Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms
  https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V
  https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

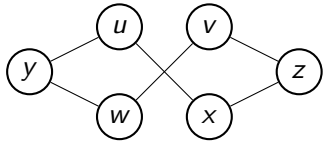
Qualquer erro é de minha responsabilidade.

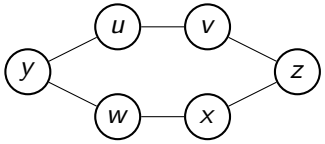
Busca Local

- Seja X o conjunto de soluções candidatas para um problema.
- Por Exemplo: Cortes em um Grafo, Circuitos do Caixeiro-Viajante, Itens em uma mochila.
- Usaremos o conceito de **vizinhança**.
- Definir para cada $x \in X$, quais soluções $y \in X$ são suas vizinhas.

Exemplos:

- Na busca local do corte máximo visto anteriormente x, y eram cortes vizinhos se e somente se diferiam na presença ou ausência de um único vértice.
- Em soluções do problema do caixeiro viajante podemos definir que dois circuitos são vizinhos se diferem em no máximo 2 arestas.





Uma busca local genérica

Algoritmo 1: Busca Local(x)

Entrada: Uma solução inicial x
Saída: Uma solução

```
1 enquanto x tem um vizinho melhor y faça
2     x := y
3 devolva x (ótimo local);
```


Como escolher uma solução inicial x?

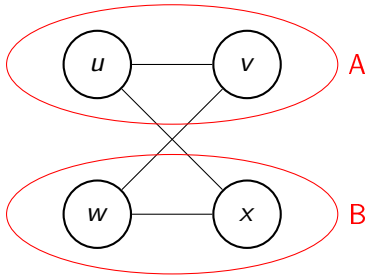
- Suponha que você não tem ideia de como obter uma boa solução.
  - Obs: Uma busca local garante que encontra um **ótimo local**.
  - Entretanto um ótimo local pode estar longe do que desejamos, o **ótimo global**.
  - Por outro lado um ótimo global também é um ótimo local.
  - Obter um ótimo local normalmente não te traz nenhuma informação sobre o quão longe você está do ótimo global.
  - Uma boa alternativa é executar várias vezes sua busca local, com diferentes soluções iniciais. **Multi-start**.
- Num cenário mais positivo, talvez você tenha alguma ideia de como encontrar uma boa solução, ou uma solução próxima da ótima. Uma heurística gulosa, uma solução relaxada.
  - A busca local garantidamente só pode melhorar a solução.

Se houverem vários y que melhoram x, qual deles escolher?

- Não há uma única resposta para essa pergunta. Existem vários métodos que podem ser utilizados e normalmente isso depende do problema, da vizinhança escolhida, etc. Uma possibilidade é fazer experimentos.
- Escolher um dos y (que melhoram) aleatoriamente, permitindo que você explore diferentes regiões de soluções (mesmo começando com a mesma solução inicial)
- Escolher o primeiro y que melhora x que você encontrar.
- Ir para a melhor. (exige mais processamento, já que você precisa explorar todos os vizinhos)
- Outras formas mais complexas. (GRASP, Busca-Tabu, VNS, etc)

Como definir uma vizinhança?

- Também não existe uma receita geral para todos os problemas. Escolher uma vizinhança depende do domínio do problema, e de certa experimentação.
- Podemos analisar o qual grande é uma vizinhança, e dessa forma analisar a complexidade de explorar ela.
- No problema do MaxCut, a vizinhança definida tinha tamanho $O(n)$.
- Poderíamos definir uma outra vizinhança no qual trocamos 2 ou menos vértices de posição simultaneamente. Qual o tamanho dessa vizinhança?$O(n^2)$
- Podemos ainda generalizar essa ideia e permitir k trocas simultâneas. essa vizinhança tem tamanho $O(\binom{n}{k})$.
- Pelo lado bom, uma vizinhança grande pode reduzir o número de ótimos locais.



- Geralmente vizinhanças grandes alcançam melhores soluções e demoram mais.
- Vizinhanças menores, param em mais ótimos locais, mas são mais rápidas de serem executadas.
- Dica: Encontrar o ponto de equilíbrio. Tentar diferentes abordagens.

Uma busca local garantidamente termina?

- Normalmente sim. Se o conjunto de soluções é finito, e os movimentos são guiados por uma função objetivo.

Uma busca local converge em tempo polinomial?

- Não necessariamente. Mas na prática, frequentemente as buscas locais são rápidas.
- Também é comum limitarmos o tempo de execução, e o algoritmo devolve a melhor solução encontrada.

Uma busca local garantem uma boa aproximação da solução ótima?

- Geralmente não.
- Garantias como a da busca local no MaxCut são exceções.
- Pode-se eventualmente comparar com um limitante (veremos depois)
- E é comum uma busca local devolver uma solução ruim. (por isso executar várias vezes introduzindo alguma aleatoriedade é importante)

Problema do 2-CNF-SAT

Dada uma fórmula booleana F na forma normal 2-conjuntiva, com n variáveis booleanas $x_1, \ldots, x_n$, e m cláusulas. Decidir se existe uma atribuição das variáveis que torna F verdadeira.

- Exemplo:

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_4)$$

- é satisfazível $x_1 = x_3 = \textit{verdadeiro}$, $x_2 = x_4 = \textit{falso}$.

2-CNF-SAT $\in P$

O 2-CNF-SAT pode ser resolvido em tempo polinomial.

- Você pode reduzir para o problema de encontrar componente fortemente conectadas.
- Você pode fazer um algoritmo de *backtraking*. (Parecido com aquele para vertex-cover)
- Busca Local Aleatorizada.
- Teremos garantia que o algoritmo converge.
- Mas pode-se estender a ideia para outros problemas de SAT.

Papadimitriou

- Christos Papadimitriou, é um cientista da computação grego, nascido em 1949.
- Graduado na Universidade Técnica Nacional de Atenas, obteve seu mestrado e doutorado na Universidade de Princeton (1974 e 1976)
- Lecionou em Harvard, MIT, Universidade Técnica Nacional de Atenas, Stanford, e Universidade de Columbia.
- Autor de vários livros e artigos.
- Fellow da Association for Computing Machinery
- em 2002 recebeu o Knuth Prize
- em 2012 recebeu o Gödel Prize, pelas contribuições em Teoria dos Jogos.

Algoritmo de Papadimitriou para o 2-CNF-SAT

Algoritmo 2: Busca Local(x)

Entrada: Uma Fórmula 2-CNF-SAT F
Saída: Uma solução

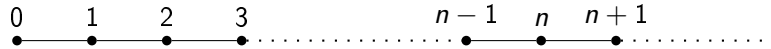
```
1 Repita log_2 n vezes
2     Gere uma atribuição inicial x;
3     Repita 2n^2 vezes
4         se F está satisfeita então devolva "sim";
5         Escolha qualquer clausula não satisfeita e troca o valor de
          uma de suas variáveis. (escolha a variável de maneira
          uniformemente aleatória);
6 devolva "não";
```

Exemplo: $(\neg x_3 \vee x_7), x_3 = V, x_7 = F$.
$(\neg x_3 \vee x_7) \wedge (x_3 \vee x_1) \wedge (x_3 \vee x_2) \ldots$.

- Executa em tempo polinomial.
- Sempre devolve a solução correta quando F é de fato inviável
- Mas e se existir uma solução que torna F factível (e exite um espaço de busca 2^n), o algoritmo vai encontrá-la?
- Talvez a melhor pergunta seja, o algoritmo provavelmente vai encontrá-la?

Passeio Aleatório nos Inteiros não Negativos

- Considere uma linha com inteiros não negativos.



- você vai andar nessa linha, entretanto você está muito perdido, a cada passo você anda 1 para a esquerda ou para direita com probabilidade 0,5.
- exceto pela posição inicial 0 que você sempre vai para a direita.
- é fácil ver que para chegar no 3 por exemplo, você pode dar 3, ou 5, ou 7, ou váááários passos.
- Seja T_n o número de passos até um passeio aleatório atingir a posição n.

Qual é a esperança de T_n, $E[T_n]$?

a. n
b. n^2
c. n^3
d. 2^n

- Seja T_n o número de passos até um passeio aleatório atingir a posição n.
- Seja Z_i o número de passos para sair de i e chegar em n.
- Note que $Z_0 = T_n$.
- $Z_n = 0$
- $E[Z_0] = 1 + E[Z_1]$, 1 passo para chegar em 1, e depois de lá para n
- Mas um pensamento semelhante pode ser aplicado para os valores intermediários $i \in \{1, \ldots, n-1\}$.

$$
\begin{aligned}
E[Z_i] &= 1 + Pr[\text{ir esq}] * E[Z_{i-1}] + Pr[\text{ir dir}] * E[Z_{i+1}] \\
E[Z_i] &= 1 + \frac{1}{2} * E[Z_{i-1}] + \frac{1}{2} * E[Z_{i+1}] \\
2 * E[Z_i] &= 2 + E[Z_{i-1}] + E[Z_{i+1}] \\
E[Z_i] + E[Z_i] &= 2 + E[Z_{i-1}] + E[Z_{i+1}] \\
E[Z_i] - E[Z_{i+1}] &= E[Z_{i-1}] - E[Z_i] + 2
\end{aligned}
$$

$$E[Z_i] - E[Z_{i+1}] = E[Z_{i-1}] - E[Z_i] + 2$$

- Informalmente, percebemos que estar em $i-1$ gasta 2 passos a mais do que estar em i.
- De outra forma estar 1 casa a frente dá uma vantagem de 2 passos.
- Além disso sabemos que $E[Z_0] = 1 + E[Z_1]$ dessa forma:

$$
\begin{aligned}
E[Z_0] - E[Z_1] &= 1 \\
E[Z_1] - E[Z_2] &= 3 \\
E[Z_2] - E[Z_3] &= 5 \\
&\ldots\ldots \\
E[Z_{n-2}] - E[Z_{n_1}] &= 2n - 3 \\
E[Z_{n-1}] - E[Z_n] &= 2n - 1
\end{aligned}
$$

$$
\begin{aligned}
E[Z_0] - \cancel{E[Z_1]} &= 1 \\
\cancel{E[Z_1]} - \cancel{E[Z_2]} &= 3 \\
\cancel{E[Z_2]} - \cancel{E[Z_3]} &= 5 \\
&\ldots\ldots \\
\cancel{E[Z_{n-2}]} - \cancel{E[Z_{n-1}]} &= 2n - 3 \\
+\cancel{E[Z_{n-1}]} - E[Z_n] &= 2n - 1 \\
\hline
E[Z_0] - 0 &= \frac{n}{2} 2n = n^2 \\
E[T_n] &= n^2
\end{aligned}
$$

Corolário

$Pr[T_n > 2n^2] \leq \frac{1}{2}$

Prova:

$$n^2 = E[T_n] = \sum_{k=0}^{2n^2} k.Pr[Tn = k] + \sum_{k=2n^2+1}^{\infty} k.Pr[Tn = k]$$
$$\geq 0 + \sum_{k=2n^2+1}^{\infty} 2n^2.Pr[Tn = k]$$
$$= 2n^2 \sum_{k=2n^2+1}^{\infty} .Pr[Tn = k]$$
$$= 2n^2 Pr[Tn > 2n^2]$$
$$n^2 \geq 2n^2 Pr[Tn > 2n^2]$$
$$Pr[Tn > 2n^2] * 2n^2 \leq n^2$$
$$Pr[Tn > 2n^2] \leq \frac{1}{2}$$

Voltando para o Algoritmo de Papadimitriou

Teorema

Para uma fórmula F satisfazível, com n variáveis, o algoritmo de Papadimitriou produz uma atribuição que satisfaz F com probabilidade $\geq 1 - \frac{1}{n}$

Prova:

- Considere uma única iteração do laço externo.
- Considere uma atribuição x^* que satisfaz F.
- Seja a_t a atribuição do algoritmo depois da iteração t do laço interno $t = 0, 1, \ldots, 2n^2$
- A chave aqui é considerar qual a medida de progresso estamos fazendo. Note que considerar o número de clausulas satisfeitas não dialoga com o Corolário anterior. Vamos considerar o número de variáveis que estão concordando com a^*, denotaremos por y_t.
- Note que se $y_t = n$ o algoritmo termina corretamente.

- Suponha então que a_t não satisfaz F, então o algoritmo escolhe uma clausula com variáveis x_i, x_j.
- Como a^* satisfaz F, ela faz uma escolha diferente para x_i ou x_j, ou ambas.

Quiz

O progresso de y_t se comporta como um passeio aleatório nos inteiros não negativos, exceto por:

a. As vezes se move para direita com probabilidade 1. (mesmo com $y_t \neq 0$)
b. talvez tenha $y_0 > 0$.
c. talvez termine antes de $y_t = n$
d. todas as anteriores.

- Note que todas essas diferenças estão a favor do algoritmo. Ou seja a probabilidade do algoritmo terminar (nessa iteração) em menos de $2n^2$ é maior do que a probabilidade de um passeio aleatório chegar em n com $2n^2$ passos.

$$Pr[\text{alg. encontrar uma atribuição satisfatória}] \geq Pr[T_n \leq 2n^2] \geq \frac{1}{2}$$

Logo

$$\begin{aligned} Pr[\text{alg. falhar}] &\leq Pr[\text{todas as } \log n \text{ tentativas falharem}] \\ &\leq \frac{1}{2}^{\log n} = \frac{1}{n} \end{aligned}$$

E portanto:

$$Pr[\text{alg. ter sucesso}] \geq 1 - \frac{1}{n}$$