UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FELIPE ZORZO PEREIRA

Proposta inicial para o trabalho final da disciplina Otimização Combinatória

Proposta apresentada como requisito parcial para a obtenção de nota no trabalho final da disciplina Otimização Combinatória

Orientador: Prof. Dr. Marcus Ritt

Porto Alegre
2018

1 FORMULAÇÃO MATEMÁTICA

A formulação matemática utilizada para o problema dos grupos balanceados de maior valor foi:

$$maximiza \sum_{i\in[n]}\sum_{k\in[n]}\sum_{j\in[g]} c_{i,k,j}d_{i,k}$$

$$sujeito\ a \sum_{i\in[n]} x_{i,j}p_i \leq U_j, \forall j \in [g],$$

$$\sum_{i\in[n]} x_{i,j}p_i \geq L_j, \forall j \in [g],$$

$$\sum_{j\in[g]} x_{i,j} = 1, \forall i \in [n],$$

$$c_{i,k,j} \leq \frac{x_{i,j} + x_{k,j}}{2}, \forall\ i \in [n], k \in [n], j \in [g],$$

$$c_{i,k,j} \in \{0,1\}, \forall\ i \in [n], k \in [n], j \in [g],$$

$$x_{i,j} \in \{0,1\}, \forall\ i \in [n], j \in [g].$$

Nessa formulação,

$$x_{i,j} = \begin{cases} 1, se\ vértice\ i \in [n]\ está\ no\ grupo\ j \in [g] \\ 0, caso\ contrário \end{cases}$$

$$c_{i,k,j} = \begin{cases} 1, se\ vértices\ i \in [n]\ e\ k \in [n]\ estão\ no\ grupo\ j \in [g] \\ 0, caso\ contrário \end{cases}$$

$d_{i,k}$ *representa o valor da aresta entre os vértices* $i \in [n]$ *e* $k \in [n]$,

p_i *representa o peso do vértice* $i \in [n]$,

U_j *representa o limite superior do peso total de um grupo* $j \in [g]$,

L_j *representa o limite inferior do peso total de um grupo* $j \in [g]$,

n *é o número de vértices e*

g *é o número de grupos*.

Notar que, caso não exista aresta entre dois vértices, consideramos que há uma aresta de valor 0.

2 DEFINIÇÃO DOS ELEMENTOS DO SIMULATED ANNEALING

A **representação de uma solução** será uma lista de listas. Cada uma das listas representará um grupo da instância do problema, e cada elemento da lista será o índice de um dos vértices que está no grupo representado por essa lista.

A **solução inicial** é feita de forma aleatória. O algoritmo adiciona vértices aleatórios da instância do problema a um grupo até que o peso desse grupo seja maior ou igual a seu limite inferior. Após realizarmos isso para todos os grupos, se ainda há vértices sem grupo, o algoritmo tenta adicioná-los a um. Após todos os vértices terem sido adicionados a um grupo, o algoritmo testa se a solução é factível. Se sim, retorna a solução; caso contrário, tentamos fazer a distribuição de vértices novamente. O algoritmo roda por um número fixo de iteraçòes; se tal número é extrapolado, o algoritmo para e retorna uma mensagem dizendo que uma solução inicial não pôde ser criada.

Vizinhanças: Até o momento estou apenas gerando vizinhos aleatórios. Para tanto, seleciono dois grupos diferentes randomicamente. Um deles perde um vértice randômico, que vai para o outro grupo. Se tal perturbação é uma solução factível, retorno ela; caso contrário, repito o processo. Depois de um número fixo de iterações, caso eu não consiga criar um vizinho novo, apenas retorno a solução que estávamos tentando perturbar. Além dessa vizinhança estocástica, pretendo testar outras heurísticas antes da entrega final, e discutir qual vizinhança gerou melhores resultados no relatório final.

Por exemplo, pretendo fazer uma vizinhança que escolha sempre o grupo com maior diferença entre seu peso total e seu limite inferior; se possível, retiro desse grupo o seu vértice de menor peso e o boto em outro grupo, o que possuir maior diferença entre seu peso total e seu limite superior. Se a solução provinda dessa perturbação for factível, retorno-a; caso contrário, tento retirar o vértice de menor peso do segundo grupo de maior diferença entre seu peso total e seu limite inferior. Caso não consiga retirar o vértice de menor peso de nenhum grupo, simplesmente retorno a solução que estávamos tentando perturbar.

Por fim, outra vizinhança que pretendo testar, e a que me dá mais esperanças até o momento, é sempre escolher a aresta de maior valor que não possui os dois vértices aos quais é incidente (v1 e v2) num mesmo grupo. O algoritmo botará o vértice v1 no mesmo grupo do v2; caso a solução seja factível, calcula seu valor. Após, o algoritmo, botará o vértice v2 no mesmo grupo de v1; caso a solução seja factível, calcula seu valor. Por fim, a solução de maior valor será retornada pelo algoritmo. Se nenhuma solução for factível, tentamos fazer o mesmo

para a próxima aresta de maior valor. Se não for possível gerar uma solução factível a partir de nenhuma aresta, o algoritmo retorna a solução que estava tentando perturbar.

Temperatura final (critério de parada): segue a ideia apresentada nas notas de aula. O algoritmo recebe uma variável pf no intervalo [0,1], chamada de probabilidade final. Para cada temperatura do algoritmo, contamos o número de vezes em que tentamos realizar um movimento para um vizinho imediatamente pior (movesTried), e o número de vezes que tal movimento efetivamente aconteceu (movesSucceded). Se, para uma dada temperatura, movesSucceded/movesTried é menor que pf, incrementamos um contador. Se uma nova solução melhor é achada, zeramos o contador. Se o contador chegar num valor pré-estabelecido, paramos o algoritmo. Isso faz com que o algoritmo pare (resfrie) quando, a partir de uma dada temperatura, não conseguimos mais achar soluções melhores nem realizar um movimento para um vizinho imediatamente pior, a fim de tentar sair de um máximo local.

Se movesTried for 0, não posso realizar a divisão citada, mas também não há a necessidade de incrementar o contador, já que todos os vizinhos gerados foram melhores que a solução que estava sendo perturbada.

Temperatura inicial: novamente, segue a ideia apresentada nas notas de aula. O próprio Simulated Annealing é executado para definir uma temperatura inicial. Para tanto, defino uma variável pi, chamada de probabilidade inicial; ela é a probabilidade que quero que meu algoritmo, em sua primeira temperatura, aceite um movimento para um vizinho pior. Assim como para a temperatura final, para cada temperatura do algoritmo, contamos o número de vezes em que tentamos realizar um movimento para um vizinho imediatamente pior (movesTried), e o número de vezes que tal movimento efetivamente aconteceu (movesSucceded). Se, para uma dada temperatura, movesSucceded/movesTried for próximo o suficiente de pi, retorno ela; próximo suficiente se refere ao fato de que defini um intervalo, próximo a pi, que é aceitável, já que alcançar exatamente pi seria difícil.

Novamente, se nenhum movimento para um vizinho pior foi tentado, há uma divisão por 0; contudo, se nenhum movimento foi tentado, posso simplesmente ignorar essa temperatura, já que ela não me informa nada sobre a probabilidade de um movimento ser aceito. Com isso evito a divisão por 0.

Algumas precauções que tive que tomar para a execução do Simulated Annealing que gera uma temperatura inicial: a sua temperatura inicial T é, inicialmente, um número grande, para que exp(delta/T), com delta negativo já que é um problema de maximização, seja extremamente próximo de 1, de forma que o algoritmo aceite praticamente todos os movimentos inicialmente. Além disso, apesar de improvável, é possível que eu nunca chegue

numa temperatura que faça com que movesSucceded/movesTried seja próximo o suficiente de pi. Portanto, sempre guardo a temperatura que tenha movesSucceded/movesTried mais próximo de pi. Se o algoritmo chegar em um ponto em que sua temperatura seja extremamente baixa, como 0.00000001, exp(delta/T), para delta negativo, será tão próximo de 0 que provavelmente movimento algum será aceito. Nesse caso, apenas retorno a temperatura que chegou mais próxima de pi.