Tópicos de Análise de Algoritmos

Busca local:
algoritmo Metropolis e simulated annealing

Secs 12.1 e 12.2 do KT

Busca local

Problema de otimização combinatória:

conjunto de instâncias
para cada instância I,
  um conjunto Sol(I) (não vazio) de soluções (viáveis) e
  uma função $\mathrm{val}(S)$ para cada solução S em Sol(I)

Busca local

Problema de otimização combinatória:

conjunto de instâncias
para cada instância I,
  um conjunto Sol(I) (não vazio) de soluções (viáveis) e
  uma função $\mathrm{val}(S)$ para cada solução S em Sol(I)

Objetivo: Dada uma instância I viável,
encontrar solução S em Sol(I) tq $\mathrm{val}(S)$ é mínimo/máximo.

Busca local

Problema de otimização combinatória:

conjunto de instâncias
para cada instância I,
  um conjunto Sol(I) (não vazio) de soluções (viáveis) e
  uma função $\mathrm{val}(S)$ para cada solução S em Sol(I)

Objetivo: Dada uma instância I viável,
encontrar solução S em Sol(I) tq $\mathrm{val}(S)$ é mínimo/máximo.

Tipicamente Sol(I) tem tamanho exponencial.

Busca local

Problema de otimização combinatória:

conjunto de instâncias
para cada instância I,
  um conjunto Sol(I) (não vazio) de soluções (viáveis) e
  uma função $\mathrm{val}(S)$ para cada solução S em Sol(I)

Objetivo: Dada uma instância I viável,
encontrar solução S em Sol(I) tq $\mathrm{val}(S)$ é mínimo/máximo.

Tipicamente Sol(I) tem tamanho exponencial.

Em uma busca local, adicionamos a
noção de vizinhança entre as soluções.

Busca local

Problema de otimização combinatória:

conjunto de instâncias
para cada instância I,
  um conjunto Sol(I) (não vazio) de soluções (viáveis) e
  uma função $\mathrm{val}(S)$ para cada solução S em Sol(I)

Objetivo: Dada uma instância I viável,
encontrar solução S em Sol(I) tq $\mathrm{val}(S)$ é mínimo/máximo.

Tipicamente Sol(I) tem tamanho exponencial.

Em uma busca local, adicionamos a
noção de vizinhança entre as soluções.

Relações que vizinhança simétricas.

Busca local

Algoritmo:

Começa com uma solução arbitrária S.
Iterativamente muda para uma solução S' vizinha a S.

Busca local

Algoritmo:

Começa com uma solução arbitrária S.
Iterativamente muda para uma solução S' vizinha a S.

Durante o processo, guarda a melhor solução visitada.

Busca local

Algoritmo:

Começa com uma solução arbitrária S.
Iterativamente muda para uma solução S' vizinha a S.

Durante o processo, guarda a melhor solução visitada.

Pontos críticos:

- como definir a vizinhança de uma solução

Busca local

Algoritmo:

Começa com uma solução arbitrária S.
Iterativamente muda para uma solução S' vizinha a S.

Durante o processo, guarda a melhor solução visitada.

Pontos críticos:

- como definir a vizinhança de uma solução
- como escolher S' na vizinhança de S

Busca local

Algoritmo:

Começa com uma solução arbitrária S.
Iterativamente muda para uma solução S' vizinha a S.

Durante o processo, guarda a melhor solução visitada.

Pontos críticos:

- como definir a vizinhança de uma solução
- como escolher S' na vizinhança de S

Grafo cujo conjunto de vértices é $\mathrm{Sol}(I)$ e adjacência é dada pela relação de vizinhança entre as soluções.

Algoritmo de busca local é um passeio neste grafo, que tenta alcançar uma boa solução.

Cobertura por vértices

Grafo $G = (V, E)$

$S \subseteq V$ é cobertura por vértices se
toda aresta e em E tem pelo menos uma ponta em S.

Cobertura por vértices

Grafo $G = (V, E)$

$S \subseteq V$ é cobertura por vértices se
toda aresta e em E tem pelo menos uma ponta em S.

Problema: Dado G,
encontrar cobertura por vértices de tamanho mínimo.

custo $c(S) = |S|$

Cobertura por vértices

Grafo $G = (V, E)$

$S \subseteq V$ é cobertura por vértices se
toda aresta e em E tem pelo menos uma ponta em S.

Problema: Dado G,
encontrar cobertura por vértices de tamanho mínimo.

custo $c(S) = |S|$

Relação de vizinhança:
$S \sim S'$ se S' é obtido de S
adicionando-se ou removendo-se um vértice.

Cobertura por vértices

Grafo $G = (V, E)$

$S \subseteq V$ é cobertura por vértices se
toda aresta e em E tem pelo menos uma ponta em S.

Problema: Dado G,
encontrar cobertura por vértices de tamanho mínimo.

custo $c(S) = |S|$

Relação de vizinhança:
$S \sim S'$ se S' é obtido de S
adicionando-se ou removendo-se um vértice.

Cada solução S tem n vizinhos, onde $n = |V|$.

Algoritmo de busca local

Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$

2 $\quad$ **enquanto** existe vizinho S' de S
tq $|S'| < |S|$ e S' é cobertura **faça**

3 $\quad\quad S \leftarrow S'$

4 $\quad$ **devolva** S

Algoritmo de busca local

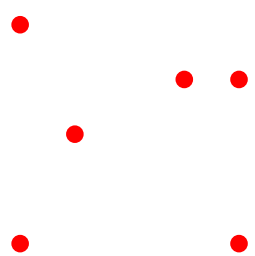
Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S

Exemplo 1: $G = (V, \emptyset)$

S decresce um vértice de cada vez até $S = \emptyset$, que é ótimo.



Algoritmo de busca local

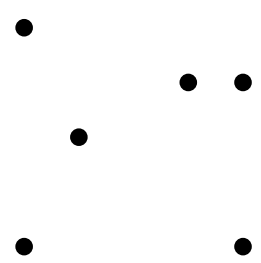
Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S

Exemplo 1: $G = (V, \emptyset)$

S decresce um vértice de cada vez até $S = \emptyset$, que é ótimo.



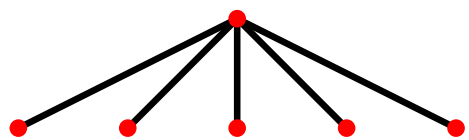
Algoritmo de busca local

Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S

Exemplo 2: G é estrela de centro v



Algoritmo de busca local

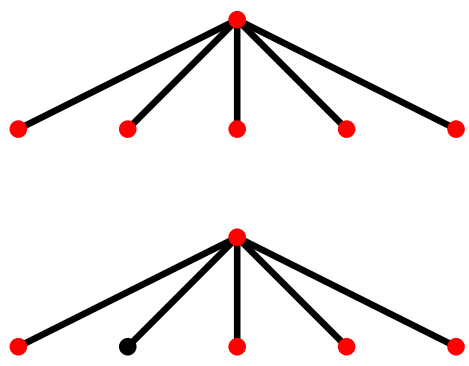
Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S

Exemplo 2: G é estrela de centro v

Se, ao final da primeira iteração, $S \neq S \setminus \{v\}$,
o algoritmo termina com $S = \{v\}$, que é ótimo.



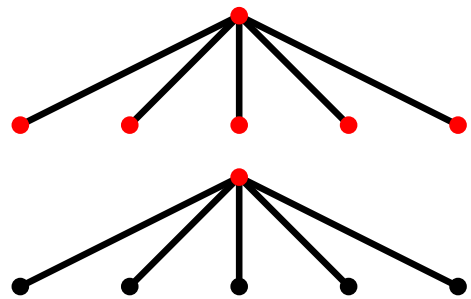
Algoritmo de busca local

Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S

Exemplo 2: G é estrela de centro v
Se, ao final da primeira iteração, $S \neq S \setminus \{v\}$,
o algoritmo termina com $S = \{v\}$, que é ótimo.



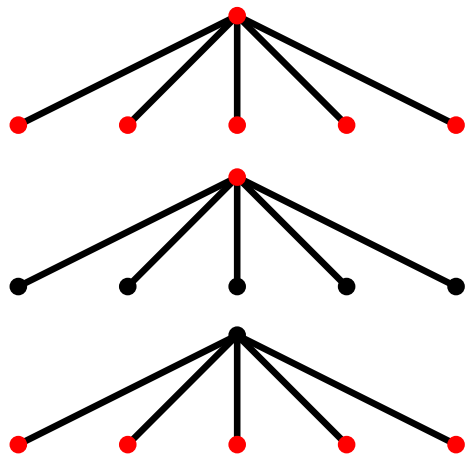
Algoritmo de busca local

Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S

Exemplo 2: G é estrela de centro v
Se, ao final da primeira iteração, $S \neq S \setminus \{v\}$,
o algoritmo termina com $S = \{v\}$, que é ótimo.
Se, ao final da primeira iteração, $S = S \setminus \{v\}$,
o algoritmo termina com esse ótimo local.

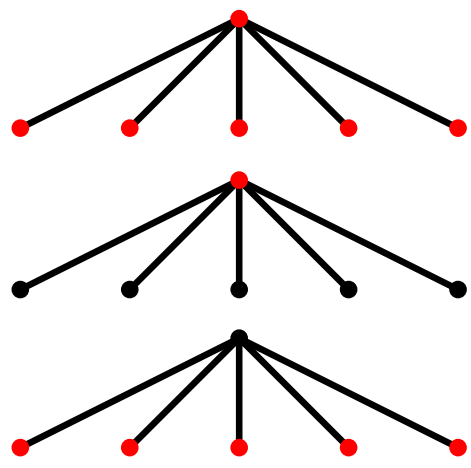


Algoritmo de busca local

Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S



Exemplo 2: G é estrela de centro v
Se, ao final da primeira iteração, $S \neq S \setminus \{v\}$,
o algoritmo termina com $S = \{v\}$, que é ótimo.

Se, ao final da primeira iteração, $S = S \setminus \{v\}$,
o algoritmo termina com esse ótimo local.

O algoritmo pode se dar arbitrariamente mal.

Algoritmo de busca local

Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S

Exemplo 3: G é um caminho com número ímpar de vértices

Algoritmo de busca local

Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S

Exemplo 3: G é um caminho com número ímpar de vértices

Solução ótima: conjunto de vértices pares

Algoritmo de busca local

Gradiente descendente

BUSCA-LOCAL (n, E) $\quad \triangleright\ G = (V, E)$ onde $V = [n]$

1 $\quad S \leftarrow [n]$
2 $\quad$ **enquanto** existe vizinho S' de S
$\qquad\qquad$ tq $|S'| < |S|$ e S' é cobertura **faça**
3 $\qquad\quad S \leftarrow S'$
4 $\quad$ **devolva** S

Exemplo 3: G é um caminho com número ímpar de vértices

Solução ótima: conjunto de vértices pares

Muitos mínimos locais.

Algoritmo Metropolis

Metropolis, Rosenbluth, Teller, e Teller (1953)

Simulação de sistema físico
de acordo com mecânica estatística.

Algoritmo Metropolis

Metropolis, Rosenbluth, Teller, e Teller (1953)

Simulação de sistema físico
de acordo com mecânica estatística.

Função de Gibbs-Boltzmann: $e^{-E/(kT)}$,
onde E é a energia, T é a temperatura e k é uma constante.

Algoritmo Metropolis

Metropolis, Rosenbluth, Teller, e Teller (1953)

Simulação de sistema físico
de acordo com mecânica estatística.

Função de Gibbs-Boltzmann: $e^{-E/(kT)}$,
onde E é a energia, T é a temperatura e k é uma constante.

Modelo: o sistema está num estado de energia E
com probabilidade dada pela função de Gibbs-Boltzmann.

Algoritmo Metropolis

Metropolis, Rosenbluth, Teller, e Teller (1953)

Simulação de sistema físico
de acordo com mecânica estatística.

Função de Gibbs-Boltzmann: $e^{-E/(kT)}$,
onde E é a energia, T é a temperatura e k é uma constante.

Modelo: o sistema está num estado de energia E
com probabilidade dada pela função de Gibbs-Boltzmann.

Para T fixo, a função decresce com E.

(Maior a chance de estar em um estado de menor energia.)

Algoritmo Metropolis

Metropolis, Rosenbluth, Teller, e Teller (1953)

Simulação de sistema físico
de acordo com mecânica estatística.

Função de Gibbs-Boltzmann: $e^{-E/(kT)}$,
onde E é a energia, T é a temperatura e k é uma constante.

Modelo: o sistema está num estado de energia E
com probabilidade dada pela função de Gibbs-Boltzmann.

Para T fixo, a função decresce com E.

(Maior a chance de estar em um estado de menor energia.)

Para T pequeno,
estado de menor energia é mais provável que estado de energia alta.

Algoritmo Metropolis

Metropolis, Rosenbluth, Teller, e Teller (1953)

Simulação de sistema físico
de acordo com mecânica estatística.

Função de Gibbs-Boltzmann: $e^{-E/(kT)}$,
onde E é a energia, T é a temperatura e k é uma constante.

Modelo: o sistema está num estado de energia E
com probabilidade dada pela função de Gibbs-Boltzmann.

Para T fixo, a função decresce com E.

(Maior a chance de estar em um estado de menor energia.)

Para T pequeno,
estado de menor energia é mais provável que estado de energia alta.

Para T grande, a diferença entre estas probabilidades é bem pequena.

Algoritmo Metropolis

Objetivo: chegar a um estado de energia mínima.

Algoritmo Metropolis

Objetivo: chegar a um estado de energia mínima.

Algoritmo Metropolis: ▷ para um dado T

Comece em um estado S.

A cada iteração, gere uma perturbação pequena S' de S.

Se $E(S') \leq E(S)$, mude para S'.

Senão seja $\Delta(E) = E(S') - E(S)$.

Mude para S' com probabilidade $e^{-\Delta(E)/(kT)}$.

Algoritmo Metropolis

Objetivo: chegar a um estado de energia mínima.

Algoritmo Metropolis: ▷ para um dado T

Comece em um estado S.

A cada iteração, gere uma perturbação pequena S' de S.

Se $E(S') \leq E(S)$, mude para S'.

Senão seja $\Delta(E) = E(S') - E(S)$.

Mude para S' com probabilidade $e^{-\Delta(E)/(kT)}$.

Seja $Z = \sum_S e^{-E(S)/(kT)}$.

Para um estado S, seja $f_S(t)$ a fração dos t primeiros passos da simulação em que o algoritmo passa em S.

$\lim_{t\to\infty} f_S(t) = \frac{1}{Z} \cdot e^{-E(S)/(kT)}$ com probabilidade indo para 1

Algoritmo Metropolis

Objetivo: chegar a um estado de energia mínima.

Algoritmo Metropolis: ▷ para um dado T

Comece em um estado S.

A cada iteração, gere uma perturbação pequena S' de S.

Se $E(S') \leq E(S)$, mude para S'.

Senão seja $\Delta(E) = E(S') - E(S)$.

Mude para S' com probabilidade $e^{-\Delta(E)/(kT)}$.

Seja $Z = \sum_S e^{-E(S)/(kT)}$.

Para um estado S, seja $f_S(t)$ a fração dos t primeiros passos da simulação em que o algoritmo passa em S.

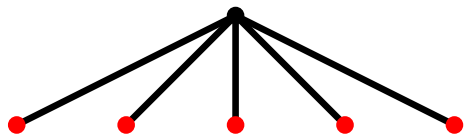
$\lim_{t\to\infty} f_S(t) = \frac{1}{Z} \cdot e^{-E(S)/(kT)}$ com probabilidade indo para 1

Fica em S o tempo esperado.

Metropolis para cobertura por vértices

Exemplo 1: G é estrela de centro v

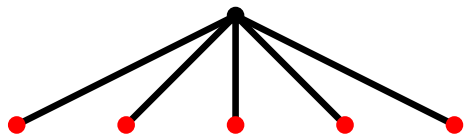
Se, ao final da primeira iteração, $S = S \setminus \{v\}$...



Metropolis para cobertura por vértices

Exemplo 1: G é estrela de centro v

Se, ao final da primeira iteração, $S = S \setminus \{v\}$...

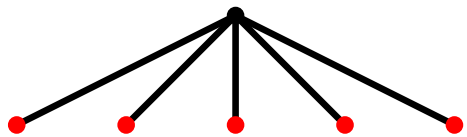


com probabilidade positiva coloca v de volta no S
e, mais adiante, encontra a cobertura mínima!

Metropolis para cobertura por vértices

Exemplo 1: G é estrela de centro v

Se, ao final da primeira iteração, $S = S \setminus \{v\}$...



com probabilidade positiva coloca v de volta no S
e, mais adiante, encontra a cobertura mínima!

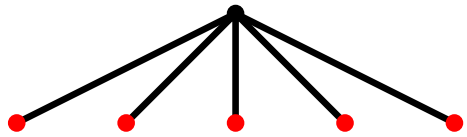
Exemplo 2: $G = (V, \emptyset)$

Quando S está grande,
tem boas chances do S diminuir.

Metropolis para cobertura por vértices

Exemplo 1: G é estrela de centro v

Se, ao final da primeira iteração, $S = S \setminus \{v\}$...



com probabilidade positiva coloca v de volta no S
e, mais adiante, encontra a cobertura mínima!

Exemplo 2: $G = (V, \emptyset)$

Quando S está grande,
tem boas chances do S diminuir.

Mas quando S fica pequeno,
tem boas chances de voltar a crescer
e é difícil chegar em $S = \emptyset$.

Algoritmo Metropolis

Algoritmo Metropolis: ▷ para um dado T

Comece em um estado S.

A cada iteração, gere uma perturbação pequena S' de S.

Se $E(S') \leq E(S)$, mude para S'.

Senão seja $\Delta(E) = E(S') - E(S)$.

Mude para S' com probabilidade $e^{-\Delta(E)/(kT)}$.

Mas e a temperatura T?

Qual é o papel e o efeito da temperatura?

Simulated Annealing

De volta ao sistema físico.

Annealing:
processo de cristalização em que o material é resfriado lentamente, para atingir o estado de energia mínima.

Simulated Annealing

De volta ao sistema físico.

Annealing:
processo de cristalização em que o material é resfriado lentamente, para atingir o estado de energia mínima.

Material em altas temperaturas não cristaliza.

Simulated Annealing

De volta ao sistema físico.

Annealing:
processo de cristalização em que o material é resfriado lentamente, para atingir o estado de energia mínima.

Material em altas temperaturas não cristaliza.

Se esfria rapidamente, cristaliza com muitas imperfeições.

Simulated Annealing

De volta ao sistema físico.

Annealing:
processo de cristalização em que o material é resfriado lentamente, para atingir o estado de energia mínima.

Material em altas temperaturas não cristaliza.

Se esfria rapidamente, cristaliza com muitas imperfeições.

Simulated annealing imita este processo, ajustando as probabilidades de mudar para um estado de energia maior de acordo com um parâmetro T que diminui com o tempo.

Simulated Annealing

De volta ao sistema físico.

Annealing:
processo de cristalização em que o material é resfriado lentamente, para atingir o estado de energia mínima.

Material em altas temperaturas não cristaliza.

Se esfria rapidamente, cristaliza com muitas imperfeições.

Simulated annealing imita este processo, ajustando as probabilidades de mudar para um estado de energia maior de acordo com um parâmetro T que diminui com o tempo.

Em geral, não tem garantia de qualidade.

Simulated Annealing

De volta ao sistema físico.

Annealing:
processo de cristalização em que o material é resfriado lentamente, para atingir o estado de energia mínima.

Material em altas temperaturas não cristaliza.

Se esfria rapidamente, cristaliza com muitas imperfeições.

Simulated annealing imita este processo, ajustando as probabilidades de mudar para um estado de energia maior de acordo com um parâmetro T que diminui com o tempo.

Em geral, não tem garantia de qualidade.

Em que velocidade diminuir o T?

Redes neurais de Hopfield

Às vezes buscamos um ótimo local, não necessariamente global.

Redes neurais de Hopfield

Às vezes buscamos um ótimo local, não necessariamente global.

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Redes neurais de Hopfield

Às vezes buscamos um ótimo local, não necessariamente global.

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

Redes neurais de Hopfield

Às vezes buscamos um ótimo local, não necessariamente global.

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

O sinal que um nó escolhe é influenciado pelo sinal de seus vizinhos.

Redes neurais de Hopfield

Às vezes buscamos um ótimo local, não necessariamente global.

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

O sinal que um nó escolhe é influenciado pelo sinal de seus vizinhos.

Cada aresta e representa um requisito:
- se e tem peso positivo, os nós querem sinais opostos
- se e tem peso negativo, os nós querem ter o mesmo sinal

Redes neurais de Hopfield

Às vezes buscamos um ótimo local, não necessariamente global.

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

O sinal que um nó escolhe é influenciado pelo sinal de seus vizinhos.

Cada aresta e representa um requisito:
- se e tem peso positivo, os nós querem sinais opostos
- se e tem peso negativo, os nós querem ter o mesmo sinal

$|w_e|$ representa a intensidade da dependência entre os extremos de e.

Redes neurais de Hopfield

Às vezes buscamos um ótimo local, não necessariamente global.

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

O sinal que um nó escolhe é influenciado pelo sinal de seus vizinhos.

Cada aresta e representa um requisito:
- se e tem peso positivo, os nós querem sinais opostos
- se e tem peso negativo, os nós querem ter o mesmo sinal

$|w_e|$ representa a intensidade da dependência entre os extremos de e.

Pode não existir configuração que satisfaça todos os requisitos.

Pense num triângulo com as três arestas com peso 1.

Redes neurais de Hopfield

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

Redes neurais de Hopfield

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

Aresta $e = uv$ é boa se u e v respeitam seu requisito:
se $w_e < 0$ então $s_u = s_v$ e se $w_e > 0$ então $s_u \neq s_v$

Senão e é ruim.

Redes neurais de Hopfield

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

Aresta $e = uv$ é boa se u e v respeitam seu requisito:
se $w_e < 0$ então $s_u = s_v$ e se $w_e > 0$ então $s_u \neq s_v$

Senão e é ruim.

Em outras palavras, e é boa sse $w_e\, s_u\, s_v < 0$.

Redes neurais de Hopfield

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

Aresta $e = uv$ é boa se u e v respeitam seu requisito:
se $w_e < 0$ então $s_u = s_v$ e se $w_e > 0$ então $s_u \neq s_v$

Senão e é ruim.

Em outras palavras, e é boa sse $w_e\, s_u\, s_v < 0$.

Nó u está satisfeito se $\sum_{e \text{ boa}} |w_e| \geq \sum_{e \text{ ruim}} |w_e|$

Redes neurais de Hopfield

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

Aresta $e = uv$ é boa se u e v respeitam seu requisito:
se $w_e < 0$ então $s_u = s_v$ e se $w_e > 0$ então $s_u \neq s_v$

Senão e é ruim.

Em outras palavras, e é boa sse $w_e\, s_u\, s_v < 0$.

Nó u está satisfeito se $\sum_{e \text{ boa}} |w_e| \geq \sum_{e \text{ ruim}} |w_e|$

Configurações estáveis: todos os nós estão satisfeitos.

Redes neurais de Hopfield

Rede neural de Hopfield:
grafo $G = (V, E)$ com peso inteiro w_e em cada aresta $e \in E$

Configuração da rede: atribuição de um sinal $+$ ou $-$ para cada nó

Aresta $e = uv$ é boa se u e v respeitam seu requisito:
se $w_e < 0$ então $s_u = s_v$ e se $w_e > 0$ então $s_u \neq s_v$

Senão e é ruim.

Em outras palavras, e é boa sse $w_e\, s_u\, s_v < 0$.

Nó u está satisfeito se $\sum_{e \text{ boa}} |w_e| \geq \sum_{e \text{ ruim}} |w_e|$

Configurações estáveis: todos os nós estão satisfeitos.

Sempre existe uma configuração estável?

Configurações estáveis

Configurações estáveis: todos os nós estão satisfeitos.

Por que o nome 'estável'?

Configurações estáveis

Configurações estáveis: todos os nós estão satisfeitos.

Por que o nome 'estável'?

Se um nó está insatisfeito, ele pode mudar seu sinal, e ficará satisfeito!

Configurações estáveis

Configurações estáveis: todos os nós estão satisfeitos.

Por que o nome 'estável'?

Se um nó está insatisfeito, ele pode mudar seu sinal, e ficará satisfeito!

Então numa configuração estável, nenhum nó quer mudar seu sinal.

Configurações estáveis

Configurações estáveis: todos os nós estão satisfeitos.

Por que o nome 'estável'?

Se um nó está insatisfeito, ele pode mudar seu sinal, e ficará satisfeito!

Então numa configuração estável, nenhum nó quer mudar seu sinal.

Note que, se u troca seu sinal,
então arestas boas incidentes a u ficam ruins
e arestas ruins tornam-se boas.

Configurações estáveis

Configurações estáveis: todos os nós estão satisfeitos.

Por que o nome 'estável'?

Se um nó está insatisfeito, ele pode mudar seu sinal, e ficará satisfeito!

Então numa configuração estável, nenhum nó quer mudar seu sinal.

Note que, se u troca seu sinal,
então arestas boas incidentes a u ficam ruins
e arestas ruins tornam-se boas.

Teorema: Toda rede neural de Hopfield com n nós
tem uma configuração estável, e tal configuração pode ser
encontrada em tempo polinomial em n e $W = \sum_e |w_e|$.

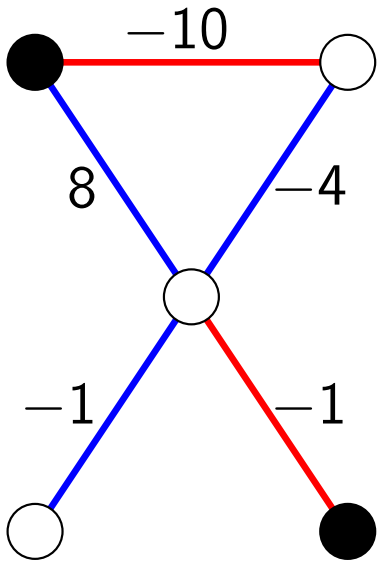
Algoritmo de busca local

Teorema: Toda rede neural de Hopfield com n nós
tem uma configuração estável, e tal configuração pode ser encontrada em tempo polinomial em n e $W = \sum_e |w_e|$.

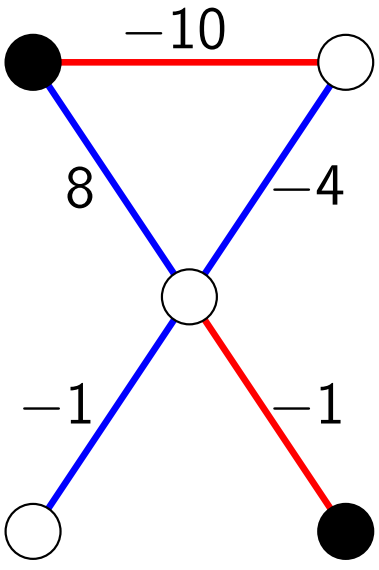
Algoritmo que busca configuração estável:

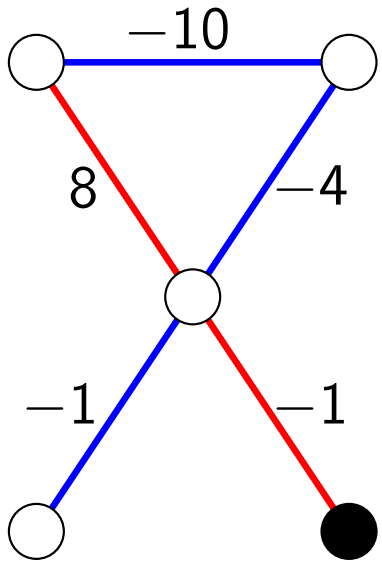
comece de uma configuração qualquer
enquanto existe nó insatisfeito, troque seu sinal

Algoritmo de busca local por configuração estável

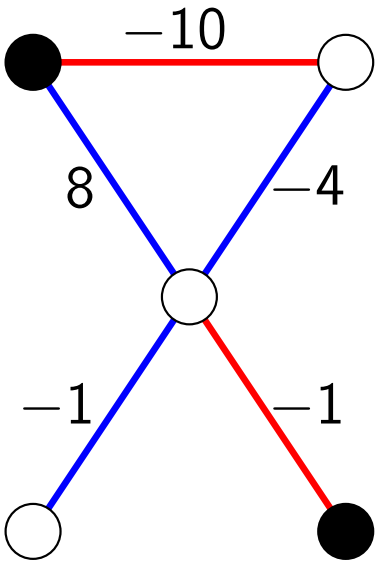


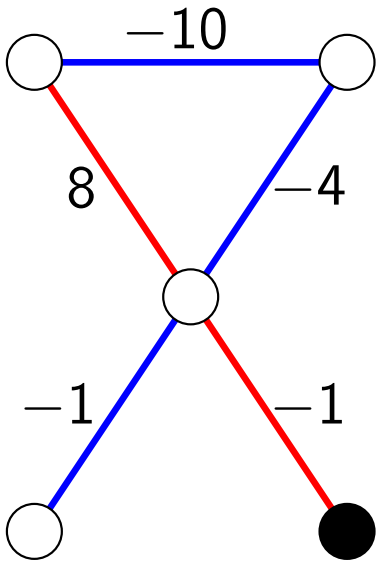
Algoritmo de busca local por configuração estável

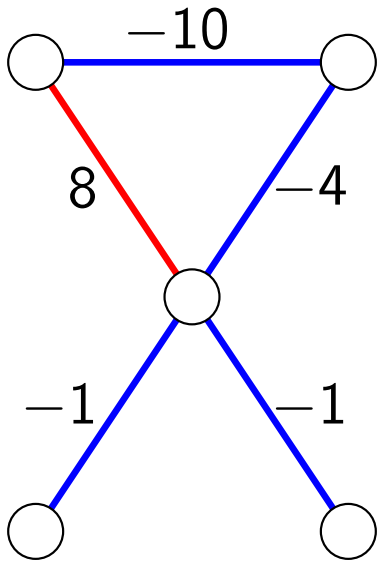




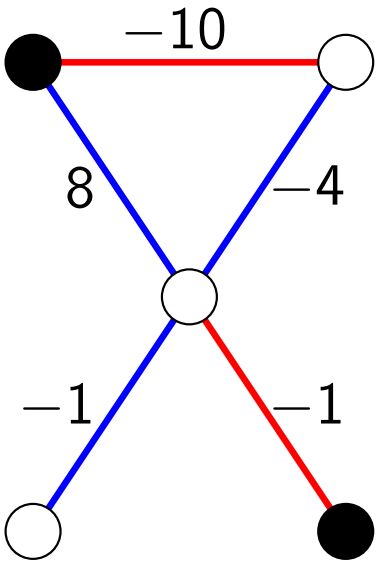
Algoritmo de busca local por configuração estável

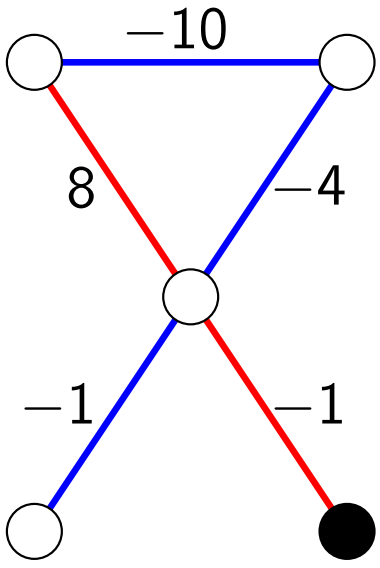


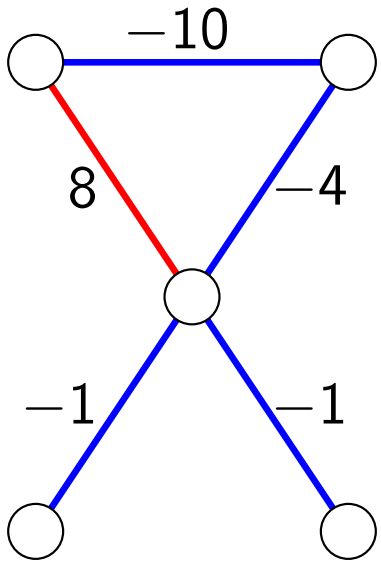


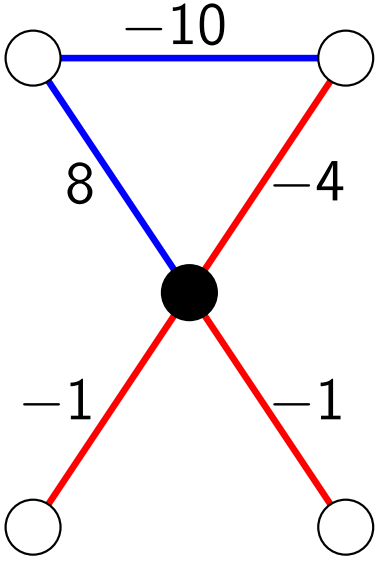


Algoritmo de busca local por configuração estável

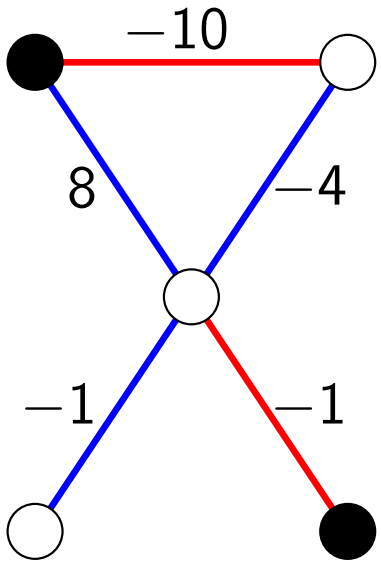


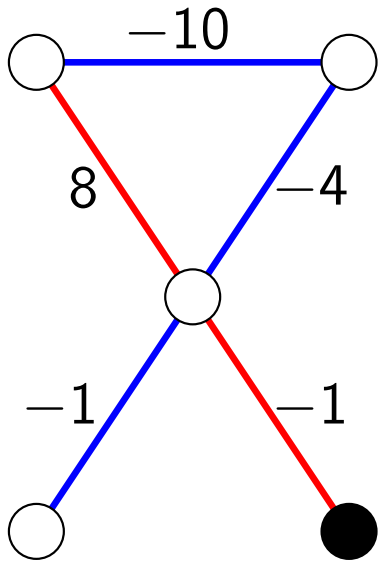


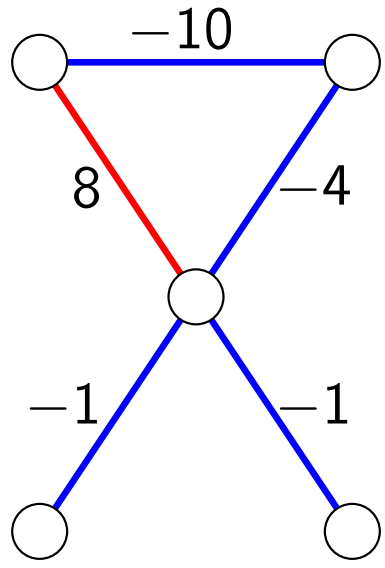


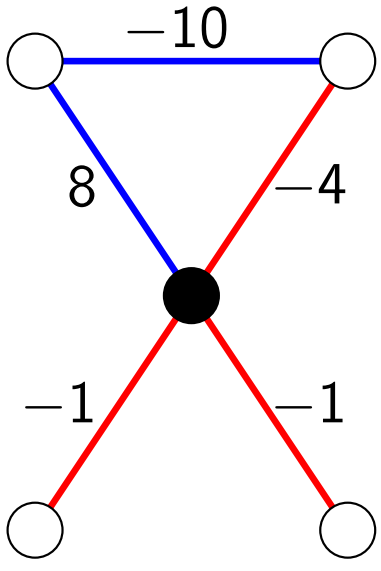


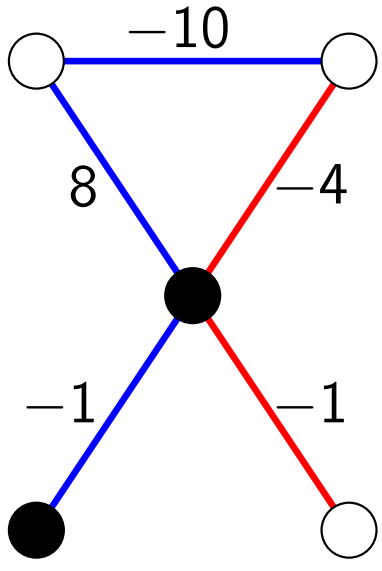
Algoritmo de busca local por configuração estável



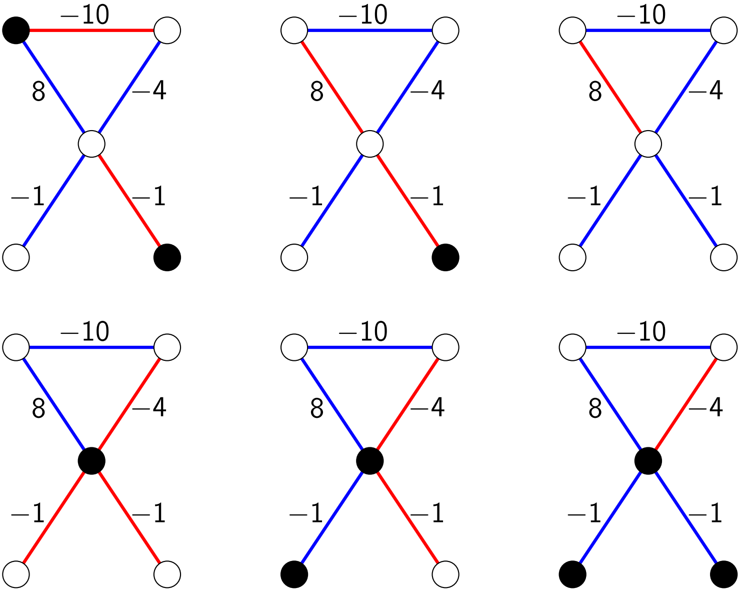








Algoritmo de busca local por configuração estável



O algoritmo

Comece de uma configuração qualquer
Enquanto existe um nó insatisfeito
    troque o seu sinal

O algoritmo termina?

O algoritmo

Comece de uma configuração qualquer
Enquanto existe um nó insatisfeito
    troque o seu sinal

O algoritmo termina?

Se termina, certamente termina numa configuração estável.

O algoritmo

Comece de uma configuração qualquer
Enquanto existe um nó insatisfeito
  troque o seu sinal

O algoritmo termina?

Se termina, certamente termina numa configuração estável.

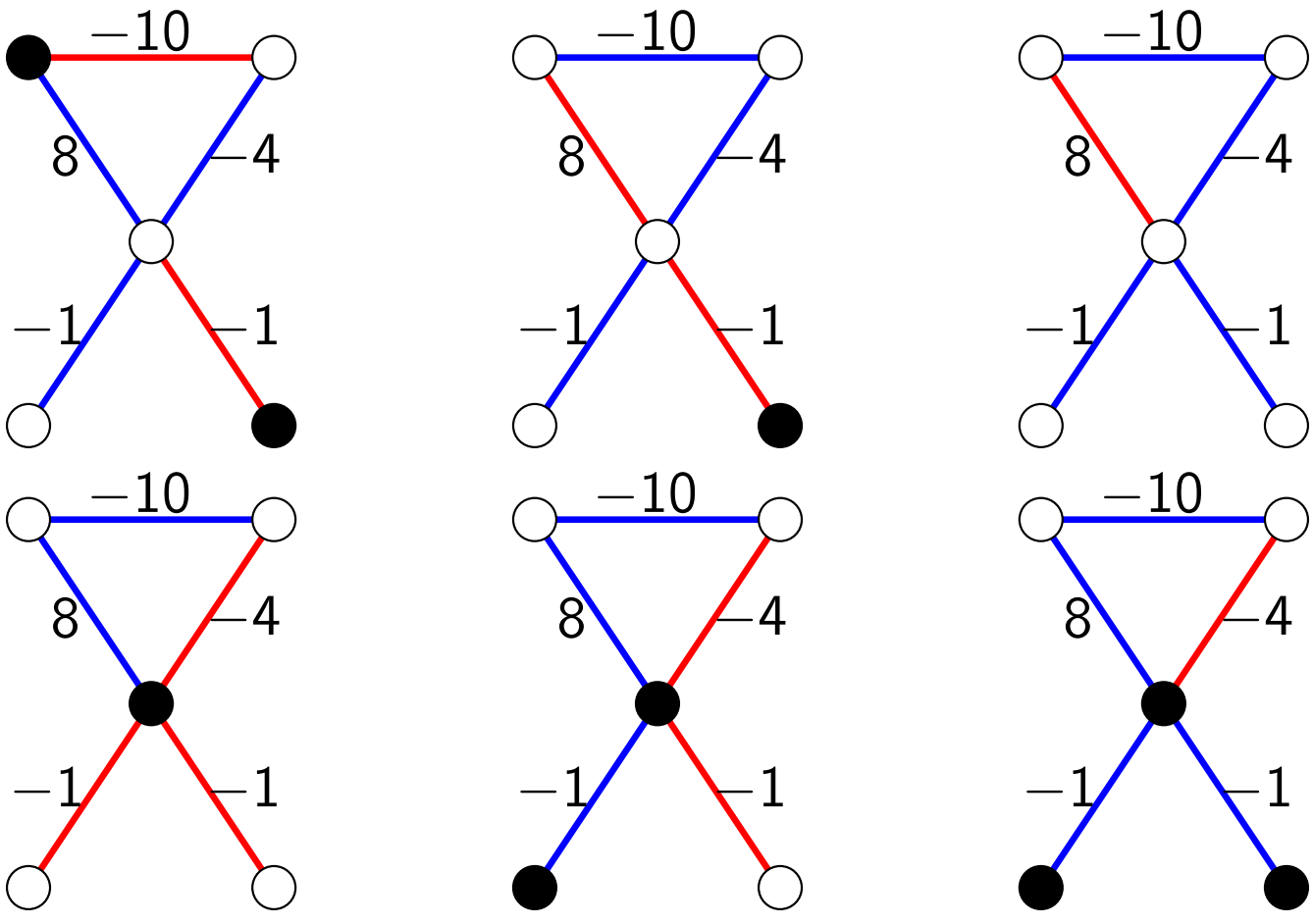
Vamos mostrar que o algoritmo acima sempre termina,
e que faz $O(n + W)$ iterações, onde $W = \sum_e |w_e|$.

Teorema: Toda rede neural de Hopfield com n nós
tem uma configuração estável, e tal configuração pode ser
encontrada em tempo polinomial em n e $W = \sum_e |w_e|$.

Quantas iterações o algoritmo pode fazer?

É verdade que

- o número de nós insatisfeitos diminui a cada iteração?



Quantas iterações o algoritmo pode fazer?

É verdade que

- o número de nós insatisfeitos diminui a cada iteração?
- cada nó troca de sinal no máximo uma vez?

Quantas iterações o algoritmo pode fazer?

É verdade que

- o número de nós insatisfeitos diminui a cada iteração?
- cada nó troca de sinal no máximo uma vez?

Como medir o "progresso" do algoritmo?

Quantas iterações o algoritmo pode fazer?

É verdade que

- o número de nós insatisfeitos diminui a cada iteração?
- cada nó troca de sinal no máximo uma vez?

Como medir o "progresso" do algoritmo?

Considere a soma do peso absoluto das arestas boas:

$$\sum_{e \text{ boa}} |w_e|$$

O que acontece com esta soma a cada iteração?

Análise do algoritmo

Como medir o "progresso" do algoritmo?

O que acontece com a soma do peso absoluto das arestas boas:

$$\sum_{e \text{ boa}} |w_e|$$

O que acontece com esta soma a cada iteração?

Análise do algoritmo

Como medir o "progresso" do algoritmo?

O que acontece com a soma do peso absoluto das arestas boas:

$$\sum_{e \text{ boa}} |w_e|$$

O que acontece com esta soma a cada iteração?

Quando o sinal de um nó troca,
as arestas em torno deste nó mudam de boas para ruins, e vice-versa.

Análise do algoritmo

Como medir o "progresso" do algoritmo?

O que acontece com a soma do peso absoluto das arestas boas:

$$\sum_{e \text{ boa}} |w_e|$$

O que acontece com esta soma a cada iteração?

Quando o sinal de um nó troca,
as arestas em torno deste nó mudam de boas para ruins, e vice-versa.

Nó u está insatisfeito se $\sum_{e \text{ boa}} |w_e| < \sum_{e \text{ ruim}} |w_e|$.

Portanto a soma acima aumenta a cada iteração!

Análise do algoritmo

A soma do peso absoluto das arestas boas

$$\sum_{e \text{ boa}} |w_e|$$

é sempre maior ou igual a zero.

Análise do algoritmo

A soma do peso absoluto das arestas boas

$$\sum_{e \text{ boa}} |w_e|$$

é sempre maior ou igual a zero.

Ela começa de um valor qualquer maior ou igual a zero
e aumenta a cada iteração.

Análise do algoritmo

A soma do peso absoluto das arestas boas

$$\sum_{e \text{ boa}} |w_e|$$

é sempre maior ou igual a zero.

Ela começa de um valor qualquer maior ou igual a zero
e aumenta a cada iteração.

Como os pesos w_e são inteiros,
ela aumenta de pelo menos um a cada iteração.

Análise do algoritmo

A soma do peso absoluto das arestas boas

$$\sum_{e \text{ boa}} |w_e|$$

é sempre maior ou igual a zero.

Ela começa de um valor qualquer maior ou igual a zero
e aumenta a cada iteração.

Como os pesos w_e são inteiros,
ela aumenta de pelo menos um a cada iteração.

A soma é no máximo $W = \sum_e |w_e|$.

Logo o algoritmo faz no máximo W iterações (pseudo-polinomial)
e termina com uma configuração estável.

Problema do corte máximo

Seja $G = (V, E)$ um grafo
com um peso w_e inteiro positivo em cada aresta $e \in E$.

Um corte é uma partição $\{A, B\}$ de V com $A \neq \emptyset$ e $B \neq \emptyset$.

Problema do corte máximo

Seja $G = (V, E)$ um grafo
com um peso w_e inteiro positivo em cada aresta $e \in E$.

Um corte é uma partição $\{A, B\}$ de V com $A \neq \emptyset$ e $B \neq \emptyset$.

O peso do corte $\{A, B\}$ é

$$w(A, B) := \sum_{e=xy : x \in A, y \in B} w_e.$$

Problema do corte máximo

Seja $G = (V, E)$ um grafo
com um peso w_e inteiro positivo em cada aresta $e \in E$.

Um corte é uma partição $\{A, B\}$ de V com $A \neq \emptyset$ e $B \neq \emptyset$.

O peso do corte $\{A, B\}$ é

$$w(A, B) := \sum_{e=xy : x \in A, y \in B} w_e.$$

Dado um grafo $G = (V, E)$ com um peso w_e inteiro positivo em cada aresta $e \in E$, determinar um corte de peso máximo.

Esse problema é NP-difícil e é conhecido como MAXCUT.

Algoritmo inspirado no anterior

Comece de um corte $\{A, B\}$ qualquer
Enquanto existe um vértice insatisfeito
    troque-o de conjunto

Algoritmo inspirado no anterior

Comece de um corte $\{A, B\}$ qualquer
Enquanto existe um vértice insatisfeito
  troque-o de conjunto

Vértice insatisfeito:

Se $v \in A$ e $\sum_{u \in N(v) \cap A} w_{uv} > \sum_{u \in N(v) \cap B} w_{uv}$,

  então v prefere mudar para B.

Se $v \in B$ e $\sum_{u \in N(v) \cap B} w_{uv} > \sum_{u \in N(v) \cap A} w_{uv}$,

  então v prefere mudar para A.

Algoritmo inspirado no anterior

Comece de um corte $\{A, B\}$ qualquer
Enquanto existe um vértice insatisfeito
  troque-o de conjunto

Vértice insatisfeito:

Se $v \in A$ e $\sum_{u \in N(v) \cap A} w_{uv} > \sum_{u \in N(v) \cap B} w_{uv}$,

  então v prefere mudar para B.

Se $v \in B$ e $\sum_{u \in N(v) \cap B} w_{uv} > \sum_{u \in N(v) \cap A} w_{uv}$,

  então v prefere mudar para A.

O algoritmo termina?

Algoritmo inspirado no anterior

Comece de um corte $\{A, B\}$ qualquer
Enquanto existe um vértice insatisfeito
  troque-o de conjunto

Vértice insatisfeito:

Se $v \in A$ e $\sum_{u \in N(v) \cap A} w_{uv} > \sum_{u \in N(v) \cap B} w_{uv}$,

  então v prefere mudar para B.

Se $v \in B$ e $\sum_{u \in N(v) \cap B} w_{uv} > \sum_{u \in N(v) \cap A} w_{uv}$,

  então v prefere mudar para A.

O algoritmo termina?

Sim! O peso do corte está sempre aumentando! Então termina!

Qualidade do corte produzido

O peso do corte produzido é

$$\sum_{v \in A} \sum_{u \in N(v) \cap B} w_{uv} = \sum_{v \in B} \sum_{u \in N(v) \cap A} w_{uv}.$$

Qualidade do corte produzido

O peso do corte produzido é

$$\sum_{v \in A} \sum_{u \in N(v) \cap B} w_{uv} = \sum_{v \in B} \sum_{u \in N(v) \cap A} w_{uv}.$$

Termina com todo vértice satisfeito:

- Se $v \in A$, então $\sum_{u \in N(v) \cap A} w_{uv} \leq \sum_{u \in N(v) \cap B} w_{uv}$.
- Se $v \in B$, então $\sum_{u \in N(v) \cap B} w_{uv} \leq \sum_{u \in N(v) \cap A} w_{uv}$.

Qualidade do corte produzido

O peso do corte produzido é

$$\sum_{v \in A} \sum_{u \in N(v) \cap B} w_{uv} = \sum_{v \in B} \sum_{u \in N(v) \cap A} w_{uv}.$$

Termina com todo vértice satisfeito:

- Se $v \in A$, então $\sum_{u \in N(v) \cap A} w_{uv} \leq \sum_{u \in N(v) \cap B} w_{uv}$.
- Se $v \in B$, então $\sum_{u \in N(v) \cap B} w_{uv} \leq \sum_{u \in N(v) \cap A} w_{uv}$.

Em outras palavras,

- se $v \in A$, então $\sum_{u \in N(v) \cap B} w_{uv} \geq \frac{1}{2} \sum_{u \in N(v)} w_{uv}$ e
- se $v \in B$, então $\sum_{u \in N(v) \cap A} w_{uv} \geq \frac{1}{2} \sum_{u \in N(v)} w_{uv}$.

Qualidade do corte produzido

O peso do corte produzido é

$$\sum_{v \in A} \sum_{u \in N(v) \cap B} w_{uv} = \sum_{v \in B} \sum_{u \in N(v) \cap A} w_{uv}.$$

Para todo vértice v, $\sum_{u \in N(v) \cap B} w_{uv} \geq \frac{1}{2} \sum_{u \in N(v)} w_{uv}$ se $v \in A$ e

$$\sum_{u \in N(v) \cap A} w_{uv} \geq \frac{1}{2} \sum_{u \in N(v)} w_{uv} \text{ se } v \in B.$$

Então

$$\begin{aligned} W = \sum_e w_e &= \frac{1}{2} \sum_{v \in V} \sum_{u \in N(v)} w_{uv} \\ &= \frac{1}{2} \left(\sum_{v \in A} \sum_{u \in N(v)} w_{uv} + \sum_{v \in B} \sum_{u \in N(v)} w_{uv} \right) \\ &\leq \sum_{v \in A} \sum_{u \in N(v) \cap B} w_{uv} + \sum_{v \in B} \sum_{u \in N(v) \cap A} w_{uv} \\ &= 2 \sum_{v \in A} \sum_{u \in N(v) \cap B} w_{uv}. \qquad \triangleright\ 2 \times \text{valor do corte produzido} \end{aligned}$$

Como um corte máximo tem peso no máximo W, o algoritmo produz um corte que tem pelo menos metade do peso de um corte máximo.

2-aproximação?

O algoritmo é então uma 2-aproximação?

2-aproximação?

O algoritmo é então uma 2-aproximação?

Não... porque ele é pseudo-polinomial, e não polinomial.
O número de iterações é $\mathrm{O}(W)$.

2-aproximação?

O algoritmo é então uma 2-aproximação?

Não... porque ele é pseudo-polinomial, e não polinomial.
O número de iterações é $\mathrm{O}(W)$.

Altere o algoritmo levemente. Para um $\epsilon > 0$,

se $v \in A$ e $\sum_{u \in N(v) \cap A} w_{uv} \geq \sum_{u \in N(v) \cap B} w_{uv} + \frac{\epsilon}{n} w(A, B)$,

então v muda para B;

2-aproximação?

O algoritmo é então uma 2-aproximação?

Não... porque ele é pseudo-polinomial, e não polinomial.
O número de iterações é $\mathrm{O}(W)$.

Altere o algoritmo levemente. Para um $\epsilon > 0$,

se $v \in A$ e $\sum_{u \in N(v) \cap A} w_{uv} \geq \sum_{u \in N(v) \cap B} w_{uv} + \frac{\epsilon}{n} w(A, B)$,

então v muda para B;

se $v \in B$ e $\sum_{u \in N(v) \cap B} w_{uv} \geq \sum_{u \in N(v) \cap A} w_{uv} + \frac{\epsilon}{n} w(A, B)$,

então v muda para A.

2-aproximação?

O algoritmo é então uma 2-aproximação?

Não... porque ele é pseudo-polinomial, e não polinomial.
O número de iterações é $\mathrm{O}(W)$.

Altere o algoritmo levemente. Para um $\epsilon > 0$,

se $v \in A$ e $\sum_{u \in N(v) \cap A} w_{uv} \geq \sum_{u \in N(v) \cap B} w_{uv} + \frac{\epsilon}{n} w(A, B)$,

então v muda para B;

se $v \in B$ e $\sum_{u \in N(v) \cap B} w_{uv} \geq \sum_{u \in N(v) \cap A} w_{uv} + \frac{\epsilon}{n} w(A, B)$,

então v muda para A.

Exercício 1: Quantas iterações no máximo o algoritmo faz agora?

Exercício 2: Mostre que essa versão é uma $(2 + \frac{\epsilon}{2})$-aproximação.