Problemas de Busca (a.k.a **NP**) - parte 1

André Vignatti

DINF- UFPR

Problemas de Busca

Até agora, vimos algoritmos eficientes $\rightarrow$ tempo é uma função polinomial no tamanho da entrada.

Nestes problemas queremos **buscar** uma solução dentre exponenciais possibilidades.

De fato, existem $n!$ permutações para ordenar n números.

O número de cortes mínimos em um grafo é exponencial.

Problemas de Busca

Poderiam ser resolvidos em tempo exponencial ao verificar todas as soluções.

Mas tempo de execução exponencial é inútil na prática.

A jornada de encontrar algoritmos eficientes é sobre maneiras inteligentes de contornar o processo de busca exaustiva.

Até agora vimos os mais brilhantes sucessos dessa jornada: divisão e conquista, algoritmos probabilísticos, etc...

Problemas de Busca

Chegou a hora de conhecer as mais **embaraçosas** e persistentes falhas dessa jornada.

Veremos problemas onde nenhum truque parece possível! $\rightarrow$ busca exaustiva $\rightarrow$ os algorimos mais rápidos **conhecidos** são exponenciais.

Satisfatibilidade

Satisfatibilidade, ou **SAT** é um problema prático de grande importância.

Aplicações:

testes de chips, projeto de computadores, análise de imagem e engenharia de software.

Uma instância de SAT se parece com isso:

$$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z}).$$

Esta é uma fórmula booleana em forma normal conjuntiva (CNF) **E's** de **OU's** .

Satisfatibilidade

CNF:

- É uma coleção de cláusulas (os parênteses),
- Cada cláusula é uma composta de uma disjunção (`ou` lógico, denotada $\vee$) de literais
- Um literal é uma variável booleana (como x) ou sua negação (como $\overline{x}$).

Satisfatibilidade

Atribuição verdadeira: atribuir `falso` ou `verdadeiro` a cada variável para que a fórmula completa seja verdadeira.

Definição: Problema SAT

Dado uma CNF, encontrar uma atribuição verdadeira, ou então dizer que não existe tal atribuição.

Satisfatibilidade

$$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z}).$$

No exemplo: fazendo todas variáveis `verdade`, por exemplo, satisfaz todas as cláusula, exceto a última.

Existe uma atribuição que satisfaça todas as cláusulas?

Pensando um pouco, não é difícil ver que tal atribuição **não** existe. (Dica: As três cláusulas do meio obrigam todas as três variáveis a ter a mesmo valor.)

Satisfatibilidade

Mas como decidir isso para qualquer instância?

Podemos buscar todas as atribuições possíveis, uma a uma $\rightarrow$ fórmulas com n variáveis, 2^n **possíveis atribuições.**

Definindo Problemas de Busca

SAT é um típico problema de busca.

É dada uma **instância** I e o objetivo é **buscar** uma solução S (um objeto que atenda uma especificação particular).

Se nenhuma solução existe, é preciso dizê-lo.

Propriedade de um Problema de Busca: verificação rápida

Em um problema de busca, qualquer solução proposta S a uma instância deve poder ser rapidamente verificado quanto à correção.

Definindo Problemas de Busca

Para formalizar a noção de **verificação rápida**, vamos dizer que há um algoritmo de tempo polinomial que:

1. Recebe como entrada I e S.
2. Decide se S é ou não é solução de I.

Para o SAT: verificar se a atribuição especificada por S realmente satisfaz todas as cláusula I.

- Claramente tempo polinomial ($O(n)$, n é o número de variáveis)

Definindo Problemas de Busca

A definição formal é dada em termos da verificação rápida:

Definição: Problema de Busca

Um **problema de busca** é dado por um algoritmo C que:

- Recebe duas entradas: uma **instância** I e uma **solução proposta** S.
- $C(I, S)$ é executado em tempo polinomial em $|I|$.
- Se S é uma **solução** para I, então $C(I, S) = verdade$, caso contrário $C(I, S) = falso$.

Definindo Problemas de Busca

No SAT, desde os anos 60 há pesquisas para encontrar algoritmo eficiente.

Algoritmos mais rápidos até agora são exponenciais no pior caso.

No entanto, duas variantes do SAT tem algoritmos bons:

1. fórmula de Horn: todas as cláusulas contém no máximo um literal positivo $\rightarrow$ algoritmo guloso.
2. 2SAT: as cláusulas têm apenas dois literais $\rightarrow$ resolvido em tempo linear.

Por outro lado, o 3SAT (cláusulas com no máximo três literais) é difícil de resolver!

O problema do Caixeiro Viajante

Definição: Problema do Caixeiro Viajante (TSP)

Temos n vértices $1, \ldots, n$ e todas as $n(n-1)/2$ distâncias entre eles, bem como um orçamento b.

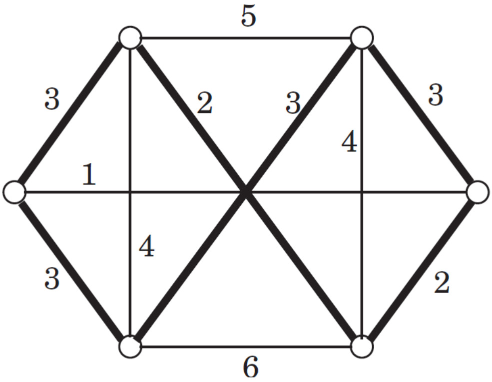
Objetivo: encontrar um **ciclo** que passa por todo vértice exatamente uma vez, com custo total $\leq b$ ou dizer que tal ciclo não existe

Ou seja, buscamos uma permutação $\tau(1), \ldots, \tau(n)$ dos vértices tal que quando percorremos o ciclo nesta ordem, a distância total percorrida é no máximo b:

$$d_{\tau(1),\tau(2)} + d_{\tau(2),\tau(3)} + \ldots + d_{\tau(n),\tau(1)} \leq b$$

O problema do Caixeiro Viajante

Veja a figura de exemplo (apenas algumas das distâncias são mostradas; assuma que as outras são distâncias muito grandes)



O problema do Caixeiro Viajante

Observe que TSP foi definido como um problema de busca: dada uma instância, encontrar o ciclo dentro do orçamento (ou dizer que não existe).

Na realidade, o TSP é um problema de otimização, e não de busca: queremos o ciclo **mais curto** possível

Pergunta:

Mas então por que expressamos o TSP como um problema de busca?

O problema do Caixeiro Viajante

Por uma boa razão.

Nosso plano é **comparar** e **relacionar** problemas.

O framework de problemas de busca é útil: abrange ambos **problemas de otimização** (como o TSP) e **problemas de busca "de verdade"** (como o SAT)

O problema do Caixeiro Viajante

Transformar um problema de otimização em um problema de busca não muda sua dificuldade: ambos se reduzem um ao outro.

Resolver a otimização do TSP também resolve o problema de busca: encontre o melhor ciclo, se estiver dentro do orçamento, devolva-o, senão, não há solução.

Por outro lado, um algoritmo para a busca serve para resolver o problema de otimização:

- Testar com orçamentos de 0 até um certo valor máximo (pseudo-polinomial).

Solução: testar os orçamentos com busca binária!

O problema do Caixeiro Viajante

Uma sutileza: Por que introduzir um orçamento?

Um problema de otimização é basicamente um problema de busca: buscamos a solução que é ótima! Então **porque introduzir o orçamento???**

- O motivo é que a solução de um problema de busca deve ser verificável em tempo polinomial.

Dada uma solução para o TSP, é fácil (tempo polinomial) verificar que "**cada vértice é visitado exatamente uma vez**" e "**tem comprimento total** $\leq b$**.**"

Mas como verificar eficientemente a propriedade "**É ótimo**"?

O problema do Caixeiro Viajante

Não se conhecem algoritmos de tempo polinomial para o TSP.

O problema da árvore geradora mínima (MST), para o qual temos algoritmos eficientes, fornece um **contraste gritante** aqui.

Para enunciá-lo como um problema de busca, temos (como no TSP) todas as distâncias entre pares de vértices um orçamento b. Pede-se para encontrar uma árvore geradora T com peso total $\sum_{(i,j)\in T} d_{ij} \leq b$.

O TSP pode ser considerado como um "primo durão" do MST, onde não é permitida a árvore se ramificar, então ela só pode ser um **caminho**.

- Essa restrição extra torna o problema muito mais difícil!