executar algoritmo de Busca Local com diferentes inícios**
- **Guardar melhor solução**
- **Exemplo para minimização (unidimensional)**



Mínimos e máximos locais em função bidimensional



Busca Local para o TSP

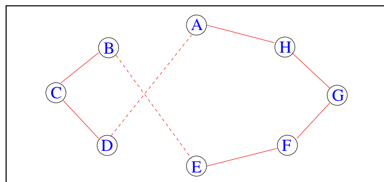
Considere grafos completos

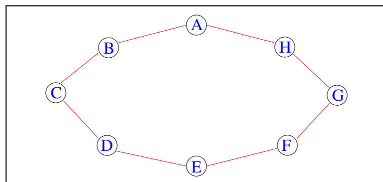
Vizinhança-K-OPT(C) $:= \{C' : C'$ é circuito hamiltoniano obtido de C removendo K arestas e inserindo outras K arestas.$\}$.

K-OPT $(G = (V, E, c))$

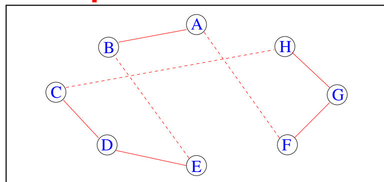
1 encontre um circuito hamiltoniano inicial C
2 repita
3 procure C' em Vizinhança-K-OPT(C) tal que $\mathrm{val}(C') < \mathrm{val}(C)$.
4 se encontrou tal C', $C \leftarrow C'$
5 até não conseguir encontrar tal C' no passo 3
6 devolva C

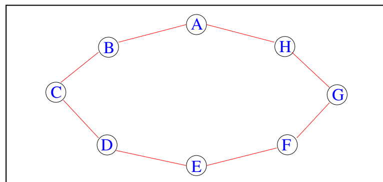
Exemplo de troca 2-OPT





Exemplo de troca 3-OPT





Uma solução viável pode ter vários vizinhos usando troca 3-OPT.
Quantos ?

Comparação em grafos euclidianos aleatórios

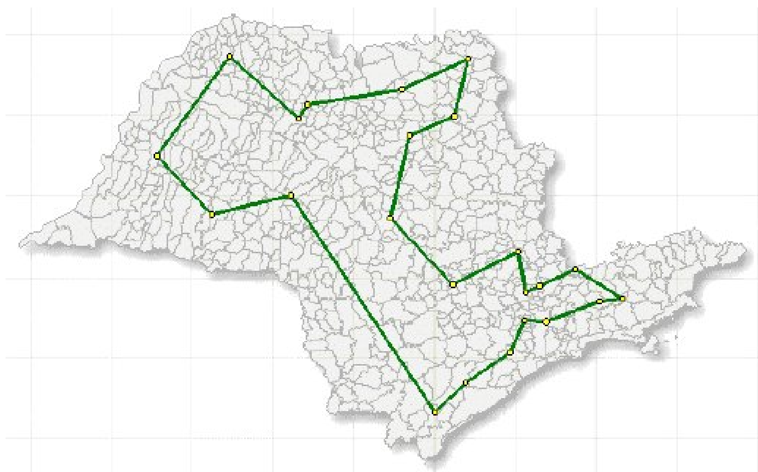
D.S. Johnson & L.A. McGeoch'97

- **Circuito inicial por algoritmo guloso (estilo Kruskal)**
  - Inserindo arestas mais leves primeiro
  - Descartando arestas que inviabilizam solução
- **Comparando com limitante inferior do ótimo**
  - Limitante de Held-Karp

$N =$	10^2	$10^{2.5}$	10^3	$10^{3.5}$	10^4	$10^{4.5}$	10^5	$10^{5.5}$	10^6
Guloso	19.5	18.8	17.0	16.8	16.6	14.7	14.9	14.5	14.2
2-OPT	4.5	4.8	4.9	4.9	5.0	4.8	4.9	4.8	4.9
3-OPT	2.5	2.5	3.1	3.0	3.0	2.9	3.0	2.9	3.0

Fator de excesso em relação ao limitante de Held-Karp

Exemplo de solução obtida pelo TSP-2-OPT



Complexidade de se encontrar Ótimos Locais

Problema de Otimização

- *Problema de Otimização Combinatória Π:*
- Dados
  - I: conjunto de instâncias
  - $F(x)$: conjunto de soluções para cada $x \in I$
  - $c(s)$: função de custo para todo $s \in F(x)$
  - funções eficientes que verificam instâncias, soluções,...
- dado $x \in I$
  - encontrar solução $s \in F_\Pi(x)$ tal que $c_x(s)$ é mínimo
- A versão de maximização é análoga.
- Exemplo:

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \ \wedge \ (x_1 \vee \overline{x_2} \vee \overline{x_3}) \ \wedge \ (\overline{x_2} \vee x_3)$$

- **MaxSat** Dada fórmula booleana ϕ em FNC encontrar atribuição maximiza o número de cláusulas verdadeiras.

Problema de Otimização Local

Um *problema de otimização local* é um problema de otimização onde

- há uma vizinhança $N_x(s) \subset F(x)$ para cada $x \in I$ e $s \in F(x)$
- e uma solução s de $F(x)$ é um mínimo local se $c_x(s) \leq c_x(s')$ para todo $s' \in N_x(s)$.

O objetivo é encontrar uma solução que é mínimo local.

Um problema de otimização local pertence à *PLS* se temos um oráculo que, para qualquer instância $x \in I$ e solução $s \in F(x)$, decide se s é ótimo local, e se não for, devolve $s' \in N_x(s)$ com $c_x(s') < c_x(s)$.

Classe PLS *(Polynomial-time Local Search)*

(Johnson, Papadimtriou, Yannakakis'88)

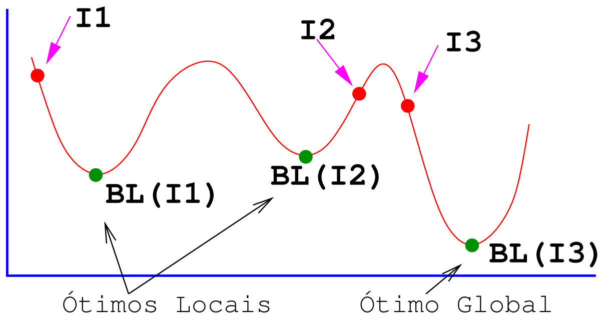
- Para problemas de otimização com grafo de vizinhança entre soluções viáveis
- Exemplo: MaxSat Dada fórmula booleana ϕ em FNC encontrar atribuição maximiza o número de cláusulas verdadeiras.

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \ \wedge \ (x_1 \vee \overline{x_2} \vee \overline{x_3}) \ \wedge \ (\overline{x_2} \vee x_3)$$

- Vizinhança Flip para MaxSat:
  - Uma atribuição A é vizinha de B se
  - B é a atribuição A trocando o valor de uma das variáveis
  - $A = (x_1 = \mathtt{V}, \quad x_2 = \mathtt{F}, \quad x_3 = \mathtt{V})$
  - $B = (x_1 = \mathtt{F}, \quad x_2 = \mathtt{F}, \quad x_3 = \mathtt{V})$
  - A e B são vizinhas

Problema de Otimização Local

- Grafo de vizinhanças G entre soluções
- Encontrar uma solução ótima local s tal que $c(s) \leq c(s')$ para toda solução s' que é vizinha de s



PLS-redução: Q tem uma PLS-redução para P se

- há funções eficientes que mapeiam soluções de Q para P
- mapeamento preserva soluções ótimas locais

P é PLS-*completo* se está em PLS e há uma PLS-redução de Q para P, para todo $Q \in$ PLS

Tempo de Convergência em Jogos Puros

Teorema: *Os seguintes problemas são* PLS*-completos*

- Min EqPartição de Grafos - vizinhança Kerninghan-Lin (Johnson, Papadimitriou, Yannakakis'88)
- TSP - vizinhança: *k*-OPT (Krentel'89)
- TSP - vizinhança: Lin&Kerninghan (Papadimitriou'90)
- Max2Sat c/ pesos - vizinhança: Flip (Schaffer e Yannakakis'91)
- MaxCut c/ pesos - vizinhança: migração de um vértice (Schaffer e Yannakakis'91)

Se existir um algoritmo eficiente que obtém um ótimo local para um deles, haverá para todos problemas de otimização local PLS

Proposição: *Se existir um algoritmo eficiente para um dos problemas* PLS*-completos, então haverá um algoritmo para o Método Simplex de Programação Linear com complexidade de tempo polinomial.*

Equilíbrio de Nash e Busca Local

- Internet: Rede gigantesca com grande quantidade de usuários e complexa estrutura sócio-econômica
- Usuários podem ser competitivos, cooperativos,...
- Situações envolvendo Teoria dos Jogos e Computação

Ref.: Cap. 12 - Local Search do livro *Algorithm Design* de Kleinberg e Tardos

Um jogo Multicast

- Jogadores podem construir links entre nós
- Há um nó origem
- Cada jogador representa um nó destino
- Cada jogador quer conectar o nó origem até seu nó destino
- Há cooperação na construção da rede. Isto é, o custo de um linké dividido igualmente entre os usuários que o utilizam

Definição

Dados

- Grafo direcionado $G = (V, E)$
- Custo positivo c_e para cada aresta e.
- Vértice fonte s
- k vértices destinos $t_1, \ldots, t_k$

Cada usuário i procura encontrar

- caminho orientado P_i do vértice s até t_i pagando menos

Custo para

- usuário i é $c(P_i) = \sum_{e \in P_i} \frac{c_e}{k_e}$, onde k_e número de caminhos usando e
- sistema é $c(P_1, \ldots, P_k) = \sum_i c(P_i)$ (custo social)

Jogo

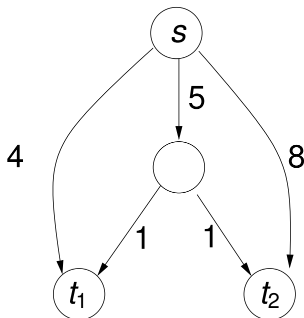
Regras do Jogo:

- Cada usuário fica estável ou muda sua rota (pagando menos) baseado apenas na configuração atual
- Em um estado do jogo com caminhos $(P_1, \ldots, P_k)$, denotamos por $E^+ \subseteq E$ as arestas usadas em pelo menos um caminho.
- O custo social é o custo dos caminhos escolhidos pelos jogadores:

$$c(P_1, \ldots, P_k) = \sum_{i=1}^{k} c(P_i) = \sum_{e \in E^+} c_e$$

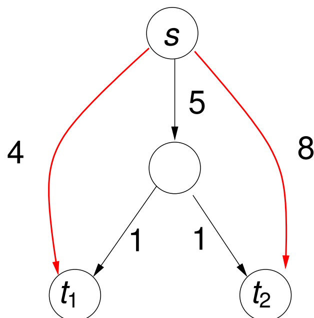
Exemplo 1

- Temos dois jogadores: 1 e 2
- Cada um tem duas alternativas: uma rota externa e uma interna.



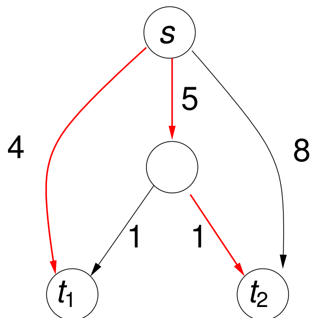
Exemplo 1

- Considere que inicialmente os jogadores usam as rotas externas.
- O jogador 1 paga 4 e o jogador 2 paga 8
- O custo social é igual a 12.



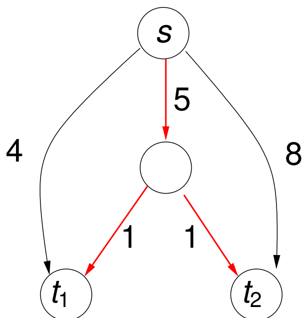
Exemplo 1

- O jogador 2 muda para a rota interna e seu custo cai para 6
- O custo social cai para 10



Exemplo 1

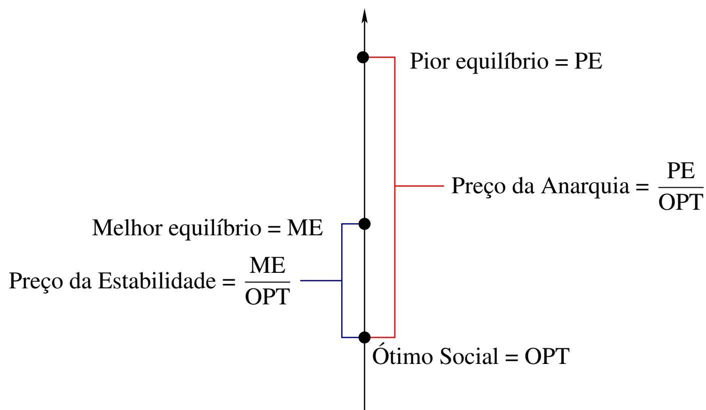
- O jogador 1 tem incentivo a mudar
- Cada jogador paga $2{,}5 + 1$, e estamos em um equilíbrio
- O custo social cai para 7 (solução final também é ótima)



Definições e Notação

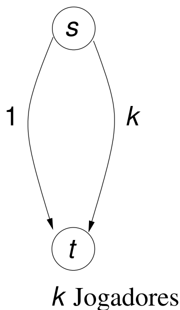
- A Estratégia do jogador i é o conjunto de rotas de s para t_i
- O estado do jogo em um momento é dado pelos k caminhos $(P_1, \ldots, P_k)$ no momento
- O ótimo social é o menor valor possível de uma solução (dos k caminhos), possivelmente não está em equilíbrio.
- Um usuário i está insatisfeito no estado atual, se ele pode mudar sua rota por outra de custo melhor
- Estado em Equilíbrio de Nash quando não há usuários insatisfeitos
- O Preço da Estabilidade razão entre a melhor solução em equilíbrio com o ótimo social
- O Preço da Anarquia razão entre a pior solução em equilíbrio com o ótimo social
- Melhor resposta: movimento para estratégia de maior ganho positivo

Preços da anarquia e do equilíbrio para minimização



Exemplo 2

- Na rede abaixo há k jogadores todos com mesmo destino t
- Considere todos usando a aresta da direita
- Estamos em um equilíbrio com custo k (cada jogador paga 1).
- O ótimo social tem custo 1 (cada jogador paga $\frac{1}{k}$).



Teorema: *O preço da anarquia deste jogo Multicast é k.*

Método da Função Potencial e o Preço da Estabilidade

Def.: *Uma* função potencial exata Φ *é uma função que*

- *mapeia cada vetor de estratégia* $\mathcal{P}$ *para um valor real tal que*
- *se* $\mathcal{P} = (P_1, \ldots, P_i, \ldots, P_k)$ *e*

  $P'_i \neq P_i$ *é uma estratégia alternativa para o jogador* i*, então*

$$\Phi(\mathcal{P}) - \Phi(\mathcal{P}') = c_i(P_i) - c_i(P'_i),$$

  onde $\mathcal{P}' = (P_1, \ldots, P'_i, \ldots, P_k)$

Fato: *Seja* Φ *uma função potencial exata para o jogo do Multicast com dinâmica de melhor resposta. Se jogador* i *muda sua estratégia de* P_i *para* P'_i*, e o vetor de estratégia muda de* $\mathcal{P}$ *para* $\mathcal{P}'$*, então*

$$\Phi(\mathcal{P}) > \Phi(\mathcal{P}').$$

Isto é, Φ *é estritamente decrescente após jogadas.*

Função Potencial para Multicast

Dado vetor de estratégias $\mathcal{P} = (P_1, \ldots, P_k)$, denote por

$$\Phi(\mathcal{P}) = \sum_e c_e \cdot H(k_e),$$

onde

$$H(t) = 1 + \frac{1}{2} + \cdots + \frac{1}{t} \quad \text{e} \quad H(0) = 0$$

k_e é o número de caminhos de $\mathcal{P}$ que usam e

Lema: *Φ é uma função potencial exata.*
Prova. Exercício □

Fato: *$\Phi(\mathcal{P})$ é limitado inferiormente.*
Prova. Exercício □

Lema: *O jogo Multicast com a dinâmica de melhor resposta converge para um equilíbrio de Nash.*
Prova. Exercício □

Preço da Estabilidade

Lema: *Se $\mathcal{P} = (P_1, \ldots, P_k)$ é um vetor de estratégia, então*

$$c(\mathcal{P}) \leq \Phi(\mathcal{P}) \leq H(k)c(\mathcal{P}).$$

Teorema: *O preço da estabilidade do jogo Multicast é no máximo $H(k)$.*

Prova. Seja:

OPT um vetor de estratégia ótimo (ótimo social)

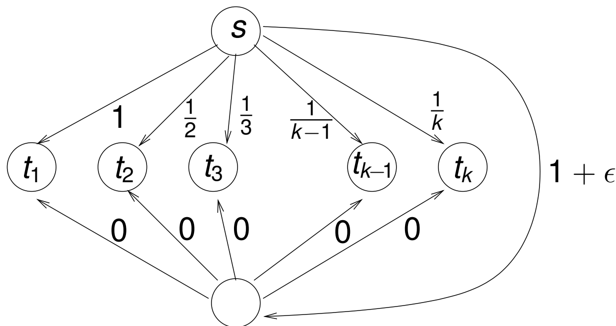
$\mathcal{O}$ um vetor de estratégia em equilíbrio obtido a partir de OPT

$\mathcal{P}$ um vetor de estratégia em Equilíbrio de Nash de menor custo

$$\begin{aligned} c(\mathcal{P}) &\leq c(\mathcal{O}) \\ &\leq \Phi(\mathcal{O}) \\ &\leq \Phi(\mathrm{OPT}) \\ &\leq H(k) \cdot c(\mathrm{OPT}) \end{aligned}$$

Lema: *O preço da estabilidade $H(k)$ do problema de Multicast é justo (melhor possível).*

Prova.



Observações

- Tempo de convergência do problema Multicast pode ser exponencial
- Encontrar ótimo social é um problema NP-difícil.

Exercício: Mostre que encontrar o ótimo social do problema Multicast é um problema NP-difícil. Sugestão: por Cobertura por Conjuntos.

Complexidade de se encontrar Equilíbrio de Nash em Jogos Potenciais

O quão difícil é encontrar um algoritmo polinomial para encontrar um equilíbrio de Nash ?

Teorema: *(Fabrikant, Papadimitriou, Talwar'04) O problema de se encontrar um equilíbrio puro de Nash em jogos potenciais, onde a melhor resposta é computada em tempo polinomial, é PLS-completo.*

Algoritmo Metropolis
Metropolis, Rosenbluth, Rosenbluth, Teller, Teller'53

- Simula um sistema físico, de acordo com princípios da mecânica estatística.
- Baseado em passos de buscas locais.
- Passos buscando melhorar a solução, mas pode obter soluções piores para sair de ótimos locais, com alguma probabilidade.
- Usa conceito de temperatura e estados de energia (Gibbs-Boltzmann).

Sem perda de generalidade, vamos supor problemas de minimização

Função de Gibbs-Boltzmann

- A probabilidade de encontrar um sistema físico em um estado de energia E é proporcional a

$$e^{-\frac{E}{kT}},$$

  onde $T > 0$ é uma temperatura e k é uma constante.

- Para qualquer temperatura $T > 0$, a função é monotonicamente decrescente em E.

- Sistema tende a estar em um estado de baixa energia

  - T grande: Estados de energia alta e baixa tem basicamente mesma chance

  - T pequeno: favorece estados de baixa energia

Algoritmo Metropolis

Notação:

$\mathcal{N}$ = Conjunto de estados (no grafo de vizinhanças)

$\mathcal{N}(S)$ = Conjunto de estados vizinhos a S (no grafo de vizinhanças)

$E(S)$ = nível de energia do estado S (custo de uma solução)

k = Parâmetro para calibrar estados (soluções)

T = Temperatura (controla busca por vizinhos piores/melhores)

- Considera temperatura T fixa
- Mantém um estado corrente S durante a simulação
- Para cada iteração, obtém novo estado $S' \in \mathcal{N}(S)$
- Se $E(S') \leq E(S)$, atualiza estado corrente para S'
- Caso contrário, atualiza estado corrente com probabilidade $e^{-\frac{\Delta E}{kT}}$, onde $\Delta E = E(S') - E(S) > 0$.

Algoritmo Metropolis

ALGORITMO METROPOLIS - MINIMIZAÇÃO

1. Seja k e T (temperatura) parâmetros para calibrar problema
2. Obtenha uma solução inicial S
3. $S^+ \leftarrow S$ (mantém a melhor solução obtida)
4. Enquanto não atingir condição de parada, faça
5. Seja $S' \in \mathcal{N}(S)$ solução vizinha escolhida aleatoriamente
6. $\Delta S \leftarrow c(S') - c(S)$
7. Se $\Delta S \leq 0$ então
8. $S \leftarrow S'$
9. Se $c(S) < c(S^+)$ faça $S^+ \leftarrow S$
10. senão
11. com probabilidade $e^{-\frac{\Delta S}{kT}}$ faça $S \leftarrow S'$
12. Devolva S^+

Algoritmo Metropolis

Teorema:
Seja $f_S(t)$ a fração das t primeiras iterações onde a simulação percorre S. Então, assumindo algumas condições técnicas, com probabilidade 1, temos

$$\lim_{t \to \infty} f_S(t) = \frac{1}{Z} e^{-\frac{E(S)}{kT}},$$

onde

$$Z = \sum_{S \in \mathcal{N}} e^{-\frac{E(S)}{kT}}.$$

Intuição: A simulação gasta basicamente o tempo correto em cada estado, de acordo com a equação de Gibbs-Boltzmann.

Simulated Annealing

- T grande $\Rightarrow$ probabilidade de aceitar uma solução pior é grande (movimentos de subida do morro)
- T pequeno $\Rightarrow$ probabilidade de aceitar uma solução pior é pequena.

Analogia física: Não é esperado obter um sólido com estrutura cristalográfica boa se

- o colocarmos em alta temperatura
- estiver quente e congelarmos abruptamente

Annealing: Cozimento de material em alta temperatura, permitindo obter equilíbrio gradual sucessivo em temperaturas menores.

Idéia: Controlar o algoritmo Metropolis variando a temperatura

Simulated Annealing - Simplificado

SIMULATED ANNEALING - MINIMIZAÇÃO

1. Faça $T \leftarrow T_0$ (temperatura inicial)
2. Seja S solução inicial
3. $S^+ \leftarrow S$ (mantém a melhor solução obtida)
4. Enquanto não atingiu condição de parada faça (loop externo)
5. Enquanto não atingir condição de parada, faça (loop interno)
6. $S' \leftarrow$ Obtenha-Solução-Vizinha(S)
7. $\Delta S \leftarrow c(S') - c(S)$
8. Se $\Delta S \leq 0$ então
9. $S \leftarrow S'$
10. Se $c(S) < c(S^+)$ faça $S^+ \leftarrow S$
11. senão
12. com probabilidade $e^{-\frac{\Delta S}{kT}}$ faça $S \leftarrow S'$
13. Atualiza-Temperatura(T)
14. Devolva S^+

Simulated Annealing

Possíveis implementações das subrotinas:

OBTENHA-SOLUÇÃO-VIZINHA(S)

- Aleatória: Escolha vizinho $S' \in \mathcal{N}(S)$ aleatoriamente
- Vizinho-Melhor: Escolha $S' \in \mathcal{N}(S)$ com $c(S')$ mínimo

ATUALIZA-TEMPERATURA(T)

- Esfriamento geométrico:
- Faça $T \leftarrow \alpha \cdot T$, para parâmetro $0 < \alpha < 1$.

Simulated Annealing

Possíveis Critérios de parada no loop externo

- Seja T_{parada} uma temperatura final
- Condição atingida se $T_n \leq T_{parada}$

Possíveis Critérios de parada no loop interno

- Seja N um inteiro positivo
- Condição atingida se número de iterações seguidas sem melhorar solução atingiu N

Exemplo: TSP (D.S. Johnson e L.A. McGeoch'97)

S.A. com 2-OPT (Simulated Annealing com 2-OPT):

- Vizinhança 2-OPT
- Temperatura decrescendo de maneira geométrica ($\alpha = 0,95$)
- Temperaturas não calculadas de maneira exata (pelo exponencial), mas aproximadas por valores em tabela.
- Média de 10 execuções para $N = 100$ e de 5 execuções para N maiores.

$N =$	10^2	$10^{2.5}$	10^3
Apenas 2-OPT	4.5	4.8	4.9
S.A. com 2-OPT	5.2	4.1	4.1
S.A. com 2-OPT + Pós 2-OPT	3.4	3.7	4.0
S.A. sofisticado*	1.1	1.3	1.6

Excesso em relação ao limitante de Held-Karp

S.A. com 2-OPT + Pós 2-OPT = S.A. com 2-OPT aplicando 2-OPT novamente no tour gerado.

* Veja mais detalhes em Johnson & McGeoch'97.

Simulated Annealing

Exercícios faça algoritmos Simulated Annealing para os seguintes problemas:

1. Caixeiro Viajante: TSP
2. Corte de Peso Máximo: MaxCut
3. Satisfatibilidade de Peso Máximo: MaxSat
4. Subárvore de peso mínimo com exatamente k arestas

Introdução à Busca Tabu

- Baseado em Busca Local
- A cada iteração, procura nova solução vizinha, preferencialmente de custo melhor
- Sempre aceita uma nova solução que tiver custo melhor
- Cada solução é formada por elementos
- Tenta escapar de mínimos locais, proibindo alterações nos elementos afetados nas últimas k iterações
- Guarda a melhor solução encontrada durante sua execução

Busca Tabu

Notação e Termos:

k = Número das últimas iterações na memória da Busca Tabu

$T(k)$ = Lista de movimentos proibidos, Lista Tabu, considerando últimas k iterações.

$\mathcal{N}(S)$ = Conjunto de soluções vizinhas de S

- O Critério de Aspiração permite aceitar uma solução baseado em sua qualidade, mesmo que tal solução faça movimentos da lista tabu.

BUSCA TABU SIMPLIFICADA - MINIMIZAÇÃO

1. Seja S uma solução inicial
2. $S^+ \leftarrow S$ (mantém atualizado a melhor solução encontrada)
3. Enquanto não atingir condição de parada faça
4. Escolha soluções candidatas $V \subseteq \mathcal{N}(S)$
5. Enquanto $V \neq \emptyset$ faça
6. Pegue $S' \in V$ de peso mínimo e faça $V \leftarrow V \setminus \{S'\}$
7. Se $c(S') \leq c(S^+)$ então // (critério de aspiração)
8. $S^+ \leftarrow S'$, $S \leftarrow S'$ e $V \leftarrow \emptyset$
9. senão
10. se não é Tabu($S' \leftarrow S$) então
11. $S \leftarrow S'$ e $V \leftarrow \emptyset$
12. Devolva S^+

Possíveis implementações das subrotinas:

Condições de parada

- Quando atingir limite de tempo de CPU
- Quando melhor solução não for atualizada por certo tempo

Escolha de soluções candidatas

- Escolha todas soluções vizinhas (se vizinhança for pequena)
- Escolha aleatoriamente um subconjunto dos vizinhos

Possíveis implementações das subrotinas:

CRITÉRIO-ASPIRAÇÃO

- No passo 7, note que mesmo uma solução com movimento Tabu, pode ser escolhida. Este é o critério de aspiração mais comum.

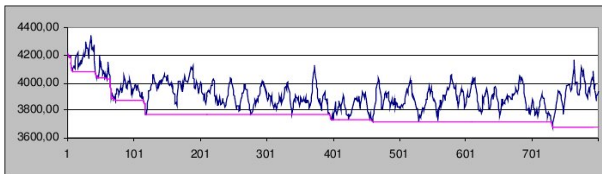
TABU($S' \leftarrow S$)

- Considere cada solução formada por elementos E
- Cada iteração do algoritmo de Busca Tabu representa um tempo
- Cada elemento $e \in E$ possui um marcador de tempo t_e (inicialmente $-\infty$)
- t_e é o último momento que e entrou ou saiu de uma solução
- Se estamos no momento t e for inserido/removido elemento e tempo t_e tal que $t - t_e \leq k$, então movimento é tabu.

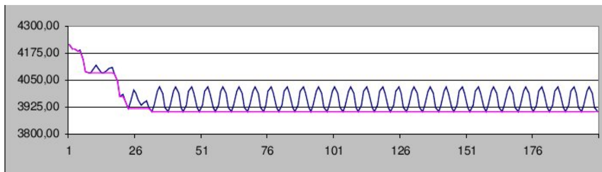
Valores adequados para k

Depende do problema, mas $k = 7$ é um bom ponto de partida

Exemplo de comportamento sem ciclo



Exemplo de comportamento com ciclo ($k = 3$)



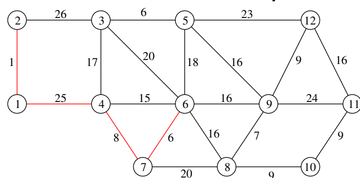
Experimentos realizado por M. Ganhoto para VRP - IC-UNICAMP

Exemplo: k-Árvore

k-Árvore: Dado um grafo $G = (V, E)$ com custo c_e em cada aresta $e \in E$, encontrar uma árvore com k vértices de peso mínimo.

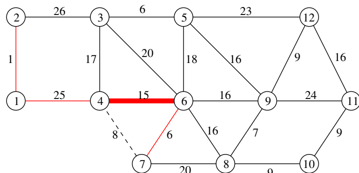
Teorema: *O problema da k-Árvore é um problema NP-difícil.*

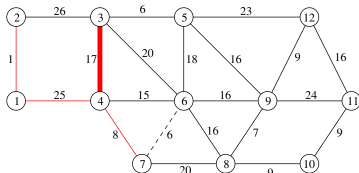
Movimentos:

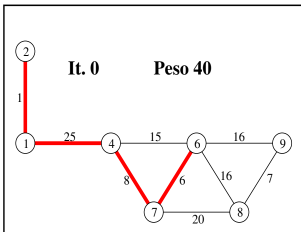


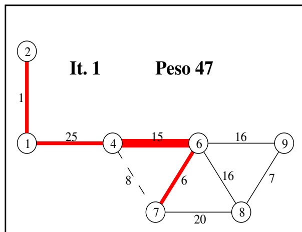
Melhor movimento estatico (mantem conjunto de vertices)

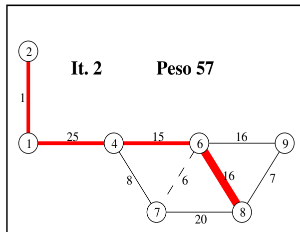
Melhor movimento dinamico

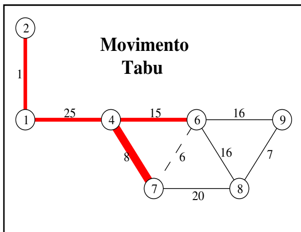




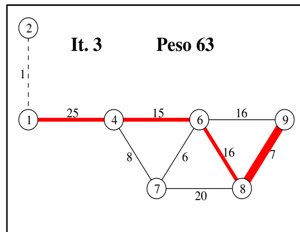






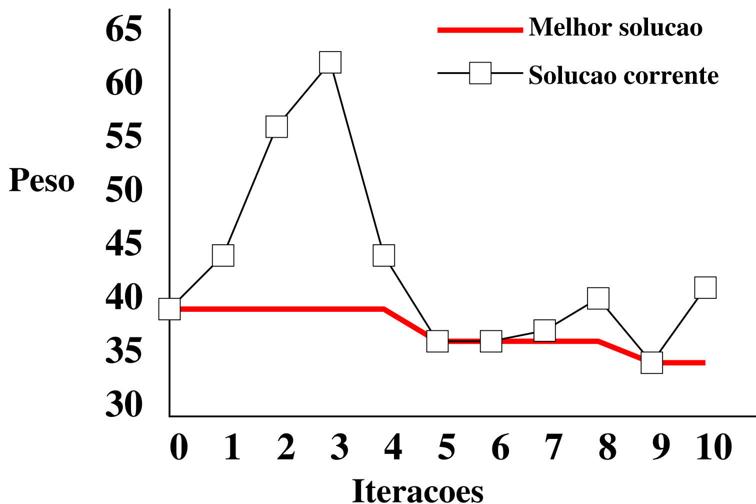


TABU



`Arestas ficam na Lista Tabu por 2 iteracoes`

Comportamento do algoritmo para 10 iterações com melhor movimento possível



Busca Tabu

Exercícios faça algoritmos Busca Tabu para os seguintes problemas:

1. Caixeiro Viajante: TSP
2. Corte de Peso Máximo: MaxCut
3. Satisfatibilidade de Peso Máximo: MaxSat

Algoritmos Genéticos

- Baseado em idéias da Evolução Natural
- Mantém uma população de indivíduos
- Há cruzamento entre indivíduos gerando novos indivíduos
- Pode haver mutação nos indivíduos
- Há seleção dos melhores indivíduos
- No decorrer do tempo, há indivíduos que se destacam

ALGORITMO GENÉTICO SIMPLIFICADO

1. Seja P uma população inicial de indivíduos
2. Repita
3. Obtenha novos indivíduos N a partir de P por
4. Cruzamentos:
5. Selecione pares de indivíduos em P
6. Produza filhos de cada par
7. Mutação:
8. Selecione indivíduos de P
9. Produza filhos de maneira assexuada
10. Atualize P selecionando indivíduos de $P \cup N$
11. Até atingir critério de parada
12. Devolva indivíduo $e \in P$ mais adaptado

Algoritmos Genéticos

Indivíduo	$\leftrightarrow$	Solução para um problema
População	$\leftrightarrow$	Conjunto de soluções
Cromossomo	$\leftrightarrow$	Codificação de uma solução
Aptidão/Fitness	$\leftrightarrow$	Qualidade da solução
Alelo	$\leftrightarrow$	cada elemento que forma a solução

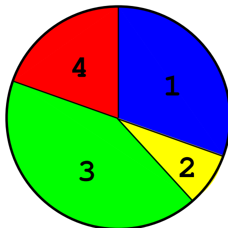
Manutenção da População

- Mantenha a população P com n elementos.
- Mantenha de uma população para outra os n_e melhores indivíduos (elite).
- Use probabilidade para escolher alguns elementos (e.g. método da roleta).
- Remova da população os indivíduos menos adaptados.

Método da Roleta (Roulette Wheel)

Suponha que temos uma população de $n = 4$ indivíduos:

Item	Aptidão/Fitness	Chance de Escolha
1	150	30%
2	50	10%
3	200	40%
4	100	20%
Total	500	100%



A chance de escolher um indivíduo é proporcional à aptidão.

- Para gerar uma nova população com m indivíduos, sorteie (rode a roleta) m vezes (pode dar repetições).

Exemplos de codificação

Vetor de bits Cada bit pode representar um elemento que forma uma solução.

Cromossomo 1: `1111000001110000111`

Cromossomo 2: `0101010101010101010101`

Permutação Usado para problemas com ordem, como o TSP.

Cromossomo 1: `1 4 2 8 5 7 6 9 3`

Cromossomo 2: `5 2 6 9 4 3 1 7 8`

Função de Aptidão/Fitness

- Função que quantifica o quão bom é uma solução (indivíduo) em relação aos outros.
- Em geral, a aptidão é o valor da solução.
- Quanto mais próximo da solução ótima, mais apta é a solução.

Seleção para Cruzamento/Crossover ou Mutação

Exemplos:

- Selecione de maneira aleatória.
- Seleção baseada na aptidão/fitness.
- Um mesmo indivíduo pode estar em vários cruzamentos ou várias mutações.

Cruzamento/Crossover

Cruzamento em 1 ponto:

```
Pai 1:      11111|111111111

Pai 2:      00000|000000000
```

Produzem os filhos:

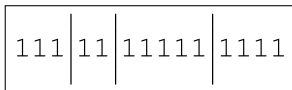
```
Filho 1:    11111|000000000

Filho 2:    00000|111111111
```

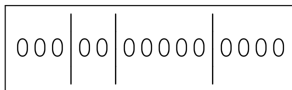
Cruzamento/Crossover

Cruzamento em k pontos:

Pai 1:

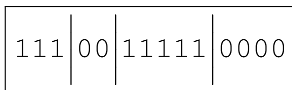


Pai 2:

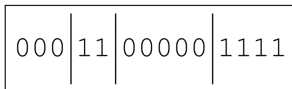


Produzem os filhos:

Filho 1:



Filho 2:



Os k pontos podem ser gerados aleatoriamente. Um filho pode ser gerado de vários pais.

Mutação

- Gera variabilidade na população, principalmente com aleatoriedade
- Em geral uma mutação é obtida por pequenas mudanças no indivíduo original

Por exemplo: Sortear posições no vetor de bits e trocar seus valores

Indivíduo:	10 **1** 1101 **0** 01011 **1** 110
Mutação:	10 **0** 1101 **1** 01011 **0** 110

Exemplo: $\max x^2$ *para* $x \in \{0, 1, \ldots, 31\}$

Codificação com vetor de 5 bits. Indivíduo = 01101 ($x = 13$)

População de tamanho $n = 4$

Cruzamento com 1 ponto

Seleção por roleta

Inicialização aleatória

Exemplo: max x^2 *para* $x \in \{0, 1, \ldots, 31\}$

Seleção de indivíduos

No. do Indivíduo	População Inicial	Valor de x	Aptidão $f(x)$	Probab. na roleta	No. Vezes selecionado
1	01101	13	169	0,14	1
2	11000	24	576	0,49	2
3	01000	8	64	0,06	0
4	10011	19	361	0,31	1
Soma			1170	1	4
Média			293	0,25	1
Máximo			576	0,49	2

Exemplo: max x^2 *para* $x \in \{0, 1, \ldots, 31\}$

Cruzamento

No. do Indivíduo	Pares p/ Cruzamento	Ponto de Cruzamento	Filhos	x	Aptidão $f(x) = x^2$
1	0110 | 1	4	01100	12	144
2	1100 | 0	4	11001	25	625
2	11 | 000	2	11011	27	729
4	10 | 011	2	10000	16	256
Soma				1	1754
Média					439
Máximo					729

Exemplo: max x^2 *para* $x \in \{0, 1, \ldots, 31\}$

Mutação

No. do Indivíduo	Antes Mutação	Após Mutação	x	Aptidão $f(x) = x^2$
1	**0**1101	**1**1100	28	784
2	11000	11001	25	625
2	11000	11011	27	729
4	10**0**11	10**1**11	23	529
Soma				2667
Média				666,75
Máximo				784

Exemplo: Caixeiro Viajante

Codificaçao

- Codificação usando ordem/seqüência
- Conjunto de cidades: $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Cada rota é uma permutação

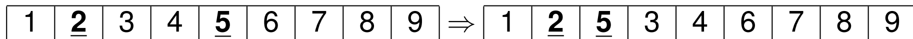
Exemplo de rota:

1	2	3	4	5	6	7	8	9

Exemplo: Caixeiro Viajante

Exemplo de Mutação: Migração

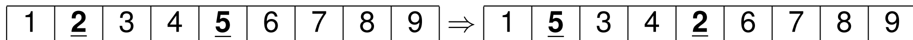
- Pegue dois alelos e coloque o segundo na seqüência do primeiro
- Esta mutação é pontual e preserva a ordem da maioria dos alelos



Exemplo: Caixeiro Viajante

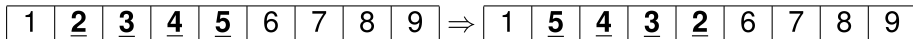
Exemplo de Mutação: Troca

- Pegue dois alelos e troque a posição deles



Exemplo de Mutação: Inversão

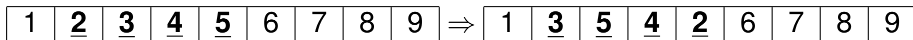
- Pegue um trecho e inverta a seqüência do trecho



Exemplo: Caixeiro Viajante

Exemplo de Mutação: Rearranjo aleatório de trecho

- Pegue um trecho e faça um rearranjo aleatório de trecho



Exemplo: Caixeiro Viajante

Exemplo de Cruzamento 1:

- Copie um trecho de um pai para o filho.

1	2	3	**4**	**5**	**6**	**7**	8	9
9	3	7	**8**	**2**	**6**	**5**	1	4

$\Rightarrow$

			4	5	6	7		

- Complete os demais elementos a partir do trecho, com elementos do segundo pai na ordem em que aparecem e sem considerar os já inseridos

1	2	3	**4**	**5**	**6**	**7**	8	9
9	3	7	**8**	**2**	**6**	**5**	1	4

$\Rightarrow$

3	8	2	4	5	6	7	1	9

Algoritmos Genéticos

Exercícios faça algoritmos genéticos para os seguintes problemas:

1. Caixeiro Viajante: TSP (faça mais rotinas para mutação, cruzamentos, etc)
2. Corte de Peso Máximo: MaxCut
3. Satisfatibilidade de Peso Máximo: MaxSat

GRASP
Greedy Randomized Adaptive Search Procedures

Algoritmo guloso $\times$ Construção aleatória

- Construção aleatória
  - Soluções com alta variabilidade
  - Baixa qualidade de soluções
- Algoritmo Guloso
  - Soluções de boa qualidade
  - Baixa ou nenhuma variabilidade nas soluções
- GRASP
  - Explorar vantagens das duas estratégias

GRASP

GRASP: Greedy Randomized Adaptive Search Procedures

- Em cada iteração, aplica método de busca local
- Insere aleatoriedade na geração das soluções iniciais
- Multi-Start Local Search + Soluções Iniciais guiadas por processo Guloso-Probabilístico
- Cada solução é formada por elementos/componentes
- Cada elemento/componente é rankeado de acordo com uma função gulosa
- Guarda a melhor solução encontrada durante sua execução

GRASP

GRASP SIMPLIFICADO - MINIMIZAÇÃO

13. $S^+ \leftarrow \emptyset$ (mantém atualizado a melhor solução encontrada)
14. Enquanto não atingir condição de parada faça
15. $\quad S \leftarrow$ *Solução-Gulosa-Aleatoria*()
16. $\quad S' \leftarrow$ *Busca-Local*(S)
17. $\quad$ Se $c(S') \leq c(S^+)$ então
18. $\quad\quad S^+ \leftarrow S'$
19. Devolva S^+

Possíveis implementações das subrotinas:

CONDIÇÕES DE PARADA

- Número de iterações limitado a um valor máximo
- Quando atingir limite de tempo de CPU
- Quando melhor solução não for atualizada por certo número de iterações

Possíveis implementações das subrotinas:

SOLUÇÃO-GULOSA-ALEATÓRIA

1. $S \leftarrow \emptyset$
2. Enquanto S não é solução
3. $\quad L \leftarrow$ *Construa-Lista-Restrita-de-Candidatos*(S)
4. $\quad e \leftarrow$ *Escolha-Gulosa-Aleatória*(L)
5. $\quad S \leftarrow$ *Insere-ou-Adapte-Novo-Elemento*(S, e)
6. Devolva S

Construa-Lista-Restrita-de-Candidatos:

- Seja $f(S, e)$ valor da função gulosa sobre acréscimo do elemento e em S
- Seja $E = (e_1, \ldots, e_m)$ elementos/componentes candidatos para serem inseridas (e alteradas) em S tal que

$$f_{\min} = f(S, e_1) \leq \ldots \leq f(S, e_m) = f_{\max}$$

CONSTRUA-LISTA-RESTRITA-DE-CANDIDATOS (Minimização)

min max α:

1. Seja $\alpha \in [0, 1]$.
2. $t \leftarrow \max\{i : f(S, e_i) \leq f_{\min} + \alpha \cdot (f_{\max} - f_{\min})\}$
3. $L \leftarrow (e_1, \ldots, e_t)$
4. Devolva L

Por cardinalidade k:

1. Seja k tamanho máximo para lista restrita de candidato
2. $L \leftarrow (e_1, \ldots, e_k)$
3. Devolva L

Escolha-Gulosa-Aleatória:

Possíveis algoritmos para Escolha-Gulosa-Aleatória(L):

- Seja $L = (e_1, \ldots, e_k)$.

ESCOLHA-GULOSA-ALEATÓRIA(L) (Minimização)

Uniforme

1. Com probabilidade $\frac{1}{|L|}$, devolva $e \in L$.

Ordem

1. $bias(e_i) \leftarrow \frac{1}{i}$ para $i = 1, \ldots, k$ (indicador de preferência).
2. Defina $p(e_i)$ probabilidade de obter e_i proporcional a $bias(e_i)$
3. Com probabilidade $p(e_i)$, devolva $e \in L$.

Outras

- Baseado no peso dos elementos.
- Baseado em $f(S, e)$.
- Combinação de regras.
- etc.

Path Relinking

Idéia:

1. Sejam S e T duas soluções
2. Suponha que fazemos movimentos que partem de S para T.

$$S = S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_k = T$$

3. Possivelmente, neste caminho podemos obter soluções melhores que obtêm características boas de ambas soluções

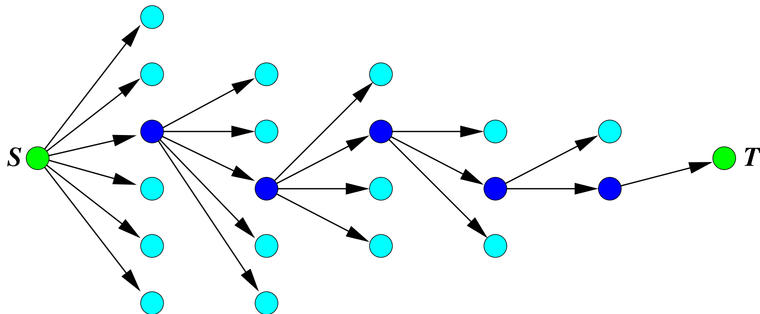
Path Relinking

Denote por $\Delta(S, T)$ a diferença simétrica de S e T

PATH-RELINKING(S, T)

1. $j \leftarrow 1$
2. $S_j \leftarrow S$
3. $S^+ \leftarrow S_j$ (manter a melhor solução)
4. Enquanto $\Delta(S_j, T) \neq \emptyset$
5. Seja S_j^e solução vizinha de S_j correspondente a $e \in \Delta(S_j, T)$
6. Seja $S_{j+1} \in \{S_j^e : e \in \Delta(S_j, T)\}$ com custo mínimo
7. Se $c(S_{j+1}) < c(S^+)$ então $S^+ \leftarrow S_{j+1}$
8. $j \leftarrow j + 1$
9. Devolva S^+

Path Relinking



Forward Path Relinking: S é uma solução melhor que T

Backward Path Relinking: S é uma solução pior que T

Back and forth Path Relinking: Busca de S para T e depois de T para S (custo computacional duplicado, mas possivelmente melhora marginal)

GRASP with Path Relinking - Minimização

- Manter um pool $\mathcal{P}$ das melhores soluções
- Seja S_j a solução obtida pelo GRASP pela busca local da j-ésima iteração.
- No fim da iteração j do GRASP, faça

1. $T \leftarrow$ Escolha-Solução-Destino$(\mathcal{P}, S')$
2. $S' \leftarrow$ Path-Relinking(S_j, T)
3. Se $c(S') < c(S^+)$ então $S^+ \leftarrow S'$
4. Atualize-Pool$(\mathcal{P}, S')$

Alternativas de implementação da rotina: Escolha-Solução-Destino

Escolha-Solução-Destino($\mathcal{P}, S'$) - Minimização

- Escolha uma solução $T \in \mathcal{P}$ com probabilidade uniforme
- Escolha uma solução $T \in \mathcal{P}$ com probabilidade proporcional a $|\Delta(T, S')|$

Possível implementação da rotina: Atualize-Pool

- Manter boas soluções destino no pool, mantendo diversidade
- Remover soluções piores ou parecidas da que está inserindo

Possível implementação para
Atualize-Pool($\mathcal{P}, S'$) - Minimização

1. Se $|\mathcal{P}| < MaxPool$ então
2. $\quad \mathcal{P} \leftarrow \mathcal{P} \cup S'$
3. Senão
4. $\quad \mathcal{Q} \leftarrow \{S \in \mathcal{P} : \ c(S) \geq c(S')\}$
5. $\quad$ Se $\mathcal{Q} \neq \emptyset$ então
6. $\quad\quad$ Seja $Q \in \mathcal{Q}$ com $|\Delta(Q, S')|$ mínimo
7. $\quad\quad \mathcal{P} \leftarrow \mathcal{P} - Q + S'$

Exemplo: Max-Sat com pesos nas cláusulas

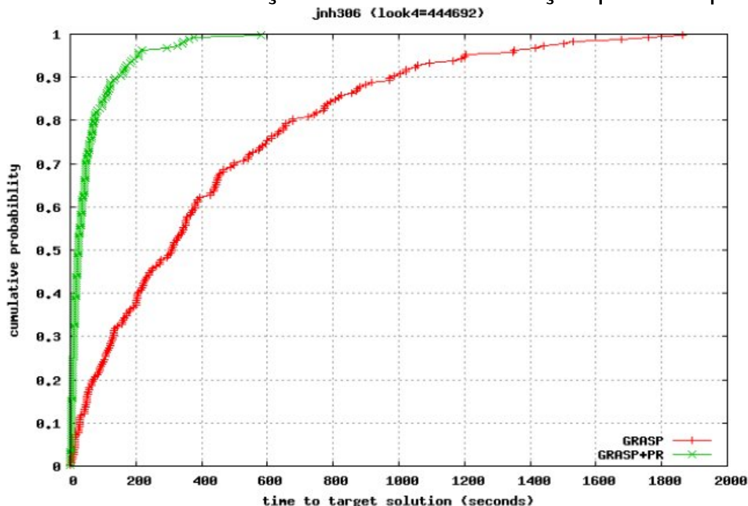
Lista-Restrita-de-Candidatos:
A cada solução parcial, calcula para cada variável X_i o peso total das novas cláusulas satisfeitas quando $X_i = 1$ ou $X_i = 0$.

Diferença simétrica
$$\Delta(X, Y) = \{i : \; X_i \neq Y_i, \quad i = 1, \ldots, n\}$$

Ex.: MaxSat (Festa, Pardalos, Pitsoulis, Resende'06)

Comparação: GRASP $\times$ GRASP+Path Relinking:

Probabilidade de se alcançar valor de uma solução pelo tempo



GRASP

Exercícios faça heurísticas GRASP para os seguintes problemas:

1. Caixeiro Viajante: TSP
2. Corte de Peso Máximo: MaxCut
3. Satisfatibilidade de Peso Máximo: MaxSat