O Problema da Atribuição Dupla de Custo Mínimo

Vinícius F. dos Santos[1] **, Sebastián Urrutia** [1]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) Belo Horizonte, MG, Brazil

{viniciussantos,surrutia}@dcc.ufmg.br

Abstract. *In this work we introduce the Minimum Cost Double Assignment Problem, in which the input is a pair X, Y of n-dimensional arrays of costs and one has to find a pair (i, j) for which a given function $f(i, j)$ is true, minimizing $x_i + y_i$. We present exact polynomial time algorithms for its solution and discuss the best scenarios for each algorithm.*

Resumo. *Neste trabalho introduzimos o Problema da Atribuição Dupla de Custo Mínimo, no qual a entrada é um par X, Y de vetores n-dimensionais de custos e é necessário encontrar um par (i, j) para o qual uma função $f(i, j)$ dada é verdadeira, minimizando $x_i + y_j$. Apresentamos algoritmos polinomiais para a solução do problema e discutimos os melhores cenários para os algoritmos apresentados.*

1. Introdução

Considere duas tarefas a serem executadas. Ambas as tarefas precisam de determinados recursos (tempo, espaço, ferramentas, matéria-prima, mão-de-obra, etc) para a sua execução. Cada uma das tarefas pode ser executada em n modos distintos e cada um destes modos tem um custo associado e utiliza quantidades diferentes de recursos. Como recursos são compartilhados pelas duas tarefas, alguns pares de modos (um de cada tarefa) são incompatíveis.

No Problema da Atribuição Dupla de Custo Mínimo, é necessário atribuir um modo de execução a cada tarefa de forma que os modos selecionados sejam compatíveis e a soma dos custos dos modos selecionados seja tão pequena quanto possível.

Como exemplo prático, considere o Problema de Alocação de Cais e Guindaste [Meisel and Bierwirth 2009] no qual duas embarcações podem precisar utilizar as instalações de um porto para entregar um número de contêineres e receber outros. Cada embarcação demanda espaço, tempo, mão-de-obra e equipamentos específicos para a execução das tarefas. A quantidade e a qualidade dos recursos designados a esta tarefa (o modo de execução) determina o custo financeiro que deve ser pago à administração do porto e estes recursos devem ser compartilhados pelas duas embarcações. Neste caso, o dono das embarcações gostaria de resolver o Problema da Atribuição Dupla de Custo Mínimo.

Neste trabalho apresentaremos diferentes soluções de tempo polinomial para este problema. Iniciaremos com uma formalização do mesmo e, em seguida, apresentaremos diferentes estratégias de solução. Todas as estratégias apresentadas são originais. Finalmente, concluiremos com direções de trabalhos futuros.

2. Formalização

Denotaremos por $[n]$ o conjunto $\{1, \ldots, n\}$. Sejam X e Y vetores inteiros ordenados de dimensão n e $f : [n] \times [n] \to \{0, 1\}$ uma função booleana, que recebe índices dos vetores como parâmetros. Neste trabalho assumiremos que f pode ser computada em tempo constante. Se $f(i, j) = 1$, dizemos que i e j são compatíveis.

O Problema da Atribuição Dupla de Custo Mínimo consiste em determinar um par de índices compatíveis de menor soma, isto é, encontrar um par (i, j) tal que $f(i, j) = 1$ e $x_i + y_j$ seja tão pequeno quanto possível. Se tal par não existe, então o problema não possui solução.

Em determinados casos, estaremos interessados em saber qual a posição k do primeiro par compatível quando todos os pares estão ordenados pela soma de seus elementos correspondentes nos vetores. Definimos $k = n^2$ se não existe par compatível. Por simplicidade e restrições de espaço, vamos assumir que não existem duas somas iguais. O caso mais geral pode ser tratado sem grandes mudanças nos algoritmos.

3. Problemas relacionados

Alguns problemas similares já foram considerados na literatura. Dados vetores X e Y, a soma cartesiana $X+Y$ é a matriz cujo elemento na linha i coluna j tem valor x_i+y_j. Três problemas foram bastante estudados para matrizes $X + Y$: ordenação, busca e seleção.

No problema de ordenação, a entrada são os vetores X e Y e deseja-se ordenar totalmente os elementos da matriz $X + Y$. Note que há n^2 elementos a serem ordenados, assim, um limite inferior natural é $\Omega(n^2)$. Em [Fredman 1976] é fornecida uma prova não construtiva de que os elementos de $X + Y$ podem ser ordenados utilizando-se $O(n^2)$ comparações, mas apenas em [Lambert 1992] é apresentado o primeiro algoritmo que resolve o problema neste número de comparações, embora o tempo total do algoritmo seja $O(n^2 \log n)$.

O problema de busca consiste em, dado um valor z, encontrar índices i e j tais que $a_i + b_j = z$. Em [Cosnard et al. 1990] é demonstrado que este problema pode ser resolvido em tempo $\Theta(n)$.

Finalmente o problema de seleção tem como entrada os vetores X e Y e um inteiro k, $1 \leq k \leq n^2$ e consiste em determinar o índices i e j tais que $x_i + y_j$ seja o k-ésimo elemento de ordenação total dos elementos de $X + Y$. Algoritmos para este problema foram apresentados resolvendo-o em tempo $O(n)$ [Frederickson and Johnson 1982, Mirzaian and Arjomandi 1985].

4. Algoritmos

Algoritmo trivial (AT). A ideia mais ingênua consiste em avaliar $f(i, j)$ para todos os n^2 pares de índices válidos e retornar o par que minimiza a soma dos valores correspondentes nos vetores. Este algoritmo claramente leva tempo $O(n^2)$. Uma otimização simples seria evitar testes de pares dominados por um par compatível já encontrado. Isto é, se (i, j) é uma solução válida já encontrada, não é necessário testar nenhum par $(i + s, j + t)$, para s e t positivos.

Algoritmo com fila de prioridades (AFP). Outra estratégia consiste em avaliar a função f em ordem crescente de $x_i + y_j$. Assim, tão logo um par válido seja encontrado, o

algoritmo encerraria a busca, uma vez que esta seria a solução ótima. Isto pode ser feito através de uma fila de prioridades. Inicialmente, apenas o par $(1, 1)$ é colocado na fila de prioridades e, a cada iteração, um par (i, j) é removido da fila e os pares $(i+1, j)$ e $(i, j+1)$ são adicionados à fila, se ainda não tiverem sido adicionados anteriormente e se os índices forem no máximo n. Alguns detalhes de implementação devem ser observados, mas o algoritmo pode ser implementado em tempo $O(k \times \log(k))$ utilizando heaps binomiais.

Note que nenhum dos algoritmos apresentados domina o outro. Para valores pequenos de k ($k = O(n)$), AFP é melhor que AT. Por outro lado, no caso em que $k = O(n^2)$, AT é melhor que AFP. Motivados por esta não dominância, na próxima seção apresentaremos algoritmos cuja complexidade depende de k, como em AFP, mas que domina AT, igualando sua complexidade no pior caso, porém melhor no caso geral.

5. Algoritmos de complexidade $O(n \times \sqrt{k})$

A ideia dos dois algoritmos apresentados a seguir é avaliar na i-ésima iteração todos os pares com soma menor ou igual que a do par (i, i). Dado que o a menor posição possível do par (i, i) na ordenação de todos os pares é i^2 os algoritmos garantidamente avaliam o par na posição k no pior caso na iteração $\sqrt{k}$.

Os algoritmos possuem algumas diferenças fundamentais. O Algoritmo 1 é conceitualmente mais simples e sempre avalia $O(n)$ elementos por iteração, e o número de iterações é limitado por $\sqrt{k}$. Já o Algoritmo 2 em algumas instâncias pode avaliar apenas $\Theta(k)$ posições, porém ao custo de espaço adicional da ordem de $\Theta(n)$. Ainda assim, embora o Algoritmo 2 possa ser executado em tempo $o(n \times \sqrt{k})$ em algumas instâncias, é possível construir instâncias nas quais o algoritmo também leva tempo $\Theta(n \times \sqrt{k})$.

6. Trabalhos futuros

O principal problema que deixamos em aberto é determinar se é possível resolver o problema em tempo linear em k e n, como ocorre em alguns problemas relacionados. Além disso, como trabalho futuro, podemos mencionar a comparação, analítica e experimental, em instâncias reais e artificiais, dos algoritmos propostos.

Referências

Cosnard, M., Duprat, J., and Ferreira, A. G. (1990). The complexity of searching in $X+Y$ and other multisets. *Information Processing Letters*, 34(2):103–109.

Frederickson, G. and Johnson, D. (1982). The complexity of selection and ranking in $X + Y$ and matrices with sorted columns. *J. of Comput. and Syst. Sci.*, 24(2):197–208.

Fredman, M. L. (1976). How good is the information theory bound in sorting? *Theoretical Computer Science*, 1(4):355–361.

Lambert, J.-L. (1992). Sorting the sums $(x_i + y_j)$ in $O(n^2)$ comparisons. *Theoretical Computer Science*, 103(1):137–141.

Meisel, F. and Bierwirth, C. (2009). Heuristics for the integration of crane productivity in the berth allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):196 – 209.

Mirzaian, A. and Arjomandi, E. (1985). Selection in $X + Y$ and matrices with sorted rows and columns. *Information processing letters*, 20(1):13–17.

Algoritmo 1: Avalia uma linha e uma coluna por iteração.

```
início
    se f(1,1) = 1 então
        retorna (1,1)
    i ← 2, best ← +∞
    enquanto i ≤ n e best > x_{i-1} + y_{i-1} faça
        para j ← i até n faça
            se f(i-1, j) = 1 e x_{i-1} + y_j < best então
                best ← x_{i-1} + y_j
                bestpair ← (i-1, j)
            se f(j, i-1) = 1 e x_j + y_{i-1} < best então
                best ← x_j + y_{i-1}
                bestpair ← (j, i-1)
        se f(i,i) = 1 e x_i + y_i < best então
            best ← x_i + y_i
            bestpair ← (i,i)
        i ← i + 1
    se best < +∞ então
        retorna bestpair
    senão
        retorna "não existe solução"
```

Algoritmo 2: Avalia apenas elementos menores que (i, i)

```
início
    i ← 1, best ← +∞
    para j ← 1 to n faça
        contourline[j] ← 1
    enquanto i ≤ n e best = +∞ faça
        para j ← 1 to n faça
            enquanto contourline[j] < n e x_j + y_{contourline[j]} ≤ x_i + y_i faça
                se f(j, contourline[j]) = 1 e x_j + y_{contourline[j]} < best então
                    best ← x_j + y_{contourline[j]}
                    bestpair ← (j, contourline[j])
                countline[j] ← contourline[j] + 1
        i ← i + 1
    se best < +∞ então
        retorna bestpair
    senão
        retorna "não existe solução"
```