Desenvolvimento de uma Heurística Construtiva para o Problema de Agrupamento Maximamente Diversificado

João Duarte Colares, Patrick Duarte Pimenta

10 de março de 2025

Resumo

O presente trabalho demonstra uma heurística construtiva para resolver o Problema de Agrupamento Maximamente Diversificado (MDGP), cujo problema representa o desafio comum em situações onde a formação de grupos com alta diversidade entre seus elementos é essencial. **A heurística busca a maximização da diversidade dos grupos, quando o espaço de soluções e explorado de forma eficiente.** Utilizaremos uma heurística construtiva com estratégia de intensificação e diversificação, permitindo uma construção incremental dos grupo com ajustes dinâmicos, afim de evitar ótimos locais. Em seguida, vamos avaliar a eficiência e a qualidade das soluções geradas através dos experimentos computacionais.

Abstract

This work presents a constructive heuristic to solve the Maximally Diverse Grouping Problem (MDGP), a problem that represents the common challenge in situations where forming groups with high diversity among their elements is essential. **The heuristic aims to maximize the diversity of the groups, while efficiently exploring the solution space.** We will use a constructive heuristic with intensification and diversification strategies, allowing for an incremental construction of the groups with dynamic adjustments, in order to avoid local optima. Subsequently, we will evaluate the efficiency and quality of the solutions generated through computational experiments.

1 Introdução

Esse artigo descreve uma heurística construtiva para encontrar uma solução viável para o problema de maximização da diversidade de grupos (MDGP), que visa formar grupos o mais diversificados possível a partir de um set de elementos. Esse problema concerne a diferentes situações práticas do mundo real, como a alocação de estudantes em grupos com características variadas [WL98], e o armazenamento de programas em memórias com paginação. Diferentes propostas de solução foram discutidas para o MDGP. O MDGP é um problema np-difícil, como foi provado por [FK90]. Dessa forma, torna-se pertinente o desenvolvimento de heurísticas visando soluções cada vez melhores para o problema.

Em [DM07] foi desenvolvida uma heurística com busca tabu e memoryless design, para o problema de formar um único grupo com máxima diversidade entre os elementos colocados nele (MDP), obtendo resultados que superaram as melhores heurísticas da época. O problema abordado no presente artigo ganha complexidade com a inclusão de um número maior de grupos para distribuir os elemento, e tem recebido atenção de cada vez mais estudos acadêmicos, principalmente após os trabalhos de [AL89] em que foi tratado o problema de agendamento de exames universitários.

Em [FCMZ11], foi desenvolvida uma heurística híbrida baseada em algoritmo genético (AG); obtendo resultados eficientes em relação aos algoritmos LCW e non-hybrid GA.

2 Definição do Problema

O problema do MDGP consiste em distribuir um set de M elementos em G subsets mutuamente disjuntos, de modo que todo elemento esteja em um, e apenas um, subset; e de modo que a soma da

diversidade total de todos os subsets seja maximizada. A diversidade total de um subset significa a soma das distâncias entre cada par de elementos do subset. O que a distância de um par de elementos representa depende do contexto específico do problema. Em [FK90], por exemplo, as distâncias representam as arestas entre dois vértices de um grafo.

Cada um dos G subsets possui um tamanho máximo e um tamanho mínimo; de modo que é necessário, para uma solução ser válida, que todo subset tenha pelo menos a quantidade correspondente ao tamanho mínimo em elementos, e não tenha mais elementos que o seu tamanho máximo.

3 Descrição da Heurística

Para o desenvolvimento da heurística construtiva, utilizaremos a linguagem C++. O motivo dessa escolha se deve ao fato de que a linguagem demonstra ter um ótimo desempenho quando se trata de processamento de algoritmos, e pela proximidade com a comunicação de baixo de nível. Linguagens como Python também seriam uma ótima escolha, pela variedade de bibliotecas e frameworks disponíveis, e tendo também uma comunidade ampla e ativa, para a tarefa de pré processamento e leitura dos dados, por exemplo. Entretanto, o desempenho da heurística construtiva desenvolvida em Python seria drasticamente afetado devido a lentidão de processamento da linguagem.

Nossa heurística inclui uma série de etapas para construir uma solução promissora para o problema. Seja G a quantidade de grupos; inicialmente, distribuímos arbitrariamente 1 elemento para cada um dos G grupos.

Em seguida, para decidir onde colocar cada um dos elementos restantes fazemos o seguinte: Antes de alocar um elemento para um grupo, calculamos, para cada grupo, um valor A que representa o quão vantajoso é incluir o elemento naquele grupo. Adicionamos, então, o elemento ao grupo com o maior valor para A, e removemos o elemento da lista de elementos que ainda faltam ser colocados em grupos.

Para calcular o valor A nos basemos nos seguintes princípios:

- É preferível colocar mais elementos em grupos com capacidades maiores à distribuí-los proporcionalmente entre os grupos, pois dessa forma aumentamos o números de pares entre elementos que se encontram no mesmo grupo.
- Ao escolher um grupo para colocar um elemento, é vantajoso que seja um grupo com elementos, em média, distantes do elemento que estamos inserindo.

Então calculamos o valor de A, para um grupo i e um elemento x, da seguinte forma:

- Somamos as distâncias entre x e cada um dos elementos já inseridos em i e dividimos pela quantidade de elementos presentes em i, para calcular a média.
- multiplicamos o resultado por (1.0 + (tamanho máximo do grupo i/ (total de elementos do problema/10.0))). Dessa forma, priorizamos grupos que tem uma capacidade de espaço maior.
- também multiplicamos o resultado por 2 caso o grupo ainda não tenha cumprido a capacidade mínima, para estimular os elementos a serem atribuídos para grupos que não tem a capacidade mínima satisfeita.

Atribuímos o elemento para o grupo que tiver o maior A naquele instante do processamento, para aquele elemento.
Garantimos que um elemento seja atribuído para um grupo que ainda não tem o tamanho mínimo satisfeito, caso haja risco de sobrar grupos com quantidade de elementos inseridos menor que o seu tamanho mínimo.
Garantimos que nenhum elemento seja inserido em um grupo sem espaço.
Alocamos cada elemento para um grupo, baseado no valor de A, até não restar elementos para atribuir a grupos.

Limite superior - algoritmo:

- calcular M(M-1)/2 para saber quantas distâncias há ao todo;
- ordenar os grupos por tamanho máximo ;
- ordenar os grupos por tamanho mínimo;
- percorrer a lista ordenada de tamanho mínimo, atribuindo para o grupo na posição i da lista ordenada por tamanho mínimo, o tamanho máximo do grupo na posição i da lista ordenada por maior tamanho máximo.
- Teremos M elementos para distribuir em G grupos.
- Atribuir um inteiro para a quantidade de elementos de cada grupo da seguinte forma:
- 1 todo grupo recebe quantidade de elementos = tamanho mínimo do grupo elementos.
- 2 Fazer até M = 0 ou até o grupo não ter espaço disponível: adicionamos 1 à quantidade de elementos do grupo com espaço disponível que tenha o maior tamanho máximo (considerando que modificamos os tamanhos máximos originais conforme explicado anteriormente) e subtraímos 1 de M.
- Criamos uma variável D para a quantidade de pares de distâncias possíveis de serem formados nesses grupos (os pares de elementos que se encontram no mesmo grupo).
- Para cada grupo com quantidade de elementos maior que 0 calculamos a quantidade de pares possíveis de serem formados nesse grupo e adicionamos o resultado a D.
- Ordenamos a lista de todas as distâncias (as distâncias entre pares formados entre quaisquer dois elementos distintos da lista de M elementos).
- Percorremos essa lista de distâncias, da maior para a menor, somando cada distância a uma variável TOTAL, até termos passado por D distâncias.
- Retornamos TOTAL.

Demonstração do limite superior - Teorema: É impossível obter uma solução melhor que TOTAL

- Chamamos de total de distâncias os M*(M-1)/2 pareamentos possíveis entre todos os M elementos e suas respectivas distâncias.
- Chamamos de distâncias para uma solução X todos os pareamentos entre elementos que se encontram em um mesmo grupo, e suas respectivas distâncias.
- Seja A um número real qualquer maior que TOTAL. Para que exista uma solução para o problema que gere um resultado igual a A, é necessário que:
    1. Ou é possível redistribuir os elementos de modo que o número de distâncias para essa solução aumente; o que significa mudar a quantidade de elementos em cada grupo, deslocando elementos individuais de um grupo para outro.
    2. Ou é possível melhorar a solução mudando os elementos de grupo de forma a manter a quantidade de distâncias da solução igual a D; por exemplo selecionando algum elemento de um grupo e algum elemento de outro grupo e trocando-os de posição, e repetindo esse processo até haver uma melhora no resultado.
    3. Ou é possível melhorar a solução a partir de uma nova disposição dos elementos em que a quantidade de distâncias da solução é menor que D.
- 1. é impossível
    - Não é possível melhorar a solução alterando elementos de grupo e mantendo a quantidade de distâncias da solução igual a D.
    - A maior quantidade de distâncias é atingida quando alocamos elementos sempre para o maior grupo, enquanto ele tiver espaço disponível, pois:

    - Suponha uma solução com n grupos onde estão alocados elementos, e que o grupo com maior quantidade máxima de elementos dessa solução tenha espaço sobrando.
    - Então o valor da quantidade de distâncias para essa solução só pode ser melhorado:
    - 1 retirando-se um elemento de um grupo e alocando-o no grupo com maior quantidade máxima de elementos, caso o grupo com maior quantidade máxima de elementos seja o grupo (ou um dos grupos) com a maior quantidade de elementos alocados para ele.
    - 2 deslocando elementos de outros grupos para o grupo com maior quantidade máxima de elementos, até que ele se torne o grupo (ou um dos grupos) com a maior quantidade de elementos alocados para ele, e então realizando (1). Essa etapa é possível pois no cenário analisado o grupo com maior quantidade máxima de elementos de um subconjunto de grupos é sempre o grupo com maior quantidade mínima de elementos desse subconjunto.
    - Como o método usado, para distribuir os elementos de forma a obter D distâncias, consiste em sempre alocar elementos no grupo que tem a maior distância máxima entre os grupos com espaço disponível, então não é possível melhorar a solução mudando elementos de grupo, e já estamos na solução que gera o maior valor para a quantidade de distâncias.

- 2. é impossível
  - Não é possível melhorar a solução alterando elementos de grupo e mantendo a quantidade de distâncias da solução igual a D.
  - A solução TOTAL consiste na soma das D distâncias.
  - Seja listaD a lista das D distâncias da solução TOTAL.
  - Para obter uma solução melhor que TOTAL trocando elementos de posição entre si, é necessário trocar elementos de posição entre si de modo que uma ou mais das D distâncias em listaD sejam substituídas por distâncias maiores. Porém listaD contém exatamente as maiores distâncias da lista de todas as distâncias entre pares de elementos do problema (os M*(M-1)/2 pareamentos possíveis entre todos os M elementos).

- 3. é impossível
  - Qualquer solução com uma quantidade de distâncias menor que D é pior que TOTAL. Pois qualquer seleção de D distâncias diferente da seleção listaD, das D maiores distâncias, gera um resultado pior que TOTAL, e continuará gerando um resultado pior que TOTAL se removermos distâncias dela, alterando a disposição dos elementos.

Infelizmente, não conseguimos desenvolver o algoritmo de limite superior descrito a tempo. Portando um algoritmo mais simples foi implementado, em que somamos todas as distâncias entre elementos (a partir da lista de pares de elementos), e retornamos o resultado.

4 Resultados

Comparação dos resultados.		
Instâncias	**Soluções Encontradas**	**Cplex.12.1**
instances/Geo/Geo_n010_ss_01.txt	3660.671	3660.67
instances/Geo/Geo_n012_ss_01.txt	666.008	716.46
instances/Geo/Geo_n030_ss_01.txt	13286.02	13776.34
instances/Geo/Geo_n060_ss_01.txt	45021.124	45374.32
instances/Geo/Geo_n120_ss_01.txt	100655.266	99906.5
instances/Geo/Geo_n240_ss_01.txt	187389.43	185973.83
instances/RanInt/RanInt_n010_ss_01.txt	1292	1292.00
instances/RanInt/RanInt_n012_ss_01.txt	897	985.00
instances/RanInt/RanInt_n030_ss_01.txt	4742	5324.00
instances/RanInt/RanInt_n060_ss_01.txt	17574	18408.00

instances/RanInt/RanInt_n120_ss_01.txt	44046	40577.00
instances/RanInt/RanInt_n240_ss_01.txt	144686	129877.00
instances/RanReal/RanReal_n010_ss_01.txt	1336.053	1427.85
instances/RanReal/RanReal_n012_ss_01.txt	806.769	956.43
instances/RanReal/RanReal_n030_ss_01.txt	5170.026	5503.12
instances/RanReal/RanReal_n060_ss_01.txt	17032.774	18164.17
instances/RanReal/RanReal_n120_ss_01.txt	42382.54	42047.6
instances/RanReal/RanReal_n240_ss_01.txt	142885.003	128619.53
instances/Geo/Geo_n010_ds_01.txt	3829.173	3864.69
instances/Geo/Geo_n012_ds_01.txt	647.732	807.68
instances/Geo/Geo_n030_ds_01.txt	13273.711	14358.40
instances/Geo/Geo_n060_ds_01.txt	46200.671	48163.77
instances/Geo/Geo_n120_ds_01.txt	106540.642	108971.98
instances/Geo/Geo_n240_ds_01.txt	194407.796	190288.26
instances/RanInt/RanInt_n010_ds_01.txt	1219	1325.00
instances/RanInt/RanInt_n012_ds_01.txt	955	1059.00
instances/RanInt/RanInt_n030_ds_01.txt	4536	5607.00
instances/RanInt/RanInt_n060_ds_01.txt	17685	19080.00
instances/RanInt/RanInt_n120_ds_01.txt	43982	44589.00
instances/RanInt/RanInt_n240_ds_01.txt	145107	137150.00
instances/RanReal/RanReal_n010_ds_01.txt	1336.053	1437.81
instances/RanReal/RanReal_n012_ds_01.txt	954.909	1050.35

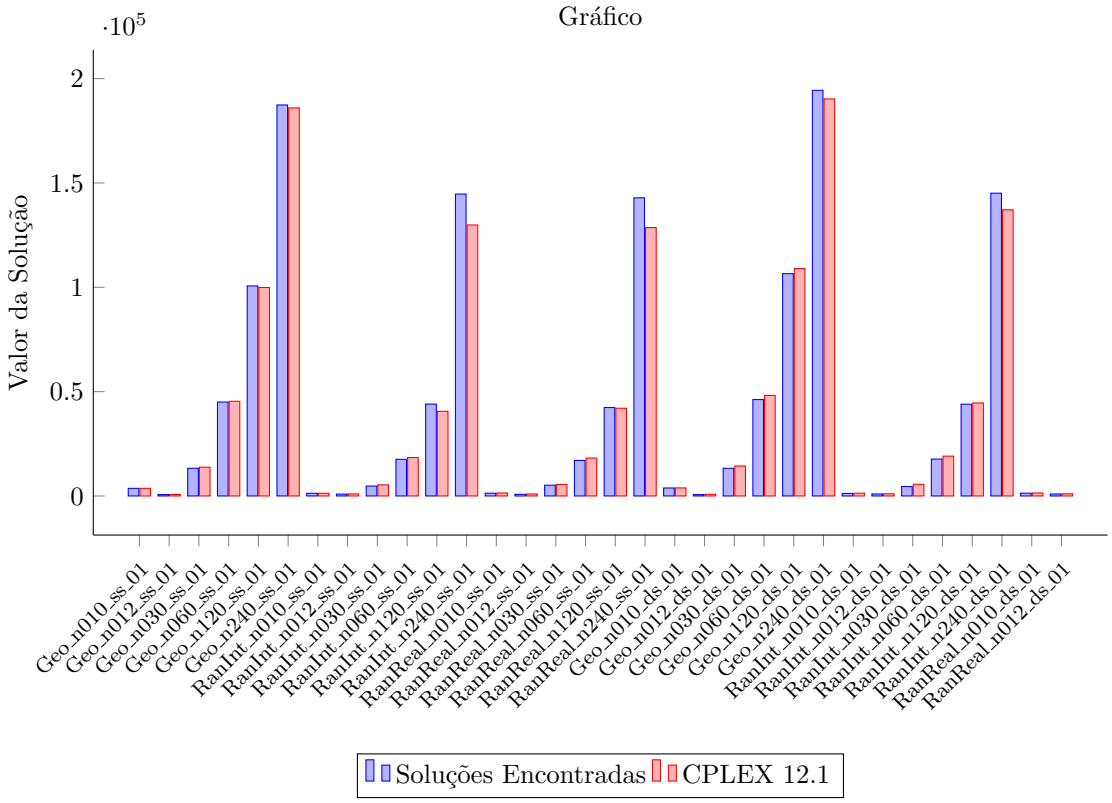

Figure 1: Comparação entre as Soluções Encontradas e o CPLEX 12.1

Foram comparadas as soluções obtidas pela heurística construtiva proposta neste artigo com as soluções fornecidas pelo CPLEX 12.1, conforme apresentado na tabela da página 8 do artigo de [GLMD13]. Para problemas com valores extremamente grandes, nossa heurística construtiva, rep-

resentada pela cor azul no gráfico, demonstrou ser mais eficaz, gerando resultados superiores aos obtidos pelo CPLEX 12.1. Em contrapartida, para soluções de valores grandes, médios, pequenos e extremamente pequenos, observou-se uma variação nos resultados comparativos entre os dois algoritmos. Em alguns casos, a heurística construtiva apresentou melhorias significativas, enquanto em outros, as soluções encontradas não superaram as do CPLEX 12.1. Em resumo, das 32 instâncias analisadas, 8 apresentaram soluções melhores geradas pela heurística construtiva, o que representa 25% das soluções obtidas, superando as fornecidas pelo CPLEX 12.1.

5 Heurística Construtiva - Conclusão

Este trabalho apresentou uma heurística construtiva para o Problema de Agrupamento Maximamente Diversificado, cujas soluções foram comparadas com as geradas pelo CPLEX 12.1, conforme demonstrado na tabela de resultados e também pelo gráfico. A análise revelou que, para instâncias com valores extremamente grandes, nossa heurística construtiva foi capaz de fornecer soluções melhores do que as obtidas pelo CPLEX 12.1, destacando sua eficácia em situações mais desafiadoras. Em 25% das instâncias analisadas, a heurística gerou soluções superiores, enquanto para as demais, os resultados obtidos por ambos os algoritmos foram semelhantes ou o CPLEX superou a heurística.

Embora os resultados da heurística construtiva não tenham sido consistentemente melhores em todos os casos, ela apresentou um desempenho competitivo, especialmente em problemas de grande escala. Este estudo demonstra que a heurística pode ser uma alternativa válida em cenários onde o CPLEX 12.1 não se mostra tão eficiente.

6 Proposta de Busca Local e Busca Tabu

Para dar continuidade no estudo e avaliar possíveis cenários com boas soluções, a presente pesquisa apresentará melhoras para as soluções encontradas pela heurística desenvolvida previamente. Para isso, serão usadas as técnicas de busca local, e posteriormente, a busca tabu. Em primeira análise, será utilizada a busca local, que consiste em um método que realiza um percurso entre as soluções pelo algoritmo, através de um espaço de busca de soluções, ocorrendo mudanças locais, até que o algoritmo pare quando todos os vizinhos (soluções candidatas) sejam considerados piores, desta forma será possível melhorar ligeiramente o resultado inicial. A busca tabu consiste em uma técnica mais elaborada de busca, que permite buscas mais extensas.

7 Definição de Vizinhança

Propomos o uso de duas vizinhanças, e partir delas, vamos desenvolver algoritmos de busca local e busca tabu para melhorar o resultado obtido de anteriormente pelo estudo.

7.1 Primeira Vizinhança

A primeira vizinhança será gerada da seguinte forma:

Uma solução contém m grupos, com elementos atribuídos a eles. Geramos um vizinho selecionando dois grupos e trocando o elemento na primeira posição de um desses grupos com o elemento na primeira posição do outro grupo. Fazendo isso para todos os pares possíveis de grupos dessa solução, geramos todos os vizinhos.

7.2 Segunda Vizinhança

A segunda vizinhança é semelhante à primeira. Porém, ao invés de trocarmos o primeiro elemento de dois grupos, trocamos o último elemento de dois grupos.

8 Proposta de Busca Local

Realizamos dois tipos de busca local. A busca local com primeira melhora e a busca local com melhor melhora. Na busca local com primeira melhora, o algoritmo segue gerando os vizinhos até que encontre um vizinho que seja melhor que a solução atual. Nesse momento, ele para de procurar por novos vizinhos, e define a solução atual como o vizinho encontrado. O algoritmo para quando, em uma busca por vizinhos, não é encontrado nenhum vizinho melhor que a solução atual.

Na busca local com melhor melhora; a diferença é que o algoritmo só para de procurar por vizinhos melhores, a partir de uma dada solução, quando todos os vizinhos tiverem sido pesquisados e comparados com a solução atual. Da mesma forma que a busca com primeira melhora, o algoritmo para quando, em uma busca por vizinhos, não é encontrado nenhum vizinho melhor que a solução atual.

9 Proposta de Busca Tabu

Na busca tabu; criamos uma lista tabu; uma array de soluções; e então, a partir de uma solução, pesquisamos todos os seus vizinhos, e selecionamos o melhor. A diferença da busca tabu para a busca local com melhor melhora é que, na busca tabu, a busca continua mesmo que todos os vizinhos sejam piores que a solução atual. Desse modo, o algoritmo salva a "melhor solução até o momento", e continua explorando as vizinhanças, atualizando continuamente a solução atual. Para evitar loops, a lista tabu salva as soluções mais recentes, e o algoritmo não permite que uma um vizinho se torne solução atual caso ele esteja presente na lista tabu. O algoritmo para após as buscas forem executadas por uma constante arbitrária X de vezes. Nesse momento, o melhor resultado encontrado até o momento é retornado.

10 Apresentação dos Resultados

Instância	**heurística**	**v1 - pm**	**v1 - mm**	**v2 - pm**	**v2 - mm**	**tabu1**	**tabu2**	**TPSDP**
RanInt_n120_ds_01.txt	43982	43995	43995	43982	43982	43995	43982	51075.85
RanInt_n120_ds_02.txt	45044	45057	45057	45192	45309	45057	45309	51294.65
RanInt_n120_ds_03.txt	43960	43960	43960	43960	43960	43960	43960	50192.05
RanInt_n120_ds_04.txt	44517	44517	44517	44517	44593	44517	44593	50343.85
RanInt_n120_ds_05.txt	42993	42993	42993	43110	43233	42993	43233	49839.20
RanInt_n120_ds_06.txt	43828	43828	43828	43977	43982	43828	43982	49602.70
RanInt_n120_ds_07.txt	42638	42638	42638	42841	42841	42638	42866	50202.85
RanInt_n120_ds_08.txt	43920	44126	44126	43971	43971	44126	43971	50282.30
RanInt_n120_ds_09.txt	44651	44744	44744	44683	44781	44744	44781	50363.10
RanInt_n120_ds_10.txt	43653	43653	43653	43779	43779	43653	43779	50269.30
RanInt_n240_ds_01.txt	145107	145107	145107	145353	145353	145107	145432	160358.10
RanInt_n240_ds_02.txt	144052	144099	144099	144052	144052	144220	144052	160277.70
RanInt_n240_ds_03.txt	144908	144979	144979	144908	144908	144979	144908	160223.05
RanInt_n240_ds_04.txt	143488	143488	143488	143488	143488	143488	143488	162420.25
RanInt_n240_ds_05.txt	143990	143990	143990	144045	144045	143990	144045	160605.25
RanInt_n240_ds_06.txt	145584	145584	145584	145584	145584	145584	145802	161040.55
RanInt_n240_ds_07.txt	143391	143542	143542	143475	143475	143542	143475	160131.20
RanInt_n240_ds_08.txt	144555	144555	144555	144677	144677	144555	144677	157980.90
RanInt_n240_ds_09.txt	144590	144590	144590	144599	144599	144590	144599	160601.10
RanInt_n240_ds_10.txt	145543	145543	145543	145543	145543	145543	145543	160082.30

Table 2: Resultados das diferentes heurísticas e buscas tabu para as instâncias RanInt_n120_ds e RanInt_n240_ds.
v1 - vizinhança1, v2 - vizinhança2, pm - primeira melhora, mm - melhor melhora, tabu1 - busca tabu 1, tabu2 - busca tabu 2

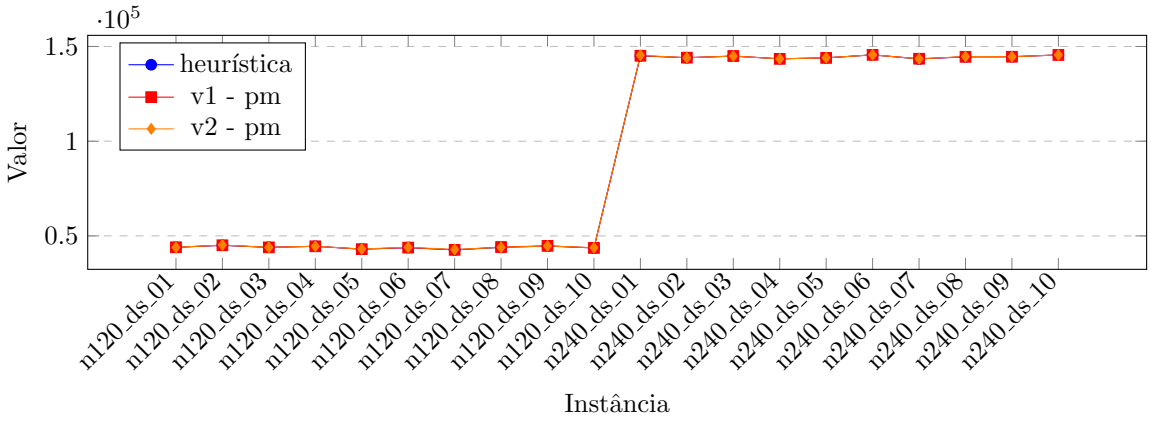

Figure 2: Gráfico comparativo de Heurística, V1-PM e V2-PM para as instâncias RanInt_n120_ds e RanInt_n240_ds.

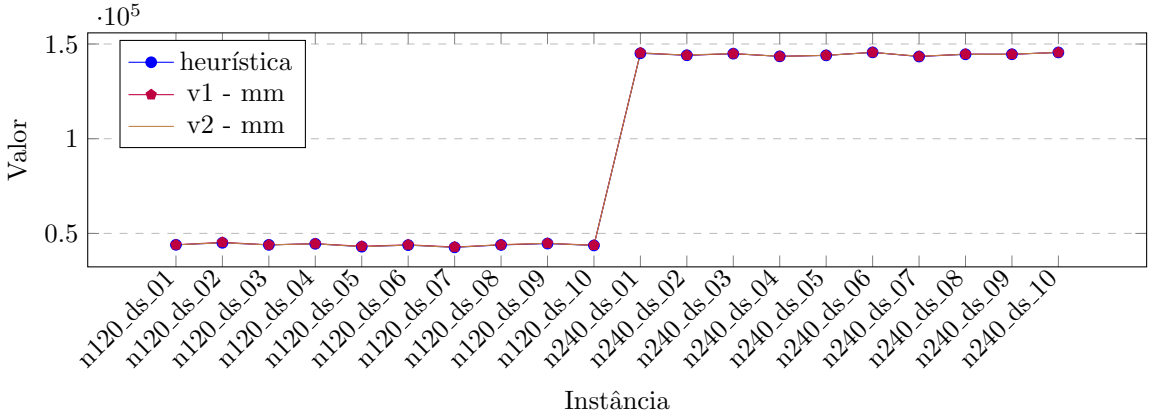

Figure 3: Gráfico comparativo de Heurística, V1-MM e V2-MM para as instâncias RanInt_n120_ds e RanInt_n240_ds.

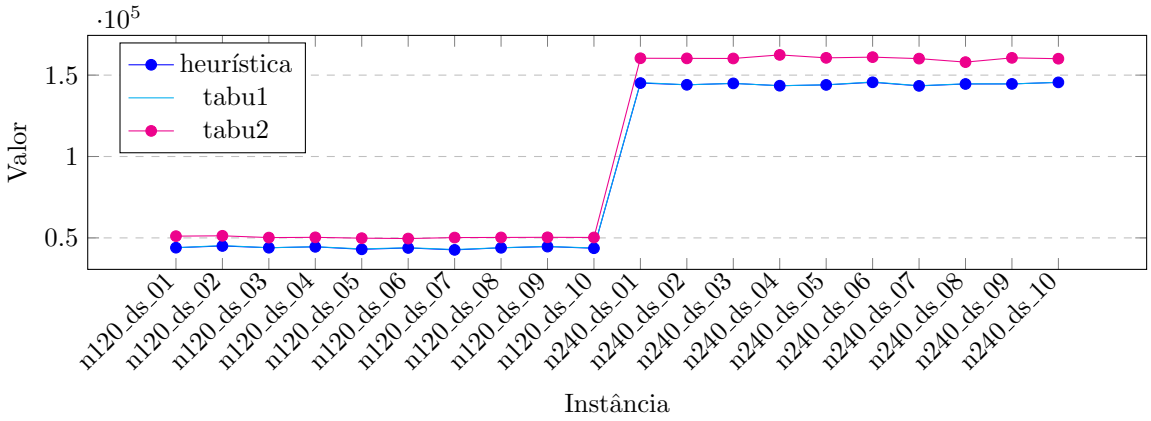

Figure 4: Gráfico comparativo de Heurística, Tabu1 e Tabu2 para as instâncias RanInt_n120_ds e RanInt_n240_ds.

11 Busca TABU - Conclusão

Apesar das buscas terem frequentemente levado a melhorias nos resultados; essas melhorias são muito pequenas. Geralmente menores que 1%. As buscas locais quanto as buscas tabu não apresentaram diferenças gritantes entre si. Muitos resultados foram parecidos, e algumas buscas tiveram desempenho

melhor para instâncias específicas. A busca tabu ficou restrita a poucas iterações, pois quando a quantidade de iterações é definida como um valor muito alto, a busca demora bastante tempo. Também é interessante notar que a busca tabu funcionou melhor quando usamos a segunda vizinhança.

Concluímos que as buscas propostas proporcionam melhoras bem pequenas nos resultados; de forma que os resultados atingidos pelas melhores pesquisas continuam muito melhores.

12 Proposta de Algoritmo Genético

Esta seção apresenta um algoritmo genético desenvolvido para solução de problemas de agrupamento de grupos maximamente diversificado. O algoritmo trabalha com uma população de soluções candidatas que evoluem através de processos inspirados na seleção natural, incluindo seleção, recombinação e mutação.

13 Modelagem de Solução

A implementação do algoritmo genético é baseada na classe Population que gerencia conjuntos de soluções candidatas. Cada solução representa uma distribuição de elementos em diferentes grupos, respeitando restrições específicas de tamanho mínimo e máximo para cada agrupamento.

13.1 Componentes

1. **Inicialização da População:**

   O processo de inicialização da população estabelece um conjunto diversificado de soluções candidatas. Para promover tanto a diversidade quanto a qualidade inicial, o algoritmo:

   - Gera (n-1) soluções através de distribuições aleatórias
   - Complementa com uma solução baseada em heurística dedicada
   - Mantém um tamanho fixo de população para controle computacional

   Desta forma, esta estratégia dual permite exploração ampla do espaço de busca enquanto garante presença de pelo menos uma solução potencialmente promissora.

2. **Mecanismo de Seleção**

   A seleção de indivíduos para reprodução utiliza o método de roleta ponderada, onde:

   - A probabilidade de seleção é proporcional ao valor de fitness
   - O fitness é transformado quadraticamente para amplificar diferenças entre soluções.
   - Soluções superiores recebem probabilidades significativamente maiores de reprodução

   Portanto, método implementa pressão seletiva controlada, favorecendo características superiores enquanto mantém diversidade genética.

3. **Operador de Recombinação**

   O operador de crossover implementa uma estratégia híbrida que:

   - Transfere aproximadamente um terço dos elementos de cada grupo do primeiro progenitor
   - Complementa os grupos seguindo a organização do segundo progenitor
   - Garante a não-repetição de elementos entre grupos
   - Preserva as restrições de tamanho mínimo e máximo para cada grupo

   Este mecanismo preserva parcialmente blocos construtivos de ambos os progenitores enquanto assegura a viabilidade da solução resultante.

4. **Operador de Mutação**

   Para evitar convergência prematura e permitir exploração de novas regiões do espaço de busca, o algoritmo implementa mutações estocásticas que:

   - Realizam múltiplas trocas de elementos entre grupos distintos
   - A quantidade de trocas é controlada por um parâmetro de intensidade
   - Diversificam a população sem comprometer totalmente as características convergentes

5. **Seleção para Sobrevivência**

   O mecanismo de substituição implementa:

   - Ordenação completa das soluções por valor de fitness
   - Eliminação das 10 piores soluções a cada ciclo

14 Ciclo Evolutivo

Em cada geração, o algoritmo executa sequencialmente:

1. Seleção de progenitores e geração de descendentes
2. Avaliação de fitness para todas as soluções
3. Seleção das melhores soluções para a próxima geração
4. Preservação da solução ótima encontrada
5. Aplicação de mutações estocásticas

15 Características Distintivas

O algoritmo implementa características que o tornam particularmente eficiente para problemas de agrupamento:

1. Balanceamento adaptativo entre exploração e explotação
2. Preservação de diversidade genética através de mutações controladas
3. Pressão seletiva ajustável via transformação quadrática do fitness
4. Viabilidade das soluções preservada em todos os operadores genéticos

16 Resultados

Conforme a análise dos dados fornecidos, observa-se um padrão crítico nos resultados: Para todas as instâncias fornecidas, o resultado obtido pelo algoritmo genético é exatamente igual ao resultado da solução heurística, sendo indicador que e o algoritmo não está conseguido melhorar as soluções geradas inicialmente pela heurística construtiva.

O algoritmo genético gera uma população inicial (com 19 soluções aleatórias e 1 solução gerada por meio da heurística construtiva), e tenta, a partir dela, encontrar soluções melhores; mas não consegue superar o resultado encontrado pela heurística construtiva.

17 Conclusão

O algoritmo genético atual mantém a solução encontrada pela heurística descrita no início do artigo, sem conseguir avançar. Isso significa que há a necessidade de revisão na implementação ou, alternativamente, pode significar que para este tipo específico de problema e instâncias, uma abordagem heurística simples seja mais eficiente que um algoritmo genético.

Referências

[AL89] TAGHl ARANI and Vahid Lotfi. A three phased approach to final exam scheduling. *IIE transactions*, 21(1):86–96, 1989.

[DM07] Abraham Duarte and Rafael Martí. Tabu search and grasp for the maximum diversity problem. *European Journal of Operational Research*, 178(1):71–84, 2007.

[FCMZ11] Zhi-Ping Fan, Y Chen, Jian Ma, and S Zeng. Erratum: A hybrid genetic algorithmic approach to the maximally diverse grouping problem. *Journal of the Operational Research Society*, 62(7):1423–1430, 2011.

[FK90] Thomas A Feo and Mallek Khellaf. A class of bounded approximation algorithms for graph partitioning. *Networks*, 20(2):181–195, 1990.

[GLMD13] Micael Gallego, Manuel Laguna, Rafael Martí, and Abraham Duarte. Tabu search with strategic oscillation for the maximally diverse grouping problem. *Journal of the Operational Research Society*, 64(5):724–734, 2013.

[WL98] Rob R Weitz and S Lakshminarayanan. An empirical comparison of heuristic methods for creating maximally diverse groups. *Journal of the operational Research Society*, 49(6):635–646, 1998.