Algoritmo de Branch and Bound para o Problema de Separação de Grupos Minimizando Conflitos

Erick Eckermann Cardoso GRR20186075
Dante Eleutério dos Santos GRR20206686

Resumo: O problema de separação de grupos minimizando conflitos é uma importante questão em diversos contextos, como otimização de equipes, formação de grupos de trabalho e design de comunidades. Neste trabalho, abordamos o problema de separar um conjunto de heróis em dois grupos de forma a respeitar todas as amizades existentes e minimizar o número de inimizades entre os grupos. Propomos a aplicação do algoritmo de Branch and Bound como uma abordagem eficiente para encontrar a solução ótima ou uma solução aproximada de alta qualidade. O algoritmo realiza uma busca exaustiva no espaço de soluções viáveis, utilizando técnicas de relaxamento e poda para melhorar a eficiência computacional. Apresentamos uma formulação matemática do problema, detalhes da implementação do algoritmo e resultados experimentais em conjuntos de dados de diferentes tamanhos. Os resultados demonstram a eficácia do algoritmo proposto em encontrar soluções de alta qualidade em tempo razoável.

Palavras-chave: Branch and Bound, Backtracking, otimização, busca, otimalidade.

1 Introdução

A formação de grupos é uma tarefa comum em diversos cenários, desde projetos em equipe até a criação de comunidades. Em muitos casos, é fundamental levar em consideração as relações sociais existentes entre os indivíduos ao realizar essa tarefa. Por exemplo, ao montar uma equipe de trabalho, é desejável que as pessoas sejam compatíveis e que as relações de amizade sejam promovidas, a fim de garantir uma colaboração eficiente e um ambiente de trabalho harmonioso.

No entanto, a formação de grupos pode se tornar desafiadora quando há relações de inimizade entre os indivíduos. O conflito entre membros do grupo pode levar a problemas de comunicação, diminuição da produtividade e clima negativo. Portanto, é importante considerar esses aspectos ao realizar a separação dos indivíduos em grupos.

Neste trabalho, concentramos nossa atenção no problema de separação de grupos minimizando conflitos, no contexto específico dos heróis. Supomos que temos um conjunto de heróis com pares de amizades e inimizades conhecidos. Nosso objetivo é encontrar uma maneira de separar esses heróis em dois grupos, de forma que todas as amizades sejam respeitadas e o número de inimizades entre os grupos seja minimizado.

Para resolver esse problema, propomos a aplicação do algoritmo de Branch and Bound. Esse algoritmo é uma técnica de busca exaustiva que divide o espaço de soluções em subconjuntos menores, explorando apenas aqueles que têm potencial para conter a solução ótima. Utilizamos técnicas de relaxamento e poda para reduzir o espaço de busca e melhorar a eficiência computacional do algoritmo. Uma solução de corte por viabilidade e duas de otimalidade: considerando os futuros triângulos de conflitos e outra considerando os futuros pentágonos de conflitos.

O restante deste artigo está organizado da seguinte forma: na Seção 2, formulamos o problema, dando uma descrição do problema e da modelagem. Na Seção 3, descrevemos o algoritmo de Branch and Bound proposto e discutimos detalhes de sua implementação e de suas funções limitantes; e finalmente na Seção 4, apresentamos os resultados experimentais obtidos e comparações entre as funções limitantes propostas, concluindo assim esse trabalho.

2 O Problema e sua modelagem

2.1 Descrição do problema

O problema abordado neste trabalho é o de separação de grupos com o objetivo de minimizar conflitos entre os elementos do grupo. Neste cenário específico, consideramos um mundo fictício onde há um grupo de **n** super-heróis. Alguns super-heróis possuem conflitos entre si, enquanto outros têm uma afinidade tão grande que só funcionam bem juntos. Com a aparição de dois vilões, os super-heróis decidiram se dividir em dois grupos. A separação dos grupos deve garantir que todos os pares de super-heróis com grande afinidade permaneçam no mesmo grupo, ao mesmo tempo em que minimiza o número de pares com conflito dentro de cada grupo.

Considere que cada super-herói é indexado pelos números de **1 a n**. Seja **C** o conjunto de pares **(ai , bi)**, com **1 ≤ i ≤ k = |C|**, tais que **ai** e **bi** tem conflito, e seja **A** o conjunto de pares **(ai , bi)**, com **1 ≤ i ≤ m = |A|**, tais que **ai e bi** tem grande afinidade. Dados **n**, **C** e **A**, queremos encontrar uma separação em dois grupos não vazios de tal forma que para todo par **(x, y)** ∈ **A, x e y** estão no mesmo grupo, e que minimiza o número de pares **(u, v)** ∈ **C** tais que **u** e **v** estão no mesmo grupo.

2.2 Modelagem

Para resolver esse problema, propomos a utilização do algoritmo de Branch and Bound. Esse algoritmo utiliza uma abordagem de busca exaustiva, porém com estratégias de corte inteligentes para evitar a exploração desnecessária do espaço de busca.

O Branch and Bound trabalha com uma árvore de soluções parciais, onde cada nó representa a atribuição de um super-herói a um grupo específico. O algoritmo percorre essa árvore de forma recursiva, dividindo o espaço de busca em subárvores que representam todas as possíveis atribuições dos super-heróis aos grupos.

Durante a busca, são aplicadas estratégias de corte para evitar a exploração de subárvores que não levarão a uma solução ótima. Uma dessas estratégias é o corte por otimalidade referente a verificação de conflitos entre os super-heróis atribuídos a um mesmo grupo. Se em um determinado nó da árvore de soluções parciais, o número de conflitos entre os super-heróis de um grupo somado ao número de vezes que um triângulo de conflitos aparece no conjunto de conflitos ainda não decididos exceder o mínimo encontrado até então, essa subárvore é podada, pois não poderá levar a uma solução ótima.

Além disso, durante a construção da árvore de soluções parciais, também são aplicadas restrições para garantir que os super-heróis com grande afinidade permaneçam juntos na mesma equipe. Essa restrição evita a criação de soluções inválidas e reduz o espaço de busca, fazendo assim o corte por viabilidade.

O algoritmo de Branch and Bound continua a explorar a árvore de soluções parciais até que todas as ramificações sejam verificadas ou até que uma solução ótima seja encontrada. A solução ótima é aquela que minimiza o número de pares com conflito dentro de cada grupo, respeitando as restrições de afinidade.

Em resumo, o problema de separação de grupo minimizando conflitos entre os super-heróis pode ser resolvido utilizando o algoritmo de Branch and Bound. Através da modelagem adequada, aplicação de estratégias de corte inteligentes e a utilização de restrições, é possível encontrar uma solução ótima que atenda aos requisitos de minimização de conflitos e agrupamento por afinidade.

3 Implementação

3.1 Fluxo geral do programa

Inicialmente é recebido um arquivo txt contendo as relações de afinidade e conflitos entre os heróis e então são feitos dois grafos: Um para as afinidades e outro para os conflitos. Além disso, é feito um vetor de pares contendo os conflitos e efetua-se o pré-processamento da busca de todos os triângulos e todos os pentágonos no grafo de conflitos.

Após isso, é efetuada a chamada da função de Branch and Bound, a qual tem como base o tamanho do vetor de heróis e ao alcançar a base confere se a solução encontrada é válida e se ela é melhor que a atual solução salva como a ótima. Depois são efetuados os cortes de viabilidade de de otimalidade e após acabar a busca, são impressos o tempo de execução, o número de nodos da solução encontrada, a solução encontrada e a quantidade de conflitos existente na mesma.

3.2 Implementação do corte por viabilidade

Ao descer um nível da árvore de soluções parciais para um nodo específico, ou seja, antes de atribuir um super herói a um grupo específico, se a flag -f não tiver sido acionada a função recursiva bnb() aplica a verificação do corte por viabilidade.

Caso o herói a ser atribuído tenha relação de amizade ao mesmo tempo com heróis que já estão no grupo A e com outro que está no grupo B, significa que é inviável que o algoritmo continue por aquela sub árvore de problemas, pois qualquer grupo que aquele herói for atribuído violará a primeira regra, que é a de respeitar todas as afinidades entre os heróis. Por essa razão o corte é feito para aquela sub-árvore de problemas, por não ser viável continuar naquele caminho.

3.3 Implementação das funções limitantes

A função limitante fornecida no enunciado do problema foi implementada na função de nome “Guedes” e funciona da seguinte maneira: Primeiramente é contado o número de conflitos existentes entre os heróis com grupos já designados e então busca-se todos os triângulos de conflitos que envolvem apenas heróis com grupos não designados e sem repetição de heróis.
A cada triângulo encontrado com essas características, o contador de conflitos é aumentado e após o fim da soma, compara-se o valor encontrado com uma variável global que guarda o número de conflitos da solução ótima encontrada até o momento e caso seja melhor, efetua-se a troca.
A função limitante escolhida para ser implementada pela dupla encontra-se na função “betterGuedes” e é um incremento da função dada. Além de procurar por triângulos de conflitos dentre os heróis sem grupo definido, também buscou-se pentágonos de conflitos visto que todo ciclo de conflitos de tamanho ímpar causaria o problema de haver um conflito inevitável.

4 Resultados e testes

4.1 Análise das funções limitantes

4.1.1 Função proposta pelos professores

Foi verificado que a função de limitante proposta pelos professores é efetiva e de fato diminui consideravelmente o número de nodos expandidos em casos gerais. Porém, comparada com a solução dos alunos, o número de nodos expandidos é maior, perdendo nesse aspecto. Em quesito tempo de execução, a função é superior a solução dos alunos, demonstrando sempre um resultado melhor.

4.1.2 Solução dos Alunos

Foi observado uma melhora no número de nodos cortados apesar de haver uma piora no tempo de execução, a qual consideramos ter sido causado pelo fato da busca pelos pentágonos apresentar complexidade O(5).

4.2 Resultados gerais dos testes

Para a seguinte entrada:

6 12 0
3 4
1 4
4 6
1 2
2 6
4 5
3 6
2 5
5 6
1 5
3 5
1 3

O resultado da execução com o corte de viabilidade ativado, porém sem a opção de cortes de otimalidade retorna a seguinte saída:

3
1 5 6
Time= 0.000115seconds
Nodes= 127

O resultado da execução com o corte de viabilidade ativado e a função de corte proposta pelos professores é o seguinte:

3
1 5 6
Time= 0.00045seconds
Nodes= 66

Agora, o resultado da execução com o corte de viabilidade ativado junto com a opção de otimalidade proposta por nós gera a seguinte saída:
3
1 5 6
Time= 0.00451seconds
Nodes= 44

Podemos ver com esses resultados que de fato a solução utilizando pentágonos na otimalidade possui o menor número de nodos expandidos, porém perde em quesito tempo de execução para os outros dois testes. Isso se dá pelos fatos citados anteriormente. A solução proposta pelos professores se mostra superior em tempo de execução, pelo fato de que contar os triângulos é um processo muito menos custoso computacionalmente do que contar os pentágonos.

Neste caso o corte por viabilidade não influenciou em nenhum aspecto. Porém, para a seguinte entrada:
4 1 3
1 3
1 2
1 4
3 4

Ao executar com as opções de corte por viabilidade e corte por otimalidade desligadas, temos o seguinte resultado:
1
1 2 3 4
Time= 2.44e-05seconds
Nodes= 31

Ao ativar o corte por viabilidade:
1
1 2 3 4
Time= 6.29e-06seconds
Nodes= 11

Percebe-se que houve uma melhora considerável no número de nodos expandidos, o que prova a eficiência do método.

REFERÊNCIAS

[1] STINSON, D. L. COMBINATORIAL ALGORITHMS : generation, enumeration,and search. S.L.: Crc Press, 2019.

[2] CORMEN, T. H. et al. Introduction to algorithms. [s.l.] MIT Press, 2009.

[3] HORST, R.; TUY, H. Global Optimization. [s.l.] Springer Science & Business Media, 2013.

[4] GEEKSFORGEEKS. GeeksforGeeks | A computer science portal for geeks. Disponível em: <https://www.geeksforgeeks.org/>.