Trabalho de Conclusão de Curso

Uma heurística Iterated Local Search para o problema da Interseção Máxima de k-Subconjuntos

Hélder Silva Ferreira Lima
hsfl@ic.ufal.br

Orientador:
Dr. Rian Gabriel dos Santos Pinheiro

Maceió, Abril de 2025

Hélder Silva Ferreira Lima

Uma heurística Iterated Local Search para o problema da Interseção Máxima de k-Subconjuntos

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Dr. Rian Gabriel dos Santos Pinheiro

Maceió, Abril de 2025

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico
Bibliotecária: Girlaine da Silva Santos – CRB-4 – 1127

L732u Lima, Hélder Silva Ferreira.
Uma heurística Iterated Local Search para o problema da interseção máxima de k-Subconjuntos / Hélder Silva Ferreira Lima. – 2025.
37 f.: il.

Orientador: Rian Gabriel dos Santos Pinheiro.
Monografia (Trabalho de Conclusão de Curso em Computação) - Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2025.

Bibliografia: f. 27-29.
Apêndice: f. 30-37.

1. Otimização Combinatória. 2. Problema da interseção máxima de k-Subconjuntos. 3. Iterated Local Search. 4. Meta- heurísticas. I. Título.

CDU: 004.023

Agradecimentos

Agradeço, antes de tudo, à Santíssima Trindade, sem cuja graça nada seria possível em minha vida. Agradeço à Virgem Maria, Mãe Amável, a quem dedico este trabalho.

Ao meu tio-avô, o padre Manoel José dos Santos, pelo suporte essencial nos anos de graduação.

Aos meus pais e à minha irmã, pelo apoio às minhas decisões e por proporcionar condições favoráveis para segui-las.

À melhor surpresa que a universidade me deu, Giselle, minha namorada, minha flor.

Ao meu orientador, professor Rian Pinheiro, por me acompanhar e me corrigir com tanta paciência e diligência.

Aos professores Fábio Coutinho e André Aquino, pelas oportunidades e pela confiança nos projetos do TATU e do Orion, respectivamente.

Aos amigos Claudemir, Rui, Pedro, Maurício, Cortez, Yuri e tantos outros, que tornaram a graduação mais proveitosa e mais leve.

Finalmente, a todos que, direta ou indiretamente, contribuíram para a minha vida acadêmica e profissional.

“Tanto ch’i’ vidi de le cose belle
che porta ’l ciel, per un pertugio tondo.
E quindi uscimmo a riveder le stelle.”

— Dante Alighieri, *Inferno*, XXXIV, 137-139

Resumo

Dada uma coleção L de n subconjuntos de um conjunto finito de elementos R, o problema da Interseção Máxima de k-Subconjuntos (kMIS) consiste em encontrar $L' \subseteq L$ com $|L'| = k$ de modo que a interseção dos subconjuntos em L' seja máxima. Este trabalho propõe uma meta-heurística de Iterated Local Search (ILS) para o kMIS. A meta-heurística proposta se baseia em uma estrutura de vizinhança de troca, que faz uso inovador de uma estrutura de dados para acelerar a fase de busca local e, assim, melhorar o desempenho. Testes computacionais comprovam a superioridade desta proposta em relação a algoritmos da literatura, encontrando, em média, soluções de qualidade superior ao estado da arte.

Key-words: Otimização Combinatória; Problema da interseção máxima de k-Subconjuntos; Iterated Local Search; Meta-heurísticas

Abstract

Given a collection L of n subsets of a finite set of elements R, the Maximum Intersection of k-Subsets problem (kMIS) consists of finding $L' \subseteq L$ with $|L'| = k$ such that the intersection of the subsets in L' is maximum. This work proposes an Iterated Local Search (ILS) metaheuristic to kMIS. The proposed metaheuristic relies on a swap neighborhood structure, which makes innovative use of a data structure to speed up the local search phase and thus improve the performance. Computational tests prove the superiority of this proposal over algorithms in the literature, finding on average solutions of higher quality than the state of the art.

Key-words: Combinatorial Optimization; Maximum k-Subsets problem; Iterated Local Search; Metaheuristics

Conteúdo

1

Introdução

Em um mundo marcado pela produção de volumes cada vez maiores de dados e pela complexidade das relações entre eles, a capacidade de analisar e identificar padrões e sobreposições torna-se essencial para o avanço científico e tecnológico. Nesse contexto, problemas de otimização caracterizados pela seleção sob restrições sempre despertaram especial interesse à comunidade científica devido às suas diversas aplicações no mundo real, em áreas como privacidade dos dados, biologia computacional e logística. O Problema da Interseção Máxima de k Subconjuntos (*Maximum Intersection of k-subsets*) (kMIS) destaca-se entre tais problemas devido à sua alta aplicabilidade e pela sua complexidade intrínseca.

Intuitivamente, o kMIS pode ser compreendido como o problema de selecionar k subconjuntos de um conjunto maior de forma que a quantidade de elementos em comum entre eles seja a máxima possível. Ele será formalmente definido no Capítulo 2, mas é possível perceber desde já o potencial do problema para capturar relações significativas em diferentes contextos.

1.1 Justificativa

Como foi dito, problemas de seleção sob restrição encontram aplicações em diversas áreas da atividade humana, como escalonamento distribuído [23], posicionamento de sensores [22], segmentação de imagens [8], entre muitos outros.

O kMIS, em específico, foi estudado na literatura lado a lado com as suas aplicações práticas. O problema foi inicialmente estudado por Vinterbo [33] no contexto de uma aplicação em controle de divulgação de dados, especificamente para dados hospitalares. No caso estudado, tem-se um conjunto de pacientes de um hospital e um conjunto dos dados dos pacientes, isto é, atributos que eles podem apresentar (Figura 1.1). Sendo necessário divulgar a maior quantidade possível de dados para futura análise, assegurando, ao mesmo tempo, que a privacidade

dos pacientes não seja violada, um dado somente pode ser divulgado se ao menos k pacientes o possuam em comum. Deste modo, o objetivo do problema é encontrar k pacientes que compartilhem a maior quantidade possível de dados.

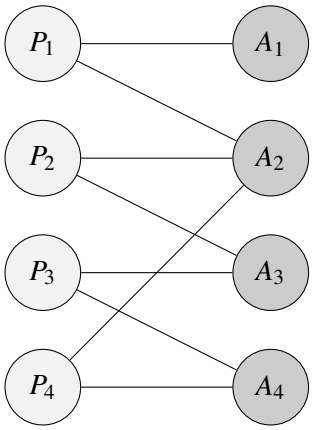


Figura 1.1: Os vértices à esquerda representam os pacientes; os vértices à direita, os atributos. Um vértice P_i está conectado a um vértice A_j caso o paciente P_i possua o atributo A_j.

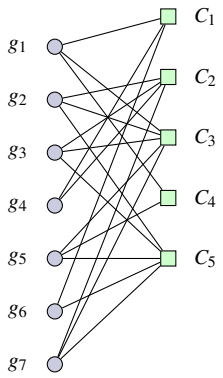
Em Nussbaum et al. [28], cita-se uma aplicação do kMIS na biologia molecular, no uso de *DNA Microarray* para estudar a interação entre genes e condições de teste. Dados de *Microarray* são normalmente apresentados na forma de uma matriz bidimensional, cujas linhas correspondem aos genes e cujas colunas, às condições de teste. Cada entrada $[i, j]$ da matriz representa o nível de expressão de um dado gene i medido sob uma dada condição j, podendo haver aumento, decréscimo ou nenhuma mudança significativa (Figura 1.2 (a)). Capturar a relação entre conjuntos de genes e conjuntos de condições é importante, na biologia, para a compreensão de processos biológicos a nível celular e molecular. Tal análise é inviável sem o uso de recursos computacionais. Assim, esse problema pode ser modelado como sendo uma instância do kMIS tal que de um lado existam os genes e do outro, as condições de teste; caso o nível de expressão de um gene aumente ou diminua significativamente sob uma condição de teste, eles são conectados (Figura 1.2 (b)). O objetivo se torna selecionar os k genes que tenham o maior número de condições em comum.

Ainda, Bogue [7] estuda uma aplicação do problema em sistemas de recomendação. Procura-se montar uma seleção de artistas musicais (para integrar o programa de um festival, por exemplo) que tenham o maior número de fãs em comum, com o intuito de agradar a um público o mais amplo possível. Assim, é possível modelar uma instância do kMIS identificando um conjunto com os artistas musicais disponíveis e outro como o das pessoas interessadas em música. O objetivo é selecionar os k artistas que tenham o maior número de fãs em comum.

Nota-se, portanto, uma fácil aplicabilidade do kMIS em áreas que envolvam seleção sob restrições, especialmente quando a similaridade ou a sobreposição entre grupos de dados ou conjuntos é um fator crucial. Assim, um estudo do problema faz-se relevante. Ao se ter em vista que o kMIS é um problema aparentemente intratável — isto é, não se conhecem algoritmos eficientes para encontrar soluções ótimas para ele — surge a necessidade de se desenvolver

	C_1	C_2	C_3	C_4	C_5
g_1	+	-	+	+	∅
g_2	∅	+	+	-	+
g_3	-	+	+	-	+
g_4	+	+	-	∅	-
g_5	-	-	+	+	+
g_6	+	∅	-	-	+
g_7	-	+	+	∅	+

(a) Matriz de condições



(b) Grafo bipartido

Figura 1.2: (a) Uma matriz de dados *microarray*; Cada símbolo $+$ ou $-$ representa um aumento ou decréscimo, respectivamente, dos níveis de expressão de um gene, enquanto o símbolo ∅ representa que não há mudança significativa. (b) Um grafo bipartido que representa os aumentos significativos na matriz de dados *microarray*. Fonte: Nussbaum et al. [28]

técnicas para encontrar boas soluções com baixo custo computacional, o que se dá através de algoritmos heurísticos.

1.2 Objetivos

1.2.1 Objetivo geral

Este trabalho tem como objetivo geral desenvolver uma meta-heurística *Iterated Local Search* (ILS) [26] para o Problema da Interseção Máxima de k Subconjuntos e mostrar a efetividade de tal proposta, visando evoluir o *state-of-the-art* do problema.

1.2.2 Objetivos específicos

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

- Implementar uma metaheurística *Iterated Local Search* (ILS) para resolver o Problema da Interseção Máxima de k Subconjuntos;
- Aprimorar o algoritmo por meio de estratégias para escapar de ótimos locais e acelerar a convergência;
- Propor novas estruturas de dados eficientes que consigam acelerar as buscas locais da literatura;
- Avaliar os componentes da meta-heurística proposta de forma a saber quais os componentes possuem um maior impacto na efetividade do algoritmo.

1.3 Contribuições

O presente trabalho apresenta uma evolução do trabalho intitulado “Algoritmo ILS-VND para o Problema da Interseção Máxima de k-Subconjuntos” [16], publicado pelo autor no LVI Simpósio Brasileiro de Pesquisa Operacional. Uma versão estendida do primeiro artigo foi submetida para publicação em periódico.

1.4 Organização do Trabalho

O restante deste trabalho se organiza da seguinte forma: O Capítulo 2 define formalmente o Problema da Interseção Máxima de k Subconjuntos e apresenta uma breve revisão da literatura do problema. O Capítulo 3 apresenta os fundamentos teóricos do desenvolvimento deste trabalho. A seguir, o Capítulo 4 descreve o algoritmo ILS proposto, detalhando cada um de seus componentes e especificidades. Finalmente, o Capítulo 5 apresenta os experimentos computacionais realizados, os resultados obtidos e algumas análises estatísticas.

2

Caracterização do Problema

Este capítulo tem como objetivo introduzir o Problema da Interseção Máxima de k Subconjuntos, estudado neste trabalho. A Seção 2.1 o define formalmente. Já a Seção 2.2 apresenta uma breve revisão da literatura relacionada.

2.1 Problema da Interseção Máxima de k Subconjuntos

O Problema da Interseção Máxima de k Subconjuntos (*k-Maximum Intersection Subsets* - kMIS) é um problema de otimização cujo objetivo é selecionar um número k de conjuntos tais que a interseção entre eles seja maximizada, ou seja, que eles compartilhem o maior número de elementos em comum.

O kMIS pode ser formalmente definido da seguinte forma: dada uma coleção $L = \{S_1, ..., S_n\}$ de n subconjuntos de um conjunto finito de elementos $R = \{e_1, ..., e_m\}$, e um inteiro positivo k, o objetivo é encontrar $L' \subseteq L$ com $|L'| = k$ tal que os conjuntos de L' tenham o número máximo de elementos em comum, ou seja, $|\bigcap_{S \in L'}|$ seja máximo. O problema pode ser modelado como um grafo bipartido $G = (L \cup R, E)$ no qual existe uma aresta (S_i, e_j) se e somente se $e_j \in S_i$.

Por exemplo, a Figura 2.1 ilustra uma instância do kMIS tal que $R = \{e_1, e_2, e_3, e_4, e_5\}$ e $L = \{S_1, S_2, S_3, S_4\}$, com $S_1 = \{e_1, e_2, e_3\}$, $S_2 = \{e_1, e_2, e_3, e_5\}$, $S_3 = \{e_2, e_5\}$ e $S_4 = \{e_3, e_4\}$. Para $k = 2$, deve-se encontrar uma combinação de dois subconjuntos que gere a interseção máxima. Assim, $L' = \{S_1, S_3\}$, com $|S_1 \cap S_3| = 1$, é uma possível solução. No entanto, $L' = \{S_1, S_2\}$ com $|S_1 \cap S_2| = 3$, é melhor, e, para a instância, é uma solução ótima.

2.2 Trabalhos Relacionados

Até onde se sabe, o kMIS foi citado inicialmente por Vinterbo [33], em que ele é definido e provado $\mathcal{NP}$-difícil. O autor apresentou o problema mais geral da k-ambiguidade, definido

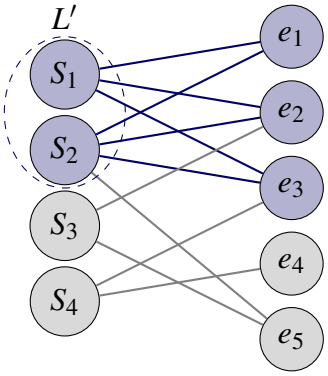


Figura 2.1: Exemplo de instância do k-MIS. Para $k = 2$, a interseção máxima de k subconjuntos será obtida por S_1 e S_2, com cardinalidade 3. Fonte: Autor

da seguinte forma: seja $U = \{x_i\}_{i=1}^{n}$ um conjunto de n objetos de interesse. Dado um inteiro positivo $k \leq n-1$, deseja-se descobrir quanta informação a respeito de um elemento x_i deve ser descartada para torná-lo indiscernível dos demais k elementos x_j em U. No mesmo artigo, o problema da k-ambiguidade é demonstrado ser equivalente ao da k-união mínima, por sua vez equivalente ao kMIS. A prova de que o kMIS (assim como os outros dois, por equivalência) é $\mathcal{NP}$-difícil se dá por redução a partir do problema do Subgrafo Completo Bipartido Balanceado [17].

Xavier [35] demonstrou que o kMIS é inaproximável, isto é, ele não admite um algoritmo $\frac{1}{N^{\varepsilon}}$-aproximado, sendo N o tamanho da entrada e ε uma constante. Novamente demonstrou-se a complexidade do kMIS, desta vez por meio de redução do problema *Maximum Edge Biclique* (MEB).

As primeiras propostas de métodos exatos para resolver o kMIS vieram com Bogue [7], em que cinco formulações de programação linear inteira são apresentadas para o problema. Três dessas formulações baseiam-se no método de *Branch-and-Bound* e duas no método de *Branch-and-Cut*. Dois dos modelos baseiam-se em formulações anteriores, desenvolvidas para o MEB; um é inteiramente novo, denominado M_{ARESTA}. O autor introduziu, também, uma heurística construtiva gulosa, denominada `kInter`, e uma meta-heurística GRASP (*Greedy Randomized Adaptative Search Procedure*) Reativo, as primeiras de que se tem notícia para o kMIS, além de um método eficiente de pré-processamento para reduzir o tamanho da entrada. A avaliação computacional foi feita utilizando instâncias geradas aleatoriamente e obtidas do serviço musical *online* LastFM.

Outra tentativa de resolver o kMIS a partir de uma meta-heurística veio com Dias et al. [14], em que se propõe um algoritmo VNS (*Variable Neighborhood Search*) Reativo para o problema. Nele, diversas heurísticas e meta-heurísticas são combinadas com componentes dinâmicos para diversificar a busca. Os autores apresentam uma versão estendida da heurística construtiva `kInter` [7] para escolher a melhor dentre diversas soluções geradas, combinada com uma fase de intensificação chamada *Path-Relinking* [18]. Incorporam, também, a fase de construção do

GRASP apresentado em Bogue [7] na fase de perturbação do VNS, criando o componente reativo. Há uma estratégia de atualização dinâmica na fase de perturbação, que aumenta a vizinhança quando a busca parece travada. Outra contribuição importante do trabalho é o uso de *bitsets* para representar as instâncias e soluções, o que acelera as operações de interseção de conjuntos por meio do paralelismo de bits. O VNS é testado com instâncias geradas pelos próprios autores e comparado ao GRASP Reativo [7], obtendo resultados superiores na maior parte dos casos.

Finalmente, os autores de Casado et al. [9] apresentaram uma meta-heurística GRASP, cuja fase de busca local funciona com uma Busca Tabu cuja estrutura de vizinhança é um operador de troca. A escolha foi feita porque a Busca Tabu, utilizando uma memória adaptativa, permite movimentos que geram soluções de pior qualidade; com isso, espera-se diversificar a busca e escapar de ótimos locais. Os autores também modelam o problema por meio de *bitsets* [14], adicionando à estratégia o armazenamento em cache de informações da solução durante a fase de busca, com o objetivo de reduzir o número de operações do algoritmo. Os experimentos demonstram que o GRASP tem, em média, um tempo de execução menor e gera soluções de melhor qualidade do que o VNS Reativo [14].

3

Fundamentação teórica

Este capítulo introduz a teoria que fundamenta o presente trabalho. A Seção 3.1 introduz a teoria da $\mathcal{NP}$-completude. A Seção 3.2 apresenta o conceito de otimização combinatória. As Seções 3.3 e 3.4 tratam de abordagens exatas e heurísticas, respectivamente.

3.1 NP-completude

A base da teoria da $\mathcal{NP}$-completude foi estabelecida por Stephen Cook, no trabalho *The Complexity of Theorem Proving Procedures* [11]. Esse trabalho trouxe à tona alguns dos assuntos que mais interessa a comunidade científica de computação e matemática, principalmente sobre a chamada questão $\mathcal{P}$ *versus* $\mathcal{NP}$.

Antes de se debruçar sobre este assunto, é necessário introduzir o conceito de problema. Segundo Garey and Johnson [17], um problema é, em geral, uma pergunta a ser respondida, geralmente com alguns parâmetros cujos valores não são especificados. Um problema é enunciado pela descrição geral dos parâmetros e pela descrição das propriedades de uma resposta, ou solução, que o satisfaça. Obtém-se uma instância de um problema especificando valores para os parâmetros.

$\mathcal{P}$ é a classe dos problemas de decisão — para os quais a resposta deve ser "sim" ou "não" — que podem ser resolvidos em tempo polinomial, isto é, que podem ser resolvidos em um tempo $O(n^k)$ para uma constante k, em que n é o tamanho da entrada do problema. Já a classe $\mathcal{NP}$ consiste dos problemas de decisão verificáveis em tempo polinomial; ou seja, dado um "certificado" de uma solução "sim", seria possível verificar se o certificado é correto em tempo polinomial. Tome, por exemplo, o kMIS; ele pode ser formulado como um problema de decisão da seguinte forma: dados um conjunto $L = \{S_1, \ldots, S_n\}$, um inteiro positivo k e um valor limite B, existe uma seleção de k subconjuntos de L cuja interseção tenha cardinalidade maior ou igual a B? Nesse caso, um certificado para a solução "sim" seria uma seleção $L' = \{S_1, ..., S_k\}$

de subconjuntos. É possível verificar em tempo polinomial que a seleção contém k subconjuntos distintos, bem como se o valor $|\bigcap_{S_i \in L'}|$ é maior ou igual a B.

A relação entre $\mathcal{P}$ e $\mathcal{NP}$ é fundamental para a teoria da $\mathcal{NP}$-completude. Primeiramente, é fácil ver que $\mathcal{P} \subseteq \mathcal{NP}$. Afinal, se um problema pertence a $\mathcal{P}$, ele pode ser resolvido em tempo polinomial. A questão em aberto seria determinar se $\mathcal{P} \neq \mathcal{NP}$.

Para tanto, volta-se a atenção para a classe dos problemas $\mathcal{NP}$-completos. Um problema p $\mathcal{NP}$-completo está em $\mathcal{NP}$ e tem a propriedade especial de que todos os demais problemas em $\mathcal{NP}$ podem ser transformados (ou reduzidos) a p em tempo polinomial. Isso significa que, se qualquer problema $\mathcal{NP}$-completo puder ser resolvido em tempo polinomial, então todos os demais problemas em $\mathcal{NP}$ terão um algoritmo em tempo polinomial e pertencerão, assim, a $\mathcal{P}$. No entanto, boa parte dos teóricos acredita que os problemas $\mathcal{NP}$-completos são intratáveis, ou seja, tão difíceis que nenhum algoritmo em tempo polinomial poderia resolvê-los. Essa crença existe devido à enorme variedade de problemas $\mathcal{NP}$-completos estudados até o momento, sem que nenhum algoritmo em tempo polinomial fosse descoberto para eles. Ao mesmo tempo, porém, empreendeu-se um grande esforço para provar que os problemas $\mathcal{NP}$-completos são intratáveis, sem que nenhuma conclusão fosse alcançada.

Caso um problema seja tão difícil quanto os problemas $\mathcal{NP}$-completos, mas não pertença a $\mathcal{NP}$, ele pertence à classe dos $\mathcal{NP}$-difíceis. Problemas $\mathcal{NP}$-completos também podem ser reduzidos a problemas $\mathcal{NP}$-difíceis, de modo que todo problema $\mathcal{NP}$-completo é também $\mathcal{NP}$-difícil. A Figura 3.1 ilustra as relações das classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-completo e $\mathcal{NP}$-difícil, caso $\mathcal{P} \neq \mathcal{NP}$.

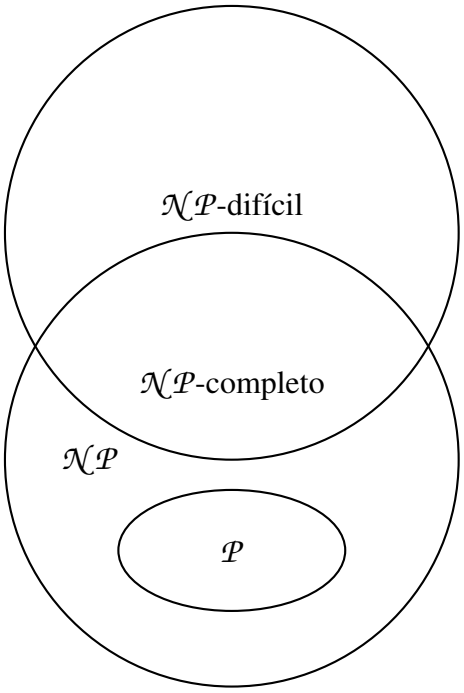


Figura 3.1: Diagrama ilustrando as relações entre as classes de problemas $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-completo e $\mathcal{NP}$-difícil, caso $\mathcal{P} \neq \mathcal{NP}$. Fonte: Autor

Em suma, resolver qualquer problema $\mathcal{NP}$-completo confirmaria que $\mathcal{P} = \mathcal{NP}$. Porém, sem uma conclusão alcançada até o momento, $\mathcal{P}$ *versus* $\mathcal{NP}$ se mantém um dos problemas abertos da computação mais intrigantes e famosos da história.

3.2 Otimização Combinatória

A Otimização Combinatória é um ramo da matemática aplicada dedicado a encontrar a melhor solução possível dentro de um conjunto de valores de uma função. Esse é um problema com aplicações em diversos ramos da atividade humana, como a Pesquisa Operacional e a Inteligência Artificial.

Segundo Papadimitriou and Steiglitz [29], um problema de otimização é aquele cuja instância é um par $(D,\ c)$, em que D é o domínio dos pontos viáveis e c a função de custo, um mapeamento

$$c : D \to \mathbb{R}$$

O problema é encontrar um $x^* \in D$ para o qual $c(x^*) \leq c(x)\ \forall x \in D$ (problemas de minimização) ou $c(x^*) \geq c(x)\ \forall x \in D$ (problemas de otimização). O ponto x^* é o chamado ótimo global. Problemas de otimização se dividem em contínuos e discretos. Em problemas contínuos, D se constitui de números reais ou até mesmo uma função; em problemas discretos (ou, ainda, combinatórios), D é formado por variáveis inteiras, como grafos ou conjuntos.

Um algoritmo para problemas de otimização é dito *exato* se sempre retorna uma solução ótima para uma dada instância do um problema. Ele será chamado *eficiente* caso o faça em tempo polinomial. No entanto, muitos problemas de otimização são intratáveis, de modo que se pode optar por buscar soluções próximas da optimalidade em espaços de busca reduzidos. Tais soluções se chamam *ótimos locais* e são normalmente encontrada por meio de exploração de vizinhanças. Uma *vizinhança* $N(x)$ de uma solução x são as soluções próximas a x, alcançadas por meio de algum mecanismo de mudança, o qual pode ser definido de várias formas a depender do problema. A exploração de vizinhanças em busca de ótimos locais costuma ser o objetivo de algoritmos heurísticos de busca local, descritos na Seção 3.4.

3.3 Métodos Exatos

Os métodos exatos são técnicas que garantem a obtenção da solução ótima para problemas de otimização, desde que certas condições matemáticas sejam satisfeitas. Esses métodos contrastam com abordagens heurísticas ou meta-heurísticas, que buscam soluções viáveis de boa qualidade, mas sem garantias de otimalidade. Entre os principais métodos exatos destacam-se algoritmos baseados em programação linear, *branch-and-bound*, programação dinâmica e

decomposição de problemas, cada um com aplicações específicas dependendo da estrutura do modelo matemático em questão.

Um dos pilares da otimização exata é o algoritmo Simplex, proposto por Dantzig [13], que resolve problemas de programação linear explorando a geometria convexa das soluções viáveis. Apesar de sua complexidade computacional teórica não ser polinomial, o Simplex é eficiente na prática para muitos problemas reais, especialmente quando combinado com técnicas de pré-processamento e pivoteamento inteligente [3]. Para problemas com variáveis inteiras, o método *branch-and-bound* [25] tornou-se fundamental. Ele divide o espaço de soluções em subproblemas, calculando limites inferiores e superiores para eliminar regiões inviáveis ou subótimas, sendo amplamente utilizado em problemas de programação inteira mista [34]. Além disso, métodos de decomposição de Benders [6] e planos de corte [19] são estratégias clássicas para lidar com problemas de grande escala, fragmentando-os em partes menores ou adicionando restrições iterativamente para convergir à solução ótima.

Já a programação dinâmica, formalizada por Bellman [4], é aplicável a problemas com propriedade de subestrutura ótima, como roteamento e planejamento de produção. Ela decompõe o problema em estágios sequenciais, armazenando soluções parciais para evitar recálculos, o que a torna eficiente em cenários com sobreposição de subproblemas [12].

A escolha do método exato adequado depende de fatores como a natureza das variáveis (contínuas, inteiras), a convexidade das funções envolvidas e a dimensão do problema. Embora métodos exatos sejam computacionalmente exigentes para instâncias muito grandes, avanços recentes em *hardware* e otimizações algorítmicas, como paralelização e técnicas híbridas (ex.: combinação de *branch-and-bound* com heurísticas), têm ampliado sua aplicabilidade [27].

3.4 Métodos Heurísticos

Normalmente, métodos heurísticos são usados para resolver problemas de otimização quando utilizar métodos exatos se torna inviável. Trata-se de uma situação bastante comum na prática, pois a resolução de problemas de otimização têm grande impacto no mundo real e, ao mesmo tempo que esses problemas vêm se tornando mais complexos, também se exige que soluções para eles sejam encontradas de forma mais rápida. Heurísticas utilizam conhecimentos específicos de um problema para encontrar boas soluções a um baixo custo computacional. As meta-heurísticas (MHs), por sua vez, são heurísticas de propósito geral, isto é, elas podem ser aplicadas a qualquer problema por meio de algumas poucas modificações na sua estrutura de funcionamento.

A principal característica das MHs é a interação entre procedimentos de intensificação, com os quais se explora o espaço de busca, e de diversificação, por meio dos quais se foge de ótimos locais. O objetivo é executar uma busca robusta no espaço de soluções de um problema.

Ao longo dos anos, surgiu uma grande diversidade de meta-heurísticas aplicadas aos mais

diversos problemas. As diversas características de tais propostas fez surgir a necessidade de classificá-las e agrupá-las a partir das suas características essenciais. Talvez a classificação mais geral seja a que separa as meta-heurísticas entre *baseadas em trajetória* e *baseadas em população*. Meta-heurísticas baseadas em população utilizam processos de busca em um conjunto finito de soluções candidatas, chamado população, as quais são melhoradas e utilizadas para gerar novas soluções ao longo das iterações. Nesse caso, as meta-heurísticas mais estudadas são os algoritmos de Computação Evolucionária, como os Algoritmos Genéticos [24] e *Ant Colony* [5]. Meta-heurísticas baseadas em trajetória, por sua vez, focam em uma única solução candidata, que é modificada e melhorada iterativamente. Por este motivo, também são chamadas de meta-heurísticas de solução única, ou ainda de meta-heurísticas de busca. Entre as meta-heurísticas baseadas em trajetória mais estudadas, estão o *Simulated Annealing* [31], *Variable Neighborhood Structure* (VND) [20] e *Iterated Local Search* (ILS) [26].

Dentre as diversas meta-heurísticas estudadas na liteatura, a mais importante para este trabalho é o *Iterated Local Search*, a qual será aprofundada na subseção a seguir.

3.4.1 *Iterated Local Search*

O ILS [26] é uma meta-heurística amplamente utilizada para resolver problemas de otimização. Seu funcionamento baseia-se na exploração iterativa de uma solução inicial por meio de procedimentos de perturbação e busca local. A perturbação consiste na substituição de uma solução s por uma nova solução s', com o objetivo de escapar de ótimos locais. Para que essa etapa seja eficaz, é comum utilizar elementos randomizados, pois eles evitam que o algoritmo fique preso em ótimos locais e permitem uma exploração mais ampla do espaço de soluções.

Já a busca local é um procedimento de intensificação que, dada uma solução s, retorna, para problemas de maximização, outra solução s^* cujo custo $\texttt{cost}(s^*)$ é maior ou igual ao custo $\texttt{cost}(s)$. Isso normalmente ocorre por meio da definição de uma ou mais estruturas de vizinhança, que estabelecem um conjunto de soluções vizinhas alcançáveis a partir de s por meio de pequenas modificações, conhecidas como movimentos.

4

Metodologia

Neste capítulo é apresentado o algoritmo ILS proposto para resolver o problema da Interseção Máxima de k Subconjuntos. Sua estrutura geral é descrita na Seção 4.1. Antes de se debruçar sobre as fases do algoritmo, é fundamental compreender a representação das soluções do problema, apresentada na Seção 4.2. Também é necessário descrever a heurística utilizada para a geração da solução inicial do algoritmo, o que se dá na Seção 4.3. As fases do algoritmo ILS são descritas nas seções seguintes. A Seção 4.4 descreve o funcionamento da fase de busca local, incluindo a estrutura de dados utilizada para acelerá-la. A seguir, a Seção 4.5 descreve a fase de perturbação do algoritmo. Por fim, a Seção 4.6 descreve o critério de aceitação para novas soluções no processo de busca.

4.1 Estrutura Geral do Algoritmo

O pseudocódigo do ILS proposto neste trabalho pode ser visto no Algoritmo 1. A solução inicial L^* é gerada (linha 2) e imediatamente refinada na função `LocalSearch` (linha 3), que executa a busca local. Em seguida, no loop, uma cópia L' de L^* se torna a solução atual, que entra na função de perturbação (linha 5), e, a seguir, na busca local (linha 6). A cada iteração ela entra na função `AcceptanceCriteria`, para ser avaliada, sendo comparada a L^*. Ao final, L^* é retornada.

4.2 Representação das Soluções

Seguindo Dias et al. [14] e Casado et al. [9], foram utilizados *bitsets* para representar as instâncias e soluções do kMIS. Essa estratégia leva em conta as desvantagens de se utilizar uma representação direta, que, para uma instância modelada como um grafo $G = (L \cup R, E)$, seria a lista dos conjuntos escolhidos de L. Com essa representação, a avaliação de uma solução se

Algoritmo 1 Iterated Local Search (ILS)

```
1:  function ILS(L, k)
2:      L* ← Construction(L, k)                    ▷ Sessão 4.3
3:      L* ← LocalSearch(L*)                       ▷ Sessão 4.4
4:      while stop condition not reached do
5:          L' ← Perturb(L*)                       ▷ Sessão 4.5
6:          L' ← LocalSearch(L')
7:          L* ← AcceptanceCriteria(L', L*)        ▷ Sessão 4.6
8:      end while
9:      return L*
10: end function
```

daria pelo cálculo da interseção de todos os conjuntos escolhidos de L, o que pode ser computacionalmente exigente para instâncias médias e grandes.

Um *bitset* é um *array* cujos elementos só podem ser 0 ou 1. Na implementação do ILS para o kMIS, cada $S \in L$ tem o próprio *bitset* B de tamanho $|R|$, em que a posição p indica a presença ou ausência do elemento $e_p \in R$ em S. Com *bitsets*, o cálculo da interseção entre dois subconjuntos S_i e S_j se dá em uma operação lógica `AND` entre B_i e B_j, feita por meio de paralelismo de bits para alcançar maior eficiência. Caso o número de elementos seja menor ou igual ao comprimento da palavra da CPU, o cálculo é feito em tempo constante; caso contrário, divide-se o *bitset* em *arrays* do tamanho da palavra da CPU e a operação é feita automaticamente em cada um dos *arrays*, também em tempo constante para eles. Trata-se de uma grande vantagem sobre a modelagem direta citada acima, na qual a interseção entre subconjuntos teria uma complexidade de $O(|R|)$.

Uma solução L' também tem um *bitset* B' associado, que guarda a interseção resultante dos subconjuntos selecionados. A função objetivo $\texttt{cost}(L')$, portanto, dá-se pelo cálculo da cardinalidade de B'.

Assim, essa modelagem acelera consideravelmente o processo de busca e já obteve sucesso em outros problemas, como mostrado em Segundo et al. [32] e em Komosko et al. [21].

4.3 Fase de Construção

Para a construção da solução inicial, utiliza-se a versão estendida do algoritmo guloso `kInter` [7]. O `kInter` original consiste no seguinte: dada uma solução parcial L', seleciona-se para ser adicionado a L' um subconjunto $S \in L \setminus L'$ tal que $cost(L' \cup S)$ seja máximo. A escolha gulosa ocorre k vezes, ao fim das quais se tem uma solução viável. No início, $L' = \emptyset$ e, seguindo o critério, o primeiro subconjunto inserido é o de maior cardinalidade.

A versão estendida do `kInter` foi proposta por Dias et al. [14], com o seguinte funcionamento: primeiramente, o conjunto $L = \{S_1, S_2, \ldots, S_n\}$ é ordenado de tal forma que $|S_1| \geq |S_2| \geq \ldots \geq |S_n|$. Então, o `kInter` é executado n vezes. Na i-ésima execução, L' é inicializada com S_i.

Uma execução do `kInter` será interrompida caso $\texttt{cost}(L') \leq \texttt{cost}(L_{best})$, sendo L_{best} a melhor solução encontrada até o momento. Ao final, retorna-se a solução de maior custo entre as n geradas. Originalmente, o `kInter` estendido possui uma fase de *Path-Relinking*, mas ela não foi incorporada na implementação do ILS.

O Algoritmo 2 mostra a versão estendida do `kInter` conforme implementada neste trabalho. Inicialmente, a solução viável L_{best} é inicializada como um conjunto vazio. A seguir, para cada subconjunto $S_i \in L$ (linha 3) uma solução L' será inicializada com S_i. O loop *while* que se inicia na linha 5 trata-se de uma execução do `kInter` que constrói L' e a torna uma solução viável. Caso L' não tenha um custo maior que L_{best} (linha 8), ela é ignorada e uma nova solução L' será gerada, iniciando com o próximo $S_i \in L$. Caso contrário (linha 12), L_{best} recebe L'. Ao final, retorna-se L_{best}.

Algoritmo 2 Extended-KInter

```
1:  function EXTENDED-KINTER(L, k)
2:      L_best ← ∅
3:      for i = 1 to |L| do
4:          L' ← {S_i}
5:          while |L'| < k do
6:              Selecione um subconjunto S ∈ L \ L' tal que cost(L' ∪ {S}) seja máximo.
7:              L' ← L' ∪ {S}
8:              if cost(L') ≤ cost(L_best) then
9:                  break while
10:             end if
11:         end while
12:         if cost(L') > cost(L_best) then
13:             L_best ← L'
14:         end if
15:     end for
16:     return L_best
17: end function
```

4.4 Busca Local

A Busca Local constitui a fase de intensificação do ILS. Dada uma solução L' como entrada, a saída da Busca Local deve ser uma solução L^* cujo custo é maior ou igual ao de L'. Normalmente isso se dá com a definição de uma ou mais estruturas de vizinhanças, isto é, uma estrutura que define um espaço de soluções que podem ser alcançadas a partir de L' por meio de pequenas modificações, conhecidas como movimentos [26].

Considerando que soluções viáveis para o kMIS devem ter exatamente k subconjuntos, define-se a estrutura de vizinhança `swap(1, 1)`. Ela utiliza um operador de troca `swap`, definido para uma solução L', um conjunto $S_i \in L'$, e para um conjunto $S_j \in L \setminus L'$ da seguinte forma:

$$\texttt{swap}(L', S_i, S_j) = (L' - S_j) \cup \{S_j\} \tag{4.1}$$

Para acelerar o procedimento e prevenir tanto a realização de cálculos redundantes quanto a avaliação de movimentos incapazes de melhorar a solução atual, foi utilizado um *array* μ de k elementos associado a L', calculado antes do início da exploração da vizinhança. Para um subconjunto $S_i \in L'$, $\mu[i]$ guarda a interseção resultante de L' após a remoção de S_i, ou seja,

$$\mu[i] = \bigcap_{S \in L' - \{S_i\}} \tag{4.2}$$

Um algoritmo força bruta para preencher μ seria custoso. O cálculo de $L' - S_i$ é uma operação de complexidade $O(k)$. Como em μ esse cálculo precisa ser realizado para cada $S_i \in L'$, o algoritmo resultaria em um custo total de $O(k^2)$. Por isso, este trabalho implementa uma estratégia eficiente inspirada na solução para o problema de programação competitiva *Product of Array Except Self* (PAES)[1]: dado um *array* A de n inteiros, deve-se construir um novo *array* p, também de tamanho n, tal que $p[i]$ seja igual ao produto de todos os elementos de A exceto $A[i]$. Isso deve ser feito em $O(n)$ e sem usar o operador de divisão.

É fácil perceber que, feitas as devidas adaptações, trata-se do mesmo problema encontrado para preencher μ; a diferença é que, para μ, em vez do produto de inteiros, deseja-se calcular a interseção de conjuntos. Para calcular μ conforme a solução clássica do PAES, observa-se, primeiramente, que $\mu[i]$ é a interseção dos subconjuntos de S_1 até S_{i-1}, chamada `prefixo`, com a interseção dos subconjuntos de S_{i+1} até S_k, chamada `sufixo`, pertencentes a L'. Isto é:

$$\mu[i] = S_1 \cap \cdots \cap S_{i-1} \cap S_{i+1} \cap \cdots \cap S_k.$$

Utilizando a técnica de Programação Dinâmica [12], ambas interseções são calculadas previamente e em seguida acessadas para obter μ.

Na Programação Dinâmica, problemas são resolvidos através da combinação de soluções recursivas de subproblemas, os quais se sobrepõem. Cada subproblema é resolvido apenas uma vez e tem a solução guardada em uma tabela, o que evita o trabalho de recalcular a solução sempre que um novo subproblema for resolvido.

Com isso em mente, pode-se definir o `prefixo` recursivamente:

$$\texttt{prefixo}[i] = \begin{cases} R & \text{se } i = 1, \\ \texttt{prefixo}[i-1] \cap S_{i-1} & \text{se } i > 1. \end{cases}$$

Isto é, como o `prefixo` armazena, na i-ésima posição, $S_1 \cap \cdots \cap S_{i-1}$, é possível usar o próprio `prefixo`, na posição anterior, para calcular a posição i. Quando $i = 1$, não existe S_{i-1}; dessa forma, `prefixo[1]` recebe R, que serve de conjunto neutro.

[1] https://www.enjoyalgorithms.com/blog/product-of-array-except-self

Pode-se definir recursivamente também o `sufixo`:

$$\texttt{sufixo}[i] = \begin{cases} R & \text{se } i = k, \\ \texttt{sufixo}[i+1] \cap S_{i+1} & \text{se } i < k. \end{cases}$$

O `sufixo` armazena, na i-ésima posição, $S_{i+1} \cap \cdots \cap S_k$, portanto a posição $i+1$ do *array* pode ser usada para calcular a posição i. Como para $i = k$ não existe S_{i+1}, `sufixo[k]` recebe R como conjunto neutro.

Assim, é possível definir μ a partir das duas interseções calculadas:

$$\mu[i] = \texttt{prefixo}[i] \cap \texttt{sufixo}[i].$$

As definições acima são utilizadas para definir um algoritmo para preencher μ de forma eficiente (Algoritmo 3). μ é inicializado como um *array* de k posições. Ao fim do primeiro *loop* (linha 4), μ será igual ao `prefixo`; por isso, sua primeira posição recebe R (linha 3) e ele é preenchido conforme a definição do `prefixo` (linha 5). Isso evita a criação de um novo *array* apenas para esta operação, reduzindo a complexidade de espaço do algoritmo. Tendo-se em vista esse mesmo objetivo, calcula-se o `sufixo` e o valor final de $\mu[i]$ em apenas um *loop*. Para tanto, guarda-se o valor do `sufixo` em uma variável simples, inicializada com R (linha 7). Um segundo *loop*, que itera da penúltima posição até a primeira (linha 8), garante que $\mu[i]$, que já guardava o `prefixo[`i`]`, receba a interseção de si mesmo com o `sufixo[`i`]` (linha 9), alcançando o seu valor final, e que `sufixo` seja atualizado com o valor do próximo `sufixo[`i`]` (linha 10). Ao final, retorna-se μ. A complexidade do algoritmo é, portanto, $O(k)$, uma melhoria em relação ao cálculo direto de μ, que se daria em $O(k^2)$.

Algoritmo 3 Calculate-μ

```
1:  function CALCULATE-μ(L' = {S_1, ..., S_k})
2:      μ ← array(k)
3:      μ[1] ← R
4:      for i ← 2 to k do
5:          μ[i] ← S_{i-1} ∩ μ[i − 1]          ▷ Inicialmente, μ[i] guarda o prefixo
6:      end for
7:      sufixo ← R
8:      for i ← k − 1 to 1 do
9:          μ[i] ← μ[i] ∩ sufixo
10:         sufixo ← sufixo ∩ S_i              ▷ Atualização do sufixo
11:     end for
12:     return μ
13: end function
```

É interessante notar que a estratégia acima descrita pode ser estendida para compreender soluções parciais com ainda menos subconjuntos. Isso se daria pelo aumento das dimensões de μ, que se tornaria uma matriz. Seu preenchimento seria mais complexo.

O Algoritmo 4 mostra o procedimento de Busca Local, que se resume à exploração da vizinhança `swap(1, 1)`. Tendo calculado μ, procede-se para o *loop* principal. A cada iteração, um subconjunto $S_i \in L'$ é escolhido como candidato a ser removido da solução atual. A interseção resultante de $L' - \{S_i\}$ já está calculada em $\mu[i]$. Apenas soluções parciais cuja interseção tenha cardinalidade maior do que o custo de L' são exploradas, pois estas são as únicas com potencial de gerar uma solução viável melhor do que a atual; assim, é possível ignorar as demais e reduzir o número de iterações (linha 3). Esse é um mecanismo de filtro que impede iterações infrutíferas e acelera a avaliação de um movimento na busca local. Ao conhecimento dos autores, o uso de uma estrutura de dados semelhante a μ, calculada de forma eficiente para servir de filtro na exploração da vizinhança, é uma abordagem inédita para o kMIS.

Dentre as possíveis opções para a escolha de qual subconjunto inserir em L', optou-se pelo critério de *first improvement*, isto é, o primeiro subconjunto que aumente a função objetivo será escolhido, pois esta opção gasta menos tempo. Nota-se que μ também é utilizado na avaliação da inserção (linha 7), o que evita calcular $L' - \{S_i\}$ diretamente. A complexidade da Busca Local é $O(k \cdot |L|)$ no pior caso. Observa-se, porém, que o filtro possibilitado por μ reduz bastante o custo computacional da busca.

Algoritmo 4 Local Search

```
1:  function LocalSearch(L')
2:      for S_i ∈ L' do
3:          if |μ[i]| ≤ cost(L') then          ▷ μ ajuda a evitar soluções parciais infrutíferas
4:              continue
5:          end if
6:          for S' ∈ L \ L' do
7:              if |μ[i] ∩ S'| > cost(L') then
8:                  L' ← swap(L', S_i, S')
9:                  return L'                   ▷ First improvement
10:             end if
11:         end for
12:     end for
13:     return L'
14: end function
```

4.5 Perturbação

A fase de perturbação do ILS serve para diversificar a solução atual, na esperança de escapar de ótimos locais. Para o kMIS, na fase de perturbação é comum remover aleatoriamente um certo número de subconjuntos da solução atual e em seguida inserir novos subconjuntos que antes estavam de fora.

Este trabalho adota a seguinte estratégia de perturbação, conforme ilustrado no Algoritmo 5. Inicialmente (linha 2), calcula-se q, a quantidade de subconjuntos a ser removidos da solução

atual L'. O cálculo é fixo em $\lfloor\sqrt{k}\rfloor$ para cada instância. A seguir, removem-se aleatoriamente q subconjuntos de L' para criar a solução parcial L_p (linha 3). L_p será completada a partir da seleção, também aleatória, de subconjuntos em $L \setminus L'$ (linhas 4-6). Ao final, retorna-se a solução perturbada L_p.

Algoritmo 5 Perturbation

```
1: function PERTURBATION(L' = {S_1, ..., S_k})
2:     q ← ⌊√k⌋
3:     L_p ← RemoveRandomSubsets(L', q)
4:     while |L_p| < k do
5:         S ← SelectRandomSubset(L \ L')
6:         L_p ← L_p ∪ {S}
7:     end while
8:     return L_p
9: end function
```

4.6 Critério de Aceitação

O Algoritmo 6 mostra o critério de aceitação de uma solução atualmente explorada L' em relação ao ótimo global L^* no ILS. Trata-se simplesmente da comparação entre as funções objetivo de cada uma. Se L' tem uma função objetivo maior (linha 2), ela passa a ser L^* e é retornada ao loop principal para ser explorada. Do contrário, L^* não é modificada.

Algoritmo 6 AcceptanceCriteria

```
1: function ACCEPTANCECRITERIA(L', L*)
2:     if cost(L') > cost(L*) then
3:         L* ← L'
4:     end if
5:     return L*
6: end function
```

5

Resultados e Discussões

Neste capítulo, os resultados computacionais obtidos a partir da implementação do ILS são apresentados. Primeiramente, a Seção 5.1 descreve as instâncias-teste utilizadas nos experimentos. A Seção 5.2 descreve o ambiente onde os experimentos foram realizados e as ferramentas utilizadas. Por fim, a Seção 5.3 apresenta os resultados computacionais obtidos e a discussão.

5.1 Instâncias

As instâncias utilizadas foram as introduzidas por Dias et al. [14]. Tratam-se de 238 instâncias divididas em três grupos: no primeiro, o número de subconjuntos é igual ao número de elementos, isto é, $|L| = |R|$; no segundo, o número de subconjuntos é maior ($|L| > |R|$) e, no terceiro, é maior o número de elementos ($|L| < |R|$). Baseando-se na densidade das instâncias, também é possível dividi-las em classes: quanto maior a classe, maior a densidade. C_1, C_4 e C_7 têm valores baixos de k, C_2, C_5 e C_8 têm valores médios e C_6 e C_9 têm valores altos.

5.2 Ambiente e Ferramentas

Todos os testes foram executados em um computador equipado com um processador Intel Core i7-9700K com clock de 3,6 GHz, 16 GB de memória RAM, tendo o Ubuntu 22.04.3 LTS como sistema operacional. O algoritmo foi implementado na linguagem `C++` e compilado com o GNU g++ versão 13.2.0 utilizando a flag de otimização `-O3`.

5.3 Resultados Experimentais

Para avaliar o desempenho do ILS, ele foi comparado com as duas últimas meta-heurísticas para o kMIS encontradas na literatura, a saber, o VNS apresentado por Dias et al. [14] e o GRASP

com Busca Tabu apresentado por Casado et al. [9]. Os autores disponibilizaram os respectivos códigos-fontes. O VNS foi implementado em `C++`, executado com o `GNU g++` versão 13.2.0 utilizando a flag de otimização `-O3`; e o GRASP foi implementado em `Java` e executado com o `JDK` 21.0.3.

Para que as comparações entre os algoritmos sejam justas, todos foram executados no mesmo computador. Além disso, o tempo de execução dos três algoritmos foi fixado em $\frac{k}{10}$ segundos para cada instância. Desejando evitar o viés produzido pelos elementos randômicos das implementações, os resultados foram computados a partir de 10 execuções independentes de cada algoritmo em cada instância. Seguindo esse protocolo, foi construída a Tabela 5.1. Trata-se de um sumário em que os resultados são exibidos em relação às classes das instâncias. Assim, a coluna *Qnt.* mostra a quantidade de instâncias em uma classe; a coluna *Best* tem a média das melhores soluções obtidas pelos algoritmos nas instâncias de uma dada classe nas 10 execuções; *Worst* é a média dos piores valores obtidos por classe; *Avg.* é o valor médio encontrado por classe, independentemente de ser o melhor ou o pior. Na última linha estão as médias das colunas ponderadas pela quantidade de instâncias em cada classe. *# Bests* é a contagem dos melhores valores encontrados pelos algoritmos em cada instância, sem levar em consideração as classes.

Tabela 5.1: Comparação do ILS com o VNS [14] e o GRASP [9]

Class	**Qnt.**	**VNS [14]**			**GRASP [9]**			**ILS**		
		Best	Worst	Avg.	Best	Worst	Avg.	Best	Worst	Avg
1	30	4,27	**4,20**	**4,22**	4,23	4,07	4,19	**4,30**	4,07	4,14
2	30	**1,10**	1,07	1,07	**1,10**	**1,10**	**1,10**	1,07	1,07	1,07
4	30	25,50	24,80	25,13	25,37	25,07	25,20	**33,80**	**26,87**	**30,09**
5	30	7,33	7,20	7,27	7,50	7,30	7,38	**11,57**	**8,37**	**9,74**
6	28	2,32	2,29	2,30	2,43	2,39	2,42	**4,39**	**3,21**	**3,78**
7	30	98,10	97,67	97,86	98,00	97,70	97,86	**102,13**	**99,10**	**100,58**
8	30	75,50	75,33	75,41	75,60	75,27	75,42	**86,40**	**84,60**	**85,44**
9	30	50,10	50,07	50,08	50,17	50,07	50,12	**76,93**	**72,10**	**74,81**
#Bests			120			130			**212**	

Nota-se que, exceto para a classe 2, em média o ILS encontra os valores mais altos em todas as classes, com larga vantagem em alguns casos. A contagem de melhores valores também mostra a superioridade do ILS: ele encontra 212 melhores valores, mais do que os 130 do GRASP e os 120 do VNS.

Uma análise aprofundada das instâncias da classe 2, em que o ILS tem performance inferior, mostra que as instâncias têm baixa densidade e valores médios de k. Além disso, grande parte dos valores encontrados é 1, isto é, na maioria dos casos só há um elemento resultante da interseção entre os k conjuntos selecionados.

Os resultados completos do experimento estão disponíveis no apêndice, na Tabela 6.1.

Portanto, o ILS surge como uma proposta competitiva de meta-heurística para o *k*MIS, obtendo em pouco tempo soluções de melhor qualidade do que os métodos encontrados na literatura. É possível atribuir esses resultados aos ganhos de performance obtidos na fase de busca local pelo uso do array μ, o qual permite mais iterações no tempo de execução e, consequentemente, uma busca mais ampla.

5.4 Impacto dos Componentes do Algoritmo

Este experimento visa analisar o impacto dos componentes usados no algoritmo apresentado neste trabalho, incluindo estrutura de dados auxiliar μ, a estratégia de perturbação e a fase de busca local. Para tanto, foram geradas quatro versões distintas do algoritmo. A primeira, nomeada `ILS-1`, é o ILS como foi apresentado neste trabalho. A segunda, denominada `ILS-2`, é o ILS sem uso da estrutura μ. Para a terceira versão, `ILS-3`, a fase de perturbação do ILS foi substituída pela estratégia reativa apresentada por Dias et al. [14], a qual adapta a fase de construção do GRASP e leva em conta o histórico das soluções geradas no algoritmo para tornar o algoritmo mais ou menos guloso. O quarto algoritmo, denominado `ILS-4`, substitui a fase de busca local do ILS por uma estratégia VND (*Variable Neighborhood Descent*) [20], na qual se soma à estrutura de vizinhança `swap(1, 1)` a estrutura `swap(2, 2)` que, semelhantemente à primeira, faz a troca de dois subconjuntos em uma solução *L* e também utiliza uma estrutura de dados auxiliar para ter soluções parciais com antecedência.

Tabela 5.2: Componentes que formam as diferentes versões do ILS

Versão	**Componentes**		
	Pertubação Reativa	**Estruturas de Dados**	**`swap(2, 2)`**
`ILS-1`		✓	
`ILS-2`			
`ILS-3`	✓	✓	
`ILS-4`		✓	✓

Para esta análise, empregaram-se gráficos de distribuição de probabilidade. *Time-to-target plots* (ou *ttt-plots*) são amplamente utilizados na literatura para comparar e caracterizar os tempos de execução de diferentes algoritmos estocásticos para problemas de otimização combinatória [15, 1, 2, 30]. Esse tipo de gráfico exibe, no eixo das ordenadas, a probabilidade de um algoritmo encontrar, para uma dada instância, uma solução no mínimo tão boa quanto um valor *target*, em um dado tempo de execução exibido no eixo das abscissas.

Seja $\mathcal{P}$ um problema de otimização e $\mathcal{H}$ uma heurística randomizada para este problema. Seja, ainda, I uma instância específica de $\mathcal{P}$ e *target* um valor objetivo para essa instância. Para construir um *ttt-plot*, segue-se o seguinte protocolo: a heurística $\mathcal{H}$ é executada N vezes em uma instância fixa I, tendo como critério de parada encontrar uma solução igual ou melhor do que o

target. Garante-se que as N execuções sejam independentes por meio da reinicialização do *seed* de números aleatórios em cada uma delas. O tempo de cada execução é salvo e, ao final, eles são ordenados em ordem crescente. Então, atribui-se ao i-ésimo tempo de execução ordenado a probabilidade $p_i = \frac{i-\frac{1}{2}}{N}$. Por fim, são gerados e plotados os pontos $y_i = (t_i, p_i) \quad \forall i \in \{1, \cdots, N\}$. Neste trabalho, definiu-se $N = 100$ e limitou-se o tempo de cada execução a 30 segundos.

A Figura 5.1 mostra os *ttt-plots* gerados a partir da execução do protocolo descrito. As instâncias foram selecionadas a partir do conjunto de Dias et al. [14], explorado no experimento anterior. Elas foram escolhidas com base no tamanho, definido pelos valores de $|R|$ e $|L|$ e levando em conta que, no conjunto de dados usado, o valor mínimo de $|R|$ e de $|L|$ é 32 e o máximo é 300. O objetivo é amostrar o comportamento do algoritmo em instâncias das mais simples às mais complexas. Assim, foi escolhida uma instância pequena, duas médias e uma grande. Os *targets* foram definidos como aproximadamente 90% dos melhores valores encontrados pelo `ILS-1` no experimento competitivo.

A Figura 5.1 agrupa os plots das instâncias escolhidas, descritas a seguir:

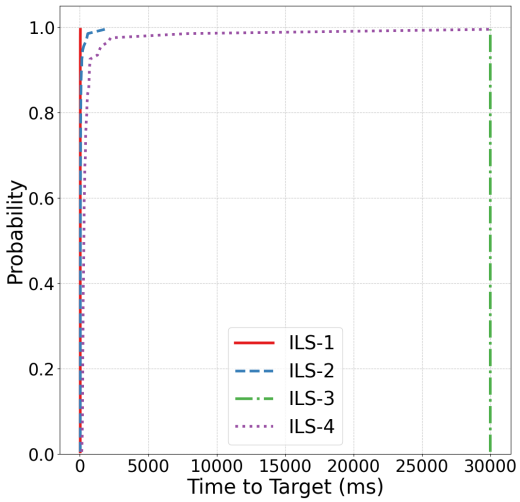
- Figura 5.1a: `classe_9_80_80`. Tem $|L| = 80$, $|R| = 80$ e $k = 72$; é considerada, portanto, uma instância pequena. *Target* usado: 66.
- Figura 5.1b: `classe_9_200_200`. Tem $|L| = |R| = 200$ e $k = 158$; é considerada, portanto, média. *Target* usado: 128.
- Figura 5.1c: `classe_4_224_280`. Tem $|L| = 224$, $|R| = 280$ e $k = 45$; é considerada média, um pouco maior do que a anterior. *Target* usado: 21
- Figura 5.1d: `classe_7_280_280`. Tem $|L| = |R| = 280$ e $k = 63$; é considerada, portanto, grande. *Target* usado: 33.

Analisando os quatro gráficos, é evidente que o `ILS-1` apresenta o melhor desempenho nas quatro instâncias. Ele apresenta larga vantagem na instância média e, nas demais, embora possa haver alguma proximidade inicial com os tempos de `ILS-2`, a estrutura de dados μ atua notavelmente na convergência do algoritmo, aumentando a probabilidade do algoritmo atingir o objetivo com maior velocidade, conforme refletido na área entre as curvas. Essa marca pode demorar mais que o dobro do tempo para o `ILS-2`, sem μ, e em alguns casos sequer é atingida, como na instância `classe_9_200_200`, na qual o `ILS-2` não atinge o alvo dentro dos 30 segundos estabelecidos algumas vezes.

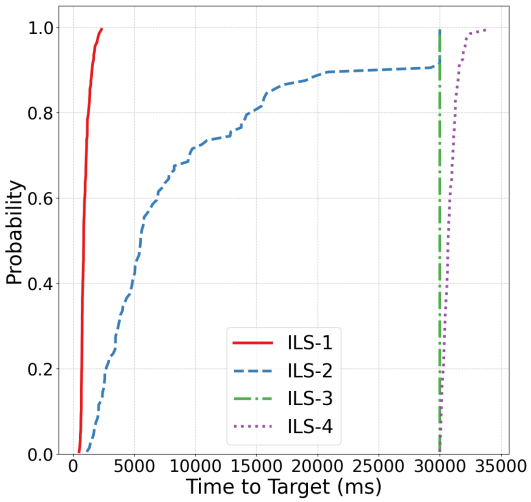
É interessante notar que o `ILS-3`, que usa a estratégia de perturbação reativa de Dias et al. [14], não atinge os alvos dentro do tempo limite na maior parte dos casos; ele tem pequenas probabilidades de atingi-lo apenas na terceira instância. Isso demonstra a pouca eficácia de estratégias complexas de perturbação para o kMIS, em contraste com o algoritmo mais simples adotado neste trabalho (Seção 4.5).

Por fim, o algoritmo `ILS-4` implementa uma estratégia de busca local mais complexa, com uma estrutura de vizinhança mais ampla e até mesmo estruturas de dados auxiliares. Não consegue, porém, atingir o alvo na instância `classe_9_200_200`. Nas demais, costuma ter um tempo pior que o `ILS-1` e o `ILS-2`, sobretudo na convergência (`classe_9_80_80`). Além disso, em nenhuma instância existe 100% de probabilidade do algoritmo atingir o alvo. Isso mostra que também estratégias de busca local mais complexas podem ser pouco eficazes para o *k*MIS.

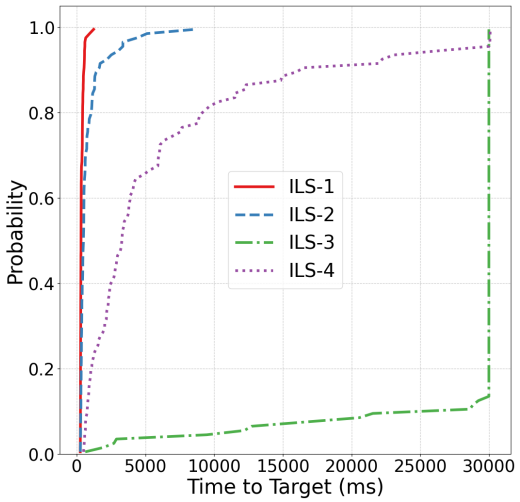
Tais observações indicam a importância das estruturas de dados utilizadas na fase de busca local do ILS e da simplicidade dos seus componentes no desempenho em termos de tempo de execução.



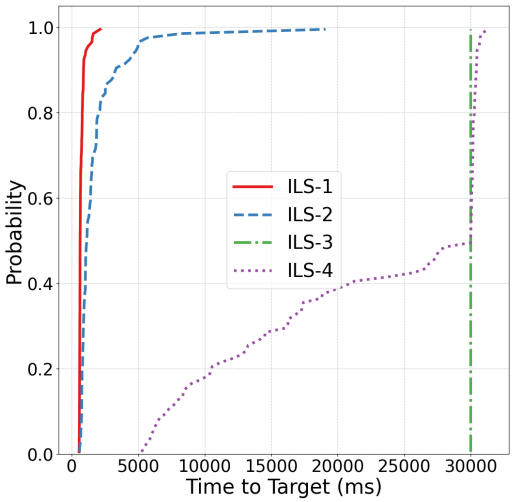
(a) classe_9_80_80 (66)



(b) classe_9_200_200 (128)



(c) classe_4_224_280 (21)



(d) classe_7_280_280 (33)

Figura 5.1: *ttt-plots* de 4 instâncias para 4 versões diferentes do ILS.

6

Conclusão

Este trabalho apresentou um algoritmo de *Iterated Local Search* (ILS) para o Problema da Interseção Máxima de k-Subconjuntos (kMIS). O método proposto foi comparado com dois algoritmos de ponta da literatura — GRASP e VNS — demonstrando desempenho superior. O ILS obteve a melhor solução em 212 instâncias, superando o GRASP (130 melhores valores) e o VNS (120 melhores valores). Além disso, a estrutura de dados μ reduz significativamente o tempo de execução da busca local, tornando o ILS mais eficiente. Além de melhorar o desempenho, μ também atua como um mecanismo de filtragem, permitindo que o algoritmo descarte soluções não promissoras precocemente, otimizando ainda mais o processo de busca.

Pesquisas futuras incluem o desenvolvimento de uma abordagem híbrida que combine as vantagens do ILS com um método exato. Por exemplo, pode-se utilizar as informações encontradas durante a busca heurística para alimentar uma formulação matemática baseada em programação inteira, a fim de encontrar soluções aprimoradas, que, por sua vez, podem ser realimentadas no procedimento heurístico, e assim sucessivamente. Planeja-se, também, estender o algoritmo proposto para resolver problemas relacionados, como o Problema da Cobertura Máxima [10].

Referências bibliográficas

[1] Renata M. Aiex, Mauricio G. C. Resende, Panos M. Pardalos, and Gerardo Toraldo. GRASP with path relinking for three-index assignment. *INFORMS Journal on Computing*, 17(2):224–247, 2005. **DOI** 10.1287/ijoc.1030.0059.

[2] Renata M. Aiex, Maurício G. C. Resende, and Celso C. Ribeiro. TTT plots: a perl program to create time-to-target plots. *SpringerNature Complete Journals*, 1(4):355–366, 2006. **DOI** 10.1007/s11590-006-0031-4.

[3] Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[4] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[5] Christian Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4):353–373, 2005. ISSN 1571-0645.
DOI https://doi.org/10.1016/j.plrev.2005.10.001. URL
https://www.sciencedirect.com/science/article/pii/S1571064505000333.

[6] J BnnoBRs. Partitioning procedures for solving mixed-variables programming problems. *Numer. Math*, 4(1):238–252, 1962.

[7] Eduardo Theodoro Bogue. O problema da máxima interseção de k-subconjuntos. Master's thesis, Universidade Federal de Campinas, 2014.

[8] Y.Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 105–112 vol.1, 2001. **DOI** 10.1109/ICCV.2001.937505.

[9] Alejandra Casado, Sergio Pérez-Peló, Jesús Sánchez-Oro, and Abraham Duarte. A grasp algorithm with tabu search improvement for solving the maximum intersection of k-subsets problem. *Journal of Heuristics*, 28(1):121–146, Feb 2022. ISSN 1572-9397.

DOI 10.1007/s10732-022-09490-8. URL https://doi.org/10.1007/s10732-022-09490-8.

[10] Reuven Cohen and Liran Katzir. The generalized maximum coverage problem. *Information Processing Letters*, 108(1):15–22, 2008. ISSN 0020-0190. **DOI** https://doi.org/10.1016/j.ipl.2008.03.017. URL https://www.sciencedirect.com/science/article/pii/S0020019008000896.

[11] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. ISBN 9781450374644. **DOI** 10.1145/800157.805047. URL https://doi.org/10.1145/800157.805047.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2022.

[13] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

[14] Fabio Dias, Wladimir Tavares, and Robertty Freitas. Reactive vns algorithm for the maximum k-subset intersection problem. *Journal of Heuristics*, 26:1–29, 12 2020. **DOI** 10.1007/s10732-020-09452-y.

[15] Thomas A. Feo, Maurício G. C. Resende, and Stuart Smith. A greedy randomized adaptive search procedure for maximum independent set. *Miscellaneous*, 42(5):860–878, 1994. **DOI** 10.1287/opre.42.5.860.

[16] Hélder Silva Ferreira and Rian Pinheiro. Algoritmo ils-vnd para o problema da interseção máxima de k-subconjuntos. In *Anais do LVI Simpósio Brasileiro de Pesquisa Operacional*, Fortaleza, Brasil, 2024. Galoá. URL https://proceedings.science/sbpo/sbpo-2024/trabalhos/algoritmo-ils-vnd-para-o-problema-da-intersecao-maxima-de-k-subconjuntos?lang=pt-br.

[17] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman and Compant, New York, 1979.

[18] Fred Glover. *Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges*, pages 1–75. Springer US, Boston, MA, 1997. ISBN 978-1-4615-4102-8. **DOI** 10.1007/978-1-4615-4102-8_1.

[19] Ralph E. Gomory. *Outline of an Algorithm for Integer Solutions to Linear Programs and An Algorithm for the Mixed Integer Problem*, pages 77–103. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-540-68279-0.

DOI 10.1007/978-3-540-68279-0_4. URL https://doi.org/10.1007/978-3-540-68279-0_4.

[20] Pierre Hansen, Nenad Mladenović, Jack Brimberg, and José A. Moreno Pérez. Variable neighborhood search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics, Second Edition*, pages 61–86. Springer, New York, USA, 2010.

[21] Larisa Komosko, Mikhail Batsyn, Pablo San Segundo, and Panos Pardalos. A fast greedy sequential heuristic for the vertex colouring problem based on bitwise operations. *Journal of Combinatorial Optimization*, 31, 03 2015. **DOI** 10.1007/s10878-015-9862-1.

[22] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(8):235–284, 2008. URL http://jmlr.org/papers/v9/krause08a.html.

[23] Fabian Kuhn, Thomas Moscibroda, and Rogert Wattenhofer. What cannot be computed locally! In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '04, page 300–309, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138024. **DOI** 10.1145/1011767.1011811. URL https://doi.org/10.1145/1011767.1011811.

[24] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm- a literature review. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 380–384, 2019. **DOI** 10.1109/COMITCon.2019.8862255.

[25] Ailsa H. Land and Alison G. Doig. *An Automatic Method for Solving Discrete Programming Problems*, pages 105–132. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-540-68279-0. **DOI** 10.1007/978-3-540-68279-0_5. URL https://doi.org/10.1007/978-3-540-68279-0_5.

[26] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search: Framework and applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics, Second Edition*, pages 363–397. Springer, New York, USA, 2010.

[27] GL Nemhauser and LA Wolsey. Integer programming and combinatorial optimization. In *Proceedings of the IPCO: International Conference on Integer Programming and Combinatorial Optimization, Houston, TX, USA*, pages 22–24. Springer, 1998.

[28] Doron Nussbaum, Shuye Pu, Jörg-Rüdiger Sack, Takeaki Uno, and Hamid Zarrabi-Zadeh. Finding maximum edge bicliques in convex bipartite graphs. *Algorithmica*, 64:311–325, 2012.

[29] Christos Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, volume 32. Dover Publications, 01 1982. ISBN 0-13-152462-3. **DOI** 10.1109/TASSP.1984.1164450.

[30] Alberto Reyes and Celso C. Ribeiro. Extending time-to-target plots to multiple instances. *International Transactions in Operational Research*, 25(5):1515–1536, 2018. **DOI** 10.1111/itor.12507.

[31] R.A. Rutenbar. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26, 1989. **DOI** 10.1109/101.17235.

[32] Pablo San Segundo, Diego Rodríguez-Losada, and Agustín Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38 (2):571–581, 2011.

[33] Stall A. Vinterbo. A note on the hardness of the k-ambiguity. Technical report, Harvard Medical School, 07 2002.

[34] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

[35] Eduardo C. Xavier. A note on a maximum k-subset intersection problem. *Information Processing Letters*, 112(12):471–472, 2012. ISSN 0020-0190. **DOI** 10.1016/j.ipl.2012.03.007.

Apêndice - Resultados Completos

Tabela 6.1: Resultados completos dos algoritmos VNS, GRASP e ILS

Instance	VNS [14]			GRASP [9]			ILS		
	Best	**Worst**	**Avg**	**Best**	**Worst**	**Avg**	**Best**	**Worst**	**Avg**
classe_1_100_100	7	7	7	7	7	7	7	6	6,9
classe_1_100_80	4	4	4	4	3	4	3	3	3
classe_1_112_140	5	5	5	5	5	5	5	5	5
classe_1_140_112	6	6	6	6	5	6	5	5	5
classe_1_140_140	4	4	4	4	4	4	4	4	4
classe_1_144_180	7	7	7	7	6	7	6	6	6
classe_1_160_200	4	4	4	4	4	4	3	3	3
classe_1_180_144	5	5	5	5	4	5	4	4	4
classe_1_180_180	3	3	3	3	3	3	3	3	3
classe_1_192_240	5	4	4,6	5	4	5	4	4	4
classe_1_200_160	2	2	2	2	2	2	2	2	2
classe_1_200_200	4	4	4	4	4	4	4	4	4
classe_1_224_280	3	3	3	3	3	3	3	3	3
classe_1_240_192	5	5	5	5	5	5	5	5	5
classe_1_240_240	2	2	2	2	2	2	2	2	2
classe_1_240_300	6	5	5,1	5	5	5	5	5	5
classe_1_280_224	4	4	4	4	4	4	4	4	4
classe_1_280_280	5	5	5	5	5	5	4	4	4
classe_1_300_240	3	3	3	3	3	3	3	3	3
classe_1_300_300	3	3	3	3	3	3	3	3	3
classe_1_32_40	3	3	3	3	3	3	5	4	4,3
classe_1_40_32	3	3	3	3	3	3	3	3	3

Continua na próxima página

Tabela 6.1 continuada

Instância	VNS [14]			GRASP [9]			ILS		
	Best	**Worst**	**Avg**	**Best**	**Worst**	**Avg**	**Best**	**Worst**	**Avg**
classe_1_40_40	4	4	4	4	4	4	6	5	5,1
classe_1_48_60	6	6	6	6	6	6	7	6	6,1
classe_1_60_48	5	5	5	5	5	5	7	6	6,3
classe_1_60_60	5	5	5	5	5	5	5	5	5
classe_1_64_80	4	4	4	4	4	4	7	5	5,6
classe_1_80_100	4	4	4	4	4	4	3	3	3
classe_1_80_64	3	3	3	3	3	3	3	3	3
classe_1_80_80	4	4	4	4	4	4	4	4	4
classe_2_100_100	1	1	1	1	1	1	1	1	1
classe_2_100_80	1	1	1	1	1	1	1	1	1
classe_2_112_140	1	1	1	1	1	1	1	1	1
classe_2_140_112	1	1	1	1	1	1	1	1	1
classe_2_140_140	1	1	1	1	1	1	1	1	1
classe_2_144_180	1	1	1	1	1	1	1	1	1
classe_2_160_200	1	1	1	1	1	1	1	1	1
classe_2_180_144	1	1	1	1	1	1	1	1	1
classe_2_180_180	1	1	1	1	1	1	1	1	1
classe_2_192_240	1	1	1	1	1	1	1	1	1
classe_2_200_160	1	1	1	1	1	1	1	1	1
classe_2_200_200	1	1	1	1	1	1	1	1	1
classe_2_224_280	1	1	1	1	1	1	1	1	1
classe_2_240_192	1	1	1	1	1	1	1	1	1
classe_2_240_240	1	1	1	1	1	1	1	1	1
classe_2_240_300	1	1	1	1	1	1	1	1	1
classe_2_280_224	1	1	1	1	1	1	1	1	1
classe_2_280_280	1	1	1	1	1	1	1	1	1
classe_2_300_240	1	1	1	1	1	1	1	1	1
classe_2_300_300	1	1	1	1	1	1	1	1	1
classe_2_32_40	2	2	2	2	2	2	2	2	2
classe_2_40_32	1	1	1	1	1	1	1	1	1
classe_2_40_40	1	1	1	1	1	1	1	1	1
classe_2_48_60	1	1	1	1	1	1	1	1	1
classe_2_60_48	1	1	1	1	1	1	1	1	1

Continua na próxima página

Tabela 6.1 continuada

Instância	VNS [14]			GRASP [9]			ILS		
	Best	**Worst**	**Avg**	**Best**	**Worst**	**Avg**	**Best**	**Worst**	**Avg**
classe_2_60_60	2	1	1,9	2	2	2	1	1	1
classe_2_64_80	2	2	2	2	2	2	2	2	2
classe_2_80_100	1	1	1	1	1	1	1	1	1
classe_2_80_64	1	1	1	1	1	1	1	1	1
classe_2_80_80	1	1	1	1	1	1	1	1	1
classe_4_100_100	41	41	41	41	41	41	47	41	42,1
classe_4_100_80	22	22	22	22	22	22	22	22	22
classe_4_112_140	38	38	38	38	38	38	55	40	48,1
classe_4_140_112	33	32	32,4	33	33	33	43	32	35,5
classe_4_140_140	29	29	29	29	29	29	54	32	41,5
classe_4_144_180	46	45	45,8	46	46	46	64	46	53,9
classe_4_160_200	27	25	25,5	26	25	26	45	33	36,8
classe_4_180_144	31	31	31	31	31	31	43	32	37,4
classe_4_180_180	23	21	22,1	23	22	22	40	29	33,7
classe_4_192_240	34	33	33,9	34	34	34	55	38	45,3
classe_4_200_160	14	12	12,8	13	13	13	16	12	14,1
classe_4_200_200	29	27	28	29	28	29	45	31	39,3
classe_4_224_280	22	20	20,4	22	20	21	36	30	33,1
classe_4_240_192	33	33	33	33	33	33	39	32	34,9
classe_4_240_240	13	12	12,3	13	12	12	13	12	12,4
classe_4_240_300	42	40	41,7	42	41	42	63	47	58,1
classe_4_280_224	30	27	28,9	30	28	29	30	29	29,9
classe_4_280_280	8	7	7,1	7	7	7	7	7	7
classe_4_300_240	15	14	14,1	14	14	14	14	14	14
classe_4_300_300	3	3	3	3	3	3	3	3	3
classe_4_32_40	19	19	19	19	19	19	29	21	23,3
classe_4_40_32	17	17	17	17	17	17	17	17	17
classe_4_40_40	21	21	21	21	21	21	26	24	25,2
classe_4_48_60	26	26	26	26	26	26	26	26	26
classe_4_60_48	25	25	25	25	25	25	26	26	26
classe_4_60_60	29	29	29	29	29	29	36	31	33,4
classe_4_64_80	31	31	31	31	31	31	31	31	31
classe_4_80_100	24	24	24	24	24	24	31	24	26

Continua na próxima página

Tabela 6.1 continuada

Instância	VNS [14]			GRASP [9]			ILS		
	Best	**Worst**	**Avg**	**Best**	**Worst**	**Avg**	**Best**	**Worst**	**Avg**
classe_4_80_64	15	15	15	15	15	15	23	15	20
classe_4_80_80	25	25	25	25	25	25	35	29	32,6
classe_5_100_100	8	8	8	8	8	8	14	9	11
classe_5_100_80	8	7	7,9	8	8	8	13	9	10,8
classe_5_112_140	9	8	8,9	9	9	9	17	11	14
classe_5_140_112	9	9	9	9	8	9	12	9	10
classe_5_140_140	10	10	10	10	10	10	12	10	10,9
classe_5_144_180	8	8	8	9	8	8	13	9	10,8
classe_5_160_200	6	5	5,1	6	5	5	5	5	5
classe_5_180_144	8	7	7,1	8	8	8	7	7	7
classe_5_180_180	5	5	5	6	5	6	5	5	5
classe_5_192_240	5	5	5	5	5	5	5	5	5
classe_5_200_160	4	4	4	5	5	5	4	4	4
classe_5_200_200	6	6	6	6	6	6	6	6	6
classe_5_224_280	5	5	5	5	5	5	5	5	5
classe_5_240_192	7	7	7	8	8	8	7	7	7
classe_5_240_240	6	6	6	6	6	6	6	6	6
classe_5_240_300	5	5	5	5	5	5	5	5	5
classe_5_280_224	7	7	7	8	7	7	7	7	7
classe_5_280_280	1	1	1	1	1	1	1	1	1
classe_5_300_240	5	5	5	5	5	5	5	5	5
classe_5_300_300	1	1	1	1	1	1	1	1	1
classe_5_32_40	14	14	14	14	14	14	24	15	20
classe_5_40_32	7	7	7	7	7	7	15	9	13,1
classe_5_40_40	10	10	10	10	10	10	16	12	13,8
classe_5_48_60	13	13	13	13	13	13	27	15	20
classe_5_60_48	11	11	11	11	11	11	19	14	16,3
classe_5_60_60	13	13	13	13	13	13	25	18	19,5
classe_5_64_80	9	9	9	9	9	9	24	14	17,3
classe_5_80_100	8	8	8	8	7	7	21	11	14,9
classe_5_80_64	6	6	6	6	6	6	10	7	8,4
classe_5_80_80	6	6	6	6	6	6	16	10	12,4
classe_6_100_100	2	2	2	2	2	2	6	3	3,8

Continua na próxima página

Tabela 6.1 continuada

Instância	VNS [14]			GRASP [9]			ILS		
	Best	**Worst**	**Avg**	**Best**	**Worst**	**Avg**	**Best**	**Worst**	**Avg**
classe_6_100_80	2	2	2	2	2	2	3	2	2,3
classe_6_112_140	3	2	2,3	3	3	3	3	3	3
classe_6_140_112	3	3	3	3	3	3	3	3	3
classe_6_140_140	3	3	3	4	3	4	3	3	3
classe_6_144_180	2	2	2	2	2	2	2	2	2
classe_6_160_200	1	1	1	1	1	1	1	1	1
classe_6_180_144	2	2	2	3	3	3	2	2	2
classe_6_180_180	1	1	1	1	1	1	1	1	1
classe_6_192_240	2	2	2	3	3	3	2	2	2
classe_6_200_160	1	1	1	1	1	1	1	1	1
classe_6_200_200	2	2	2	2	2	2	2	2	2
classe_6_224_280	1	1	1	1	1	1	1	1	1
classe_6_240_192	3	3	3	3	3	3	3	3	3
classe_6_240_240	2	2	2	2	2	2	2	2	2
classe_6_240_300	1	1	1	1	1	1	1	1	1
classe_6_280_224	3	3	3	3	3	3	3	3	3
classe_6_300_240	1	1	1	1	1	1	1	1	1
classe_6_32_40	7	7	7	7	7	7	16	10	13,1
classe_6_40_32	2	2	2	2	2	2	6	3	4,4
classe_6_40_40	3	3	3	3	3	3	10	8	8,5
classe_6_48_60	2	2	2	2	2	2	9	4	6,9
classe_6_60_48	5	5	5	5	5	5	10	8	8,7
classe_6_60_60	4	4	4	4	4	4	13	9	11,3
classe_6_64_80	3	3	3	3	3	3	11	6	8,6
classe_6_80_100	2	2	2	2	2	2	6	4	5,3
classe_6_80_64	1	1	1	1	1	1	1	1	1
classe_6_80_80	1	1	1	1	1	1	1	1	1
classe_7_100_100	100	100	100	100	100	100	100	100	100
classe_7_100_80	80	80	80	80	80	80	80	80	80
classe_7_112_140	140	140	140	140	140	140	140	140	140
classe_7_140_112	112	112	112	112	112	112	112	112	112
classe_7_140_140	140	140	140	140	140	140	140	140	140
classe_7_144_180	180	180	180	180	180	180	180	180	180

Continua na próxima página

Tabela 6.1 continuada

Instância	VNS [14]			GRASP [9]			ILS		
	Best	**Worst**	**Avg**	**Best**	**Worst**	**Avg**	**Best**	**Worst**	**Avg**
classe_7_160_200	197	197	197	197	197	197	199	197	197,8
classe_7_180_144	144	144	144	144	144	144	144	144	144
classe_7_180_180	173	173	173	173	173	173	179	176	177,4
classe_7_192_240	233	233	233	233	233	233	238	233	235,4
classe_7_200_160	151	151	151	151	150	151	159	157	158,2
classe_7_200_200	192	192	192	192	192	192	197	194	195,6
classe_7_224_280	43	41	42,3	42	41	42	58	47	54
classe_7_240_192	192	192	192	192	192	192	192	192	192
classe_7_240_240	39	37	38	39	37	38	58	44	49
classe_7_240_300	82	81	81,9	82	82	82	103	82	91,9
classe_7_280_224	56	53	54,2	56	55	55	71	58	64,8
classe_7_280_280	31	31	31	31	30	31	43	37	40,2
classe_7_300_240	31	27	27,8	29	28	29	39	34	36,8
classe_7_300_300	23	22	22,6	23	21	22	28	22	24,3
classe_7_32_40	40	40	40	40	40	40	40	40	40
classe_7_40_32	32	32	32	32	32	32	32	32	32
classe_7_40_40	40	40	40	40	40	40	40	40	40
classe_7_48_60	60	60	60	60	60	60	60	60	60
classe_7_60_48	48	48	48	48	48	48	48	48	48
classe_7_60_60	60	60	60	60	60	60	60	60	60
classe_7_64_80	80	80	80	80	80	80	80	80	80
classe_7_80_100	100	100	100	100	100	100	100	100	100
classe_7_80_64	64	64	64	64	64	64	64	64	64
classe_7_80_80	80	80	80	80	80	80	80	80	80
classe_8_100_100	86	86	86	86	86	86	100	97	97,9
classe_8_100_80	77	77	77	77	77	77	80	79	79,2
classe_8_112_140	115	115	115	115	115	115	134	132	133,1
classe_8_140_112	103	103	103	103	103	103	111	110	110,2
classe_8_140_140	129	129	129	129	129	129	138	137	137,5
classe_8_144_180	149	149	149	149	149	149	172	167	170,2
classe_8_160_200	144	144	144	144	144	144	179	176	176,8
classe_8_180_144	121	120	120,8	121	120	120	140	136	137,6
classe_8_180_180	126	126	126	126	126	126	167	163	165,1

Continua na próxima página

Tabela 6.1 continuada

Instância	VNS [14]			GRASP [9]			ILS		
	Best	**Worst**	**Avg**	**Best**	**Worst**	**Avg**	**Best**	**Worst**	**Avg**
classe_8_192_240	148	148	148	148	148	148	197	191	194
classe_8_200_160	111	110	110,2	110	109	110	145	136	139,9
classe_8_200_200	142	142	142	142	142	142	176	170	172,7
classe_8_224_280	9	9	9	10	10	10	9	9	9
classe_8_240_192	157	155	156	156	154	155	179	174	176,9
classe_8_240_240	12	11	11,2	12	11	12	12	11	11,4
classe_8_240_300	9	9	9	10	9	9	9	9	9
classe_8_280_224	14	14	14	14	13	14	14	13	13,1
classe_8_280_280	10	10	10	10	9	10	10	10	10
classe_8_300_240	9	9	9	10	9	9	9	9	9
classe_8_300_300	7	7	7	9	8	8	7	7	7
classe_8_32_40	40	40	40	40	40	40	40	40	40
classe_8_40_32	32	32	32	32	32	32	32	32	32
classe_8_40_40	39	39	39	39	39	39	40	40	40
classe_8_48_60	60	60	60	60	60	60	60	60	60
classe_8_60_48	48	48	48	48	48	48	48	48	48
classe_8_60_60	60	60	60	60	60	60	60	60	60
classe_8_64_80	78	78	78	78	78	78	80	80	80
classe_8_80_100	94	94	94	94	94	94	100	99	99,9
classe_8_80_64	62	62	62	62	62	62	64	63	63,8
classe_8_80_80	74	74	74	74	74	74	80	80	80
classe_9_100_100	57	57	57	57	57	57	92	83	87,6
classe_9_100_80	57	57	57	57	57	57	77	74	75,8
classe_9_112_140	80	80	80	80	80	80	124	116	118,5
classe_9_140_112	68	68	68	68	68	68	105	99	101,3
classe_9_140_140	92	92	92	92	92	92	130	124	126,8
classe_9_144_180	102	102	102	102	102	102	158	146	152,3
classe_9_160_200	90	90	90	90	90	90	157	145	151,4
classe_9_180_144	74	74	74	74	74	74	123	116	119,4
classe_9_180_180	58	58	58	58	58	58	136	121	130,6
classe_9_192_240	108	108	108	108	108	108	175	167	172,7
classe_9_200_160	59	59	59	59	59	59	118	108	113,7
classe_9_200_200	81	81	81	81	81	81	147	137	142,2

Continua na próxima página

Tabela 6.1 continuada

Instância	VNS [14]			GRASP [9]			ILS		
	Best	**Worst**	**Avg**	**Best**	**Worst**	**Avg**	**Best**	**Worst**	**Avg**
classe_9_224_280	3	3	3	3	3	3	3	3	3
classe_9_240_192	97	96	96,3	97	96	96	153	146	150,2
classe_9_240_240	3	3	3	4	3	4	4	3	3,1
classe_9_240_300	2	2	2	3	2	3	2	2	2
classe_9_280_224	5	5	5	5	5	5	5	5	5
classe_9_280_280	3	3	3	3	3	3	3	3	3
classe_9_300_240	3	3	3	3	3	3	3	3	3
classe_9_300_300	2	2	2	2	2	2	2	2	2
classe_9_32_40	39	39	39	39	39	39	40	40	40
classe_9_40_32	25	25	25	25	25	25	32	31	31,7
classe_9_40_40	32	32	32	32	32	32	39	34	37,3
classe_9_48_60	48	48	48	48	48	48	59	55	57,3
classe_9_60_48	40	40	40	40	40	40	48	47	47,9
classe_9_60_60	53	53	53	53	53	53	60	59	59,8
classe_9_64_80	60	60	60	60	60	60	79	76	77,4
classe_9_80_100	72	72	72	72	72	72	97	93	95,6
classe_9_80_64	40	40	40	40	40	40	61	54	59,9
classe_9_80_80	50	50	50	50	50	50	76	71	73,7
Average	33.29	33.08	33.18	33.31	33.13	33.22	40.37	37.71	39.00
#Best		120			130			212	