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

André Vignatti

DINF- UFPR

Problemas de Busca

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

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

Exemplos:

Existem $n!$ permutações para ordenar n números.

O número de cortes em um grafo é 2^n.

Solução: verificar todas as possibilidades $\Rightarrow$ tempo exponencial $\Rightarrow$ inútil na prática.

Problemas de Busca

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...

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

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

Satisfatibilidade

Uma instância de SAT se parece com isso:

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

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

CNF:

- É uma coleção de cláusulas (os parênteses),
- Cada cláusula é 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 `V` ou `F` a cada variável para que a fórmula completa seja `V`.

Definição: Problema SAT

Dado uma CNF, encontrar uma atribuição verdadeira, ou 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: existe uma atribuição que satisfaça todas as cláusulas?

Não existe! (Dica: As três cláusulas do meio obrigam todas as três variáveis a ter a mesmo valor.)

Mas como decidir isso para qualquer instância?

Podemos buscar todas as atribuições possíveis, $\rightarrow$ n variáveis, 2^n **possíveis atribuições** $\rightarrow$ tempo exponencial.

Definindo Problemas de Busca

SAT é um típico problema de busca.

Dada uma **instância** I, o objetivo é:

- **buscar** uma solução S
- se nenhuma solução existe, é preciso dizê-lo.

Propriedade de Problemas de Busca: verificação rápida

Qualquer solução proposta S a uma instância I deve poder ser rapidamente verificada quanto à correção.

Definindo Problemas de Busca

Formalmente, o que é **verificação rápida**?

Definição: Verificação Rápida

Um par de **instância** e **solução (I,S)** tem **verificação rápida** se existe algoritmo C que:

1. C recebe duas entradas: I e S.
2. $C(I, S)$ é executado em **tempo polinomial** em $|I|$.
3. Se S é uma **solução** para I, então $C(I, S) = \textit{verdade}$, caso contrário $C(I, S) = \textit{falso}$.

SAT tem verificação rápida. (Porque?)

Definindo Problemas de Busca

A definição formal é dada em termos da verificação rápida:

Definição: Problema de Busca

Um **problema de busca** é um problema que, para toda instância *I*:

- Se há solução **S**, o par (I,S) tem verificação rápida.
- Se não há solução, devolve NÃO.

SAT é um problema de busca.

Definindo Problemas de Busca

Não se conhecem algoritmos de tempo polinomial para o SAT. Duas variantes do SAT tem algoritmos bons:

1. fórmula de Horn: todas as cláusulas têm 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 e todas as $\binom{n}{2}$ distâncias entre eles, bem como um orçamento b.

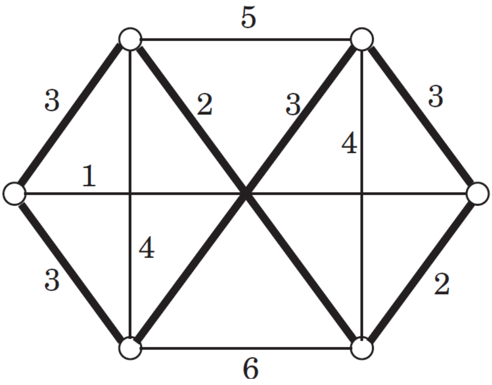
Objetivo: encontrar um **ciclo** que passa por todo vértice exatamente uma vez, com custo total $\leq b$ ou dizer que tal ciclo não existe

Ou seja, buscamos uma permutação $\tau(1), \ldots, \tau(n)$ dos vértices tal que percorridos 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 TSP é um problema de busca. (Porque?)

O problema do Caixeiro Viajante

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

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

Pergunta:

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

O problema do Caixeiro Viajante

Por uma boa razão.

Problemas de busca abrangem ambos **problemas de otimização** (como o TSP) e **problemas de busca "de verdade"** (como o SAT)

Um problema de otimização pode ser transformado em um problema de busca e vice-versa! Ambos se reduzem um ao outro.

O problema do Caixeiro Viajante

($\rightarrow$) Usar TSP de otimização para resolver TSP de busca:

Encontre o melhor ciclo, se $\leq b$, devolva-o, senão, não há solução.

($\leftarrow$) Usar TSP de busca para resolver TSP de otimização:

- Testar com b de 0 até um 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: a solução 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.

MST: dado n vértices, todas as $\binom{n}{2}$ distâncias entre eles e um orçamento b.(igual o TSP!) **Objetivo:** 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: não é permitida a árvore se ramificar, então ela só pode ser um **caminho**.

- Essa restrição extra torna o problema muito mais difícil!