Pesquisa Operacional para o Desenvolvimento
Vol. 9, n. 3, p. 109-118, 2017

ISSN 1984-3534

TUTORIAL CONVIDADO

UMA BREVE INTRODUÇÃO A ALGORITMOS DE APROXIMAÇÃO

Lehilton L. C. Pedrosa[a*]

[a]Instituto de Computação
Universidade Estadual de Campinas - UNICAMP, Campinas-SP, Brasil

RESUMO

Este tutorial convida o leitor a estudar e projetar algoritmos de aproximação para dois problemas com naturezas e estruturas diferentes, descobrindo algumas noções fundamentais para se obter um algoritmo de aproximação e passeando por algumas técnicas básicas existentes na literatura. Os conceitos e definições, que algumas vezes podem parecer bastante densos em livros avançados, são dados aqui somente de maneira amigável, servindo como um primeiro contato com a área e a fim de despertar o interesse e focar no mais importante, que é o projeto de algoritmos. No final, indicamos leituras de livros-textos especializados àqueles interessados em se aprofundar no assunto.

Palavras-chave: Algoritmo de aproximação.

ABSTRACT

This guide briefly introduces the field of approximation algorithms by studying two problems with different natures and structures, discovering the fundamental notions to obtain an approximation algorithm, and walking through basic techniques in the literature. Concepts and definitions, which might seem dense in advanced books, are given here only in a friendly manner. This tutorial serves as a first contact with the area and thus focuses on the project of algorithms. In the end, we refer the reader interested in further studying the subject to specialized textbooks.

Keywords: Approximation algorithms.

*Autor para correspondência. E-mail: lehilton@ic.unicamp.br
DOI: 10.4322/PODes.2017.009

[1]*Todos os autores assumem a responsabilidade pelo conteúdo do artigo.*

1. Por que Projetar Algoritmos de Aproximação?

Diversas vezes, resolver um problema significa encontrar uma melhor solução. Se você quer sair de casa e vai a um cinema, então você quer descobrir qual é o percurso que gasta menos tempo; se você é dono de alguma loja, então quer dar preço a um novo produto de forma a obter o maior lucro. Em um problema computacional simples, para cada entrada existe um conjunto de uma ou mais soluções que satisfazem determinada condição; nesse caso, basta encontrar qualquer uma. Nos *problemas de otimização*, damos um valor para cada uma das soluções, que é definido pela chamada função-objetivo. Dependendo do problema, queremos ou encontrar uma solução de maior valor, ou encontrar uma de menor valor. Uma tal solução é chamada de *ótima*.

O que torna os problemas de otimização particularmente intrigantes é o fato de que não sabemos qual é o valor da solução que buscamos! Isso fica mais evidente quando alguma decisão depende desse valor. No primeiro exemplo acima, suponha que a melhor rota conhecida até o cinema leva meia hora. Se a sessão do filme a que quer assistir começa em uma hora, então você pode decidir por ir seguramente, mas se a sessão começa em dez minutos, saber que não existe rota mais rápida é um argumento definitivo para não ir ao cinema. Similarmente, no segundo exemplo, pode existir um nível mínimo de lucro esperado acima do qual se decide incluir um produto na lista de ofertas da loja. Mais do que isso, saber que a solução encontrada é ótima é evidência de que sua loja se manterá competitiva, já que outra loja não pode obter solução melhor para o mesmo problema.

O que os dois exemplos acima têm em comum é que em algumas situações não precisamos conhecer o valor ótimo para de fato tomar uma decisão: se a rota que conheço já me deixa no cinema antes do início do filme, pouco importa se existe alguma rota mais rápida; se já sei que determinado preço vai levar a lucro suficiente, então é melhor vender o produto do que não o vender. Nesses casos, conhecer o valor ótimo exato não foi essencial para tomar uma decisão. Não obstante, independentemente se o problema é de minimização ou maximização, conhecer o valor da solução ótima adiciona informação útil para tomar decisões mais conscientes e bem fundamentadas. Logo, mais do que encontrar uma solução viável e de valor satisfatório, queremos descobrir qual é o valor de uma solução ótima. Só assim podemos dar um atestado da qualidade de nossa solução e, portanto, da decisão a ser tomada.

Para que uma solução possa de fato ser útil para tomar uma decisão, deve ser possível obtê-la em tempo razoável, ou seja, devemos construir um algoritmo *rápido* que encontre uma solução ótima. O significado de rápido depende do problema, mas um bom indicativo de velocidade é se esse algoritmo executa em tempo polinomial. Dizemos que um algoritmo é polinomial se o número de passos executados, quando ele recebe uma entrada com n bits, é limitado por um polinômio em n. O conjunto de problemas de decisão que possuem um algoritmo polinomial é denotado por P. Um problema em P é considerado tratável.

Para vários problemas, encontrar uma solução viável rapidamente é tarefa mais simples do que encontrar uma solução ótima. É o que acontece, em particular, com os chamados problemas de *otimização combinatória*, quando para uma dada entrada existe um conjunto finito de soluções candidatas e é possível enumerar todas elas. Nesse caso, o que distingue um problema tratável de um não tratável é o quão difícil é selecionar uma solução ótima dentre o conjunto de soluções candidatas, normalmente de tamanho exponencial em n.

Embora conheçamos algoritmos polinomiais para muitos problemas recorrentes, para vários problemas importantes não existem ou não conhecemos algoritmos polinomiais. Um exemplo de problema para o qual há algoritmo rápido é o problema de, dados dois nós de uma rede, encontrar um caminho de menor comprimento que os conecta. Curiosamente, mudanças sutis no enunciado dos problemas podem torná-los intrinsecamente mais difíceis: se ao invés de perguntar por um caminho mais curto, quisermos um caminho mais longo de uma rede, então já não conhecemos algoritmos rápidos. Vamos considerar um problema diferente. Pensemos em uma empresa que tem diversos computadores em sua sede e precisa conectá-los de forma a usar o menor comprimento de cabos possível. Esse problema pode ser prontamente reescrito como o problema da árvore

geradora de custo mínimo: dado um grafo conexo G com pesos w nas arestas, encontrar uma subárvore T com os mesmos vértices de G e que tenha o menor peso possível. Segue um algoritmo polinomial para o problema:

Algoritmo 1: Solução para o problema da árvore geradora de custo mínimo.

```
Entrada: grafo conexo G e peso w das arestas
Saída: árvore T
Crie um subgrafo vazio T;
para cada aresta e de G, em ordem de peso faça
    se T + e não contém ciclo então
        Adicione e a T;
retorna T;
```

O leitor atento pode reconhecer o algoritmo de Kruskal (1956) acima, que claramente executa em tempo polinomial. Foge do escopo desse documento demonstrar formalmente que esse algoritmo está correto, mas é instrutivo entender porque ele devolve uma solução ótima. Enquanto a priori não conhecemos uma solução ótima para o problema, em cada iteração, mantemos a invariante de que T é um subgrafo de alguma solução ótima T^*. Sempre que adicionamos uma aresta e a T, ela é uma aresta mais leve que liga dois subconjuntos X, Y de vértices e, portanto, mais leve que alguma aresta e^* de T^* que deve ligar X a Y. O ponto importante é que, embora não saibamos qual das arestas de G corresponde a e^*, podemos comparar o peso de e com o peso de e^*.

Se o problema da empresa puder ser modelado como o problema da árvore geradora de custo mínimo, então o algoritmo acima fornece uma solução garantidamente ótima e a empresa pode prosseguir com a implantação da rede com a certeza de que tem a rede mais barata. Em algumas situações, no entanto, a descrição acima é apenas uma simplificação do problema de fato enfrentado. Por exemplo, pode ser que existam roteadores instalados em diversos pontos da sede e que poderiam ser utilizados para conectar a rede, assim como os computadores. Podemos modificar ligeiramente o problema da árvore geradora de custo mínimo a fim de descrever esse problema mais precisamente: dado um grafo conexo G com pesos w nas arestas e um subconjunto de vértices R, encontrar uma subárvore T que contenha R e que tenha o menor peso possível. Esse problema é o problema da árvore de Steiner de custo mínimo. Os vértices de R são chamados de terminais e os vértices de T que não estão em R são chamados de vértices de Steiner. No exemplo, os terminais correspondem a computadores e os vértices de Steiner aos roteadores utilizados.

Agora, além de escolher que arestas serão utilizadas, torna-se preciso também decidir quais roteadores serão utilizados. Ao contrário do problema anterior, para o problema de agora não conhecemos algoritmos polinomiais. Mais do que isso, sabemos que o problema de decidir se há uma árvore de Steiner com certo custo é NP-difícil. Isso significa que, se de fato houver algoritmo polinomial para esse problema, então também há para todos os problemas da classe NP. A classe NP contém os problemas cuja solução podem ser verificadas em tempo polinomial, como é o caso do problema da árvore de Steiner e de uma série de problemas igualmente difíceis (Garey e Johnson, 1979). Esse é um resultado com implicações tão fortes que diversos pesquisadores simplesmente não acreditam que haja algoritmo polinomial para todo problema em NP e, portanto, P $\neq$ NP.

Com a dificuldade de projetar um algoritmo rápido que resolva o problema de maneira ótima, uma possibilidade é se ater ao problema da árvore geradora de custo mínimo a fim de obter uma solução que pode não ser ótima, mas que é suficientemente boa e pode ser calculada de maneira rápida. Repare que uma vez escolhidos os vértices de Steiner, o problema se reduz a encontrar uma árvore geradora de custo mínimo no subgrafo induzido pelos terminais e vértices de Steiner. Assim, ao utilizar o algoritmo para o problema da árvore geradora de custo mínimo sobre

os terminais, consideramos a solução em que nenhum vértice de Steiner é escolhido. Obtemos seguinte algoritmo.

Algoritmo 2: Uma solução para o problema da árvore de Steiner.

Entrada: grafo conexo G com peso w nas arestas e subconjunto de vértices R
Saída: árvore T

```
1 Construa o subgrafo G[R] induzido pelos terminais R;
2 Execute o Algoritmo 1 sobre G[R] para obter a árvore T;
3 retorna T;
```

Uma pergunta natural é: quão *próximo* o valor da solução com nenhum vértice de Steiner está do valor ótimo? Para responder essa pergunta precisamos comparar o valor da solução obtida com o valor da solução ótima. Mas como comparar com o valor ótimo, se não sabemos quanto ele vale? Para isso, ao invés de compararmos a nossa solução diretamente com uma solução ótima, utilizamos um limitante sobre o valor ótimo; como queremos saber o quão mais cara é a solução com relação à ótima, precisamos obter um *limitante inferior* sobre o valor ótimo.

Primeiro, vamos estudar as propriedades das instâncias do problema. No exemplo acima, os pesos das arestas correspondem aos comprimentos de cabos a serem instalados. Assim, instalar um cabo entre dois computadores u e v não pode ser mais caro do que instalar um cabo entre u e t e entre t e v. Essa propriedade é a chamada desigualdade triangular de um espaço métrico. Formalmente pode-se escrever o seguinte.

Hipótese 1. *Para todos vértices $u, v, t \in V(G)$, vale $w(u, v) \leq w(u, t) + w(t, v)$.*

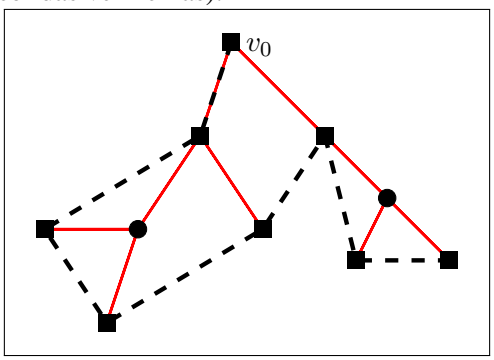
Agora podemos comparar o valor de uma árvore geradora de terminais de custo mínimo T com o valor de uma árvore de Steiner de custo mínimo T^*. O asterisco em T^* é uma notação bastante comum e nos ajuda a lembrar de que estamos falando de uma solução ótima. Iremos mostrar que os pesos de T e de T^* não estão muito distantes. Mais precisamente, para um subgrafo H de G, o peso de H é definido como $w(H) = \sum_{e \in E(H)} w(e)$. Com essa notação, podemos comparar T e T^* da seguinte maneira.

Lema 1. *Seja T uma árvore geradora de $G[R]$ de custo mínimo e T^* uma árvore de Steiner de custo mínimo. Então $w(T) \leq 2w(T^*)$.*

Para demonstrar esse lema, queremos usar o fato de que as duas árvores, T e T^*, têm estruturas similares. Relembre que a diferença entre as duas é que T é uma árvore de Steiner que não contém nenhum vértice de Steiner, enquanto T^* é uma árvore de Steiner que pode conter um ou mais vértices de Steiner. Assim, uma maneira de analisar esse problema é questionar de quanto aumentaria o valor de T^* se removêssemos todos os vértices de Steiner.

Iremos construir um caminho C a partir de T^* que também conecta todos os terminais, mas que não tem nenhum vértice de Steiner. Primeiro vamos construir um passeio fechado P que contém todos os vértices de T^* e percorre cada aresta exatamente duas vezes. Para isso, basta escolher arbitrariamente um terminal v_0 de T^* e realizar uma busca em profundidade a partir de v_0. O passeio realizado pela busca induz um passeio fechado $P = (v_0 v_1 \dots v_s)$, em que s é comprimento do passeio. Como cada aresta de T^* aparece duas vezes em P, o peso total de C é $w(P) = 2w(T^*)$. Para obter o caminho C como descrito acima, devemos remover quaisquer ciclos e quaisquer vértices de Steiner de P. Para isso, começamos um caminho em v_0, que é um terminal por escolha; em seguida, percorremos P em ordem e adicionamos ao caminho cada terminal v_i percorrido que já não tiver aparecido no caminho. Seja C o caminho obtido no final desse procedimento (veja Fig. 1).

Figura 1: Exemplo de caminho (linhas tracejadas) sobre terminais (quadrados) obtido fazendo atalho sobre terminais já percorridos ou vértices de Steiner (discos) de uma solução ótima (linhas sólidas vermelhas).



Fonte: Elaborada pelo autor.

A seguir, argumentamos que o caminho C é mais leve que o passeio P. Intuitivamente, podemos pensar que C faz atalhos em P e, portanto, é um caminho mais curto. Para formalizar isso, primeiro escrevemos o caminho como $C = (v_{e_1} v_{e_2} \dots v_{e_r})$, em que r é o número de terminais. Utilizando a Hipótese 1, podemos mostrar por indução que para todo i, tal que $1 \leq i < r$, vale $w(v_{e_i}, v_{e_{(i+1)}}) \leq w(v_{e_i}, v_{e_i+1}) + w(v_{e_i+1}, v_{e_i+2}) + \dots + w(v_{e_{(i+1)}-1}, v_{e_{(i+1)}})$. Combinando essa desigualdade para cada i, obtemos:

$$\begin{aligned} w(C) &\leq w(v_{e_1}, v_{e_2}) + w(v_{e_2}, v_{e_3}) + \dots + w(v_{e_{(r-1)}}, v_{e_r}) \\ &\leq w(v_1, v_2) + w(v_2, v_3) + \dots + w(v_{s-1}, v_s) \leq w(P). \end{aligned}$$

Como C é uma caminho, C também é uma árvore. Portanto, C é uma árvore cujos vértices correspondem ao conjunto de terminais. Entre todas tais árvores, uma de menor peso é T, do que concluímos que $w(T) \leq w(C)$. Finalmente, podemos comparar o peso de T com o de T^*. Temos $w(T) \leq w(C) \leq w(P) = 2w(T^*)$, que é o que queríamos demonstrar.

Note que na análise acima foi feita a comparação direta de $w(T)$ com $w(C)$ e somente indiretamente com $w(T^*)$. Nessa análise, $w(P)/2$ é um *limitante inferior* para o valor ótimo. Duas observações são importantes: o Algoritmo 2 executa em tempo polinomial e o valor da solução devolvida é no máximo 2 vezes o valor de uma solução ótima. Nessa caso, dizemos que o Algoritmo 2 é uma 2-aproximação para o problema da árvore de Steiner de custo mínimo. De maneira geral, podemos definir o seguinte.

Definição 1. *Considere um problema de minimização $\mathcal{P}$ e um algoritmo A correspondente. Se para cada instância $I \in \mathcal{P}$, tal que o valor de uma solução ótima para I é $OPT(I)$, o algoritmo devolver uma solução de valor $A(I) \leq \alpha OPT(I)$ em tempo polinomial no tamanho de I, então A é uma α-aproximação para $\mathcal{P}$, em que $\alpha \geq 1$ é chamado fator de aproximação.*

Para um problema de maximização podemos definir uma aproximação de forma similar, bastando inverter as desigualdades. Mas será que a análise acima realmente dá uma noção precisa da qualidade da solução obtida pelo Algoritmo 2? Uma observação importante é que a definição acima considera o pior caso, isso é, entre todas as instâncias do problema, um pior caso é uma instância para a qual a razão entre o valor da solução obtida e o valor ótimo é a maior possível. Portanto, para certas instâncias, essa razão pode ser bem menor e, no melhor caso, a solução obtida pode ter o valor ótimo.

Mas a pergunta ainda persiste, existe alguma instância para a qual o Algoritmo 2 obtém uma solução que é de fato 2 vezes pior do que ótimo? Responder essa pergunta significa dar um exemplo de instância. Vamos considerar a seguinte instância formada por $r + 1$ vértices, para

algum inteiro positivo r. O grafo consiste de um vértice central u que não é terminal ligado a r vértices terminais por arestas de peso 1. Todas as demais arestas têm peso 2.

Primeiro, observe que essa instância obedece à Hipótese 1. Por um lado, se executarmos o Algoritmo 2 sobre essa instância, então obteremos uma árvore T composta por $r-1$ arestas, cada uma com peso 2 (note que todas as árvores geradoras de terminais são isomorfas) e, assim, $w(T) = 2(r-1)$. Por outro lado, seja T^* uma solução ótima e considere a estrela S centrada em u cujas folhas são todos os terminais. Como S tem r arestas de peso 1, $w(S) = r$ e, como S é uma árvore de Steiner viável, sabemos que $w(T^*) \leq w(S) = r$. Calculando a razão entre $w(T)$ e $w(T^*)$ temos $w(T)/w(T^*) \geq (2(r-1))/r = 2 - 2/r$. Como r é arbitrário, essa razão pode estar tão próxima de 2 quanto se queira. Concluímos que não existe constante α com $\alpha < 2$ para a qual o Algoritmo 2 é uma α-aproximação. Nesse caso, dizemos que a análise do algoritmo foi *justa*.

2. Bem Próximo do Ótimo

Na seção anterior, vimos um exemplo de uma 2-aproximação para o problema da árvore de Steiner de custo mínimo. Existem outros algoritmos para esse problema que obtêm fatores de aproximação melhores para esse problema, sendo que o menor fator conhecido é 1,39, de Byrka et al. (2013). Para um problema de minimização, queremos um fator de aproximação tão pequeno quanto possível. Como obter uma aproximação com fator 1 é equivalente a resolver um problema NP-difícil, acreditamos que o melhor algoritmo de aproximação tem fator estritamente maior do que 1. Mas quão próximo de 1 podemos chegar? Para o problema da árvore de Steiner sabemos que qualquer algoritmo de aproximação tem fator pelo menos $\frac{96}{95}$, a não ser que P = NP (Chlebík e Chlebíková, 2008).

Para alguns problemas a situação é diferente. Embora resolvê-los de maneira ótima é NP-difícil, podemos obter uma aproximação com fator arbitrariamente próximo de 1. Em outras palavras, para cada constante $\epsilon > 0$, podemos construir um algoritmo que executa em tempo polinomial em n e tem fator de aproximação $1+\epsilon$. A família de todos esses algoritmos é chamada de *esquema de aproximação de tempo polinomial*.

Considere o problema da cobertura de pontos por quadrados. Nesse problema, a entrada é um conjunto de pontos $S = \{(x_1, x_2), (x_1, x_2), \ldots, (x_n, x_n)\}$ no plano euclideano. O objetivo é encontrar um conjunto R de quadrados de largura constante w que contém todos os pontos de S e cuja cardinalidade seja mínima. Por simplicidade, vamos supor que todas as coordenadas, bem como a largura de um quadrado, são inteiras. Além disso, supomos que os pontos estão contidos no retângulo com um extremo na origem e outro extremo em (X, Y), em que X, Y são inteiros positivos limitados somente por um polinômio em n. Iremos chamar essa região de *caixa delimitadora*. Esse problema tem aplicações no desenvolvimento de bancos de dados de imagens, em que os pontos representam informações sobre determinadas características da imagem e o objetivo é armazenar pontos próximos agrupadamente.

Decidir se existe uma cobertura de pontos por quadrados de um tamanho dado k é um problema NP-difícil (Fowler et al., 1981), assim vamos projetar um algoritmo de aproximação. Primeiro, estudamos a estrutura de uma solução ótima: cada quadrado da solução tem as mesmas dimensões e, portanto, pode ser representado simplesmente pela posição do vértice superior direito. Também, embora se possa posicionar um vértice do quadrado em cada ponto do espaço, basta considerar os pontos inteiros positivos dentro da caixa delimitadora. Assim, uma solução pode ser representada por um subconjunto desses pontos.

Uma dificuldade do problema é que o número de pontos na caixa delimitadora é XY e, como X e de Y são arbitrários, o número de soluções candidatas pode ser muito grande. Esse número é 2^{XY}, que corresponde ao número de subconjunto de pontos inteiros na caixa delimitadora. No entanto, se XY fosse limitado por uma constante, então poderíamos listar todas soluções candidatas. Seja r uma constante e considere o caso particular do problema em que $X = Y = rw$. Podemos executar o seguinte algoritmo de força-bruta.

Algoritmo 3: Uma solução para o problema da cobertura de pontos por quadrado considerando $X = Y = rw$.

Entrada: conjunto de pontos S no plano, largura w e constante r
Saída: conjunto R de quadrados

```
1 Inicialize R* como o conjunto de todos os pontos da caixa delimitadora;
2 para cada subconjunto R de pontos da caixa delimitadora faça
3     se |R| < |R*| e a união dos quadrados de R contém S então
4         Atualize R* com R;
5 retorna R*;
```

É claro que o Algoritmo 3 devolve uma solução ótima. Como $X = Y = rw$ é constante, o número de subconjuntos considerados também é constante e, daí, o tempo de execução do algoritmo é polinomial em n, já que em cada iteração, basta verificar se cada ponto é coberto por um quadrado em R.

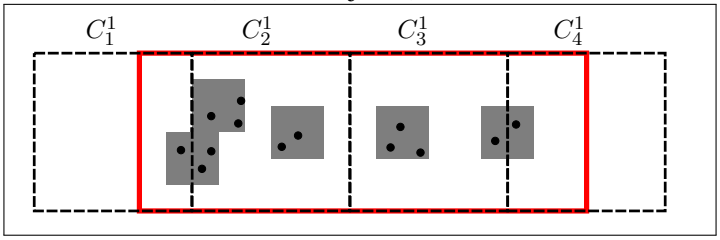
Mas se uma das dimensões não fosse constante, isso é, se tivermos $Y = rw$ constante, mas X for um inteiro arbitrário? Nesse caso, particionamos a caixa delimitadora de forma a obter diversas instâncias adjacentes, cada uma limitada por uma caixa menor de altura e largura rw. A intuição desse algoritmo é que, se existir uma solução ótima em que nenhum quadrado cruza as fronteiras das caixas menores, então a solução ótima em cada subproblema é tão boa quanto uma parte da solução ótima para o problema geral. Assim, queremos encontrar uma partição da caixa delimitadora do problema original tal que o número de quadrados que cruza as fronteiras das caixas menores é o menor possível. Cada deslocamento da primeira caixa induz uma partição distinta (veja a Fig. 2). Utilizamos o seguinte algoritmo, que é chamado algoritmo do deslocamento.

Algoritmo 4: Algoritmo de deslocamento

Entrada: conjunto de pontos S no plano, largura w e constante r
Saída: conjunto de quadrados de menor cardinalidade

```
1 para cada d = 1, 2, ..., r faça
2     Crie um conjunto de caixas C^d = {C_1^d, C_2^d, ...}, tal que o vértice superior direito
        de C_i^d é (dw + (i − 1)rw, rw);
3     para cada C_i^d faça
4         Execute o Algoritmo 3 sobre a instância correspondente e obtenha uma
            solução R_i^d;
5     Faça R^d ser a união de todos R_i^d;
6 retorna o conjunto R^d de menor cardinalidade;
```

Figura 2: Exemplo de partição da caixa delimitadora (linha sólida vermelha) em quatro caixas delimitadoras menores (linhas tracejadas).



Fonte: Elaborada pelo autor.

O seguinte lema mostra que esse algoritmo obtém uma solução que não é muito distante de

uma solução ótima no caso particular em que $Y = rw$.

Lema 2. *Seja R a solução devolvida pelo Algoritmo 4 e R^* uma solução ótima. Então vale $|R| \leq (1 + 1/r)|R^*|$.*

Para demonstrar esse lema precisamos de alguma maneira comparar R^* com R. Como sabemos que R é a união de $R_1^d, R_2^d, \ldots$ para algum d, então iremos utilizar R^* para construir soluções viáveis para cada instância induzida por C_i^d. Mais precisamente, seja V_i^d o conjunto de quadrados de R^* que intersectam C_i^d. Observe que V_i^d é uma solução viável para C_i^d, já que cada ponto em C_i^d deve ser coberto por algum quadrado de R^* que está em V_i^d. Assim, concluímos:

$$|R_i^d| \leq |V_i^d|, \tag{1}$$

já que R_i^d é ótima para C_i^d. Se utilizarmos apenas a desigualdade (1) para um valor de d fixo, obtemos o seguinte:

$$|R| \leq |R^d| = |R_1^d| + |R_2^d| + \cdots \leq |V_1^d| + |V_2^d| + \ldots. \tag{2}$$

No caso afortunado em que os conjuntos V_i^d são disjuntos, saberíamos que a última soma seria no máximo $|R^*|$ e concluiríamos que R é, de fato, uma solução ótima. Acontece que um quadrado pode ter interseção com duas caixas e, portanto, essa soma pode ser tão grande quanto $2|R^*|$. Esse limite já dá um fator de aproximação 2, mas podemos fazer uma análise bem melhor! Para isso, vamos definir I^d como o conjunto de todos os quadrados que cruzam a fronteira de alguma caixa de $\mathcal{C}^d$, ou, equivalentemente, que intersectam duas caixas de $\mathcal{C}^d$.

A observação principal aqui é que os conjuntos I^d são disjuntos. De fato, cada quadrado tem lado w e cada caixa tem lado $rw \geq w$, assim, se um quadrado cruza uma linha de fronteira da partição $\mathcal{C}^d$ para algum d, então esse quadrado não cruza a fronteira de outra partição $\mathcal{C}^{d'}$ para qualquer $d' \neq d$. Agora somamos a desigualdade (2) para cada d e obtemos:

$$r|R| \leq \sum_{d=1}^{r}(|V_1^d| + |V_2^d| + \ldots) = \sum_{d=1}^{r}(|R^*| + |I^d|) \leq r|R^*| + |R^*|.$$

Essa desigualdade implica que $|R| \leq (1 + 1/r)|R^*|$, que é o enunciado do lema.

Mas ainda não resolvemos o problema original. O que acontece quando ambos X e Y são arbitrários? Podemos decompor esse problema em problemas com caixas delimitadoras de altura limitada, mas largura arbitrária, da mesma forma que no caso anterior. Repetimos então o mesmo procedimento, mas utilizando o Algoritmo 4 como sub-rotina. A análise da solução devolvida por esse procedimento é bastante similar à análise anterior. A principal diferença é que como só temos uma aproximação para o subproblema, a equação (1) é substituída por uma equação da forma $|R_i^d| \leq (1 + 1/r)|V_i^d|$ e, como consequência, o fator obtido é $(1 + 1/r)^2$. Note que para cada $\epsilon > 0$, podemos escolher r constante tal que $(1 + 1/r)^2 \leq 1 + \epsilon$. Nos termos definidos no início da seção, obtemos o seguinte lema.

Lema 3. *Para todo $\epsilon > 0$, existe uma $(1 + \epsilon)$-aproximação para o problema da cobertura de pontos por quadrados.*

3. Indo Além

Neste pequeno tutorial, percorremos o caminho para desenvolver algoritmos de aproximação para dois problemas bastante diferentes: o problema da árvore de Steiner de custo mínimo e o problema da cobertura de pontos por quadrados de cardinalidade mínima. O primeiro é um problema típico de projeto de redes, enquanto o segundo é basicamente um problema de geometria. Além de introduzir a área, esses dois exemplos ilustram quão diversos são os problemas tratados sob a ótica de aproximação e, igualmente variados, os algoritmos para tratar esses problemas.

São muitas as técnicas utilizadas para construir um algoritmo de aproximação. Elas incluem algoritmos gulosos, busca local, enumeração, programação dinâmica e métodos probabilísticos. Uma técnica especialmente importante é a formulação de um problema como um programa linear inteiro e a obtenção de sua relaxação linear. Nessa abordagem, os algoritmos podem ser de arredondamento, quando a relaxação é resolvida explicitamente e uma solução inteira é obtida a partir da solução fracionária, ou baseados no método primal-dual, quando é explorada a dualidade do programa linear para obter um limitante para o valor da solução ótima.

Obter algoritmos de aproximação com fatores cada vez melhores é uma tarefa desafiadora e utiliza métodos cada vez mais sofisticados. Por exemplo, obter uma 2-aproximação para o problema da árvore de Steiner é tão simples quanto calcular a árvore geradora de terminais de custo mínimo (Gilbert e Pollak, 1968), enquanto obter uma 1,39-aproximação envolve estudar famílias de soluções para um problema restrito (Borchers e Du, 1997) e o desenvolvimento de técnicas avançadas de arredondamento de programa linear (Byrka et al., 2013).

O problema da cobertura de pontos por quadrados foi estudado por Fowler et al. (1981), que mostrou que tanto esse quanto o problema de empacotamento semelhante são NP-difíceis. O algoritmo de aproximação discutido é de Hochbaum e Maass (1985), que introduziram a técnica de deslocamento (em inglês, *shifting technique*) para esses e diversos outros problemas geométricos no espaço d-dimensional. Uma técnica de deslocamento similar também foi estudada por Baker (1994), que obteve esquemas de aproximação de tempo polinomial para o problema do conjunto independente máximo em grafos planares e outros.

Algoritmos de aproximação formam hoje uma área consolidada e direcionamos o leitor interessado a estudá-los nos livros-textos de Carvalho et al. (2001), também no de Vazirani (2001) ou, mais recentemente, no de Williamson e Shmoys (2011). Obter uma aproximação para um problema significa obter uma solução que não é só boa o suficiente, mas que tem uma garantida de qualidade. Mais do que isso, algoritmos de aproximação são uma ferramenta valiosa para medir, entender e superar as dificuldades intrínsecas de obter uma solução ótima.

Agradecimentos. O Autor agradece o apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo nº 2015/11937-9.

Referências

Baker, B. S. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM (JACM)*, v. 41, n. 1, p. 153–180, 1994.

Borchers, A. e Du, D.-Z. Thek-steiner ratio in graphs. *SIAM Journal on Computing*, v. 26, n. 3, p. 857–869, 1997.

Byrka, J., Grandoni, F., Rothvoss, T., e Sanità, L. Steiner tree approximation via iterative randomized rounding. *Journal of the ACM (JACM)*, v. 60, n. 1, p. 6:1–6:33, 2013.

Carvalho, M. H., Cerioli, M. R., Dahab, R., Feofiloff, P., Fernandes, C. G., Ferreira, C. E., Guimarães, K. S., Miyazawa, F. K., Pina, J. C., Soares, J., e Wabayashi, Y. *Uma Introdução Sucinta a Algoritmos de Aproximação*. Rio de Janeiro: IMPA - Instituto de Matemática Pura e Aplicada, 2001.

Chlebík, M. e Chlebíková, J. The steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, v. 406, n. 3, p. 207–214, 2008.

Fowler, R. J., Paterson, M. S., e Tanimoto, S. L. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, v. 12, n. 3, p. 133–137, 1981.

Garey, M. e Johnson, D. *Computers and Intractability:* A Guide to the Theory of NP-completeness. New York: W. H. Freeman, 1979.

Gilbert, E. N. e Pollak, H. O. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, v. 16, n. 1, p. 1–29, 1968.

Hochbaum, D. S. e Maass, W. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM (JACM)*, v. 32, n. 1, p. 130–136, 1985.

Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, v. 7, n. 1, p. 48–50, 1956.

Vazirani, V. V. *Approximation Algorithms*. Berlin: Springer, 2001.

Williamson, D. P. e Shmoys, D. B. *The Design of Approximation Algorithms*. Cambridge: Cambridge University Press, 2011.