# Relatório Projeto 2 – Separação de Grupo Minimizando Conflitos

## Identificação

* **Nome do projeto:** Separação de Grupo Minimizando Conflitos
* **Data de análise:** 21 de Junho de 2023
* **Autores da análise:** Gabriel Razzolini Pires De Paula(GRR20197155) & Rubens Zandomenighi Laszlo(GRR20206147)

## Explicação do Problema

O programa é destinado a resolver um problema de otimização envolvendo o agrupamento de super-heróis. Dado um conjunto de super-heróis, pares de conflitos entre super-heróis e pares de afinidades, o objetivo é dividir os super-heróis em dois grupos de forma a minimizar o número total de conflitos dentro dos grupos e garantir que os pares de afinidades estejam no mesmo grupo.

## Modelagem

* **Variáveis Globais**:
  + **n**: Número total de super-heróis.
  + **k**: Número de pares de conflitos entre super-heróis.
  + **m**: Número de pares de afinidades entre super-heróis.
  + **conflicts**: Array 2D que armazena os pares de super-heróis em conflito.
  + **affinity**: Array 2D que armazena os pares de super-heróis com afinidade.
  + **groups**: Array que armazena a atribuição de grupo para cada super-herói.
  + **min\_conflicts**: Armazena o mínimo número de conflitos encontrado.
  + **best\_groups**: Armazena a melhor configuração de grupos.
  + **disable\_viability**: Se verdadeiro, desativa a verificação de viabilidade.
  + **disable\_optimality**: Se verdadeiro, desativa a verificação de otimidade.
  + **use\_professors\_bounding\_function**: Se verdadeiro, usa uma função de limitação personalizada.
  + nodes: Contador do número de nós explorados na árvore de pesquisa.
* **Funções Principais**:
  + **areInSameGroup(int a, int b)**: Retorna verdadeiro se os super-heróis 'a' e 'b' estão no mesmo grupo.
  + **satisfiesAffinity()**: Verifica se todos os pares de afinidades estão no mesmo grupo.
  + **calculateConflicts()**: Calcula o número total de conflitos para a atribuição atual de grupos.
  + **bdada()**: Função fornecida pelo professor de limitação personalizada que utiliza conflitos e triângulos.
  + **alternative\_bounding\_function()**: Função de limitação alternativa usando penalidades e recompensas.
  + **branch\_and\_bound(int i)**: Implementa o algoritmo de ramificação e poda para encontrar a melhor atribuição de grupos.
  + **main(int argc, char \*argv[])**: Função principal que controla a execução do programa.

## Detalhes da Implementação

O programa utiliza o algoritmo de ramificação e poda para encontrar a configuração de grupos de super-heróis que minimiza os conflitos. Os nós da árvore de pesquisa representam as atribuições de grupo de super-heróis.

Para cada super-herói, há duas possibilidades - ele pode ser atribuído ao grupo 1 ou ao grupo 2. A árvore de pesquisa é construída recursivamente, com cada nível representando a atribuição de um super-herói a um grupo.

A função de ramificação e poda, **branch\_and\_bound**, verifica os casos base, utiliza funções de limitação para podar subárvores que não são promissoras, e realiza chamadas recursivas para continuar construindo a árvore.

O programa também permite que as verificações de viabilidade e otimidade sejam desativadas através de argumentos de linha de comando, o que pode acelerar o processo à custa de resultados possivelmente subótimos.

Por fim, o programa mede o tempo de execução e imprime informações de diagnóstico, incluindo o número de nós na árvore de pesquisa e o tempo gasto.

## Análise das Funções Limitantes

1. **bdada**: Esta função limitante utiliza contagem de conflitos e triângulos para calcular uma estimativa do número de conflitos. É potencialmente menos eficiente devido ao uso de três loops aninhados, tornando a complexidade de tempo O(k^3), onde k é o número de conflitos.
2. **alternative\_bounding\_function**: Esta função usa penalidades para conflitos e recompensas para afinidades para estimar o número de conflitos. É mais eficiente em termos de tempo em comparação com a função bdada, com uma complexidade de tempo de O(k + m), onde k é o número de conflitos e m é o número de afinidades.

* **Análise de volume de Nodos utilizado pelas Funções**

Para os testes das funções geramos mais casos de testes para as funções para possuir mais dados das execuções para comparação, sendo esses listados no diretório *examples*. E os outputs gerados pelas funções em *examples\_output*. Conforme os casos de teste, a quantidade de nodos em BALTERNATIVA equivale a 0.6771% da função BDADA, sendo listada abaixo a quantidade de heróis e nodos utilizados, e plotado graficamente os dados.

|  |  |  |
| --- | --- | --- |
|  | BALTERNATIVA | BDADA |
| 3 | 11 | 11 |
| 4 | 13 | 15 |
| 10 | 49 | 657 |
| 13 | 597 | 6421 |
| 16 | 3209 | 51737 |
| 19 | 16601 | 418845 |
| 22 | 65873 | 3352609 |
| 25 | 295465 | 26824741 |
| 28 | 1278901 | 214736937 |
| Total | 1660719 | 245391973 |