MC558-Teste3

Lucca Ferreira Paiva 240229

December 2021

Questão 1

Corretude

Para definir o conjunto independente de vértices máximo iremos utilizar o um algoritmo de fluxo máximo para determinar o corte mínimo.

Seja G um grafo bipartido, tal que V_1 seja um dos conjunto de vértices e V_2 o outro, de forma que $V_1 \cap V_2 = \phi$, e E é o conjunto de arestas. Além disso, existe uma função $w(v)$ que indica o peso do vértice v. Vamos começar construindo o grafo em que desejamos realizar a análise de fluxo, para tal, iremos criar dois vértices, uma fonte s e um sorvedouro t. Iremos criar o grafo G', que começa como uma copia de G, e adicionaremos a ele os vértices s e t. Depois, vamos ligar o vértice s a todos os vértices de V_1, e a capacidade de cada arestada e será dada pelo peso do vértice ao qual ela se liga, tal que para cada $v \in V_1$ sera criada uma aresta e de capacidade $w(v)$. Faremos algo similar para o t, mas ele sera ligado aos vértices do outro conjunto, para cada $v \in V_2$ será criada uma aresta e de capacidade $w(v)$. Para todas as arestas do conjunto inicial E, elas terão a capacidade de infinito, para $e \in E$ temos $w(e) = \infty$.

Agora iremos procurar pelo corte mínimo de s para t na rede construída G', por meio de algum algoritmo de fluxo máximo, como o de Edmonds-Karp. O valor do corte é o peso da menor cobertura de vértices. Vamos classificar as possíveis arestas do corte em 3 tipos:

1. Arestas originais
2. Arestas de $s$para V_1
3. Arestas de V_2 para t

O corte mínimo não passa pelas arestas do tipo 1, pois, se passassem, existira um caminho de capacidade miníma igual a infinito, o que indicaria que o fluxo máximo da rede é infinito. Isso porque o peso de cada vértice é um valor finito, e o fluxo que sai de s é composto pela soma dos pesos dos vértices de V_1, e este valor é finito. Dessa forma, não pode haver um fluxo máximo infinito, de tal forma que não há arestas do tipo 1 no corte mínimo.

Sendo assim, as arestas do corte mínimo podem ser apenas do tipo 2 ou 3. Seja $u \in V_1$, e seja E_u um subconjunto de arestas de E' cuja origem estão nos vértices vizinhos de u, com V_u sendo o conjunto de vértices vizinhos de u. Temos que ou a aresta $e_s = (s, u)$ esta no corte, ou todas as arestas E_u estão no corte.

Seja $c_{in} = w(u)$, e $c_{out} = \sum^{V_u} w(v_i)$, temos 3 possibilidades:

1. $c_{in} < c_{out}$

   Se este for o caso, então o corte mínimo passa pela aresta $e = (s, u)$, e portanto as arestas de E_u não passam, já que o fluxo que chega a elas de u é limitado pelo próprio fluxo que chega em u.

2. $c_{in} > c_{out}$

   Se este for o caso, então o corte mínimo passa por todas as arestas de E_u, isso porque o fluxo de u para t é limitado pelas arestas de E_u.

3. $c_{in} = c_{out}$

   Neste caso o corte mínimo é limitado por ambas as quantidades, que por serem iguais e por estamos interessados no valor, é indiferente qual conjunto escolhemos, mas neste caso vamos optar sempre pelo corte que passa pela aresta (s, v).

Essa mesma lógica também é válida para os vértices $v \in V_2$.

Dessa forma, para cada vértice uma das duas possibilidades é verdadeira: uma de suas arestas passa pelo corte ou as arestas de seus vizinhos passam pelo corte. Isso é consequência direta do que foi listado acima, pois

para cada vértice $v \in V$ há apenas uma única possibilidade, ou o corte passa pela aresta (s, v) ou ele passa pelo conjunto E_u. Agora, vamos dividir os vértices em dois novos grupos, S_1 e S_2, de forma que para todo vértice $v \in V$ se uma aresta adjacente a v foi cortada $v \in S_1$, caso contrário, $v \in S_2$. Segue que $S_1 \cap S_2 = \phi$. Como para todo vértice v, temos que $v \in S_1$ e seus vizinhos pertencem a S_2, ou o contrário, temos dois conjuntos independentes.

Se W_1 for a o peso dos vértices de S_1, e W_2 for o peso dos vértices de S_2, temos que $W_1 \leq W_2$. Isso decorre porque S_1 são os vértices cortados pelo corte mínimo, de tal forma que esta é a capacidade limitante de fluxo. Logo, a capacidade W_2 tem que ser no mínimo igual a W_1. Portanto, S_2 representa o conjunto independente máximo.

Sendo assim, o conjunto independente máximo é S_2.

Complexidade de Tempo

Para a complexidade de tempo temos que analisar os seguintes passos:

1. **Criar grafo G'**

   Para criar o grafo G' temos que isso pode ser feito em tempo polinomial, pois começa,os copiando G, e adicionamos $V_1 + V_2$ arestas, sendo que tudo isso pode ser feito em tempo polinomial.

2. **Corte Mínimo em G'**

   Podemos obter o corte mínimo por um dos algoritmos visto em aula, sendo que eles gastam tempo polinomial.

3. **Analise do Fluxo para obter o conjunto independente**

   Por ultimo, temos que realizar a análise dos vértices, mas vamos passar uma vez por cada vértice, o que é claramente de tempo polinomial.

Conclusão

Concluímos portanto que o algoritmo gasta tempo polinomial.

Questão 2

Corretude

Decisão

Primeiramente, vamos re-escrever o problema de escalonamento de tarefas em maquinas uniformes - ETMU como um problema de decisão. Para tal, queremos saber se dada uma entrada existe um makespan k, dessa forma, as possíveis respostas são 0 para se não existe, e 1 se existir uma configuração cujo makespan é k. Agora possuímos o problema como um de decisão.

NP

Vamos agora provar que o problema pertence a NP. Para tal, precisamos mostrar que uma solução pode ser verificada em tempo polinomial. Suponha um certificado A, com duas entradas, da forma A(x,y), em que x corresponde a instância do problema, e y a uma possível solução. A entrada x irá conter o conjunto T de tarefas, bem como o valor ti de cada tarefa, o número m de maquinas idênticas, e, por ultimo, um inteiro k, que corresponde ao makespan máximo que queremos encontrar. A entrada y, irá conter m conjuntos de tarefas, um para cada máquina, que corresponde ao makespan dessa configuração.

A verificação então é bem simples, basta percorrer cada um dos conjuntos de y verificando duas coisas: se as tarefas desse conjunto pertencem a T, e não estão repetidas, e o tempo total de cada configuração. No final, comparamos o tempo total de cada máquina, determinamos o maior deles e verificamos se é igual que K. Se for igual, temos que a saída é 1, caso contrario, a saída é 0. Esse processo claramente pode ser feito em tempo polinomial.

NP-Completo

Para provar agora que o problema é NP-Completo iremos reduzir um problema NP-Completo para este, mais especificamente utilizaremos o o problema da Soma do Subconjunto - SS (Subset Sum). Este problema consiste em, dado um conjunto de números S, verificar se existe um subconjunto S_1 cuja a soma é k.

Para transformarmos o problema de SS em uma instancia de ETMU precisamos realizar algumas coisas antes. Primeiro, precisamos analisar o valor de k com relação ao conjunto, e vamos definir W como a soma de todos os valores do conjunto S, temos duas possibilidades:

1. $k < W - k$
2. $k \geq W - k$

Como para o problema de ETMU estamos interessados no makespan, caso $k < W - k$, temos que realizar uma mudança, pois o valor de S_1 é maior que o de $S - S_1$. Dessa forma, vamos definir k' como:

$$k' = max\{k, W - k\}$$

Agora estamos prontos para instanciar SS como um ETMU. O conjunto de tarefas T corresponde ao conjunto S, em que o peso t_i de cada elemento é o próprio valor do elemento i de S. O número de máquinas idênticas corresponde a $m = 2$, pois desejamos apenas dois conjuntos, S_1 e o outro será $S - S1$. Por ultimo, o makespan de K será o valor k' calculado.

Executando o algoritmo para EMTU (a partir do algorítimos de decisão é possível construir um que retorne os valores de cada conjunto), temos as seguintes possibilidades:

1. **Existe configuração com makespan = K**

   Então teremos como resposta dois conjuntos de configurações para as máquinas. E o algoritmo irá retornar os dois conjuntos desejados. Basta agora em SS verificar qual dos dois apresenta soma dos elementos igual a k, este será o conjunto S_1.

2. **Não existe configuração com makespan = K**

   Então não há conjunto que satisfaça a soma k. Dessa forma, para o problema original SS também não existe conjunto S_1 com a soma igual a k. Isso porque temos duas possibilidades:

   (a) $k' = K = k$
   Nesse caso é trivial perceber que não existe configuração com soma k.

   (b) $k' = K = W - k$
   Neste caso, como W corresponde a soma do conjunto, e não existe configuração de tamanho $W - k$, não vai existir o complemento de tamanho k.

   Concluímos então que para o problema SS não há solução de um subconjunto de tamanho k.

Portanto, agora em SS caso exista um subconjunto S_1 de S com a soma k o teremos, e caso contrario saberemos que a resposta foi 0.

Complexidade de Tempo

Basta mostrar que a transformação de SS para EMTU pode ser feita em tempo polinomial, assim como a transformação reversa.

1. $SS \Rightarrow EMTU$

   Precisamos apenas determinar a soma de todos os elementos de S.

2. $EMTU \Rightarrow SS$

   Dado dois conjuntos, basta identificar qual deles tem soma dos elementos igual a k.

Podemos ver claramente que ambas as operações podem ser feitas em tempo polinomial.

Conclusão

Podemos ver então que o problema de escalonamento de tarefas em maquinas uniformes é NP-Completo.

Questão 3

Corretude

NP

Primeiro precisamos provar que há um certificado em tempo polinomial para o problema de Programação Linear de Inteiros - PLI. Ou seja, dado um vetor x, uma matriz A e o vetor inteiro b, verificar que $Ax \leq b$. Como A tem dimensões $m \times n$, e x tem tamanho n, podemos fazer uma simples multiplicação de matrizes, e comparar o resultado com o vetor b, tudo em tempo polinomial. Dessa forma, este problema é NP.

NP-Completo

Agora, precisamos provar que algum problema NP-Completo pode ser reduzido para PLI para provar que PLI é NP-Completo. Utilizaremos o problema de Satisfação de 3 Forma Normal Conjuntiva - 3-CNF-SAT, ou seja, $3 - CNF - SAT \leq_P PLI$.

Seja α uma formula 3-CNF com n variáveis e k clausulas. Vamos construir uma instancia de PLI da seguinte forma, seja A uma matriz de $2n + k$ por $2n$. Para $1 \leq i \leq k$, temos $A_{i,j} = -1$ se $1 \leq j \leq k$ e a clausula C_i contem o literal x_j, caso contrario, tem-se $A_{i,j} = 0$. Para $n + 1 \leq j \leq 2n$, temos $A_{i,j} = -1$ se a clausula C_i contem o literal $\neg x_{j-n}$, caso contrario, tem-se $A_{i,j} = 0$. Quando $k + 1 \leq i \leq k + n$ temos $A_{i,j} = 1$ se $i - k = j$ ou $i - k = j - n$, e caso contrario $A_{i,j} = 0$. Quando $k + n + 1 \leq i \leq k + 2n$ temos $A_{i,j} = -1$ se $i - k - n = j$ ou $i - k - n = j - n$, e caso contrario $A_{i,j} = 0$. Deixe b ser um vetor de tamanho $k + 2n$. As primeiras k entradas de b serão iguais a -1, as próximas n entradas iguais a 1, e as ultimas n entradas para -1. Podemos construir tanto A quanto b em tempo linear.

Agora, vamos mostrar que α tem uma configuração que satisfaz os requisitos se e somente se existe um vetor de 0-1 tal que $Ax \leq b$. Primeiro, suponha que α satisfaça a designação. Para $1 \leq i \leq n$, se x_i é verdadeiro, faça $x[i] = 1$ e $x[n+1] = 0$, se x_i for falso, faça $x[i] = 0$ e $x[n+1] = 1$. Como a clausula C_i esta satisfeita, deve existir um inteiro literal que a satisfaça. Se for x_j, então $x[j] = 1$ e $A(i, j) = -1$, então temos contribuição de -1 da i-ésima linha de b. Como toda entrada na parte superior da submatriz de A dada por k por $2n$ é não-positiva e toda entrada de x é não-negativa, não pode haver contribuição positiva para a i-ésima linha de b. Dessa forma, garantimos que que a i-ésima linha de Ax é no máximo -1. O mesmo é valido se o literal $\neg x_j$ faz a clausula i verdadeira. Para $1 \leq m \leq n$, no máximo um entre x_m e $\neg x_m$ pode ser verdadeiro, então no máximo um de $x[m]$ e $x[m + n]$ pode ser verdadeiro. Quando multiplicamos a linha $k + m$ por x, obtemos o número 1 entre $x[m]$ e $x[m + n]$. Logo, a (k + m)-ésima linha de b é no máximo 1, como necessário. Finalmente, ao multiplicar a linha $k + n + m$ de A por x, obtemos 1 negativo vezes o número de 1s entre $x[m]$ e $x[m + n]$. Como isso é pelo menos 1, a (k + n + m)-ésima linha de b é no máximo -1. Sendo assim, todas as desigualdades estão satisfeitas, então x é uma solução 0-1 para $Ax = b$.

Agora, é preciso mostrar que qualquer 0-1 solução para $Ax = b$ também satisfaz as restrições. Seja x uma dessas soluções. As desigualdades asseguradas pelas ultimas $2n$ linhas de b garantem que exatamente um de $x[m]$ e $x[n + m]$ é igual a 1 para $1 \leq m \leq n$. Em particular, isso quer dizer que cada x_i é ou falso ou verdadeiro, mas não ambos. Seja este o nosso candidato para satisfazer as restrições. Pela construção de A, temos uma contribuição de -1 para a linha i de Ax toda vez que um literal em C_i é verdadeiro, baseado no nosso candidato as restrições, e uma contribuição de 0 toda vez que um literal é falso. Como a linha i de b é -1, isso garante que pelo menos um literal seja verdadeiro por restrição de cada clausula. Como isso é valido para cada uma das k clausulas, a restrição candidata é uma restrição satisfeita. Sendo assim, LPI é NP-completo.