Algoritmo Colônia de Formigas com uma Busca VND para o Problema da Máxima Interseção de k-Subconjuntos

Fábio C. S. Dias
Campus Quixadá – Universidade Federal do Ceará (UFC)
Av. José de Freitas Queiroz, 5003, Cedro. Quixadá - CE
fabiodias@ufc.br

José Robertty de Freitas Costa
Departamento de Computação – Universidade Federal do Ceará (UFC)
Av. Humberto Monte, s/n. Bloco 902, Campus do Pici. Fortaleza - CE
robertty@alu.ufc.br

Wladimir Araújo Tavares
Campus Quixadá – Universidade Federal do Ceará (UFC)
Av. José de Freitas Queiroz, 5003, Cedro. Quixadá - CE
wladimirtavares@ufc.br

RESUMO

Neste trabalho, apresentamos um algoritmo baseado na meta-heurística Colônia de Formigas para o Problema da Máxima Interseção de k-Subconjuntos. Esse problema é definido como: dado uma coleção $\mathcal{S}$ de subconjuntos do conjunto finito B e um inteiro k, deseja-se selecionar k subconjuntos $S_1, S_2, \ldots, S_k \in \mathcal{S}$, tal que $|S_1 \cap \ldots \cap S_k|$ seja máximo. O algoritmo proposto, denominado ANT_{VND}, utiliza regras de proporcionalidade pseudoaleatória como regra de transição de estados e da busca local VND. Os resultados dos testes computacionais mostram que o algoritmo ANT_{VND} supera o algoritmo do estado da arte com relação a qualidade de solução encontrada.

PALAVRAS CHAVE. Metaheurística. Colônia de Formigas. Problema da Máxima Interseção de k-Subconjuntos.

Tópicos: Metaheurísticas, Otimização Combinatória.

ABSTRACT

In this article, we present an algorithm based on the Ant Colony metaheuristic for the Maximum k-Subset Intersection problem. This problem is defined as: given a collection $\mathcal{S}$ of subsets of the finite set B and an integer k, we want to select k subsets $S_1, S_2, \ldots, S_k \in \mathcal{S}$, such that $|S_1 \cap \ldots \cap S_k|$ is maximum. The proposed algorithm, called ANT_{VND}, uses a pseudorandom proportionality rule as a state transition and VND local search rule. The results of the computational tests show that the ANT_{VND} outperforms the state-of-the-art algorithm regarding the quality of the solution found.

KEYWORDS. Metaheuristic. Ant Colony. k-Subset Maximum Intersection Problem.

Topics: Metaheuristics, Combinatorial Optimization.

1. Introdução

O Problema da Máxima Interseção de k-Subconjuntos (kMIS) pode ser definido como: Dado uma coleção $\mathcal{S}$ de subconjuntos do conjunto finito B e um inteiro k, deseja-se selecionar k subconjuntos $S_1, S_2, \ldots, S_k \in \mathcal{S}$, tal que $|S_1 \cap \ldots \cap S_k|$ seja máximo.

O problema pode ser modelado com um grafo bipartido $G = (L \cup R, E)$, sendo que cada vértice $u \in L$ está associado a um subconjunto $S_u \in \mathcal{S}$ e cada vértice $v \in R$ está associado a um elemento $b_v \in B$. Existe uma aresta entre um vértice u (associado a um subconjunto $S_u \in \mathcal{S}$) e um vértice v (relacionado a um elemento $b_v \in B$), se e somente se, $b_v \in S_u$. Portanto, queremos determinar $L' \subset L$, com $|L'| = k$, de forma que $|\cap(L')|$ seja máximo, onde $\cap(L') = \bigcap_{u \in L'} N_G(u)$, com $N_G(u)$ sendo a vizinhança de u em G. Por simplicidade, iremos cometer o seguinte abuso de notação: algumas vezes iremos nos referir aos vértices de L pelos subconjuntos $S_u \in \mathcal{S}$ e mesmo denomina-los de vértices ou de subconjuntos.

Vejamos um exemplo, seja $B = \{1, 2, 3, 4, 5\}$, $\mathcal{S} = \{S_1, S_2, S_3\}$, onde $S_1 = \{2, 3, 4\}$, $S_2 = \{1, 4, 5\}$ e $S_3 = \{1, 2, 3, 4\}$. Para $k = 2$, o problema consiste em selecionar a máxima interseção entre todos os pares de subconjuntos. A Figura 1 ilustra esse exemplo. Note que $|S_1 \cap S_2| = 1, |S_1 \cap S_3| = 3$ e $|S_2 \cap S_3| = 2$. Portanto, a solução ótima para essa instância é dada por $L' = \{S_1, S_3\}$. Observe que o subgrafo induzido pelos vértices S_1 e S_3 e todos os vértices associados com os elementos em $S_1 \cap S_3$ definem um grafo bipartido completo.

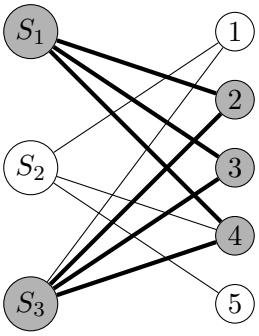


Figura 1: Para $k = 2$, a máxima interseção de k subconjuntos será obtido com os subconjuntos S_1 e S_3. O tamanho dessa interseção será 3.

O problema kMIS possui aplicações em privacidade de dados [Vinterbo, 2002], análise de dados em bioinformática [Nussbaum et al., 2012] e no contexto de redes sociais [Bogue et al., 2014]. No último trabalho citado, cada usuário curte um subconjunto de páginas e devemos encontrar k usuários de uma rede social com a intersecção máxima de páginas curtidas.

Em [Vinterbo, 2002] e [Xavier, 2012], provaram que o problema é $\mathcal{NP}$-Difícil. [Xavier, 2012], mostrou que kMIS não pode ser aproximado dentro um fator constante $\frac{1}{n^{\varepsilon}}$, onde n é o número de subconjuntos e $\varepsilon > 0$, considerando que $NP \not\subset BPTIME(2^{n^{\varepsilon}})$. [Shieh et al., 2012], mostrou que kMIS não pode ser aproximado dentro um erro absoluto $\frac{1}{2}n^{1-2\varepsilon} + O(n^{1-3\varepsilon})$ a menos que $P = NP$.

Em [Bogue et al., 2013], são apresentadas uma heurística gulosa chamada $kInter$, e três formulações de Programação Linear Inteira (PLI). Duas dessas formulações foram obtidas a partir de adaptações da formulação da Biclique Máxima em Arestas (MEB, do inglês Maximum Edge Biclique Problem) e uma nova formulação desenvolvida para o problema, denotada por M_{EDGE}. Para a realização dos testes computacionais, os autores utilizaram instâncias geradas de maneira aleatória separadas em classes variando a densidade do grafo e o valor de k. Os testes computaci-

onais mostraram que a nova formulação conseguiu resultados computacionais superiores as outras duas formulações adaptadas.

Em [Bogue et al., 2014], os autores apresentam um procedimento de redução do tamanho da entrada do problema, uma meta-heurística do tipo GRASP para encontrar uma solução inicial de qualidade e um modelo de programação linear inteira denominado M_{CLIQUE}, obtido fazendo um lifting de uma restrição de M_{EDGE} para uma desigualdade indutoras de facetas. Os resultados computacionais mostraram que a formulação M_{CLIQUE} consegue resolver de maneira ótima mais instâncias do que a formulação M_{EDGE}. Nesse trabalho, os autores consideraram a seguinte aplicação: cada vértice de L representa artistas musicais (cantor solo, banda, etc) e cada vértice de R representa uma pessoa e existe uma aresta se uma pessoa gosta de um artista musical. Portanto, o objetivo do problema neste contexto é selecionar k artistas musicais com o maior número de fãs em comum. Foram utilizadas instâncias de teste reais, extraídas da base de dados LAST FM.

[Costa et al., 2018] apresentam duas novas heurísticas gulosas, obtidas a partir heurística gulosa kInter [Bogue et al., 2013], denominada *kInter Estendida* e *kInter Estendida com path-reliking* para fornecer uma solução inicial de qualidade, e um algoritmo de busca em vizinhança variável reativo (VNS Reativo). Os resultados computacionais mostraram que o algoritmo VNS superou o algoritmo GRASP, apresentado em [Bogue et al., 2014].

Em [Dias et al., 2020], os autores apresentam uma versão estendida do algoritmo VNS apresentado em [Costa et al., 2018]. Além disso, uma análise estatística dos resultados computacional é realizada e conclui que existe uma diferença significativa entre a qualidade das soluções encontradas pelo algoritmo GRASP Reativo[Bogue et al., 2014] e o algoritmo VNS Reativo apresentado e a diferença é favorável para VNS Reativo.

Em [Costa et al., 2020], um algoritmo de branch-and-bound, denominado $BBKMIS$, é apresentado juntamente com um procedimento de limite superior e um critério de ramificação. Além disso, um novo procedimento de redução é apresentado, onde os autores provam que esse novo procedimento domina o de [Bogue et al., 2014]. Os testes computacionais mostraram que esse novo procedimento tem um poder de redução maior do que o anterior nas instâncias testadas. Por fim, o algoritmo BBKMIS superou o algoritmo do estado da arte exato M_{CLIQUE} na maioria das classes testadas, exceto nas classes de densidade mais alta.

Em [Casado et al., 2022], os autores apresentam um procedimento de busca guloso, aleatório e adaptativo (GRASP, do inglês greedy randomized adaptive search procedure) substituindo o procedimento de busca local pela meta-heurística completa da busca tabu denominado $GRASP_{RG+TS}$. Os testes computacionais realizados mostram que o algoritmo supera o algoritmo apresentado em [Dias et al., 2020].

Neste artigo, apresentamos um algoritmo de colônia de formigas utilizando regras de proporcionalidade pseudoaleatória como regra de transição de estados e na etapa de atualização dos feromônios utilizamos apenas a atualização global. Além disso, incorporamos uma busca local do tipo VND (Descida em Vizinhança Variável) ao algoritmo. Este artigo está organizado da seguinte forma: na Seção 2 apresentamos o algoritmo de colônia de formigas para o kMIS, na Seção 3 apresentamos os testes computacionais realizados e, finalmente, na Seção 4 temos as conclusões deste trabalho.

2. Algoritmo Colônia de Formigas

O algoritmo de Colônia de Formigas é uma meta-heurística bioinspirada no comportamento das formigas que utilizam o rastro de feromônios deixados por cada formiga como uma forma de comunicação indireta entre elas. Os rastros de feromônios são usados para construir uma solução para o problema de maneira probabilística. Quanto maior o número de formigas em um

mesmo caminho, maior será a quantidade de feromônios depositado nele e, consequentemente, tem a maior probabilidade deste caminho ser o melhor encontrado pela colônia de formigas. Além do componente probabilístico, o algoritmo possui um componente heurístico na escolha das arestas do caminho construído pelo algoritmo.

A literatura possui diversas versões do algoritmo colônia de formigas, sendo as três principais: Ant System (AS) [Dorigo et al., 1996], Ant Colony System (ACS) [Dorigo e Gambardella, 1997] e MAX-MIN Ant System (MMAS) [Stützle e Hoos, 2000]. As principais diferenças entre essas versões se encontram na fórmula da probabilidade e no procedimento de atualização dos feromônios. Desenvolvemos nossa versão combinando estratégias das duas primeiras versões citadas anteriormente.

Conforme [Dorigo e Stützle, 2003], um algoritmo Colonia de Formigas é dividido em três etapas: Construção de Soluções, Ações Globais e Atualização dos Feromônios. A etapa de ações globais é opcional e decidimos não adicionar em nosso trabalho. O Algoritmo 1 apresenta o pseudocódigo da meta-heurística Colônia de Formigas.

Algoritmo 1: Pseudocódigo da Meta-heurística Colônia de Formigas

Entrada: Parâmetros e feromônios iniciais
Saída: $L^* \subseteq L$ melhor solução encontrada

```
início
    L* → ∅
    ite = 0;
    enquanto ite ≤ MAX_ITER faça
        Etapa de Construção de Soluções;
        Atualização de L*;
        Aplicação da busca local VND;
        Atualização dos Feromônios;
        ite + +;
    retorna L*
```

A etapa de construção de soluções tem como objetivo gerar um conjunto de soluções viáveis através do que é conhecido como formigas artificiais. Essas formigas são agentes que constroem as soluções viáveis de forma iterativa, onde a cada passo, um elemento é escolhido para fazer parte da solução. Cada formiga artificial deposita uma certa quantidade de feromônio no caminho encontrado por ela. Essa informação é utilizada por todas as formigas. Para o kMIS, podemos ter até $|L|$ formigas, com cada formiga t iniciando com um subconjunto S_t de L em sua solução. A partir deste ponto, cada formiga deve selecionar $k-1$ subconjuntos distintos para gerar uma solução viável.

Em nosso algoritmo, denominado ANT_{VND}, consideramos um grafo obtido a partir do grafo de entrada $G = (L \cup R, E)$, que denominamos de $G_a = (V_a, E_a)$, um grafo completo onde os vértices representam todos os subconjuntos em L, ou seja, para todo $S_i \in L$, temos $i \in V_a$. É nesse grafo que as formigas percorrem, depositando feromônios nas suas arestas e visitando os vértices. O objetivo de cada formiga é encontrar um caminho passando por k vértices distintos de G_a. Os vértices visitados por cada formiga representam uma solução para o kMIS.

Dado uma formiga t, seja $A_t \subset V_a$ o conjunto com os vértices visitados pela formiga e $A_t^L \subset L$ o conjunto contendo os subconjuntos em L indicado pelos índices presentes em A_t, ou seja, a solução parcial/completa da formiga t, com inicialmente $A_t = \{t\}$ e $A_t^L = \{S_t\}$. Seja p_{ij}^t a probabilidade da formiga t visitar o vértice j, dado que i foi o último vértice visitado.

O algoritmo AS possui uma regra de transição de estados chamada regra proporcionalidade aleatória que atribui a probabilidade de uma formiga t selecionar o vértice j, dado que i foi o

último selecionado, como:

$$p_{ij}^t = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}^t]^\beta}{\sum_{k \in N(i)} [\tau_{ik}]^\alpha [\eta_{ik}^t]^\beta} & , j \in V_a \setminus A_t \\ 0 & , \text{caso contrário} \end{cases} \quad (1)$$

onde τ_{ij} é a quantidade de feromônio na aresta $ij \in E_a$, com valor inicial sendo uma constante τ_0. Já $\eta_{ij}^t = \frac{1}{d_{ij}^t}$ é um fator heurístico (guloso), onde $d_{ij}^t = \cap(A_t^L) - \cap(A_t^L \cup \{S_j\}) + 1$. Note que d_{ij}^t é proporcional ao decréscimo da cardinalidade da intersecção dos subconjuntos escolhidos pela formiga. Quanto maior o decréscimo, menor a atratividade do conjunto representado por η_{ij}^t. Os parâmetros α e β indicam o peso dos feromônios e do fator heurístico para o valor de p_{ij}^t. Cada formiga t irá selecionar o vértice $j \in V_a \setminus A_t$ de forma aleatória com a distribuição de probabilidade dada por (1). Denotamos essa escolha como j_{AS}.

Já o algoritmo ACS possui uma regra de transição de estados chamada regra proporcionalidade pseudoaleatória que utiliza um parâmetro q_0 com o intuito de equilibrar a exploração de novas arestas (regra proporcional do AS) e a exploração do conhecimento acumulado (quantidade de feromônio acumulado na aresta). A regra de transição de estados escolhe o vértice j a partir do vértice i utilizando a seguinte regra:

$$j = \begin{cases} argmax_{k \in V_a \setminus A_t} \{\tau_{ik} [\eta_{ik}^t]^\beta\} & q \leq q_0 \\ j_{AS} & , \text{caso contrário} \end{cases} \quad (2)$$

onde q é um número aleatório uniformente distribuído entre 0 e 1, q_0 é um parâmetro e j_{AS} é o vértice escolhido pela regra do algoritmo AS. Quando q_0 se aproxima de 1, o algoritmo tende a privilegiar a exploração do conhecimento acumulado. Após testes computacionais, optamos por usar q_0 igual a 1.

Na literatura, normalmente os autores aplicam uma busca local sobre as soluções encontradas pelas formigas. [Stützle e Hoos, 2000] destacam que os algoritmos do tipo colônia de formigas com os melhores resultados são aqueles que utilizam busca local.

Como busca local, aplicamos a busca local do tipo VND. O método Descida em Vizinhança Variável (Variable Neighborhood Descent - VND) é um método de busca local cuja sistemática de funcionamento é a mudança de estrutura de vizinhança. Partindo de uma solução inicial, o VND realiza a busca por uma solução de melhor qualidade dentro de uma estrutura de vizinhança inicial. O VND que aplicamos é o usado em [Dias et al., 2020] que de forma resumida: dada uma solução viável L', f vértices de L' são removidos de forma aleatória e aplica-se a heurística `kInter` [Bogue et al., 2013] para gerar uma nova solução. Esse processo é repetido 3 vezes com a garantia que os vértices removidos sejam todos diferentes das repetições anteriores, mas permitindo que os vértices que foram removidos sejam adicionados pela `kInter` e $f = \max(0, 3 * k; 3)$, exceto se $k = 2$ ou 3, nesse caso $f = 1$ e 2, respectivamente.

Ao final da etapa de construção, a melhor solução pode ter sido atualizada, nesse caso aplicamos a busca local nessa nova melhor solução, caso contrário, aplicamos na melhor solução encontrada na iteração atual.

Na etapa de atualização de feromônios ocorre tanto o depósito quanto a evaporação de feromônio. A evaporação é utilizada para evitar ótimos locais. Em nosso algoritmo utilizamos a estratégia de atualização da versão AS, ou seja, não ocorre a atualização local, apenas a global, que ocorre quando todas as formigas encontram sua solução completa. Todas as arestas são atualizadas

segundo a fórmula:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\Delta_{ij} \tag{3}$$

onde ρ é uma constante que determina a evaporação e o depósito do feromônio, ou seja, vai determinar a porcentagem de feromônio que irá ficar preservada de uma iteração a outra e a porcentagem que será depositada de Δ_{ij}. Seja avg_{ij} a média dos valores das soluções encontradas pelas formigas ao final da etapa de construção da iteração atual onde os vértices i e j estão contidos. Será zero se nenhuma solução foi encontrada com os dois vértices juntos. Seja L^* a melhor solução encontrada até o momento, então definimos $\Delta_{ij} = avg_{ij}/|\cap(L^*)|$.

Podemos ter até $|L|$ formigas, mas o custo computacional é alto se usarmos todas. Por isso, optamos por uma estratégia que a cada iteração seleciona algumas formigas para estarem ativas. Formigas ativas em uma iteração serão formigas que irão gerar uma solução viável. Seja g a quantidade de formigas ativas. Definimos duas estratégia para a seleção das formigas ativas. Na primeira estratégia selecionamos g formigas de forma aleatória. A segunda será de forma reativa. Para cada formiga, teremos a média de todas as soluções encontradas pela formiga. Selecionamos g formigas com as maiores médias.

Nosso algoritmo usa as duas estratégias de forma alternada. Seja MAX_{ITER} quantidade iterações do algoritmo. Nas G primeiras iterações, com $G < MAX_{ITER}$, todas as formigas ficarão ativas. A ideia das G primeiras iterações executarem com todas as formigas é que a escolha reativa tenha informações suficientes. A partir da iteração $G + 1$, de forma alternada, usamos a estratégia aleatória e a estratégia reativa.

3. Testes Computacionais e Resultados

Nesta seção, apresentamos os resultados computacionais obtidos neste trabalho, além das instâncias utilizadas nos testes computacionais e o ambiente computacional.

3.1. Instâncias e Ambiente Computacional

As instâncias consideradas neste trabalho foram as instâncias criadas em [Costa et al., 2020], que são grafos bipartidos gerados de maneira aleatória usando o modelo de [Gilbert, 1959]. De forma resumida, são 216 instâncias (270, mas apenas 216 são viáveis) de tamanhos 40, 60, 80, 100, 140, 180, 200, 240, 280 e 300 vértices, diferenciando-se pela densidade e valor de k. Essas instâncias são divididas em 9 classes de acordo com essas propriedades, como podemos ver na Tabela 1. Além disso, temos instâncias balanceadas ($|L| = |R|$) e desbalanceadas ($|L| > |R|$ e $|L| < |R|$), que denominamos de Grupo 1, 2 e 3, respectivamente. Todas as instâncias da classe C_3 se apresentaram inviáveis e foram removidas dos testes computacionais.

Tabela 1: Classes de Instâncias por Propriedade [Costa et al., 2020]

k \ Densi.	Baixa	Média	Alta
Baixo	C_1	C_4	C_7
Médio	C_2	C_5	C_8
Alto	C_3	C_6	C_9

Neste trabalho, os subconjuntos são representados por meio de strings de bits, desta forma, todas as operações sobre os subconjuntos em nossos algoritmos são feitas por meio de paralelismo de bits. Em [Costa et al., 2018] e [Dias et al., 2020], os autores relataram melhoria considerável no tempo computacional quando se utiliza strings de bits. Os algoritmos foram implementados na linguagem C++ e executados em um computador Intel Xeon, com 2.30 GHz, 64 GBytes de memória

RAM e usando o CentOS 7.0. Todas as instâncias, assim como os códigos fontes desse trabalho encontram-se no repositório do GitHub no link: https://github.com/fabiocsd/kMIS.

3.2. Experimentos Preliminares

Inicialmente realizamos experimentos computacionais preliminares para a seleção dos parâmetros do algoritmo. Para isso, selecionamos as instâncias de tamanhos 60, 180 e 280 de cada classe para que esses experimentos fossem executados. No total, foram 22 instâncias balanceadas ($|L| = |R|$) e 43 desbalanceadas (21 com $|L| > |R|$ e 22 com $|L| < |R|$).

Primeiramente, definimos o $MAX_{ITER} = 30$ e $G = 5$, a quantidade máxima de iterações e a quantidade de iterações iniciais com todas as formigas ativas. Os valores de MAX_{ITER} e G foram escolhidos de forma arbitrária, por considerarmos serem valores apropriados as suas finalidades. O valor de MAX_{ITER} pode parecer pequeno no primeiro momento, mas como vamos executar várias formigas em cada iteração, é uma valor suficiente.

O primeiro experimento preliminar foi para a escolha da quantidade de formigas ativas. Executamos inicialmente com todas as formigas, mas o tempo computacional foi proibitivo. Alguns trabalhos da literatura sugerem um valor proporcional ao tamanho da entrada, já outros, um valor fixo (constante) que seja independente do tamanho da entrada. Para instâncias menores é indiferente a forma de escolha, mas para instâncias maiores a escolha proporcional pode ser bastante cara em relação ao tempo computacional.

Por exemplo, testamos com $0,10 \cdot |L|$ formigas (um valor sugerido na literatura). Para instâncias $|L| = 60$ o tempo foi competitivo, mas a partir de 180, não. Baseado nesse resultado, buscamos um meio termo entre 6 e 18 formigas. Como a diferença de tempo entre essas instâncias foi proporcionalmente alto, limitamos a no máximo 10 formigas ativas. Mas resolvemos deixar as instâncias menores com valor proporcional. Então, a quantidade de formigas ativas será dado por $\min\{10; 0,10 \cdot |L|\}$. No máximo 10 formigas representa uma boa relação entre qualidade das soluções encontradas e tempo computacional.

O segundo experimento preliminar foi com o objetivo de escolher os melhores parâmetros, β e ρ. Os valores dos parâmetros usados nesse experimento foram baseados em relatos na literatura como sendo os melhores valores para β e ρ. O conjunto de valores para β foram $\{0,5; 0,8; 1,0; 1,5\}$ e o conjunto de valores para ρ foram $\{0,05; 0,1; 0,2; 0,3\}$. Executamos experimentos para cada par ordenado desses conjuntos.

Para cada instância, executamos 10 vezes o algoritmo armazenando as seguintes propriedades: a melhor, pior e médias dos valores das soluções e a média dos tempos. Portanto, para cada instância teremos essas 4 propriedade.

Definimos duas métricas para avaliarmos os parâmetros, tanto de forma qualitativa quanto quantitativa. Na primeira, calculamos a média dessas propriedades de todas as instâncias. Observe que para algumas propriedades, como na média dos valores das soluções e a média dos tempos das 10 execuções, teremos a média das médias. Na segunda métrica, contamos quantas vezes cada par de parâmetro encontra o melhor valor de cada propriedade das instâncias. A segunda métrica também será útil para detectarmos possíveis outliers nas propriedades analisadas que podem afetar a métrica qualitativa, apesar de que aos analisarmos os resultados, observamos que a diferença do valor da solução entre o melhor e pior algoritmo varia de 1 a 3.

Na Tabela 2 temos nas duas primeiras colunas os valores do β e ρ. Nas terceira, quarta, quinta e sexta colunas temos a avaliação qualitativa com as médias das 4 propriedades. Nas próximas 4 colunas temos a avaliação quantitativa. Por fim, na última linha temos o melhor valor encontrado de cada propriedade.

Tabela 2: Resultados da avaliação dos parâmetros.

		Qualitativa				Quantitativa			
β	ρ	Avg melhor	Avg pior	Avg média	Avg tempo	#Melhor	#Pior	#Avg média	#Avg Tempo
0,50	0,05	15,54	15,45	15,47	2,30	56	57	51	1
0,50	0,10	15,55	15,48	15,49	2,30	57	59	53	1
0,50	0,20	15,51	15,46	15,48	2,30	55	58	53	1
0,50	0,30	15,52	15,48	15,50	2,30	55	58	52	0
0,80	0,05	15,55	15,43	15,48	2,30	57	56	50	2
0,80	0,10	**15,60**	15,49	**15,54**	2,31	**60**	59	55	1
0,80	0,20	15,57	15,48	15,50	2,30	58	58	53	2
0,80	0,30	**15,60**	**15,51**	**15,54**	2,30	**60**	**60**	**56**	1
1,00	0,05	15,52	15,46	15,48	1,52	56	58	52	31
1,00	0,10	15,55	15,45	15,49	1,51	57	57	51	41
1,00	0,20	15,54	15,48	15,50	1,51	56	59	55	44
1,00	0,30	15,55	15,46	15,51	**1,50**	58	58	53	52
1,50	0,05	15,57	15,48	15,51	2,25	58	58	52	1
1,50	0,10	**15,60**	15,48	15,53	2,25	**60**	58	54	1
1,50	0,20	15,58	15,48	15,52	2,25	59	58	52	0
1,50	0,30	15,57	15,46	15,52	2,26	58	57	52	0
Melhor Resultado		**15,60**	**15,51**	**15,54**	**1,50**	**60**	**60**	**56**	**52**

Escolhemos como valores dos parâmetros $\beta = 0,80$ e $\rho = 0,30$ por esse par de valores terem alcançando o melhor resultado de seis dentre as oito propriedades. Além disso, as únicas propriedades que este par de parâmetros não apresentou os melhores resultados foram as propriedades relacionadas ao tempo computacional, requisito que estamos propositalmente relegando em parte para obter melhores soluções. Mas observe que os pares de parâmetros com os melhores resultados para essa propriedades relacionada ao tempo ($\beta = 1,00$ e $\rho = 0,30$), são bastante ruins quando analisadas as propriedades relacionada aos valores das soluções.

Como valor de τ_0 usamos $\frac{1}{|L_{kInter}|}$, onde L_{kInter} é a solução encontrada pela heurística $kInter$. Optamos por usar a solução encontrada por esse heurística por sua simplicidade e, consequentemente, baixo tempo computacional.

3.3. Avaliação do Algoritmo

Nesta seção iremos avaliar o desempenho de nosso algoritmo de colônia de formiga comparando-o com o melhor algoritmo heurístico encontrado na literatura, o $GRASP_{RG+TS}$ [Casado et al., 2022]. Os dois algoritmos foram executados na mesma máquina e aplicamos as mesmas métricas utilizadas na seleção dos parâmetros da Seção 3.2. Aqui iremos agrupar as instâncias por classes (independente dos grupos) e por grupo (independente das classes). Dessa forma podemos avaliar o comportamento de cada algoritmo em relação a densidade, valor de k e ao balanceamento das instâncias.

Na Tabela 3 temos a comparação qualitativa e na Tabela 4 a comparação quantitativa. Os melhores resultados estão destacado. Deixaremos a análise do tempo computacional para outro momento e iremos começar a análise apenas levando em conta as propriedades relacionadas ao valor das soluções encontradas, mas de antemão, o $GRASP_{RG+TS}$ obteve os melhores tempos computacionais.

Nas classes C_2 e C_6, ambos os algoritmos encontraram apenas soluções de valor 1. Acreditamos que esse seja o valor ótimo.

Nas classes C_1 e $C4$, classes de densidades baixa e média, com k baixo e médio, respectivamente, os algoritmos se mostram equivalentes, ambos encontrando a melhor propriedade em cerca 7 das 12 propriedades.

Na classe C_5 de densidade e k médio, em termos qualitativos os algoritmo foram equivalentes, cada um sendo melhor em 2 das 3 propriedades, mas quando analisamos os dados quantita-

tivos, o Ant_{VND} foi melhor nas 3 propriedades, enquanto o $GRASP_{RG+TS}$ apenas em 1.

A $GRASP_{RG+TS}$ mostra-se melhor na classe C_7, classes de densidade alta e k baixo, sendo melhor nas 6 propriedades, seja qualitativa quanto quantitativa, já o Ant_{VND} foi melhor em 2 de 6 propriedades.

O Ant_{VND} se destaca nas classes C_8 e C_9, classes de densidades alta e com k médio e alto, respectivamente, onde tem dominância tanto em termos qualitativos quanto quantitativos. Nessas duas classes, foi melhor nas 12 propriedades e o $GRASP_{RG+TS}$ em nenhum. Observe que enquanto na classe C_7 o $GRASP_{RG+TS}$ é melhor com uma pequena diferença, já o Ant_{VND} nas classes C_8 e C_9 a diferença é bem maior.

Quando analisamos os grupos, o Ant_{VND} se destaca, sendo melhor em 17 das 18 propriedades e o $GRASP_{RG+TS}$ em apenas 3 das 18.

Tabela 3: Comparação Qualitativa

	Avg melhor	Avg pior	Avg média	Avg tempo	#Melhor	#Pior	#Avg média	#Avg Tempo
	Ant_{VND}				$GRASP_{RG+TS}$			
C_1	3,07	**3,07**	**3,07**	0,82	**3,07**	2,93	3,02	**0,54**
C_2	**1,00**	**1,00**	**1,00**	0,72	**1,00**	**1,00**	**1,00**	**0,32**
C_4	11,23	**11,03**	11,09	0,84	**11,37**	10,97	**11,14**	**0,59**
C_5	**2,50**	**2,47**	2,47	2,96	**2,50**	2,43	**2,48**	**1,78**
C_6	**1,00**	**1,00**	**1,00**	1,25	**1,00**	**1,00**	**1,00**	**0,62**
C_7	66,40	**66,37**	66,38	0,85	**66,67**	**66,37**	**66,56**	**0,67**
C_8	**20,27**	**19,77**	**20,02**	2,99	19,83	19,17	19,47	**2,00**
C_9	**7,57**	**7,47**	**7,49**	4,42	7,47	7,10	7,26	**2,46**
G_1	**15,31**	**15,19**	**15,23**	2,33	15,29	15,00	15,13	**1,39**
G_2	**13,80**	**13,69**	**13,75**	2,27	**13,80**	13,54	13,66	**1,32**
G_3	**17,60**	**17,47**	**17,52**	1,28	17,56	17,32	17,46	**0,88**

Tabela 4: Comparação Quantitativa

	Avg melhor	Avg pior	Avg média	Avg tempo	#Melhor	#Pior	#Avg média	#Avg Tempo
	Ant_{VND}				$GRASP_{RG+TS}$			
C_1	**30**	**30**	**30**	11	**30**	26	26	**19**
C_2	**17**	**17**	**17**	4	**17**	**17**	**17**	**13**
C_4	25	**27**	22	11	**29**	25	**25**	**19**
C_5	**30**	**29**	**29**	10	**30**	28	28	**20**
C_6	**19**	**19**	**19**	5	**19**	**19**	**19**	**14**
C_7	23	**28**	23	9	**30**	**28**	**29**	**21**
C_8	**28**	**28**	**26**	13	19	17	15	**17**
C_9	**30**	**28**	**28**	10	27	18	17	**20**
G_1	68	**70**	**66**	22	**68**	59	58	**50**
G_2	63	**67**	**61**	20	**66**	58	59	**51**
G_3	**71**	**69**	**67**	31	67	61	59	**42**

Quando analisamos o tempo computacional, o $GRASP_{RG+TS}$ se destaca em relação ao Ant_{VND}, obtendo os melhores tempos, seja a análise feita por classe ou grupo. Na Tabela 5, calculamos a média dos tempos agrupados por $|L|$. Observe que até o tamanho 100, os algoritmos possuem tempos equivalentes, com uma pequena vantagem para o Ant_{VND}, mas a partir do tamanho 140, o $GRASP_{RG+TS}$ tem os melhores tempos, chegando a diferença de 78% no tamanho 300.

4. Conclusão

Neste trabalho, apresentamos uma meta-heurística do tipo colônia de formigas, que denominamos de Ant_{VND}, para o Problema de Máxima Interseção de k-Subconjuntos (kMIS). Nossa

Tabela 5: Média dos tempos computacionais agrupados pelo tamanho de L.

$\|L\|$	Ant_{VND}	$GRASP_{RG+TS}$
40	0,03	0,06
60	0,09	0,13
80	0,21	0,23
100	0,41	0,36
140	0,97	0,69
180	1,89	1,18
200	2,44	1,52
240	3,64	2,22
280	5,67	3,20
300	6,69	3,76

meta-heurística combina estratégias das duas principais versões dessa meta-heurística existentes na literatura, a Ant System [Dorigo et al., 1996] e a Ant Colony System [Dorigo e Gambardella, 1997]. Da versão Ant System incorporamos a não atualização local dos feromônios e da Ant Colony System incorporamos a fórmula da probabilidade utilizada na escolha do caminho feita pelas formigas.

Além disso, adicionamos uma estratégia reativa na escolha de quais formigas ficarão ativas, baseado nos valores da soluções encontradas pelas formigas. Também adicionamos uma busca local do tipo VND que é aplicada nas melhores soluções encontradas pela formigas ativas.

Baseado nesses resultados mostrado na Seção 3.3, podemos concluir que para instâncias de densidades altas e k de médio para alto, o Ant_{VND} é o melhor algoritmo. Essas são instâncias mais difíceis de resolver via os algoritmos exatos [Costa et al., 2020]. Para as instâncias de densidade alta e k baixo o $GRASP_{RG+TS}$ é o mais eficaz.

Agora, quando não temos informações acerca da densidade e valor de k, apenas sobre o balanceamento/desbalanceamento da instância, o Ant_{VND} é o mais indicado, por ter as melhores métricas quando agrupados por grupos. E por fim, destacamos que para instâncias maiores e que o tempo é um fator importante, o $GRASP_{RG+TS}$ é o mais indicado.

Em geral, o Ant_{VND} é o melhor algoritmo quando o valor da solução é determinante para a escolha do algoritmo. Agora, quando o tempo é o fator determinante, o $GRASP_{RG+TS}$ é o mais eficaz.

Como trabalho futuro, iremos procurar melhorar o tempo computacional, um fator que o nosso algoritmo foi inferior ao $GRASP_{RG+TS}$. Um caminho que continuaremos a seguir será em procurar uma quantidade menor de formigas ativas que o Ant_{VND} continue a encontrar boas soluções com tempo computacional menor.

Outro trabalho futuro que pretendemos analisar será uma implementação paralela do algoritmo. Repare que como não ocorre atualização local dos feromônios, então as formigas podem trabalhar de forma paralela. Consequentemente, podemos tirar proveito disso e possivelmente obter melhores tempos computacionais.

Agradecimentos

Essa pesquisa foi parcialmente apoiado pela FUNCAP PS1-0186-00155.01.00/21, FUNCAP PNE 0112.00061.01.00-16, CNPq Proc. 309315/2019-0, CNPq Proc. 422912/2021-2 e Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de

Financiamento 001. Agradecemos ao revisor por seus comentários e sugestões que nos ajudaram a melhorar o artigo.

Referências

Bogue, E. T., de Souza, C. C., Xavier, E. C., e Freire, A. S. (2013). O problema da máxima interseção de k-subconjuntos. In *Anais do XLV SBPO*, p. 2416–2425, Natal. Sobrapo.

Bogue, E. T., de Souza, C. C., Xavier, E. C., e Freire, A. S. (2014). An integer programming formulation for the maximum k-subset intersection problem. In *Combinatorial Optimization*, p. 87–99, Cham. Springer International Publishing. ISBN 978-3-319-09174-7.

Casado, A., Pérez-Peló, S., Sánchez-Oro, J., e Duarte, A. (2022). A grasp algorithm with tabu search improvement for solving the maximum intersection of k-subsets problem. *Journal of Heuristics*, 28(1):121–146.

Costa, J. R. d. F., Dias, F. C. S., e de A. Tavares, W. (2018). Um algoritmo vns para o problema da máxima interseção de k-subconjuntos. In *Anais do 50 Simpósio Brasileiro de Pesquisa Operacional*. Sobrapo.

Costa, J. R. d. F., Dias, F. C. S., e de A. Tavares, W. (2020). Um algoritmo branch-and-bound para o problema da máxima interseção de k-subconjuntos. *Anais do LII Simpósio Brasileiro de Pesquisa Operacional*.

Dias, F. C., Tavares, W. A., e de Freitas Costa, J. R. (2020). Reactive vns algorithm for the maximum k-subset intersection problem. *Journal of Heuristics*, 26(6):913–941.

Dorigo, M. e Gambardella, L. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66.

Dorigo, M., Maniezzo, V., e Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1): 29–41.

Dorigo, M. e Stützle, T. (2003). *The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*, p. 250–285. Springer US, Boston, MA.

Gilbert, E. N. (1959). Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144.

Nussbaum, D., Pu, S., Sack, J.-R., Uno, T., e Zarrabi-Zadeh, H. (2012). Finding maximum edge bicliques in convex bipartite graphs. *Algorithmica*, 64(2):311–325.

Shieh, M.-Z., Tsai, S.-C., e Yang, M.-C. (2012). On the inapproximability of maximum intersection problems. *Information Processing Letters*, 112(19):723–727.

Stützle, T. e Hoos, H. H. (2000). Max min ant system. *Future Generation Computer Systems*, 16 (8):889–914.

Vinterbo, S. (2002). A note on the hardness of the k-ambiguity problem. Technical report, Harvard Medical School, Boston, MA, USA.

Xavier, E. C. (2012). A note on a maximum k-subset intersection problem. *Information Processing Letters*, 112(12):471–472.