Resumo da Tese de Doutorado de Mark Ziegelmann

Joel Silva Uchoa

20 de junho de 2011

1 Capítulo 3 - Caminhos Mínimos com Recursos Limitados

Neste capítulo nós consideramos uma variante do clássico problema de caminhos mínimos (*SP*), o problema de caminhos mínimos com restrições (sobre recursos) (*CSP*). Diferentemente do problema original *SP*, o *CSP* é $\mathcal{NP}$-completo. Como existem importante aplicações que podem ser modeladas como uma instância do *CSP*, nós estamos interessados em resolver o problema tão eficiente quanto possível.

Nós começamos o capítulo com uma discussão sobre trabalhos anteriores. Como em problemas muito difíceis, nós vamos primeiros vamos olhar uma relaxação de um programa linear que modela o *CSP*. Começando da formulação do programa linear inteiro, nós extraímos um método combinatório para resolver a relaxação com a combinação de intuição geométrica simples e teoria em otimização. No caso de um único recurso nossa abordagem é equivalente a métodos previamente propostos, contudo, nós somos os primeiros a provar um limite de tempo polinomial para o algoritmo. Nós também obtemos o primeiro método combinatório para resolver, de forma exata, a relaxação no caso de múltiplos recursos. Resolver a relaxação nos dá um limite superior e inferior. Nós iremos então rever velhos métodos e apresentar novos métodos para eliminar ao máximo a falga da dualidade para obter um método exato com duas fases para o *CSP*. Nós fechamos o capítulo com uma detalhada comparação experimental dos diferentes métodos.

1.1 Definição do problema

O problema de caminhos mínimos com restrições (de recursos) (*CSP*) requer a computação de um caminho de menor custo que seja limitado por um conjunto de restrições de recursos. Mais precisamente, nós temos um grafo $G = (V, E)$ com $|V| = n$ e $|E| = m$, um vértice de origem s e um vértice de destino t, e k limites de recursos $\lambda^{(1)}$ até $\lambda^{(k)}$. Cada aresta e tem um custo c_e e consume $r_e^{(i)}$ unidades do recurso i, $1 \leq i \leq k$. Assumimos custos e recursos como não negativos. Eles são aditivos ao longo dos caminhos. A meta é encontrar um caminho com menor custo de s até t que satisfaça as restrições de consumo de recursos.

O caso especial $k = 1$ é chamado de caso com recurso simples (*single resource case*) que tem sido mais estudado previamente (ver Sessão 3.2) e nós iremos considerar este caso caso na Sessão 3.3.2. O caso com múltiplos recurso ($k > 1$) irá ser discutido na Sessão 3.3.4.

1.2 Trabalhos anteriores

Nós damos aqui uma visão geral dos trabalhos anteriores sobre o problema de caminhos mínimos com restrições.

1.3 Complexidade

Mesmo que o problema de caminhos mínimos (sem restrições), nós iremos ver que a introdução de uma única restrição de recurso torna o problema $\mathcal{NP}$-completo.

CSP é listado como um problema ND30 (*shortest weight-constrained path*) em Garey e Johnson (1979) e reportado como sendo $\mathcal{NP}$-completo por redução ao problema da Partição. Hanler e Zang (1980) fizeram uma interessante redução ao bem conhecido problema da Moshila (*knapsack problem*) que é muito similar:

Nós temos um conjunto de $n-1$ itens, cada um tem um valor v_j e um peso w_j, para $j = 1, \cdots, n-1$. A meta é colocar itens na Mochila tal que o limite de peso λ não seja excedido e o valor dos itens escolhidos seja maximizado. O problema da Mochila (*KP*) pode ser modelado como:

$$\begin{array}{lrcl} \text{maximizar} & \displaystyle\sum_{j=1}^{n-1} v_j x_j & & \\ \text{sujeito à} & \displaystyle\sum_{j=1}^{n-1} w_j x_j & \leq & \lambda \\ & x_j \in \{0,1\} & & j = 1, \cdots, n-1 \end{array}$$

Onde v_j, w_j, λ são inteiros positivos. Agora nós vamos configurar um grafo direcionado de n vértices com dois arcos paralelos entre cada de vértices j e $j+1$, para $j = 1, \cdots, n-1$. Seja $c^{(1)}_{j,j+1} = M - v_j$, $r^{(1)}_{j,j+1} = w_j$ e $c^{(2)}_{j,j+1} = M$, $r^{(2)}_{j,j+1} = 0$ os parâmetros do primeiro e segundo arco para $j = 1, \cdots, n-1$, respectivamente, onde $M = max\{v_j : j = 1, \cdots, n-1\}$ (ver Figura 3.1).

Figura 3.1

Então é evidente que *KP* pode ser resolvido encontrando um caminho mínimo (em relação ao parâmetro c) do vértice 1 até o vértice n, sujeito a consumir uma quantidade de recursos (parâmetro r) limitada por λ. Como o *KP* é $\mathcal{NP}$-completo, podemos deduzir que o *CSP* é também $\mathcal{NP}$-completo.

1.4 Programação dinâmica recursiva

O primeiro estudo lidando como o *CSP* com um único recurso foi feito por Joksch (1966) que apresentou um algoritmo baseado em programação dinâmica (ver tambem Lawler (1976)).

Nós chamamos um caminho de i até j de r-caminho se o consumo de recursos do ij-caminho é menor ou igual a r. Nós procuramos o λ-caminho mínimo de

1 à n. Digamos que $c_j(r)$ seja o custo do menor r-caminho do vértice i até o vértice j. Então temos a seguinte definição recursiva:

$$c_j(r) = min\{c_j(r-1), \min_{(i,j)\in E, r_{ij}\leq r}\{c_i(r - r_{ij}) + c_{ij}\}\}$$

Setando $c_1(r) = 0$ para $1 \leq r \leq \lambda$ e $c_j(0) = \infty$ para $j = 2, \cdots, n$, nós podemos recursivamente computar o valor ótimo para o problema *CSP* que é dado por $c_n(\lambda)$.

A complexidade de tempo deste método é $O(m\lambda)$ e o consumo de espaço é $O(n\lambda)$. Assim a programação dinâmica dá-nos um pseudopolinômial algoritmo para o *CSP*, e também para o problema da MOCHILA.

Como no caso do problema da MOCHILA, a abordagem por programação dinâmica pode também ser usada para obter um *PTAS* para o *CSP* utilizando arredondamento e escala, que é discutido na próxima sub-sessão.

As abordagens de *labeling* discutidas na Sessão 3.2.4 construídas em cima da programação dinâmica recursiva faz uso do fato de que $c_j(r)$ é uma função escada (?) (*step function*).

1.5 ϵ-aproximação

CSP é fracamente $\mathcal{NP}$-completo, já que nós temos uma simples formulação pseudo polinomial por programação dinâmica. HASSIN (1992) aplicou a técnica padrão de *rounding and scaling* para obter um *fully polynomial ϵ-approximation scheme* (*PTAS*) para o *CSP*. Vamos rever, resumidamente, este método agora.

Na sessão anterior nós vimos um procedimento simples baseado em programação dinâmica. Para nossos propósitos aqui, um outro algoritmo recursivo é mais útil: Digamos que $g_j(c)$ denota o consumo de recursos de um $1j$-caminho mínimo que custa no máximo c. Então a seguinte recursão pode ser definida:

$$g_j(c) = min\{g_j(c-1), \min_{(i,j)\in E, c_{ij}\leq c}\{g_i(c - c_{ij}) + r_{ij}\}\}$$

Setando $g_1(c) = 0$ para $c = 0, \cdots, OPT$ e $g_j(0) = \infty$ para $2, \cdots, n$, nós podemos iterativamente computar o valor ótimo para nosso problema.

Note que OPT não é um valor conhecido a priori, mas temos que $OPT = min\{c|g_n(c) \leq \lambda\}$. Assim, $g_j(c)$ deve ser computado primeiramente para $c = 1$ e $j = 1, \cdots, n$, depois para $c = 2$, e assim sucessivamente até que $g_j(c) \leq \lambda$ pela primeira vez, e então que setamos OPT com o valor atual de c. A complexidade desde algoritmo é $O(mOPT)$.

Agora, digamos que V seja um certo valor, e suponha que queremos testar se $OPT \geq V$ ou não. Um procedimento polinomial que responde essa questão pode ser estendido em um algoritmo polinomial parar encontrar OPT simplesmente usando uma uma busca binário. Como nosso problema é $\mathcal{NP}$-difícil, temos que nos satisfazer com um teste mais fraco.

Tomemos um ϵ fixo, $0 < \epsilon < 1$. Agora, nós explicamos um teste polinomial ϵ-aproximado com as seguintes propriedades: se tal teste devolve uma saída positiva, então definitivamente $OPT \geq V$. Se ele revolver uma saída negativa, então nós sabemos que $OPT < (V + \epsilon)$.

O teste arrendonda o custo c_{ij} das arestas, substituindo seu valor por:

$$\left\lfloor \frac{c_{ij}}{\epsilon V/(n-1)} \right\rfloor \cdot \frac{\epsilon V}{(n-1)}$$

.

Isto diminui todos os custos de arestas em no máximo $\epsilon V/(n-1)$, e todos os custos de caminhos em no máximo ϵV. Agora o problema pode ser resolvido aplicando o algoritmo anterior ao grafo com os custo das arestas escalados para $\lfloor c_{ij}/(\epsilon V/(n-1)) \rfloor$. Os valores de $g_j(c)$ para $j = 2, ..., n$, são primeiro computados para $c = 1$, depois para $c = 1, 2, \cdots$ até que $g_n(c) \leq \lambda$ para algum $c = \hat{c} < (n-1)/\epsilon$, ou $c \geq (n-1)/\epsilon$.

No primeiro caso, um λ-caminho de custo de no máximo

$$\frac{V\epsilon}{n-1}\hat{c} + V\epsilon < V(1+\epsilon)$$

foi encontrado, e segue que $OPT < V(1+\epsilon)$. No segundo caso, cada λ-caminho tem $c' \geq (n-1)/\epsilon$ ou $c \geq V$, então $OPT \geq V$. Assim, o teste funciona como queríamos.

A complexidade de tempo é polinomial cara fixado o ϵ é explicada em seguida: Tomar a parte inteira de um número não negativo no intervalo $\{0, \cdots, U\}$ pode ser feito em tempo $O(\lg(U))$ usando busca binária. Arrendondar o custo das aresta toma tempo $O(mlg(n/\epsilon))$ desde que nós escalamos somente as arestas com custo menor que V (o resultado é no máximo $(n-1)/\epsilon$). Depois executamos $O(n\epsilon)$ iterações do algoritmo acima que novamente toma tempo $O(|E|\lg(n/\epsilon))$. E essa é também a complexidade do procedimento de teste inteiro.

Agora nós usamos este teste para chegar a um $PTAS$ baseado em escalar e arrendondar: Para aproximar OPT nós primeiramente determinamos um limite superior (UB) e um limite inferior (LB). O limite superior UB pode ser setado como a soma das $n-1$ arestas com maior custo, ou o custo da caminho que consome menos recursos. O limite inferior LB pode ser setado como 0 ou o caminho de menor custo.

Se $UB \leq (1+\epsilon)LB$, então UB é uma ϵ-aproximação de OPT. Suponha que $UB > (1+\epsilon)LB$. Seja V um dado valor $LB < V < UB/(1+\epsilon)$. O procedimento TESTE pode agora ser aplicado para melhorar os limites para OPT. Especificamente, ou LB é aumentado ou UB é diminuído para $V(1+\epsilon)$. Executando uma sequência de testes, a razão UB/LB pode ser reduzida. Uma vez que a razão atinga o valor de uma constante predefinida, digamos 2, então uma ϵ-aproximação pode ser obtida aplicando-se o algoritmo por programação dinâmica para o grafo com os custo das arestas escalados para $\lfloor c_{ij}/(LB/(n-1)) \rfloor$. O erro final é no máximo $\epsilon LB < \epsilon OPT$.

A complexidade de tempo para o último passo é $O(|E|n/\epsilon)$. A redução da razão UB/LB é melhor alcançada por busca binária no intervalo (LB, UB) em escala logarítmica. Depois de cada teste nós atualizamos os limites. Para garantir uma rápida redução de razão nós executamos o teste com o valor x tal que $UB/x = x/LB$, que é $x = (LB \cdot UB)^{1/2}$. O número de testes necessários para reduzir a razão para abaixo de 2 é $O(\lg\lg(UB/LB))$ e cada teste toma tempo $O(|E|n/\epsilon)$. HASSIN (1992) mostrou como computar um valor inteiro próximo a $O(UB \cdot LB)^{1/2}$ em tempo $O(\lg\lg(UB/LB))$. Isto dá uma complexidade de tempo, total de $O(\lg\lg(UB/LB)(|E|(n\epsilon) + \lg\lg(UB/LB)))$ para este algoritmo ϵ-aproximado como um $PTAS$ para o CSP.

Hassin (1992) também mostrou uma $PTAS$ cuja a complexidade depende somente do número de variáveis e $1/\epsilon$ e possui complexidade de tempo de $O(|E|(n^2/\epsilon)\lg(n/\epsilon))$. O melhor $PTAS$ foi obtido por Phillips (1993) que atingiu a complexidade de tempo de $O(|E|(n/\epsilon) + (n^2/\epsilon)\lg(n/\epsilon))$.